Trial Guide for StackOps Elastic (Serverless)

Objective

The StackOps Elastic trial allows users to explore observabilities in StackOps Elastic. The intent of this document aims to guide users on how to setup StackOps Elastic to monitor a manageable subset of their workloads during the trial and understand how to use Elastic observability for their day-to-day operation needs using capabilities like:

- Cloud resource monitoring
- Applications monitoring
    - Synthetic monitoring
    - Application Performance Monitoring (APM)
- Alert setting to zoom in failure or performance degradation
- Log query for troubleshooting

Background

The Strategic Imperative for Observability in Government

The focus moving from monitoring toward observability is fundamentally an architectural response to the complexity of distributed systems and microservices.

Traditional monitoring typically focuses on "inside-out" metrics—simple indicators of whether a server is up or down—which often results in the "watermelon effect," where dashboards appear green while users experience significant degradation.

Observability adopts both “inside-out” and "outside-in" approach, prioritizing user centricity and multi-signal correlation across logs, metrics, and traces to answer why and how a problem occurred rather than merely when it started.

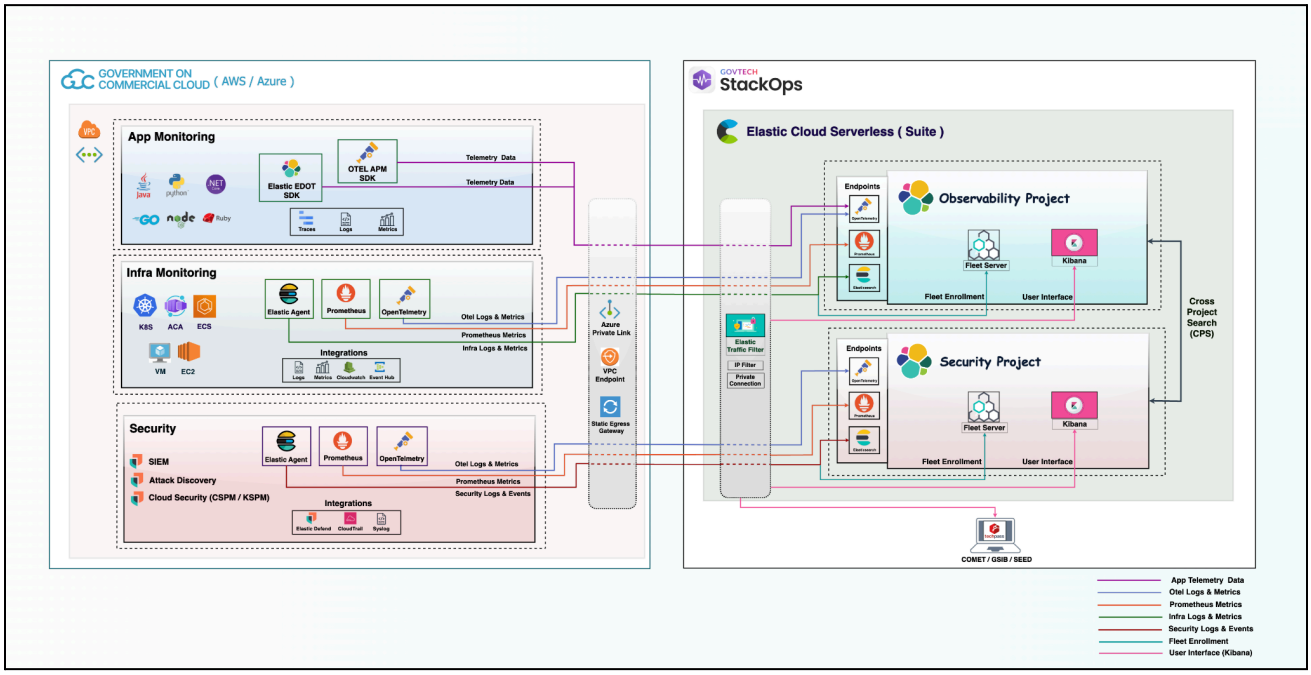
StackOps Elastic Overview

StackOps Elastic is one of the central monitoring/observability components within the Singapore Government Tech Stack (SGTS). Tailored for the Whole-of-Government (WOG), StackOps Elastic offers enhanced observability, centralized log monitoring, data aggregation, visualization, and security features across cloud applications and infrastructure.

By leveraging StackOps Elastic, agencies can improve system reliability, enhance application security, and streamline DevSecOps collaboration.

Architecture

The following reference diagrams outline how data is collected from the tenant's environment, securely transmitted, and processed within the StackOps platform for monitoring and analysis.



BEFORE requesting for StackOps Elastic Trial

To make the most of your trial, please ensure to have the following ready:

1. **A non-production GCC environment**
   This is the environment to be monitored during the trial.

2. **Capability to access and provision cloud resources**
   The trial will require provisioning and configuration of cloud resources including, but not limited to, the below:
   a. Virtual machine- for installation of an Elastic Agent.
   b. Private endpoint - for secure connectivity between your environment and StackOps Elastic project.
   c. Private hosted zone - to resolve your StackOps Elastic project to the private endpoint instead of going to the Internet.
   d. IAM permissions - an IAM role attached to your virtual machine
   e. Configuration of security groups, NACLs and route tables (where required)

   It is best to go through this document to identify where these resources are required. For scenarios that require other Elastic integrations you may want to use for the trial, you may refer to Elastic integrations (Elastic official documentation) as different integration may require different cloud resources.

3. **Network Connectivity**
   Port 443 is required for your environment to reach StackOps Elastic (outbound via the private endpoint)

What is an Elastic Agent?
A single, unified agent that simplifies the collection of logs, metrics, and security data across your entire infrastructure for monitoring.

Getting Started

Request For A StackOps Elastic Trial

Refer to Get a trial StackOps Elastic subscription (StackOps Documentation), it covers trial eligibility criteria, prerequisites and how-to guides to submitting a trial request, conversion to a paid subscription post trial and more up-to-date information.

Upon successful submission of your trial request, an email will be sent indicating that your StackOps Elastic trial is ready!

Setting up your StackOps Elastic Trial

Follow these steps to setup your StackOps Elastic for monitoring:

1. **Grant your teammates access** - refer to Manage Elastic users.

2. **Verify your access to Kibana URLs** - refer to Verify Kibana URL.
    a. There are TWO Kibana URLs: :
        i. **Observability Kibana URL**
        This directs to the StackOps Elastic **Observability** project
        ii. **Security Kibana URL**
        This directs to the StackOps Elastic **Security** project
    b. For the purpose of this trial, we will use the **Observability** project mostly for the guide (unless explicitly stated).

What is Kibana?
Kibana is the user interface (UI) for your StackOps Elastic project, acting as a real-time window to manage your data and your project.

What is the use case of each project?
StackOps Elastic consists of two projects in a suite. Each project type serves a different purpose. Refer to this link for more information.

3. **Secure the connection** between your workloads and StackOps Elastic so that your traffic never traverses the public Internet.

   FOR THE PURPOSE OF THIS TRIAL, the steps for this differ from Configure secure connection in the Developer Portal, follow the below steps instead:

   a. Carry out Step 1: Create private endpoints to Elastic Cloud

   b. **Skip** Step 2: Create traffic filters.
   To create a traffic filter in trial, raise a service request following the format below:

Field	**Value**
Summary	Create traffic filter for Trial
Description *Follow the format provided here.*	***Traffic Filter Name:*** *{Enter a preferred name, no underscore and limited to 80 characters}* ***Traffic Filter Description (optional):*** *{Enter a description}* *CHOOSE ONLY ONE FROM BELOW:* ***AWS Private Endpoint(s):*** *{VPC Endpoint IDs}* ***Azure Private Endpoint:*** *{Resource Name and Resource GUID}* ***IP Filtering:*** *{Source IP addresses or CIDR ranges}*
StackOps SR Category	Serverless
StackOps SR Sub Category	Virtual Private Cloud (VPC) Endpoint / Traffic Filter
Account Short Name	*{Account short name}*
Deployment Name / Kibana URL	*{Suite Name}*
Current Elastic Version	Serverless
Preferred Date/Time	*{Preferred date/time)*
Environment	QA/UAT/DEV
Reporting Agency	*{Your Agency}*

   c. StackOps support team will respond that your approval is required to complete the traffic filter creation. Approve the traffic filter request in GovTech Console - Approvals:
      i. Find the request named ***StackOps - [Create]<Account Short Name>_<Filter Name>***.
      ii. Click **Manage,** check the traffic filter details match your service request information.

iii. Click **Approve** once all information is verified to be correct.

d. Carry out Step 3: Associate traffic filters to associate your traffic filter to your project.

e. Carry out Step 4: Verify your endpoints upon completion of the service request.

Serverless Trial Guide (AWS)

Table of Contents

Serverless Trial Guide (AWS)..... 1
Setting up Monitoring..... 2
Stage 1: Single Page Browser Health Check..... 2
Stage 2: Cloud Resource Monitoring..... 5
EC2 Monitoring (Elastic Agent Approach)..... 7
EC2 Monitoring (Elastic Managed Integrations - Agentless)..... 18
Stage 3: Application Monitoring..... 20
Synthetic Monitoring (Multistep Browser Health Check)..... 20
Application Performance Monitoring (APM) with Elastic Observability and Opentelemetry..... 25
Java Instrumentation Strategy..... 25
.Net Instrumentation Strategy..... 26
Web Server Monitoring..... 30
Nginx Log Integration and Visualization..... 30
IIS Monitoring Integration and Visualization..... 31
Stage 4: Monitoring SaaS that provide a web page on its SaaS status..... 33
Day 2 Operations - Observing your environment..... 34
Stage 1: Setting up Alerts and Cases..... 34
Phase 1: Setting up Alerts..... 35
Phase 2: Using Cases for case management..... 38
Stage 2: Interrogate your data with Discover..... 41
Stage 3: Personalize your data..... 44
Phase 1: Onboard AWS VPC Flow Logs..... 45
Phase 2: Create a Personalized Dashboard..... 48
Stage 4: Tracing your Application flow..... 59
Annex..... 63

Setting up Monitoring

In the upcoming sections, you are going to focus on the following items in your 14-day StackOps trial journey.

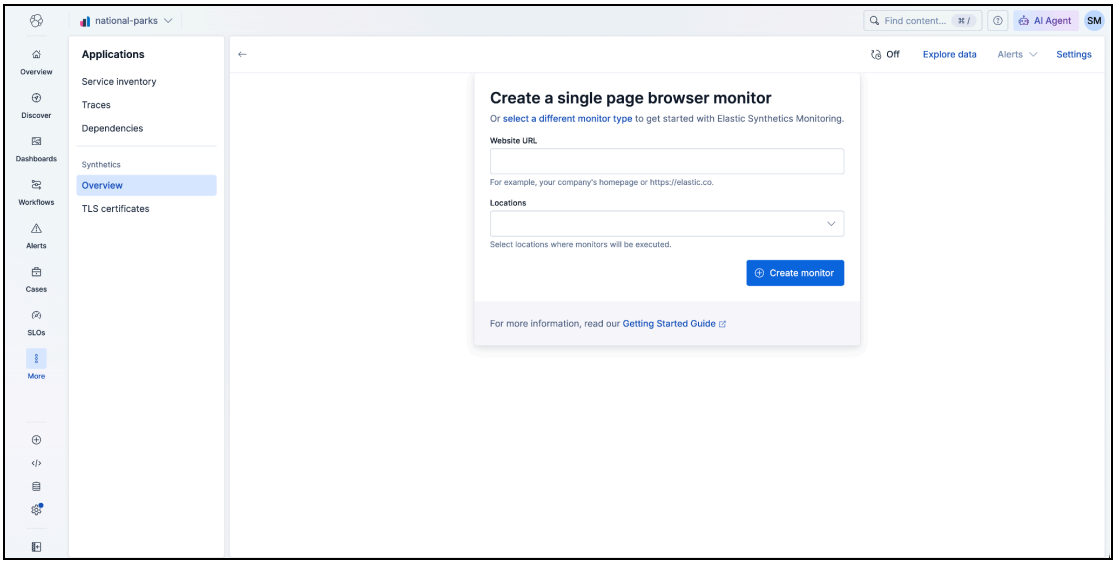


Stage 1: Single Page Browser Health Check

The trial begins with lightweight monitors, including ICMP, TCP, and HTTP pings. An ICMP monitor verifies network reachability, while an HTTP monitor checks that a specific URL returns a successful status code (e.g., 200 OK) and contains expected response text. These monitors can be configured to run at high frequencies—every minute or less—from multiple global locations or private locations within the agency's own network.

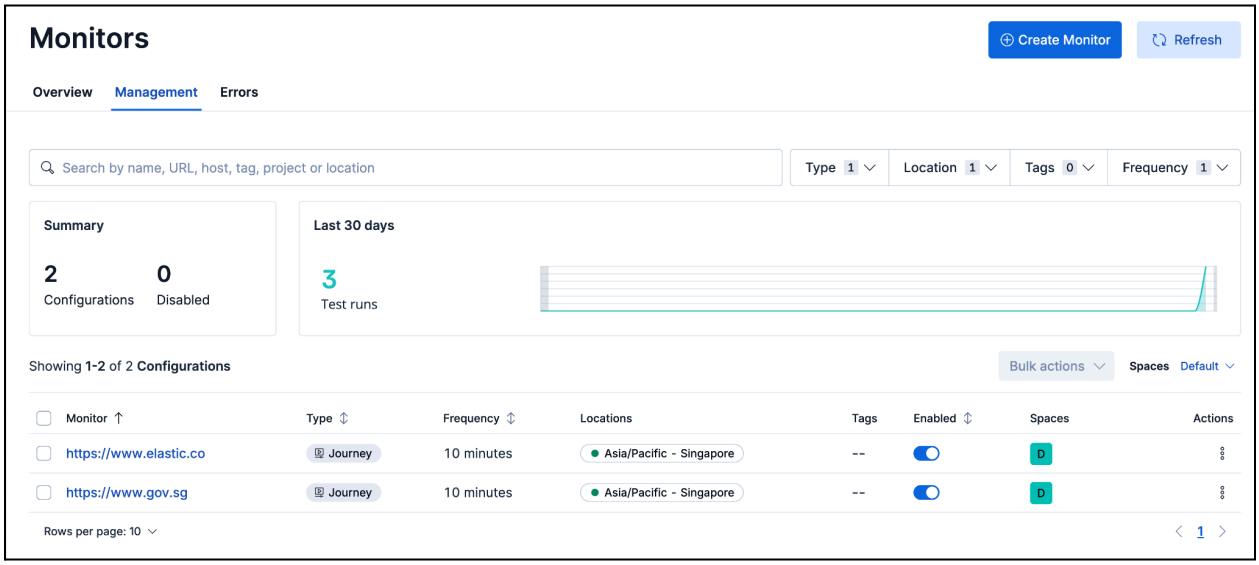
To configure the Single Page Browser Health Check, follow the steps below to monitor from Elastic multiple global locations.

1. In Kibana, go to **Applications → Synthetics → Overview**
2. Create a **Single Page Browser Test** monitor.

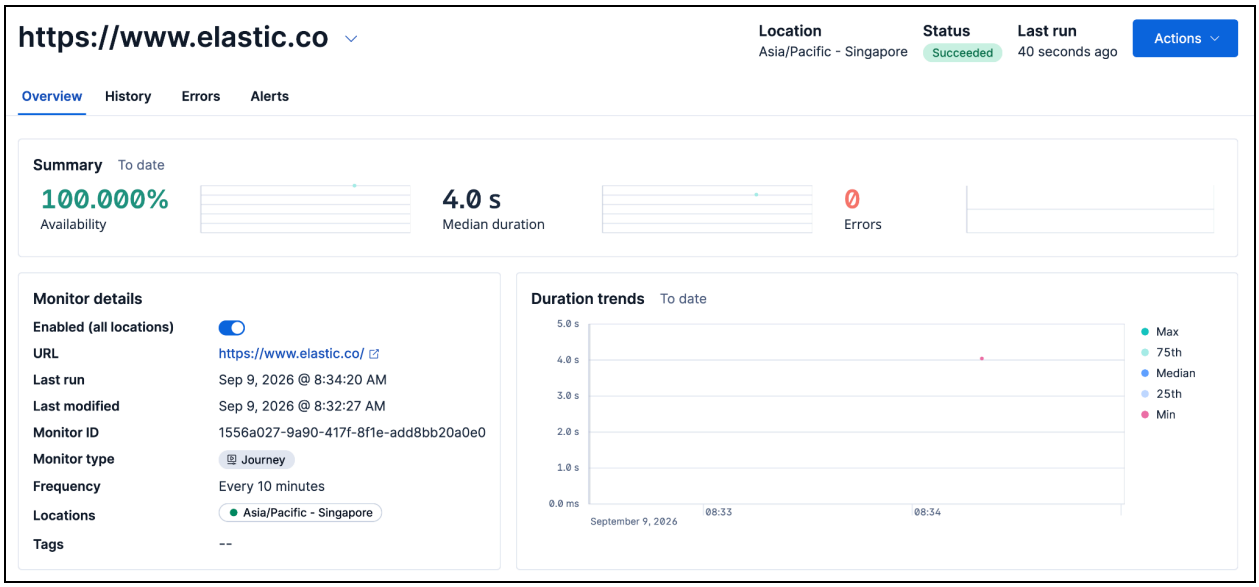


3. Enter your agency's public URL and select a location (Asia/Pacific - Singapore) → **Create monitor**

4. Click on your newly created monitor

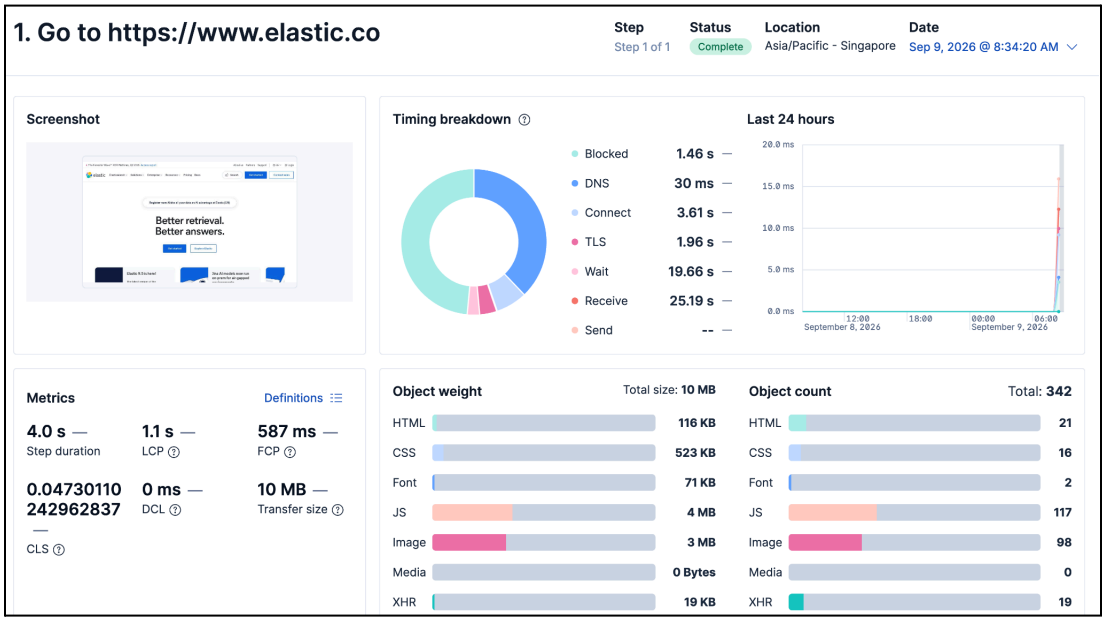


Dopamine Moment: Within 60 seconds, you'll see a green checkmark!



- Scroll down to the **Last test run** and click on the View step 1 details - you'll see a breakdown of webpage load times

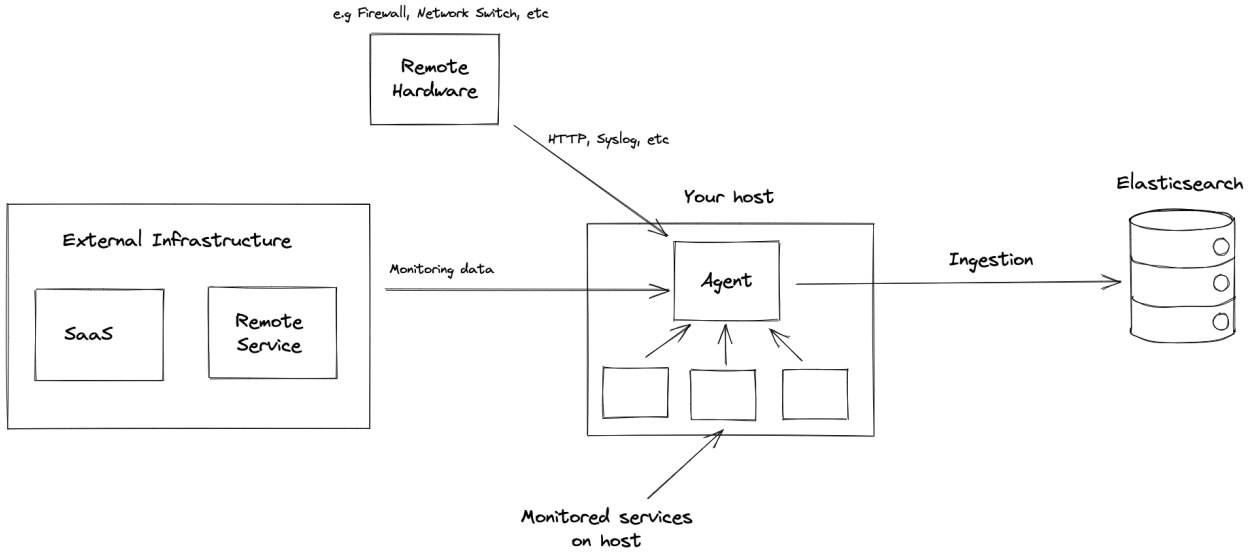




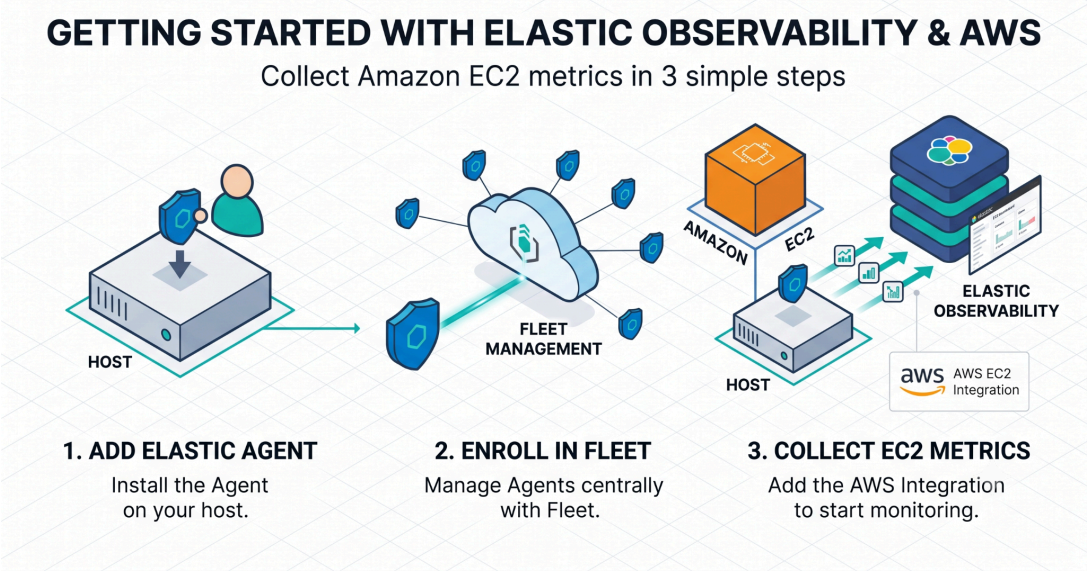
Stage 2: Cloud Resource Monitoring

Now we move into the monitoring of your Cloud Resources. For this section the trial shall focus on the monitoring of EC2 instances and AWS Cloud Infrastructure.

Elastic provides an Agent (Elastic Agent) which allows you to collect logs and metrics from multiple data sources via one unified Agents, this Agent then ships all data over to the Elasticsearch backend



For this next part, we are going to set up an Elastic Agent in order to collect telemetry from AWS EC2 Instances.



The following steps will guide on setting up monitoring for an AWS EC2. It involves setting up an Elastic Agent and configuring an agency policy to integrate with AWS CloudWatch to monitor the EC2 instance. Similarly, the agency policy can be configured to pull other AWS Cloud Resource telemetries for monitoring.

There are two options for this, we are going to use the Elastic Agent option, however, moving forward you can also use the Agentless option.

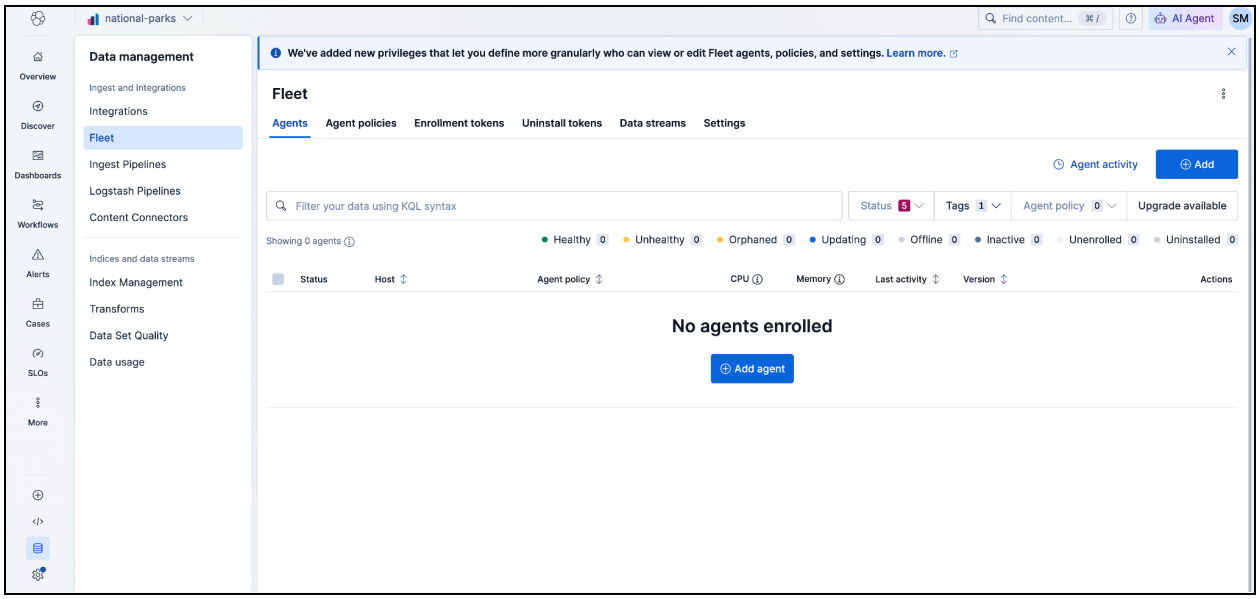
1. Use the Elastic Agent (You host the Elastic Agent)
2. Use Elastic Managed Integrations (Agentless)

EC2 Monitoring (Elastic Agent Approach)

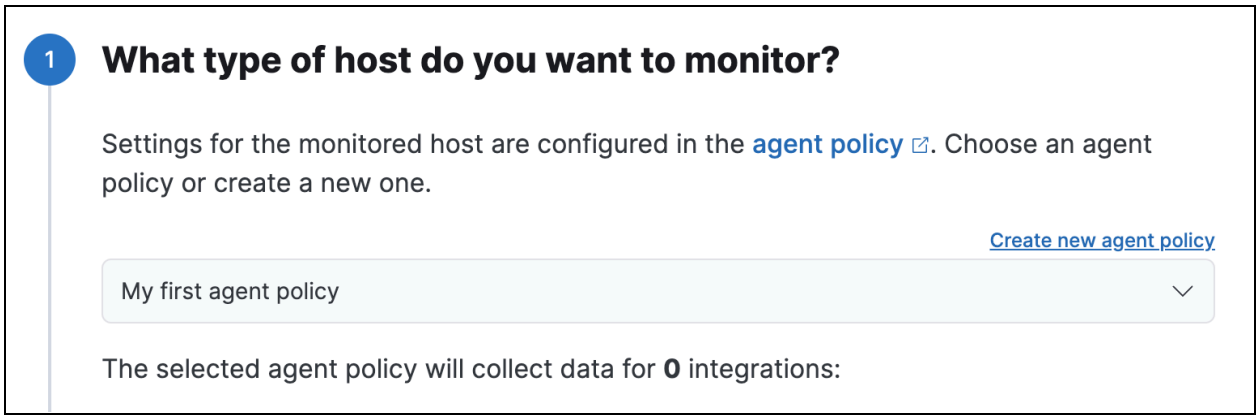
Add the Elastic Agent and Enroll in Fleet Management

1. **Go to the Centralised Agent Management:**
    a. In Kibana, go to Agents in **Data Management → Fleet**
    b. Or, **the quick route** - use the Find content search bar in the top right of the screen and search for **Fleet**

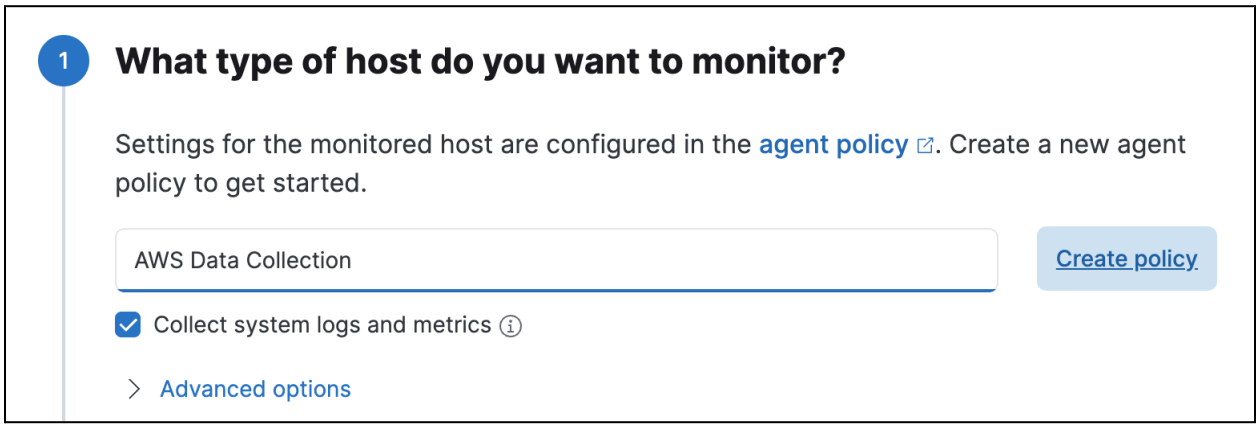
FYI: Fleet Management in Elastic Observability acts as a central hub for deploying and managing Elastic Agents. It allows you to monitor the health and status of enrolled agents and manage applied policies, eliminating the need for manual SSH access to individual servers for configuration changes.



2. **Add an Elastic Agent:**
    a. Click "**Add Agent**" to configure the agent.
3. Click "**Create new agent policy**", this shall bring up the Create Policy wizard



4. **What type of host do you want to monitor:**
    a. Enter a **policy name** → ensure "**Collect system logs and metrics**" is checked → select "**Create policy**"
        - **FYI**: In this step, you create a new or select an existing agent policy. This policy acts as a configuration template, telling the agent exactly which data to collect—such as system logs and metrics—without needing to configure individual files on the host.

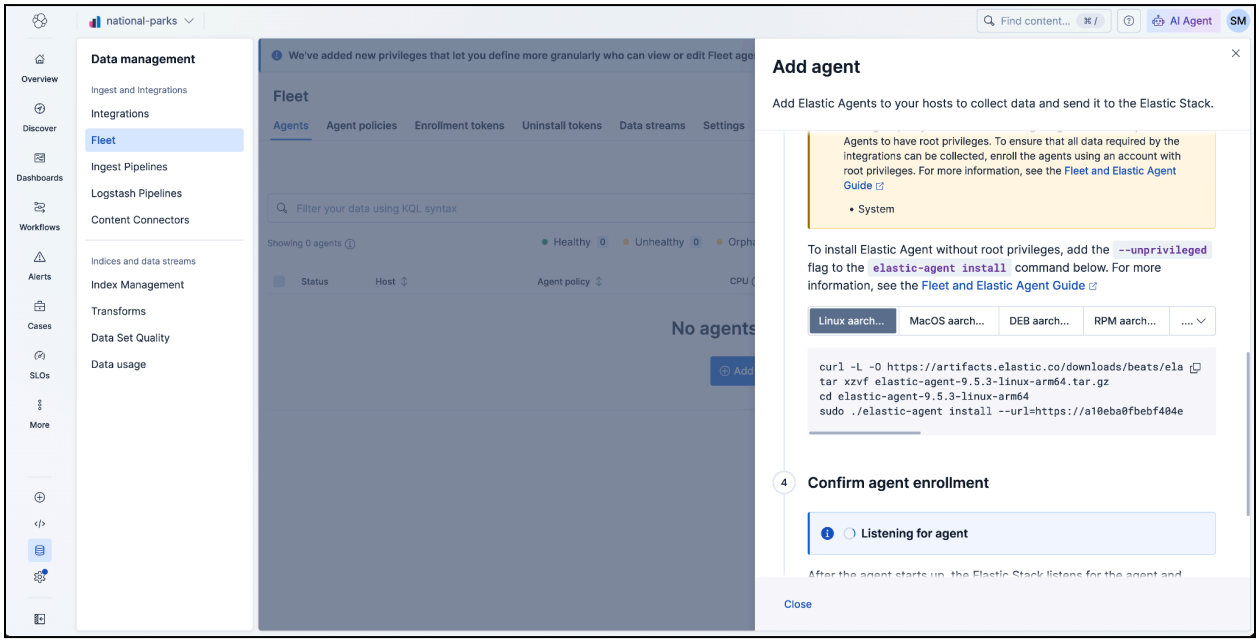


5. **Enroll into Fleet:**
    a. Ensure "**Enroll in Fleet (recommended)"** is selected
        - **FYI:** This allows the agent to be centrally managed via Kibana Fleet Management interface. Alternatively, you can run the agent in Standalone mode to be individually managed.
6. **Install the Elastic Agent on to your host:**
    a. Select the OS that the Elastic Agent will be hosted, to copy the command lines

to install the agent onto the target host.

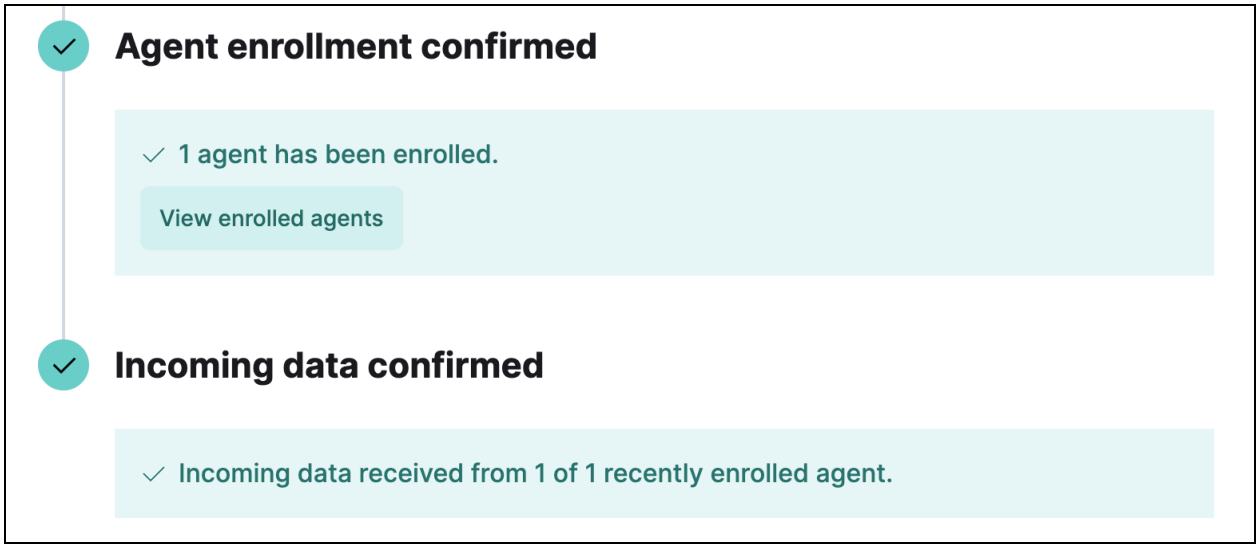
- **Note:** If you need to run the Agent without root privileges, add the **--unprivileged** flag to the **elastic-agent install** command

b. Run the commands on the desired EC2 instance to install the Agent and enroll this in fleet



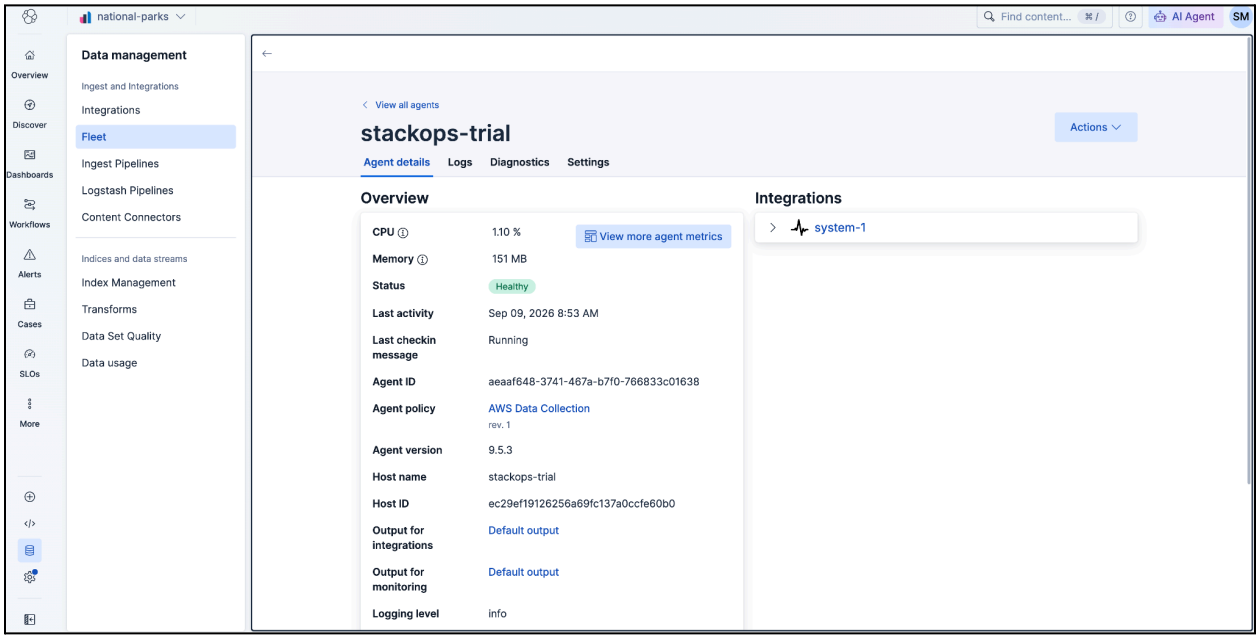
7. Confirm agent enrolment:

a. Once the commands are executed on the host and the Elastic Agent is correctly installed, you will get the following confirmation.



8. **View information on the installed elastic agent:**
    a. Click “**View enrolled agents”,** this will show the host in the Fleet Management dashboard.
    b. Select the new **Agent** in the Fleet Management dashboard
        - **FYI:** The Fleet management provides the following information and capabilities on the selected Elastic Agent:
            - **Real-Time Resource Monitoring:** The Overview section provides an immediate snapshot of the agent's footprint on the host.
            - **Operational Status:** The agent is marked as "Healthy," with a "Last activity" timestamp that confirms it is actively communicating with the Fleet server and successfully shipping data.
            - **System-Level Integrations:** The right-hand panel lists the active system integrations configured for this host, which are responsible for capturing standard OS metrics and logs to provide baseline observability.
            - **Version Control & Maintenance:** The interface identifies the current Agent version and includes an "Upgrade available" notification (if one is available), demonstrating how Fleet simplifies the lifecycle management of agents across an infrastructure.
            - **Troubleshooting Tools:** Dedicated tabs for Logs and Diagnostics are

available at the top, allowing administrators to view internal agent events or generate diagnostic bundles without needing to SSH directly into the host.



Dopamine Moment:

c. You have successfully created your first Elastic Agent policy, Installed your first Elastic Agent, and assigned this policy to the Elastic Agent.

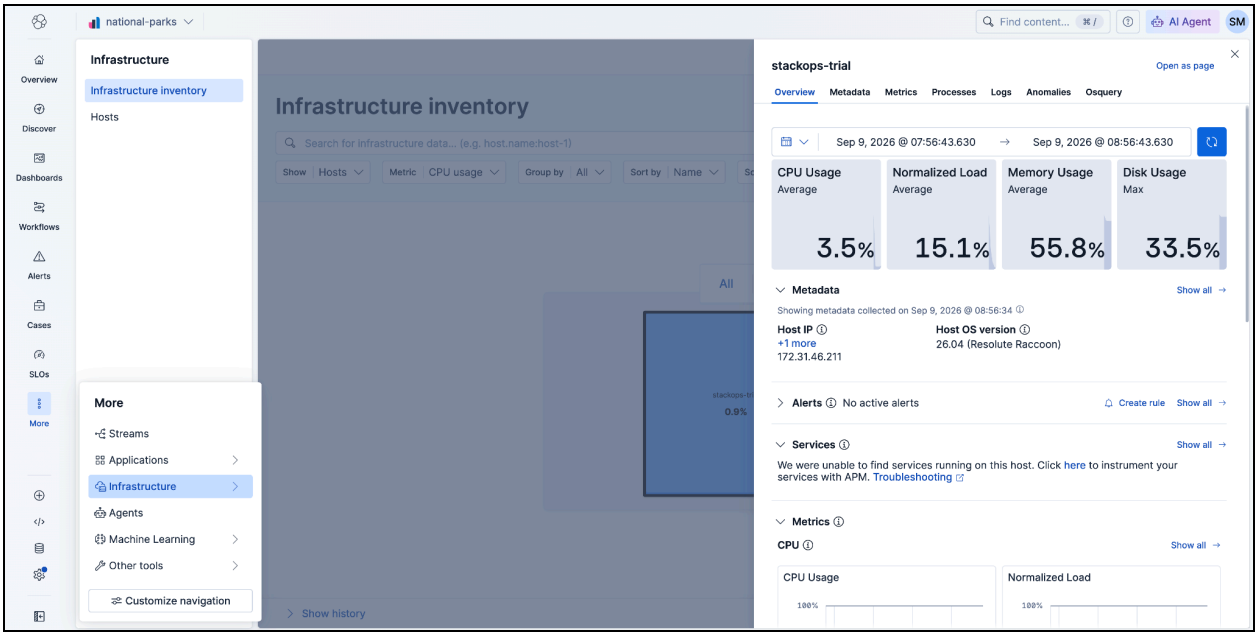
d. With this configured, the agents allow you to bridge the gap between raw data and the Observability features.

e. By default, using the system integration, the System metrics will be pulled from the OS automatically.

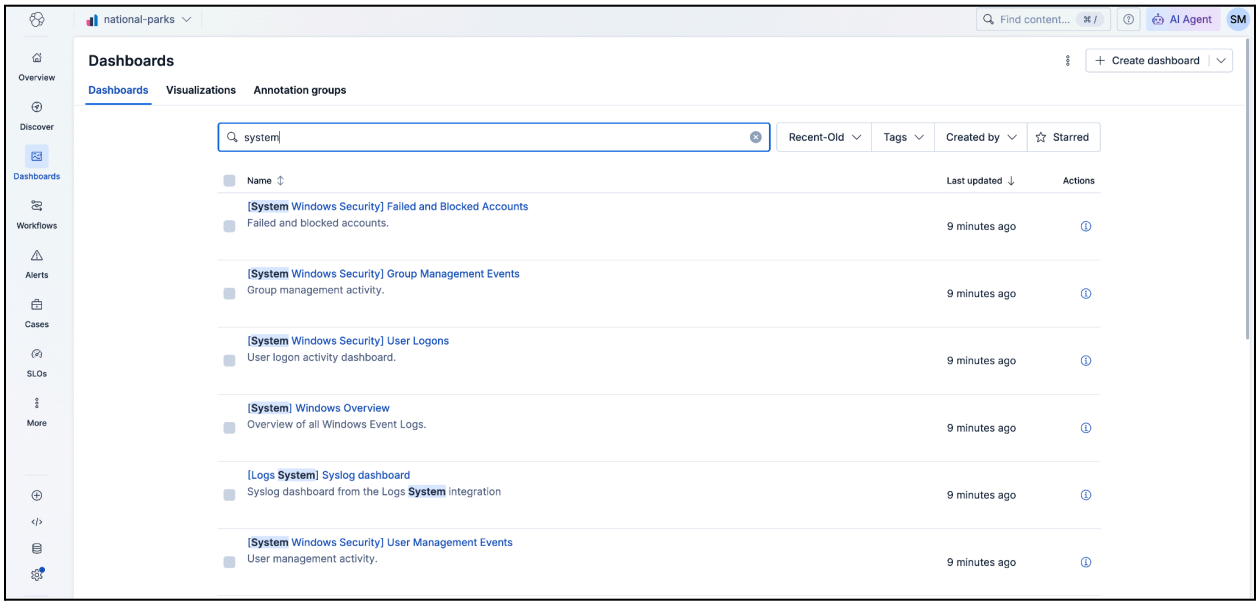
- This was added to the policy when we selected **Collect system logs and metrics** in step 4

f. Here are some suggestions for you to explore the out-of-the-box capabilities from Elastic:

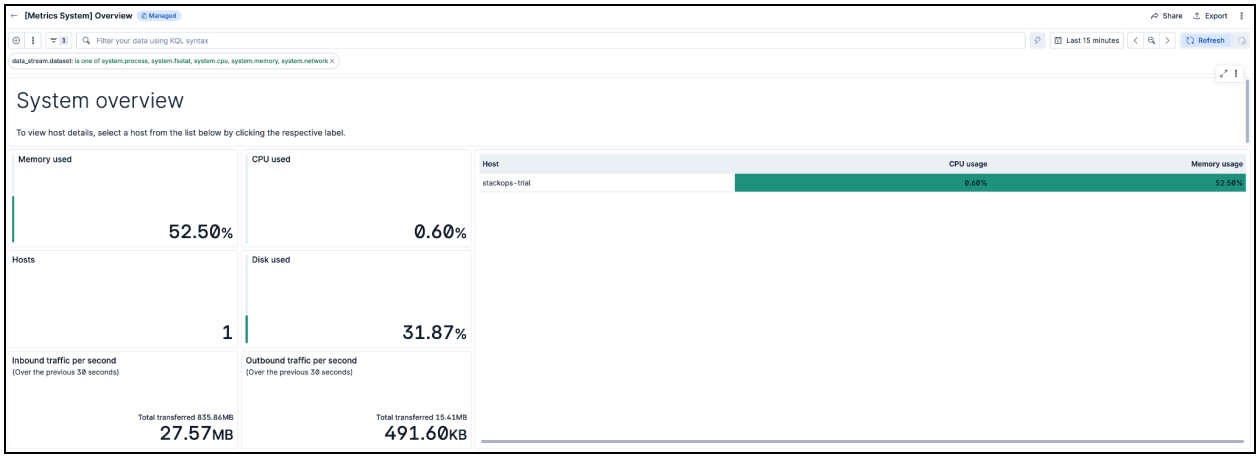
- Navigate to the **Infrastructure Inventory**, this EC2 instance shall show up here
  - Click on the EC2 host under **Hosts** to learn more on the status of the EC2



- Navigate to Dashboards, in here you will see out-of-the-box dashboards for the system metrics and logs, simply filter by “**system**”

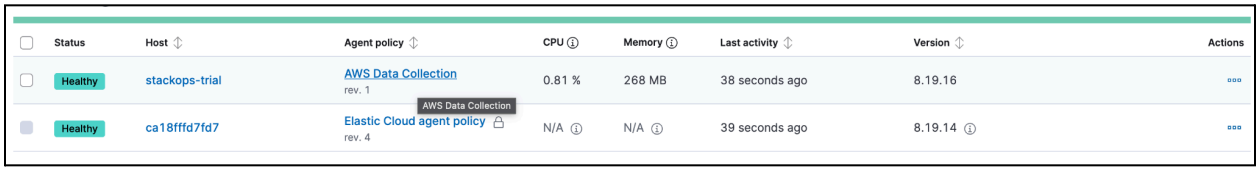


- For example: **[Metrics System] Overview**
    - This shall show the EC2 Instance where the Elastic Agent is installed

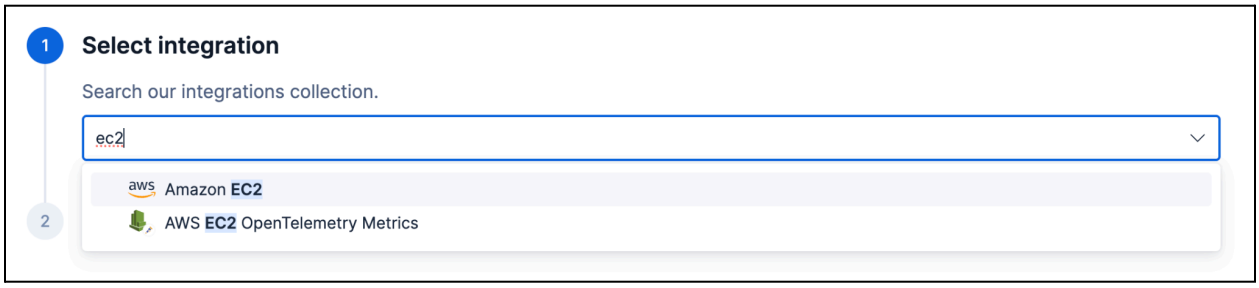


Collect Amazon EC2 Metrics

1. Ensure the **EC2 Instance** where the **Elastic Agent** resides has an an IAM role attached to with **CloudWatchReadOnlyAccess**
    a. **FYI:** This should have already been applied as per the pre-requisites
2. **Go to the Centralised Agent Management:**
    a. In Kibana, go to **Agents** in **Data Management → Fleet**
3. Select the **Agent Policy** you created in the previous stage (**AWS Data Collection** in the example below)



4. Select Add Integration
5. Search for “**ec2**” in the list of Integrations, this shall show an Integration for Amazon EC2



6. Click on the **Amazon EC2** Integration → **Add Amazon EC2**

7. Configure the Integration with the following settings
    a. **Configure Integration**
        - **Integration Settings**
            1. Give the Integration a name, such as "**aws-ec2**"
            2. You **do not** need to use any access keys, as the Elastic Agent will use the Instance Role permissions assigned to the EC2 Instance by default to access the AWS Resources
        - **Collect EC2 logs from S3**
            1. Ensure this is **disabled**
        - **Collect EC2 logs from CloudWatch**
            1. Ensure this is **disabled**
                a. If you want to collect the logs, just simple enable this and provide the Log Group ARN for the EC2 logs in CloudWatch
        - **Collect EC2 metrics**
            1. Ensure this is **enabled**
    b. Select **Add integration**
8. **Dopamine Moment:**
    a. That's it! Wait a few minutes, the Elastic Agent will now start collecting Amazon EC2 Instance Metrics
        - **Want to add additional data sources?** Sure, just follow the same process
            1. For AWS Integrations, they follow a similar pattern:
                a. Collect the Metrics from CloudWatch - just simply enable them
                b. Collect the logs from CloudWatch or S3
                    i. Ensure you have the necessary permissions in the Instance Role assigned to the EC2 instance to access the underlying Logs in S3 or CloudWatch

b. Now you have the EC2 Metrics being collected, check out the out-of-the-box dashboard called **[Metrics AWS] EC2 Overview** for some quick visualizations

Congratulations! You have now successfully created an Elastic Agent policy, installed an Elastic Agent, and started collecting the local System Metrics and logs, along with the Amazon EC2 metrics via the pre-built Elastic Integrations.

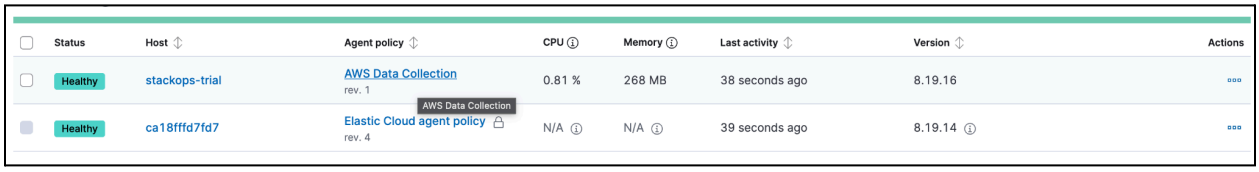
Added Bonus! Collect Amazon RDS Metrics (Optional)

Objective:

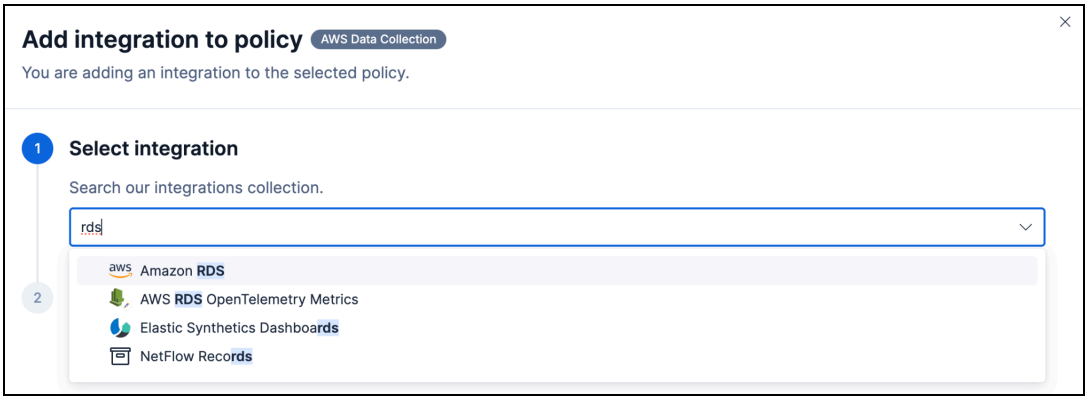
- If you're using Amazon RDS for any of your workloads, let's quickly add those in.

Steps:

1. Ensure the **EC2 Instance** where the **Elastic Agent** resides has an an IAM role attached to with **CloudWatchReadOnlyAccess**
    a. **FYI:** This should have already been applied as per the pre-requisites
2. **Go to the Centralised Agent Management:**
    a. In Kibana, go to Agent Policy in **Data Management → Fleet**
3. Select the **Agent Policy** you created in the previous stage (**AWS Data Collection** in the example below)



4. Select Add Integration
5. Search for "**rds**" in the list of Integrations, this shall show an Integration for Amazon RDS



6. Click on the **Amazon RDS** Integration ⟶ **Add Amazon RDS**
7. Configure the Integration with the following settings
    a. **Configure Integration**
        i. **Integration Settings**
            1. Give the Integration a name, such as “**aws-rds**”
            2. You **do not** need to use any access keys, as the Elastic Agent will use the Instance Role permissions assigned to the EC2 Instance by default to access the AWS Resources
        ii. **Collect RDS metrics**
            1. Ensure this is **enabled**
    b. Select **Add integration**

Dopamine Moment:

1. After a few moments, the Metrics shall start collecting
2. And you guessed it! There are **out-of-the-box** dashboards for Amazon RDS related data, go check them out!

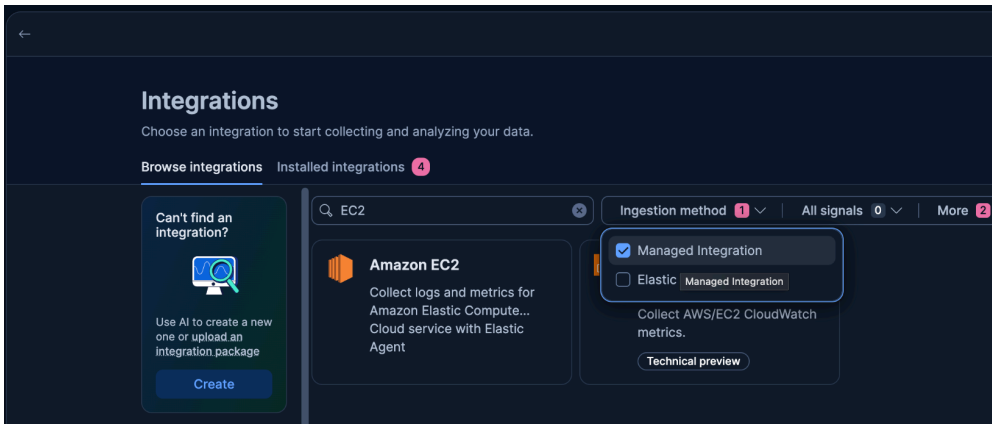


Want to know more?

- For a detailed step-by-step guide on configuring monitoring other AWS components (like Lambda, S3, etc), you may refer to the online documentation here or the video guide here.

EC2 Monitoring (Elastic Managed Integrations - Agentless)

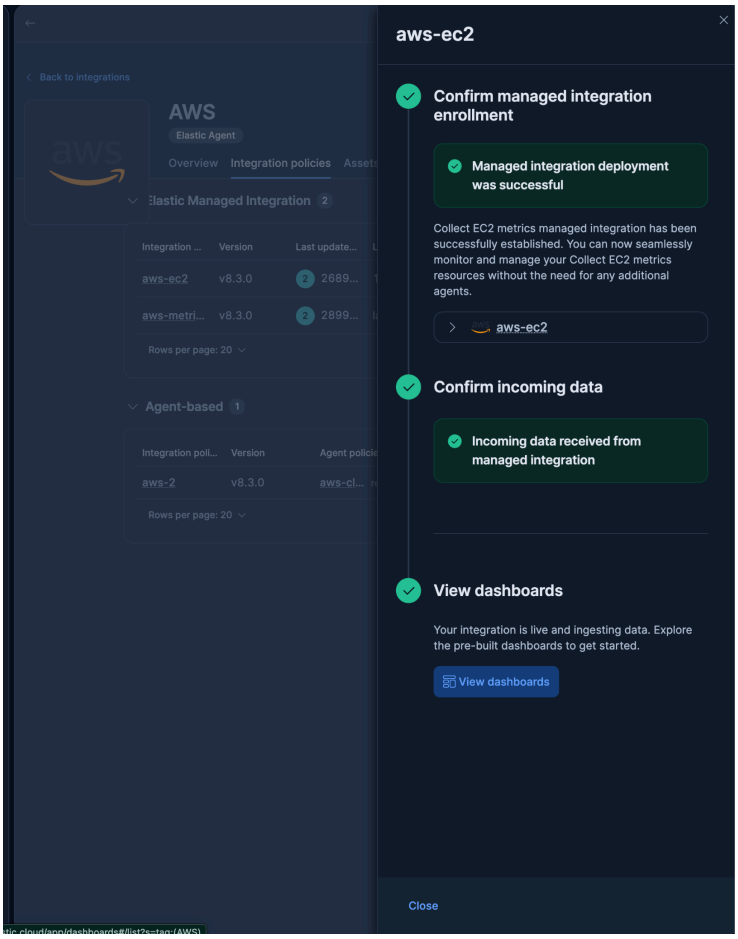
1. **Go to the Centralised Agent Management:**
    a. In Kibana, go to Agents in **Data Management → Integrations**
    b. Or, **the quick route** - use the Find content search bar in the top right of the screen and search for **Integrations**
2. **Add EC2 Metrics**
    a. Filter the Integrations for **Amazon EC2** where the **Ingestion method** is **Managed Integration**



    b. Select the **Amazon EC2** integration **->** select **Add Amazon EC2** in the top right
    c. Complete the configuration steps:
        - Configure Integration:
            - **Integration Name:** aws-ec2
            - **Advanced Options:**
                - Set the **Default AWS Region** to ap-southeast-1
            - **Deployment Options:** Elastic Managed Integration
            - **Setup Access** (Identity Federation is recommended):
                - **Identity Federation** - Follow the instructions to launch the CloudFormation templates
                - **Direct Access Keys**

- Collect EC2 logs from CloudWatch (**Optional**)
    - Specify the **Log Group ARN**
        - **Note:** This requires the EC2 logs to be output to CloudWatch via the CloudWatch agent
- Collect EC2 metrics
    - Ensure this is **enabled**
- Save and Continue

d. You should see the EC2 metrics now being collected

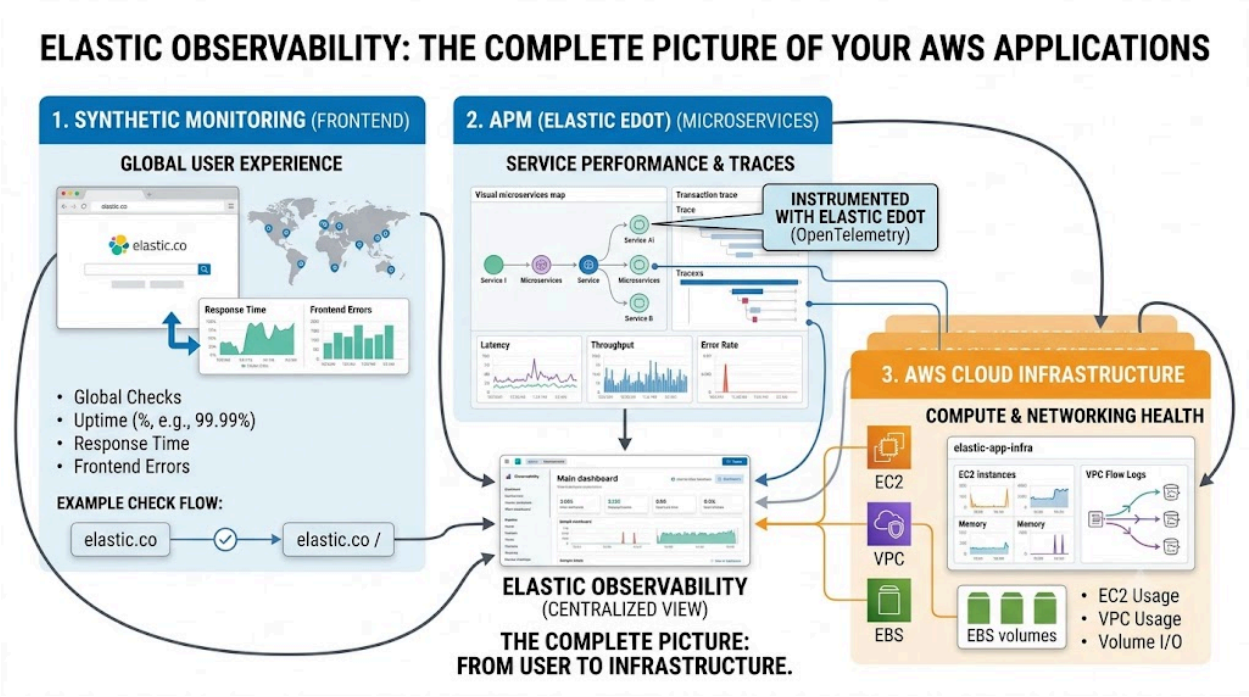
- 

e. Select **View Dashboards** to view the **[Metrics AWS] EC2 Overview** dashboard

Stage 3: Application Monitoring

In the next part of the trial, you will be guided on how to monitor your applications using Synthetic Monitoring and Application Performance Monitoring.

Combining Application Monitoring with Infrastructure Monitoring creates end-to-end visibility into your application and environment.



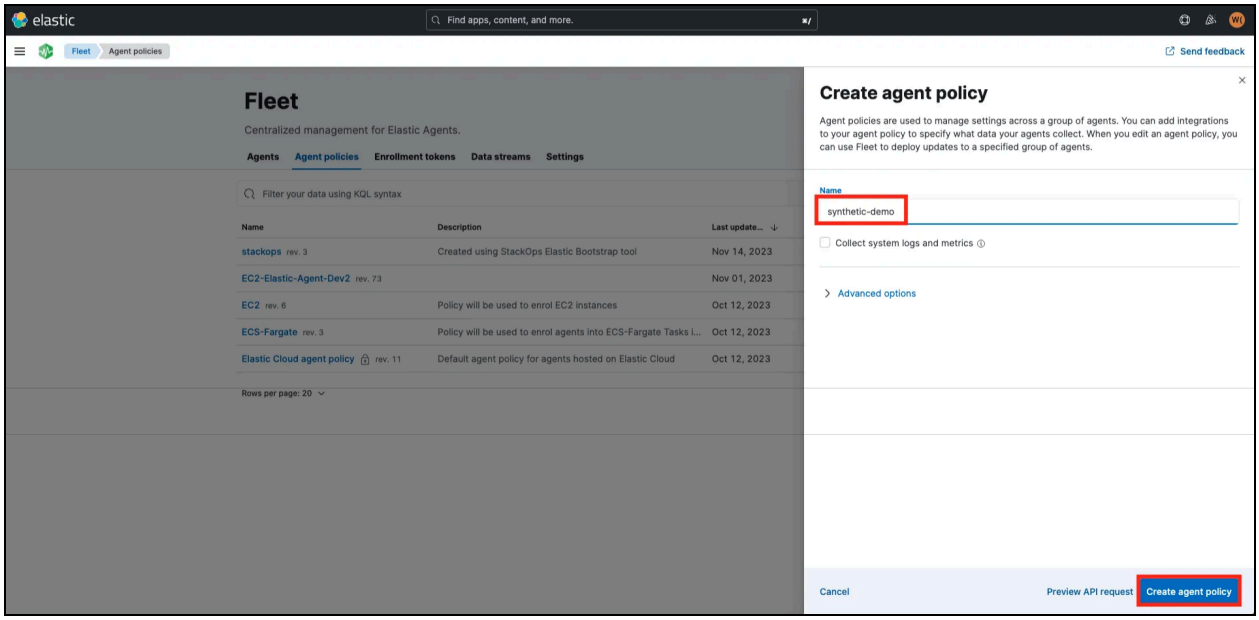
Synthetic Monitoring (Multistep Browser Health Check)

Synthetic monitoring allows you to establish an external baseline of service health quickly. This "outside-in" monitoring simulates citizen interactions, allowing teams to detect failures before they impact the public.

- **Note:** StackOps recommends the use of a Private Location to run the synthetic monitor from as it is more economical and secured when compared to public endpoints.

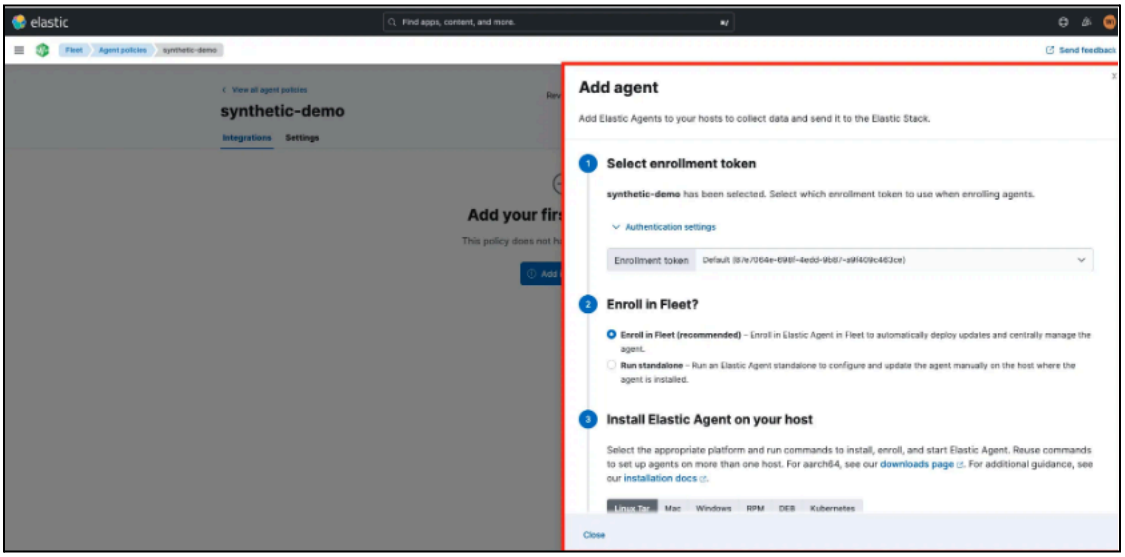
To implement the Multistep browser health check using a private location within your network:

1. Add a new agency policy in the Fleet Management Dashboard
    - In Kibana, go to **Data Management → Fleet → Agent Policies → Create agent policy**
        1. You can name this policy as "**synthetic-demo**"
    - Select "**Create agent policy**"



2. **Add an agent:**

    - Click on the Actions for this newly created Agent Policy → **Add agent**
    - This shall give you the relevant details you need to enroll the agent in Fleet with your newly created "**synthetic-demo**" policy



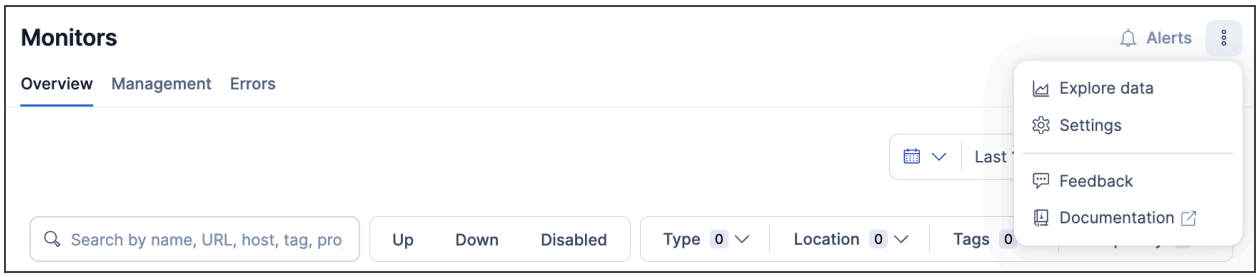
3. Download the Docker Image as per the document outlined in the document here.

    - The **ENROLMENT_TOKEN** and **FLEET_URL** will be obtained by the screen opened in Step 2

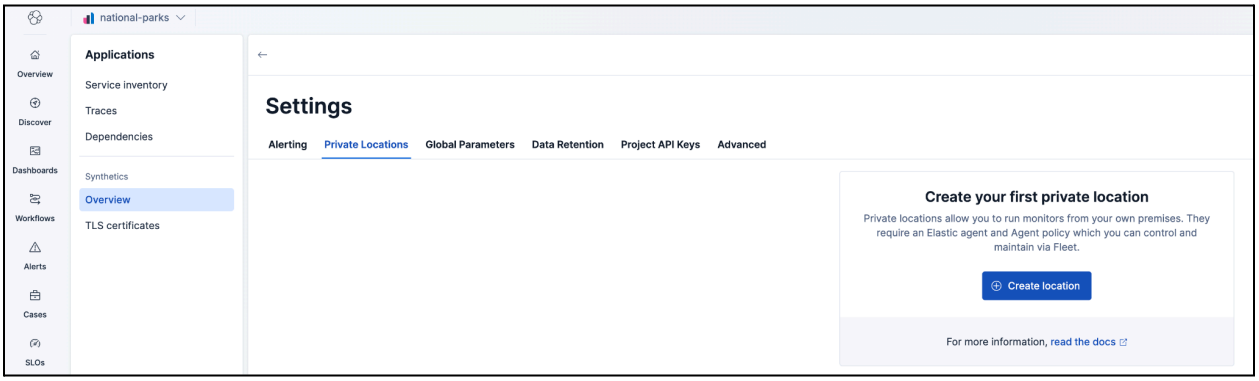
4. Once the Docker Container for the Elastic Agent has been enrolled, set up Synthetic

Monitoring in Private Location:

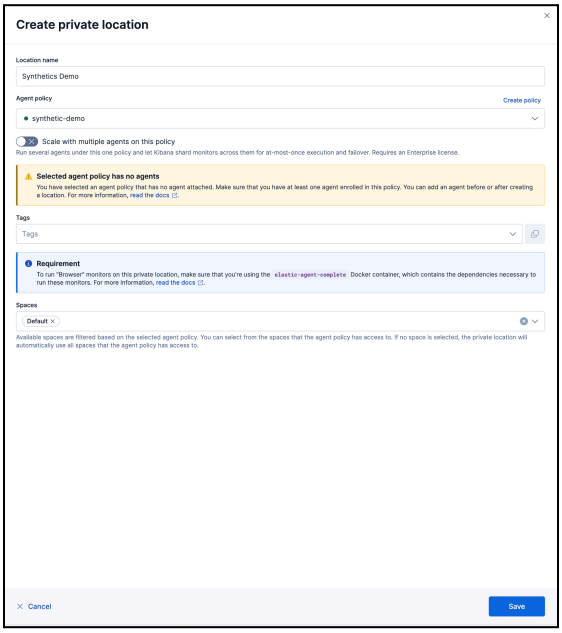
- In Kibana, go to **Applications → Overview** & click **Settings** from the triple dot icon on the top left corner.



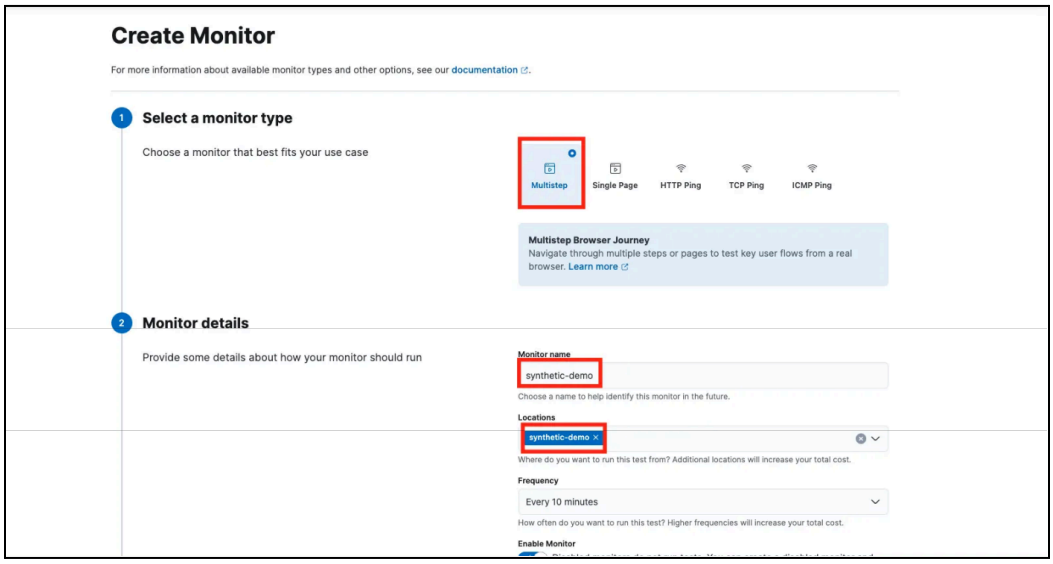
- Click **“Private Locations”** & Click **“Create location”**



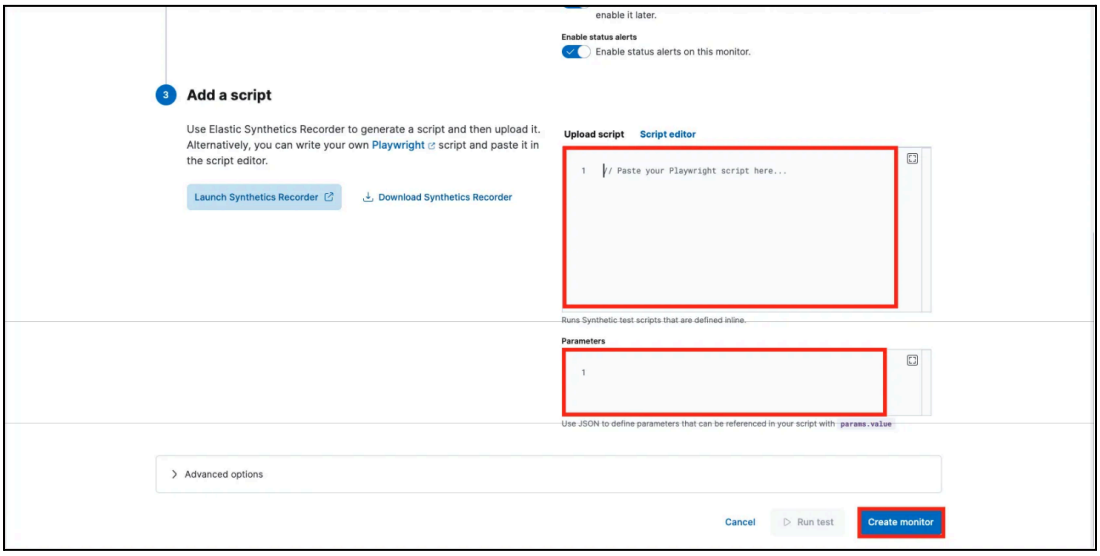
- Key in a **Location name**, and the **agent policy** that you have created in step 1 and click **“Save”.**



5. Create a Browser Synthetic Monitoring:
    - In Kibana, go to **Applications → Overview** > **Create Monitor**
    - Select **Multistep**, create a **Monitor Name** and specific a **Location** (use the location in the step above)



    - Download the Synthetic Recorder to your PC and click Launch Synthetic Recorder. Follow the video guide to use the Synthetic recorder to create the playwright script and upload the script.

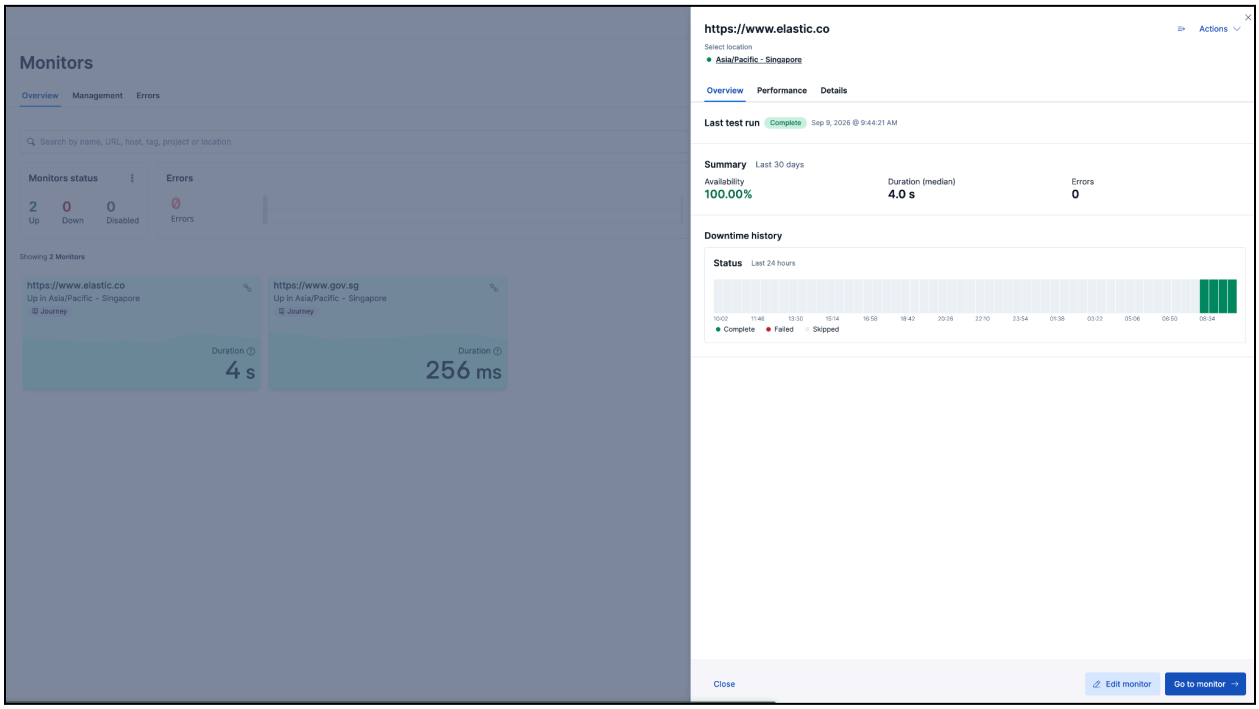


    - Click “Create Monitor” after you have uploaded the script.

6. **Dopamine Moment:** In Kibana, go **Applications** > **Synthetics** > **Overview** to view the

synthetic monitoring that you have created.

- Click the monitor and click “Go to monitor ” to learn more on what is monitored in the multistep browser health check.



For a detailed step-by-step guide, you may refer to the online documentation here or the video guide here.

Application Performance Monitoring (APM) with Elastic Observability and Opentelemetry

The most transformative aspect of observability is APM, which provides code-level visibility into how requests flow through an application. This documentation guides you on how to instrument the APM implementation on Java or .NET applications using Elastic's "**no-code**" **auto-instrumentation** capabilities.

In microservices architectures, a single user request may traverse multiple services. OpenTelemetry provides observability using distributed tracing, allowing teams to follow a request from the front-end Nginx proxy through multiple backend Java or.NET services down to the database.

The trial also highlights the Elastic Distributions of OpenTelemetry (EDOT), which provide a vendor-agnostic way to collect traces. OpenTelemetry allows agencies to own their data generation while leveraging Elastic's superior analysis and visualization UI.

Java Instrumentation Strategy

The Elastic OpenTelemetry (EDOT) Java Agent is a lightweight JAR file that instruments the JVM at runtime. To implement "no-code" auto-instrumentation APM for your app, key in the following command when starting up your application.

Steps:

To instrument your Java application, do the following:

- From the opentelemetry-java-instrumentation repository, download the **opentelemetry-javaagent.jar**, which contains the OTel agent and relevant instrumentation libraries. You can place the JAR file in your preferred directory.

- Obtain your OTEL_EXPORTER_OTLP_HEADERS and Elastic mOTLP Server URL from **Applications → Service Inventory → Add Data (Top Right) → OpenTelemetry (**Under **Connect directly to the endpoint)**
    - **Note:** The Elastic mOTLP Server URL should be your VPCE endpoint, e.g.
        i. **https://<example>.ingest.private.ap-southeast-1.aws.elastic.cloud**

- Open a Terminal window and run **either one** of the commands:
    - Include the OTel attributes directly in the **startup** command:

```
java -javaagent:/path/to/opentelemetry-javaagent.jar \
     -Dotel.service.name=my-java-app \
```

```
      -Dotel.traces.exporter=otlp \
      -Dotel.exporter.otlp.endpoint=<ELASTIC_mOTLP_URL> \
      -Dotel.exporter.otlp.headers="Authorization=Bearer
<your-secret-bearer-token>" \
      -jar my-java-app.jar
```

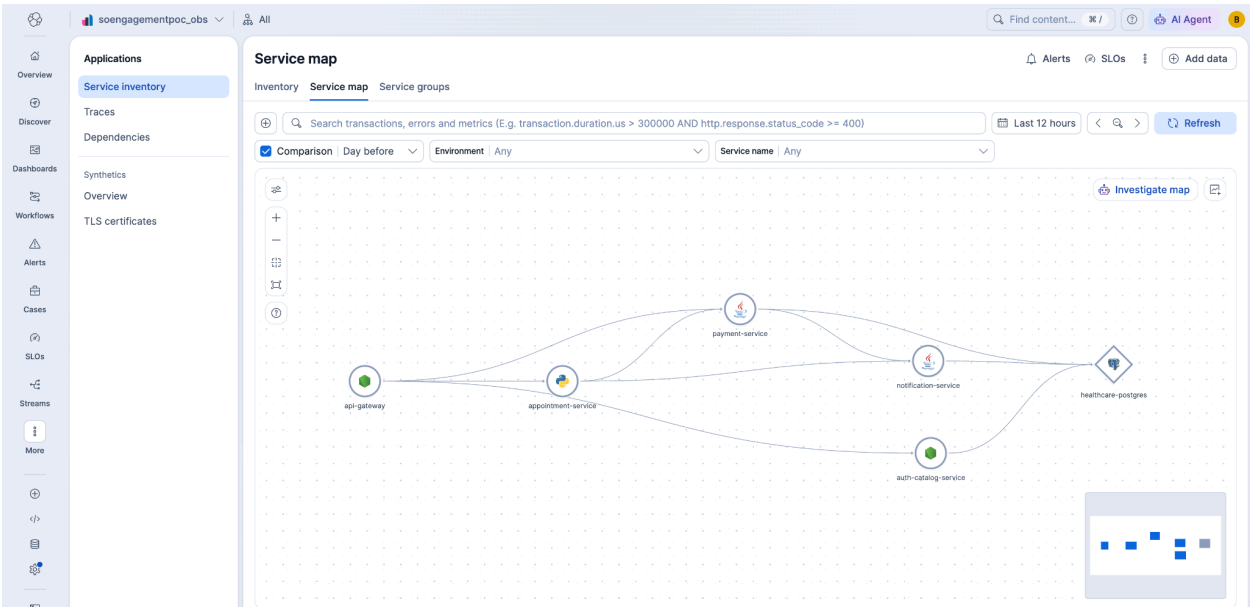
- Export OTel settings as environment variables

```
export OTEL_EXPORTER_OTLP_ENDPOINT=<ELASTIC_mOTLP_URL>
export OTEL_EXPORTER_OTLP_HEADERS=Authorization=Bearer
<your-secret-bearer-token>
export JAVA_TOOL_OPTIONS="-javaagent:path/to/opentelemetry-javaagent.jar"
export OTEL_SERVICE_NAME="<your-service-name>"

# Start up the Java service

java -jar myapp.jar
```

Dopamine Moment: Once the Java service has started up, you should now see it listed in the Service Inventory under **Applications → Service Inventory**. Once you instrument multiple services that talk together, you shall see how they communicate with each other in the Service Map.



.Net Instrumentation Strategy

The Elastic Distribution of OpenTelemetry (EDOT) .NET uses a CLR profiler-based approach to

deliver zero-code auto-instrumentation. Rather than modifying source code or adding NuGet packages, a lightweight installer script injects the profiler into the .NET runtime at startup — automatically capturing incoming request traces, outgoing HTTP calls, database queries, and more.

Note: EDOT .NET supports .NET 8, .NET 9, and .NET 10 on Linux and Windows, as well as .NET Framework 4.6.2+ on Windows. If you are unsure which runtime your app targets, open its .csproj file and check the <TargetFramework> element.

Objective:

- Automatically instrument a .Net application with OpenTelemetry

Steps:

- Obtain your OTEL_EXPORTER_OTLP_HEADERS and Elastic mOTLP Server URL from **Applications → Service Inventory → Add Data (Top Right) → OpenTelemetry** (Under **Connect directly to the endpoint)**
    - **Note:** The Elastic mOTLP Server URL should be your VPCE endpoint, e.g.
        i. **https://<example>.ingest.private.ap-southeast-1.aws.elastic.cloud**

- Install EDOT .NET and Configure OTel Settings
    - **Option A — Linux (bash):** Download and run the EDOT .NET installer, then set your OTel environment variables and start your application:

```
# Step 1: Download the EDOT .NET installer script
curl -sSfL
https://github.com/elastic/elastic-otel-dotnet/releases/latest/download/elastic-dotnet
-auto-install.sh -O
# Step 2: Install core files
sh ./elastic-dotnet-auto-install.sh
# Step 3: Make the instrumentation script executable and set up for the current shell
session
chmod +x $HOME/.otel-dotnet-auto/instrument.sh
. $HOME/.otel-dotnet-auto/instrument.sh
# Step 4: Set OTLP export settings pointing to Elastic Observability
export OTEL_SERVICE_NAME="my-dotnet-app"
export OTEL_TRACES_EXPORTER="otlp"
export OTEL_METRICS_EXPORTER="otlp"
export OTEL_LOGS_EXPORTER="otlp"
export OTEL_EXPORTER_OTLP_ENDPOINT="<ELASTIC_mOTLP_URL>"
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer <your-api-token>"
export OTEL_RESOURCE_ATTRIBUTES="deployment.environment=staging,service.version=1.0.0"
# Step 5: Start your .NET application (instrumentation is active)
./MyNetApp
```

- **Note:** The installer script downloads the EDOT .NET profiler binaries and places them under $HOME/.otel-dotnet-auto/. The instrument.sh script sets the required CLR profiler environment variables for the current shell session so any .NET process started in that session is automatically instrumented. The environment variables must be set before starting the application process.
    - **Option B — Windows (PowerShell, run as Administrator):**
        i. For .NET applications running on Windows (including IIS or Windows Services), use the PowerShell module. PowerShell 5.1 Desktop edition is required (run as Administrator):

```
# Download and import the PowerShell module
$module_url =
"https://github.com/open-telemetry/opentelemetry-dotnet-instrumentation/releases/latest/download/OpenTelemetry.DotNet.Auto.psm1"
$download_path = Join-Path $env:temp "OpenTelemetry.DotNet.Auto.psm1"
Invoke-WebRequest -Uri $module_url -OutFile $download_path -UseBasicParsing
Import-Module $download_path
Install-OpenTelemetryCore
# Set OTel environment variables for this PowerShell session
$env:OTEL_TRACES_EXPORTER="otlp"
$env:OTEL_METRICS_EXPORTER="otlp"
$env:OTEL_LOGS_EXPORTER="otlp"
$env:OTEL_EXPORTER_OTLP_ENDPOINT="<ELASTIC_mOTLP_URL>"
$env:OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer <your-api-token>"
# Register instrumentation for the current session and start your app
Register-OpenTelemetryForCurrentSession -OTelServiceName "<my-dotnet-app>"
dotnet MyApp.dll
./MyNetApp
```

- ***Note:*** *For IIS-hosted applications, use Register-OpenTelemetryForIIS instead (this performs an IIS restart). For Windows Services, use Register-OpenTelemetryForWindowsService -WindowsServiceName "YourServiceName" -OTelServiceName "my-dotnet-app". These environment variables only affect the current PowerShell session and have no impact on machine-wide settings.*

For further instructions on how to auto-instrument .Net applications, check out the Elastic EDOT documentation here and the OpenTelemetry documentation here.

Dopamine Moment: Once the .Net service has started up, you should now see it listed in the Service Inventory under **Applications > Service Inventory**. Once you instrument multiple services that talk together, you shall see how they communicate with each other in the Service Map

Dopamine Moment: In Kibana, go to **Applications**, to discover the following:

- **Service Inventory:** overview of all monitored services
- **Service Map:** upstream/downstream dependencies mapping
- **Traces:** Transaction traces
- **Dependencies:** to explore upstream dependencies of each monitored service

FYI: If you want to try this out with a sample application, you can leverage this GitHub repo which is used for some of the StackOps in-person workshops

Web Server Monitoring

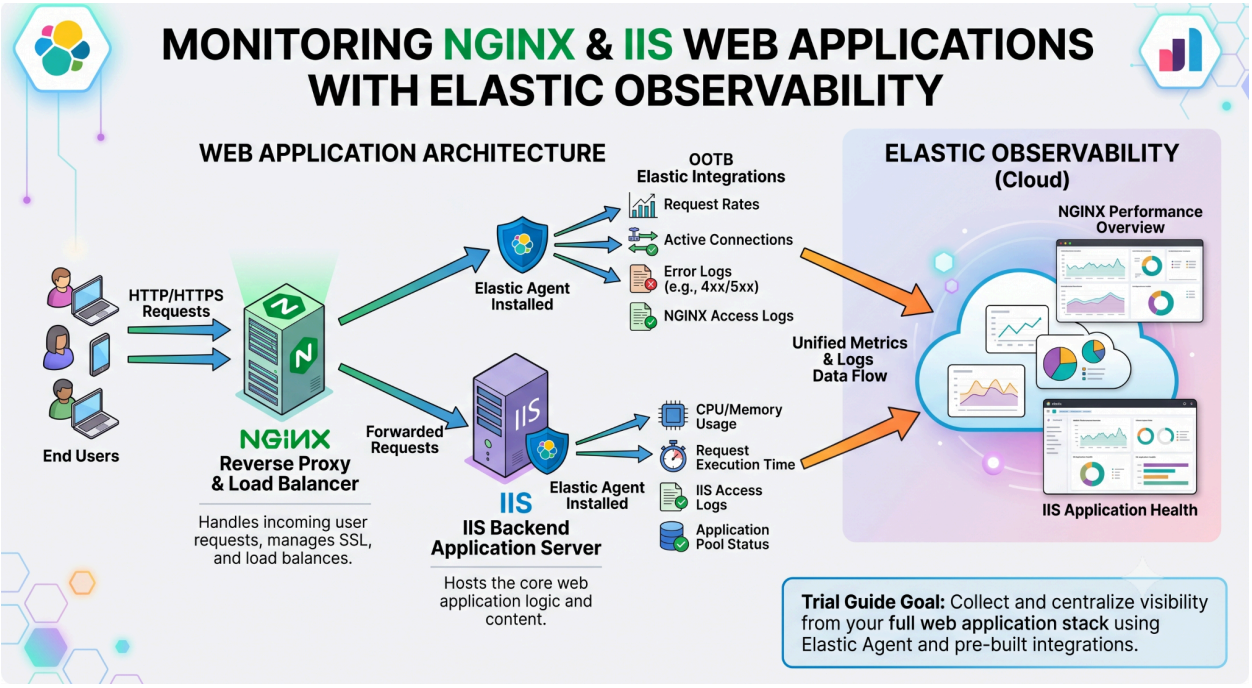
Let's add some more context!

With external visibility established, the trial shifts to "inside-out" monitoring using the Elastic Agent.

NGINX and IIS are commonplace in government web architectures, often serving as an entry point for traffic.

- If you have either one of these configured, then why not collect the Metrics and Logs from it?!

In order to collect Logs and/or Metrics from these, we shall utilize the Elastic Agent and the **out-of-the-box** integrations we saw in the EC2 Monitoring steps earlier.



Nginx Log Integration and Visualization

Following the steps we used for EC2 Monitoring, we shall deploy a new Elastic Agent on the Nginx server to collect the Logs and Metrics from the Nginx server itself:

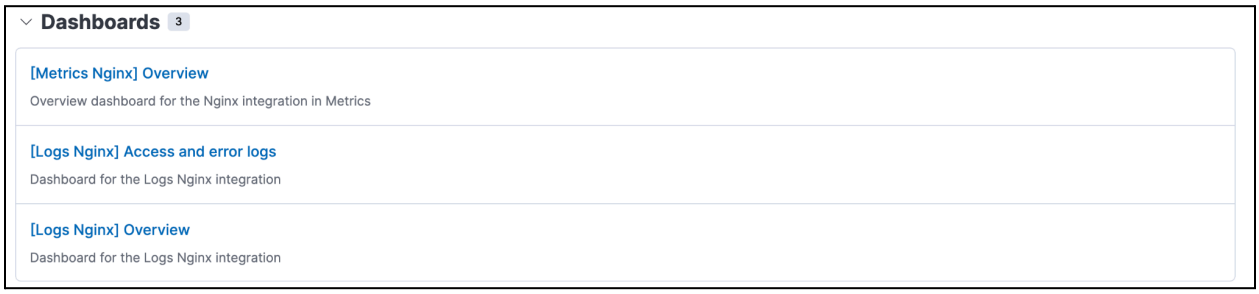
1. Create a new Agent Policy, this time it shall be for NGINX monitoring:
    - In Kibana, go to Agent Policy in **Data Management → Fleet**
    - Select **Add Agent → Create new policy** → Name the policy as **"NGINX Monitoring" → Create new policy**

- **Enroll** the Agent in Fleet and Install the Agent on the NGINX server using the commands provided

2. Go into the Agent Policy created in Step 1
    - Select Add Integration
    - Filter for the "**Nginx**" integration → **Add Nginx**
        i. Name the Integration
        ii. Ensure **Collect logs from Nginx Instances** is **selected**
            1. Ensure the log file paths are correct
        iii. Ensure **Collect metrics from Nginx Instances** is **selected**
            1. Ensure the listening port is correct
    - Add this Integration to an **Existing Host** via the policy created in Step 1 "**NGINX Monitoring**" → **Save and continue**

3. Dopamine Moment:

- After a few moments, the Logs and Metrics shall start collecting
- And you guessed it! There are **out-of-the-box** dashboards for NGINX related data, go check them out



- **FYI:** Have more than one Nginx server? Just simply go into the Agent Policy for **"NGINX Monitoring" → Actions → Add Agent**
    i. Then follow the instructions to install the Elastic Agent on this additional host(s) for Nginx data collection

IIS Monitoring Integration and Visualization

Following the steps we used for EC2 Monitoring, we shall deploy a new Elastic Agent on the IIS server to collect the Logs and Metrics from the Nginx server itself:

1. Create a new Agent Policy, this time it shall be for NGINX monitoring:
    - In Kibana, go to Agent Policy in **Data Management → Fleet**
    - Select **Add Agent → Create new policy** → Name the policy as **"IIS Monitoring" → Create new policy**
    - **Enroll** the Agent in Fleet and Install the Agent on the IIS server using the commands provided
2. Go into the Agent Policy created in Step 1
    - Select Add Integration
    - Filter for the "**IIS**" integration → **Add IIS**
        i. Name the Integration
        ii. Ensure **Collect IIS access and error logs (input: log)** is **selected**
            1. Ensure the log file paths are correct
        iii. Ensure **Collect IIS application_pool, webserver and website metrics** is **selected**
            1. Adjust any collection settings you deem necessary
    - Add this Integration to an **Existing Host** via the policy created in Step 1 "**IIS Monitoring**" → **Save and continue**

3. Dopamine Moment:

- After a few moments, the Logs and Metrics shall start collecting
- And you guessed it! There are **out-of-the-box** dashboards for IIS related data, go check them out



- **FYI:** Have more than one IIS server? Just simply go into the Agent Policy for **"IIS Monitoring"** → **Actions** → **Add Agent**
    i. Then follow the instructions to install the Elastic Agent on this additional host(s) for IIS data collection

Stage 4: Monitoring SaaS that provide a web page on its SaaS status

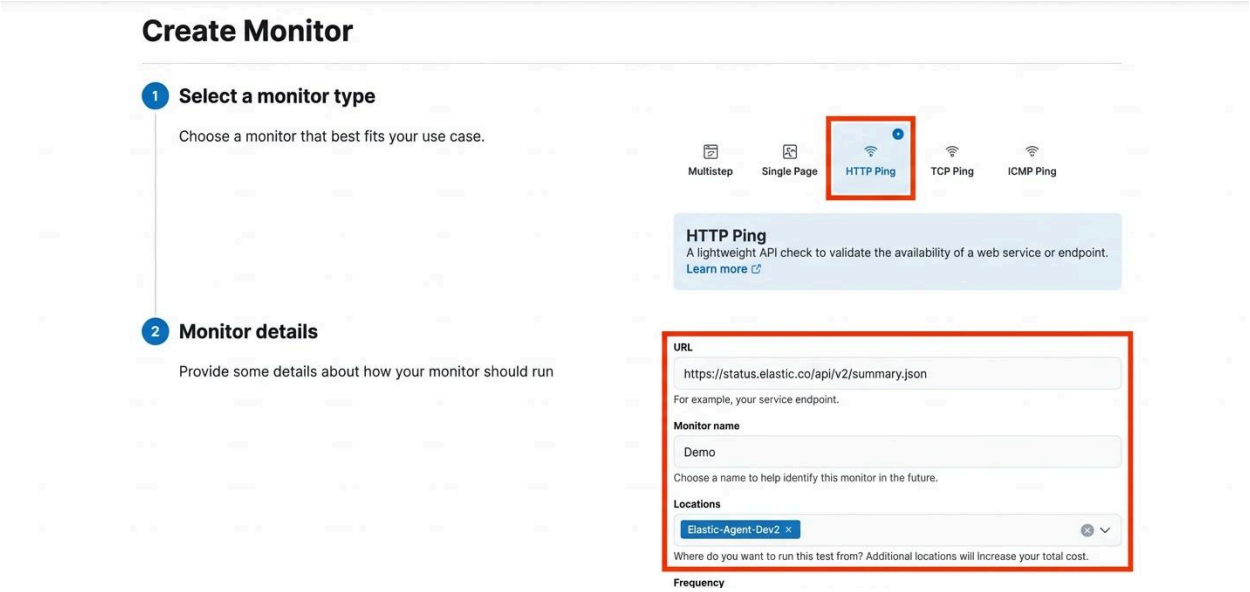
Want to monitor a SaaS provider's web page? This is how:

If SaaS provides a page to report on the SaaS health, you can use Synthetic Monitoring to ingest the data from the web page for monitoring within Elastic.

- In this example, we will use Elastic's health page (https://status.elastic.co/api/v2/summary.json) to guide you how to ingest & monitor Elastic service health via JSON APIs.
- In this context, we are using keywords "**Elasticsearch connectivity: AWS ap-southeast-1**" and "**status":"operational**" from the JSON data to monitor the AWS Singapore status.

To do this, you are setting up an HTTP Ping monitor that performs a "**Response body contains**" check for specific strings.

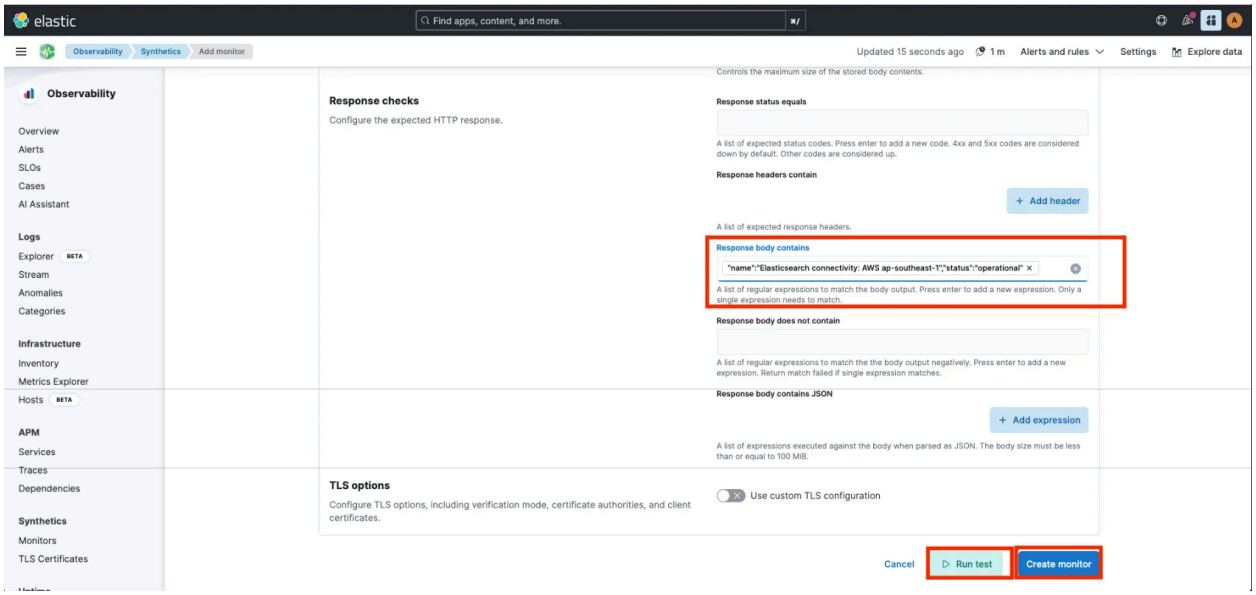
1. Create a HTTP Ping monitor:
    - Go to **Applications → Synthetics → Overview**
    - Select HTTP Ping and fill in the **URL**, **Monitor Name** and **Locations**



2. Input the following under the **Advanced Option** of the HTTP Ping monitor

   - In the "**Response body contains"** field and enter the following:

     "name":"Elasticsearch connectivity: AWS ap-southeast-1","status":"operational"



   - Click "Create Monitor"

3. **Dopamine Moment:** In Kibana, go to **Applications** > **Synthetics** > **Overview** to view the synthetic monitoring that you have created.

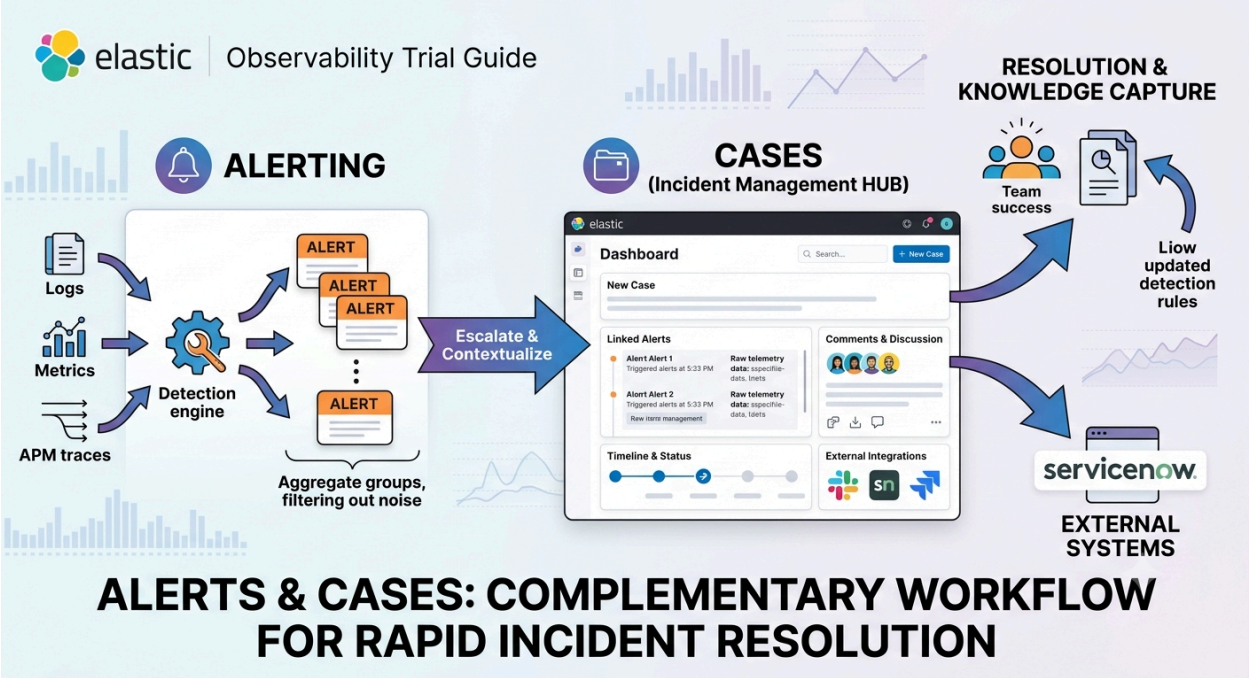
FYI: For a detailed step-by-step guide, you may refer to the online documentation here.

Day 2 Operations - Observing your environment

Stage 1: Setting up Alerts and Cases

In Elastic Observability (StackOps), **Alerting** and **Cases** are not isolated features; they are two sides of the same coin designed to optimize your incident management workflow. Setting them up together transforms raw system telemetry into organized, collaborative, and actionable resolutions.

Here is a visual overview of how these two components link together to create a closed-loop incident management system:



- **Alerts are the "Smoke Detectors":** Elastic Alerting continuously evaluates your logs, metrics, traces, and SLOs (Service-Level Objectives). When a threshold is breached or an anomaly is detected, an alert fires instantly. This ensures you know *what* went wrong and *when*, completely automatically.
- **Cases are the "Mission Control Room":** A Case acts as a central repository for an investigation. Instead of chasing loose threads across separate communication channels, teams can open a case inside Elastic Observability to track key details, assign owners, set severities, and record notes.
- **The Interconnection:** You can directly attach one or multiple related alerts to a single Case. By doing this, you instantly anchor the real-time telemetry (the exact log errors, metric spikes, or trace details that triggered the issue) directly into your team's workspace.

Phase 1: Setting up Alerts

Following on from our EC2 Metrics story, lets create an Observability Rule to help us detect if service CPU exceeds a specific threshold.

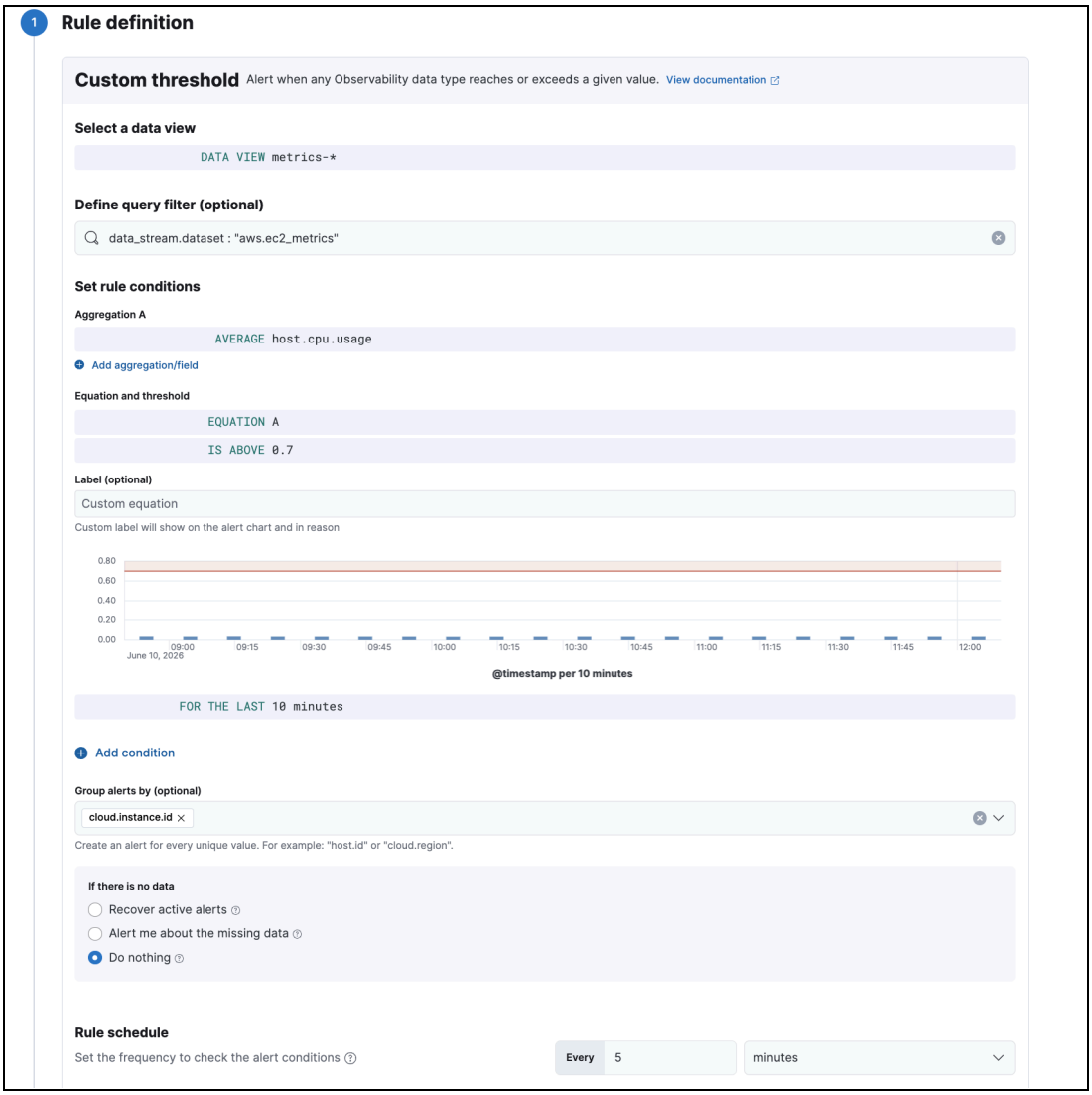
Elastic Observability and Security come with multiple capabilities for creating Alert Rules which are available out-of-the-box with StackOps, however, for this section lets focus on the Observability Rules.

Objective:

- Create an Alert rule to trigger a Warning event if the CPU of an EC2 instance exceeds 70% on average

Steps:

1. In Kibana, navigate to **Alerts**
2. Select Manage **Rules → Create Rule**
3. For this use case, we shall select the **Custom Threshold** Observability Rule
    - This shall bring up a template
4. Fill out the template as follows:
    - Name the rule as “AWS EC2 - CPU Usage”
    - Data View: **metrics-***
    - Define query filter: **data_stream.dataset : "aws.ec2_metrics"**
    - Equation and threshold: **Equation A Is above 0.7**
    - For the last: **10 minutes**
        - **Note:** AWS CloudWatch can be delayed to populate metrics so you should account for the delay on AWS side
    - Group By: **cloud.instance.id**
    - Rule Schedule: **Every 5 minutes**
5. **Create rule**



Dopamine Moment:

- This rule shall now run every 5 minutes to see if any EC2 instances exceed the CPU Usage threshold
- You can view any active alerts within the **Alerts** UI
    - These alerts shall remain **active** until the threshold condition is no longer met (The CPU drops below the threshold specified), at which point they shall change to **resolved** status



Phase 2: Using Cases for case management

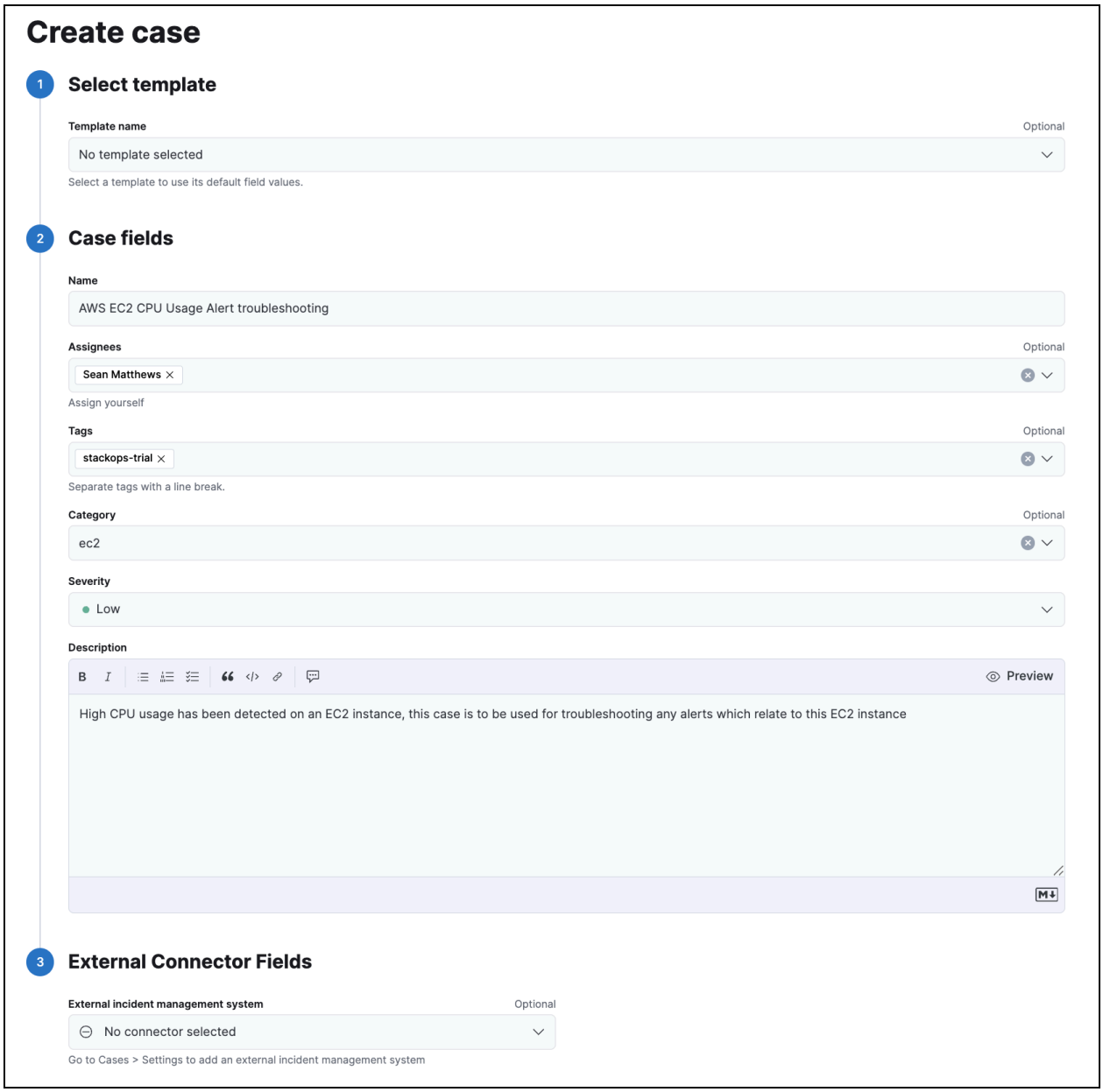
Leveraging the Alert rule created in the previous phase, we can leverage the Cases section of Elastic Observability to combine multiple Alerts together into a central repository to allow multiple analysts to work together and track all communication/troubleshooting as they work towards a root cause.

Objective:

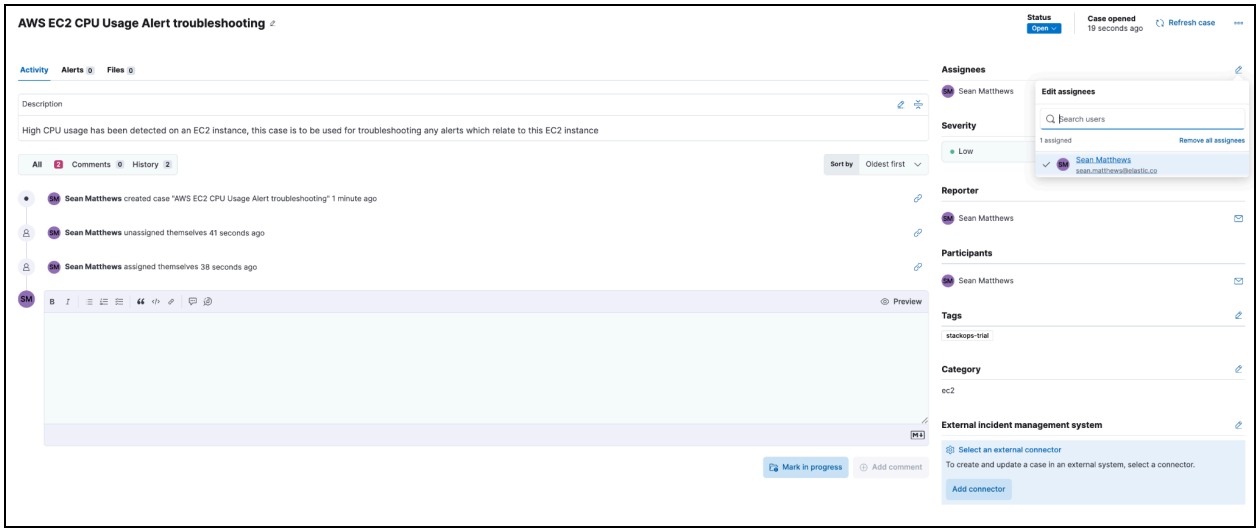
- Create a case within the Cases application
- **Optional**: Add an alert to the case

Steps:

1. In **Kibana** navigate to **Cases**
2. **Select**: Create case
3. Fill out the Case template details:
    - **FYI**: You can use the below as an example



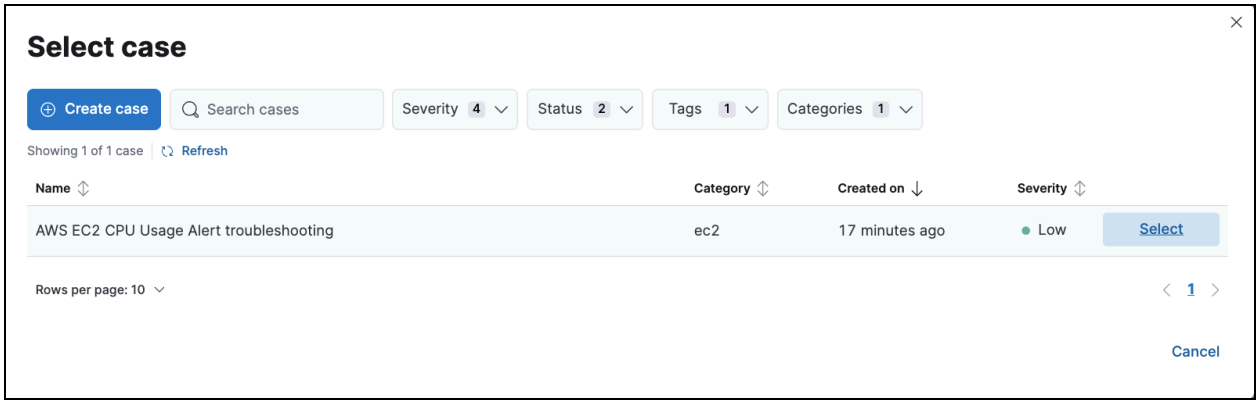
4. **Select:** Create case
    - This will create the case, take a look at some of the key capabilities:
        i. You can add additional Assignees to the case
        ii. You can add files to the Case, such as playbooks, evidence etc..
        iii. You can also attach Alerts to the case



Note: The following step 5 is optional if you have an active alert for the EC2 CPU usage based on the rule created in phase 1 (If you want to trigger an alert, you could simply reduce the threshold for the rule to some that may trigger, such as 0.1 (10%) - but be wary, this could trigger a lot of alerts so change it back afterwards).

5. **Optional - Add an Alert to the Case:**
    - In **Kibana** navigate to **Alerts**
    - If you have any active alerts in here, you can select them and add them to the **existing case** which was created in **Phase 1**





- Now navigate back to your case, you shall see the Alert listed under the Alerts tab

6. You now have the ability to cycle through the lifecycle of the case, working it from **Open → In Progress → Closed**

Dopamine Moment:

- You have now created an Alert Rule, a Case and gone through some of the Incident Management features which allow you to be able to quickly triage and work through incidents in a collaborative way
- **FYI**: If you happen to use an external ITSM tool such as Jira, you can connect to this tool and sync all cases with it

Stage 2: Interrogate your data with Discover

Discover is the primary tool for exploring your Elasticsearch data in Kibana. Search and filter documents, analyze field structures, visualize patterns, and save findings to reuse later or share with dashboards. Whether investigating issues, analyzing trends, or validating data quality, Discover offers a flexible interface for understanding your data.

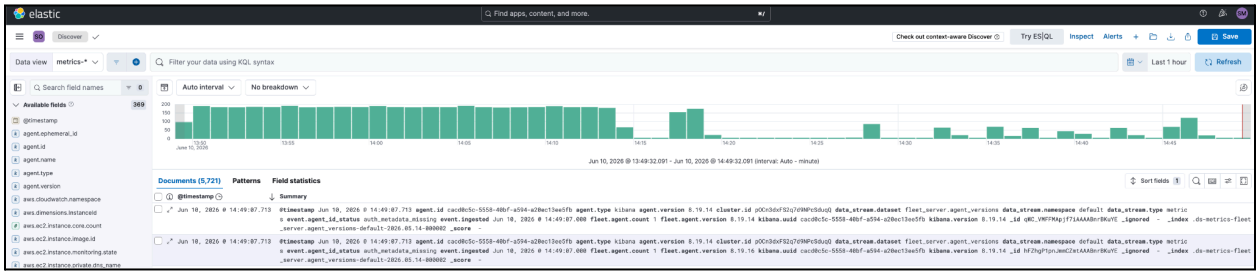
Objective:

- Investigate EC2 metrics within the Discover UI
- Apply quick filters to allow you to see key metrics quickly and easily
- Create a Data View to reduce the data showing and noise presented in the Discovery UI

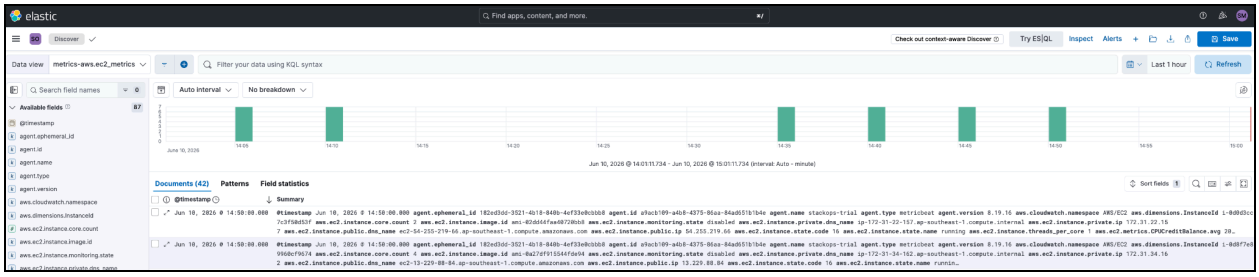
Steps:

1. In **Kibana** navigate to **Discover**
    - This will present you with the Discovery UI which allows you to interrogate your data

- Get comfortable with what data is showing in the UI
- **Note:** When we ingest data sources into StackOps, the data shall be stored in indices:
    i. **Metric** will go into **metrics-{data source name}**
    ii. **Logs** will go into **logs-{data source name}**
    iii. **Traces** will go into **traces-{data source name}**
- We can **pre-filter** the UI based on the **Data View** which can be seen in the top left
    i. Select the Data View drop down and flick between the **metrics-*** and **logs-*** Data Views
- In the **top right** you can specify the **time range** of the data being shown
- In the **top middle** you can specify a **query** to filter the data down

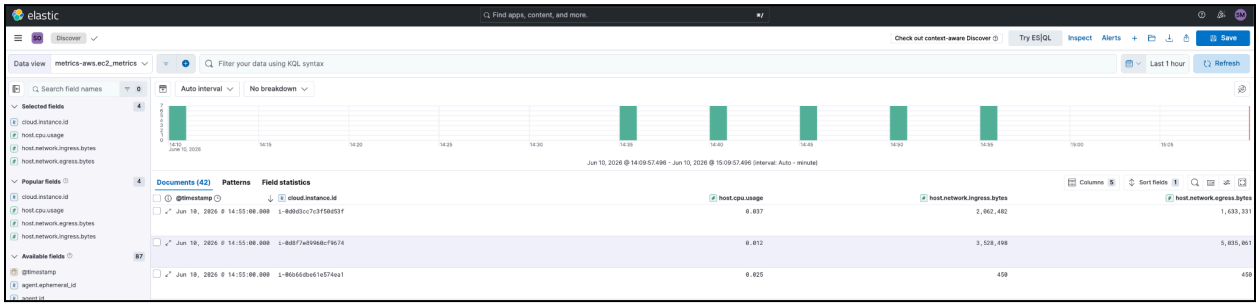


2. **Create a Data View to be able to filter the UI to our EC2 metrics**
    - In the Data View drop down, select Create a data view
    - **Enter the following:**
        i. **Name:** metrics-aws.ec2_metrics
        ii. **Index pattern:** metrics-aws.ec2_metrics-*
        iii. **Timestamp field:** @timestamp (Should already be this by default)
    - Select: Save data view to kibana
        i. This will now filter the Discover to your EC2 metrics (You can change the Data View at any time)



3. **Use some of the Available fields to provide clearer insights in the UI**

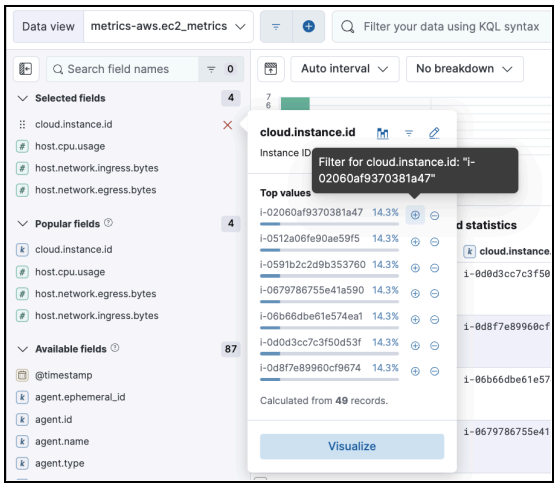
    - On the left hand side you shall see Available fields - these are the available fields within the Data view (EC2 Metrics in this case)
    - Find the following fields and select the **+** icon next to them:
        i. cloud.instance.id
        ii. host.cpu.usage
        iii. host.network.ingess.bytes
        iv. host.network.egress.bytes
    - You now have a simple table format showing these fields



4. **Add a filter to filter this discover to a specific InstanceId**

    - Often, you will want to easily filter the discover by a key field, such as the EC2 InstanceId (**cloud.instance.id**)
        i. To do this, you can either enter a KQL syntax, this will auto-complete as you start typing in
        ii. Alternatively, you can use a drag and drop approach to add a filter
    - Add a filter by clicking on the **cloud.instance.id** field in the left pane, this will give you a preview of the available InstanceId's show in your data
        i. You can then simply select the **+** next to the **InstanceId** to filter the UI to this

InstanceId



Dopamine Moment:

- This is a very quick and easy way to interrogate your data to understand what is happening within it
- This becomes extremely useful when you want to understand why system A could not connect to system B (Potentially being requested by a Security Group configuration?)
- However, typically you will also want to visualize this data on a dashboard, check out the Personalize your data section to understand how to create a personalized dashboard

Stage 3: Personalize your data

Unlock Clarity with Personalized Dashboards:

- Stop jumping between isolated tools. StackOps empowers you to unify disparate data streams into a single, intuitive dashboard tailored specifically to your operational needs. By correlating contextually relevant data sets, you can rapidly diagnose issues that would otherwise remain hidden.
- Consider these powerful use cases for your personalized dashboards:
    - **System Integrity & Access:** Overlay Windows EC2 resource usage with Event Logs to immediately correlate performance spikes with user logins or service restarts.
    - **Rapid Root-Cause Analysis:** Map Firewall events against Application error codes to instantly visualize how blocked ports directly impact application failure rates.
- By customizing your view to fit your unique architecture, you transform complex data into actionable insights, making your day-to-day operations smoother, faster, and more decisive.

Question: Now you have EC2 Metrics coming into your environment, wouldn't it be nice to be able to correlate your VPC FLow Logs (Accepts/Rejects) with your EC2 Instance Usage where your applications reside?

- Well, with StackOps, you can do this simply in a single dashboard by simply combining multiple data sources into one single place.

Expected end result:

- A single personalized dashboard combining EC2 Metrics with VPC Flow Logs



Value:

- The ability to filter the dashboard by **InstanceId**, allowing the user to be able to quickly see the Network Utilization for a given **InstanceId**

Phase 1: Onboard AWS VPC Flow Logs

Objective:

- Onboard AWS VPC Flow Logs to allow users to be able to quickly view and observe their VPCs within StackOps

Steps:

Note: For AWS VPC Flow Logs, you can output them with different field formats, as well as outputting them to S3 or CloudWatch. There is no one size fits all, however, you can adapt the below steps to fit your needs.

In this example we shall be outputting the logs to a CloudWatch log group with a specific format.

1. Within AWS, configure your VPC to generate VPC Flow Logs and output them into a

CloudWatch Log Group. Some suggested settings are:

a. **Filter:** All

b. **Maximum aggregation interval:** 10 minutes

c. **Destination:** Send to CloudWatch Logs

d. **Log record format:** Select the following fields under Custom Format:

```
${version} ${account-id} ${interface-id} ${srcaddr} ${dstaddr} ${srcport} ${dstport} ${protocol}
${packets} ${bytes} ${start} ${end} ${action} ${log-status} ${vpc-id} ${subnet-id} ${instance-id}
${tcp-flags} ${type} ${pkt-srcaddr} ${pkt-dstaddr} ${region} ${az-id} ${sublocation-type}
${sublocation-id} ${pkt-src-aws-service} ${pkt-dst-aws-service} ${flow-direction} ${traffic-path}
${ecs-cluster-arn} ${ecs-cluster-name} ${ecs-container-instance-arn} ${ecs-container-instance-id}
${ecs-container-id} ${ecs-second-container-id} ${ecs-service-name} ${ecs-task-definition-arn}
${ecs-task-arn} ${ecs-task-id} ${reject-reason}
```

FYI: the order of the fields is important.

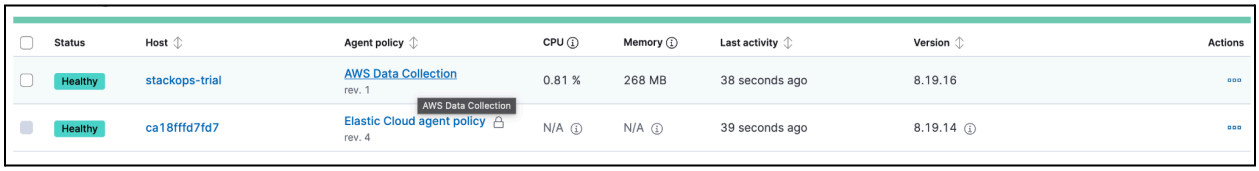
2. Note down the ARN for the Log Group where the VPC Flow Logs are being output to, as you shall require this in the upcoming steps

    a. **Note**: Also ensure the Instance Role for Elastic Agent has permissions to view this Log Group

3. **Go to the Centralised Agent Management:**

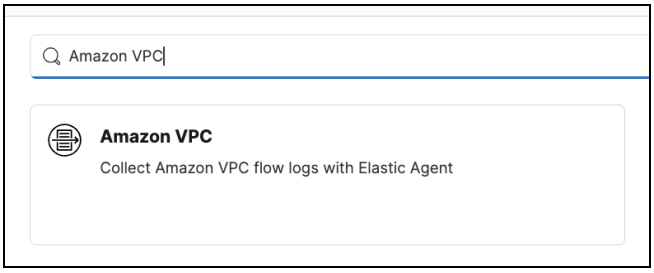
    a. In Kibana, go to Agent Policy in **Data management → Fleet**

4. Select the **Agent Policy** you created previously (**AWS Data Collection** in the example below)

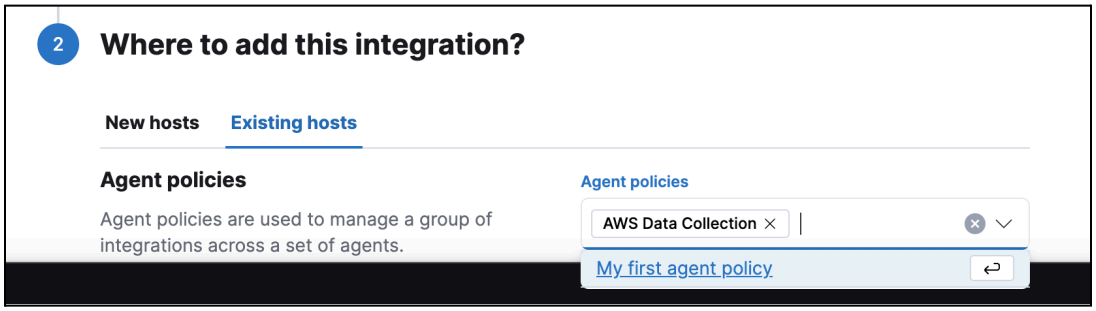


5. Select Add Integration

6. Search for "**Amazon VPC**" in the list of Integrations, this shall show an Integration for Amazon EC2

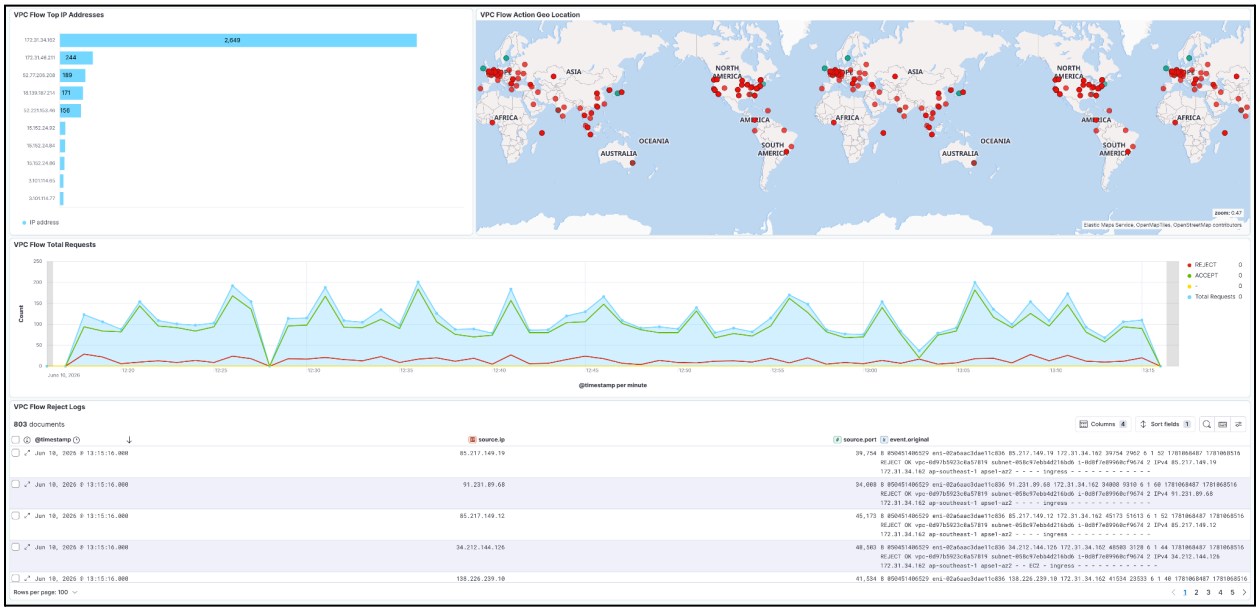


7. Click on the **Amazon VPC** Integration ⟶ **Add Amazon VPC**
8. Configure the Integration with the following settings
    a. **Configure Integration**
        i. **Integration Settings**
            1. Give the Integration a name, such as "**aws-vpc-flow-logs**"
            2. You **do not** need to use any access keys, as the Elastic Agent will use the Instance Role permissions assigned to the EC2 Instance by default to access the AWS Resources
        ii. **Collect VPC flow logs from S3**
            1. Ensure this is **disabled**
        iii. **Collect VPC flow logs from CloudWatch**
            1. Ensure this is **enabled**
            2. Enter the **ARN** obtained from **step 2**
    b. **Where to Add this Integration?**
        i. Select **Existing Hosts** and then select the **Agent Policy** created previously ⟶ **Save and continue**



Dopamine Moment:

- This shall now start collective VPC flow logs from your CloudWatch log group.
- **Note**: This could take 10-15 minutes to show meaningful data in StackOps post setting up the VPC flow logs in AWS due to the aggregation interval
- **Want an out-of-the-box dashboard?** We've got you covered!
    - Navigate to the dashboard named [Logs AWS] VPC Flow Log Overview]



Phase 2: Create a Personalized Dashboard

Objective:

- Create a personalized Dashboard to show EC2 Resource Utilization combined with AWS VPC FlowLogs, allowing the user to quickly correlate the resource and network utilization for a given InstanceId

Option 1 - The super easy way - get AI to do it!

- Provide the Elastic AI Agent with the following prompt, this should build you a dashboard based on AWS VPC Flow Logs, EC2 Instances metrics and EBS Volume metrics

None

```
Build a Kibana dashboard "AWS VPC Resource & Traffic Overview" using the dashboards and
    visualization-creation skills. All panels use ES|QL and follow the dashboard time
    range (default: last 24h). Check each field with a test query first; skip panels with
    no data and list what you skipped at the end.
```

```
DATA
- Flow logs: logs-aws.vpcflow-default — aws.vpcflow.vpc_id, aws.vpcflow.instance_id,
    source.ip, destination.ip, destination.port, network.transport, network.bytes,
    network.packets, aws.vpcflow.action (ACCEPT/REJECT), aws.vpcflow.pkt_src_service,
    aws.vpcflow.pkt_dst_service

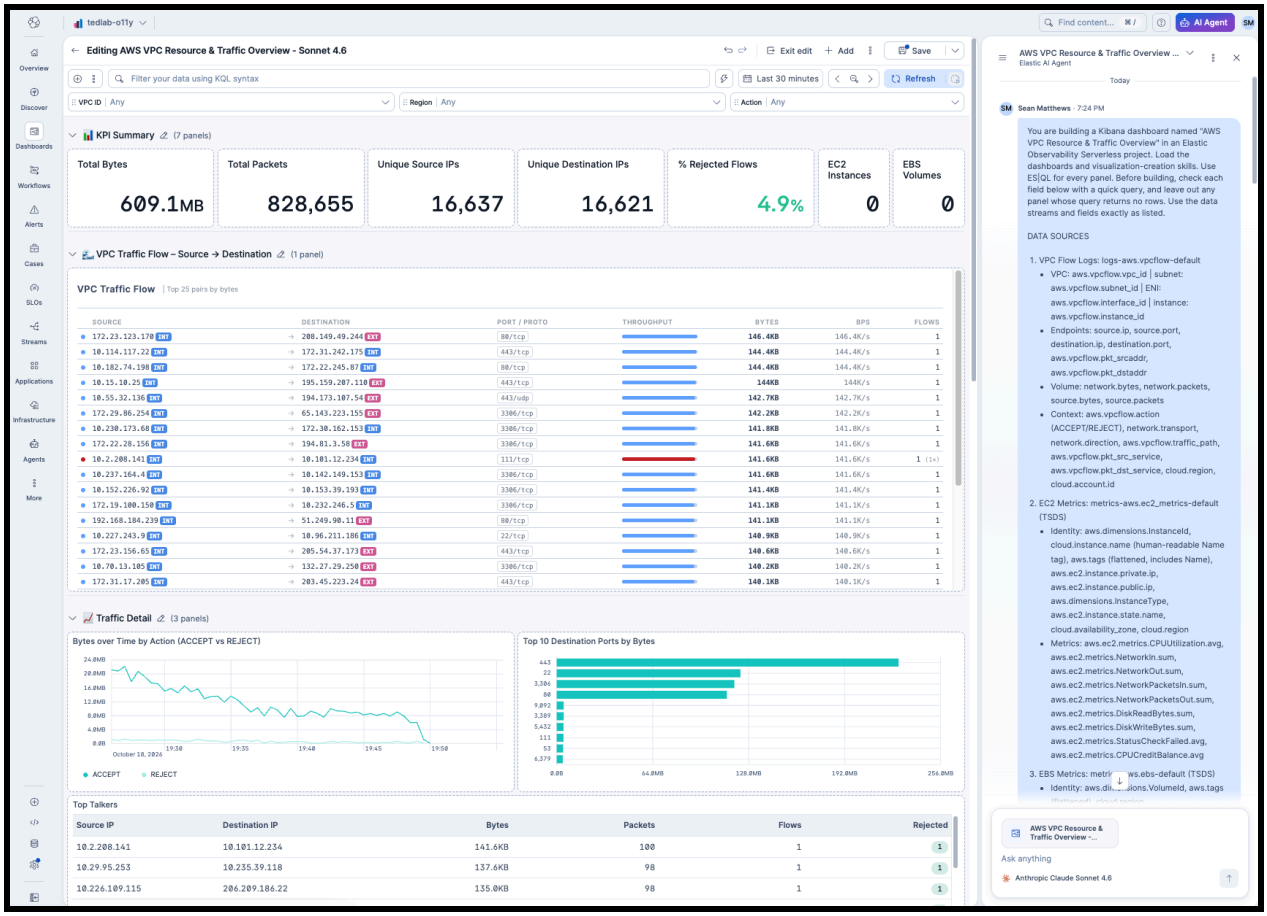
- EC2: metrics-aws.ec2_metrics-default — aws.dimensions.InstanceId, cloud.instance.name
    (Name tag), aws.ec2.instance.private.ip, aws.ec2.instance.public.ip,
    aws.dimensions.InstanceType, aws.ec2.instance.state.name, cloud.availability_zone,
    aws.ec2.metrics.{CPUUtilization.avg, NetworkIn.sum, NetworkOut.sum,
    DiskReadBytes.sum, DiskWriteBytes.sum, StatusCheckFailed.avg}
- EBS: metrics-aws.ebs-default — aws.dimensions.VolumeId, aws.tags,
    aws.ebs.metrics.{VolumeReadBytes.avg, VolumeWriteBytes.avg, VolumeReadOps.avg,
    VolumeWriteOps.avg, VolumeQueueLength.avg, VolumeThroughputPercentage.avg,
    BurstBalance.avg}
NAMING
Translate flow-log IPs to EC2 names by matching aws.ec2.instance.private.ip (or
    InstanceId to aws.vpcflow.instance_id) to cloud.instance.name. Use LOOKUP JOIN if
    possible, otherwise EVAL CASE(). Fallback map: 172.31.37.39=tedlab-o11y-agent,
    172.31.30.123=shedlab-elastic-agent, 172.31.46.211=stackops-trial,
    172.31.19.118=edot-collector-dbs. Label unmatched endpoints as the AWS service name
    if pkt_*_service is set, "Internal" for RFC1918 addresses, otherwise "External".

CONTROLS
ES|QL control ?vpc_id (values from aws.vpcflow.vpc_id) applied to all flow-log panels.

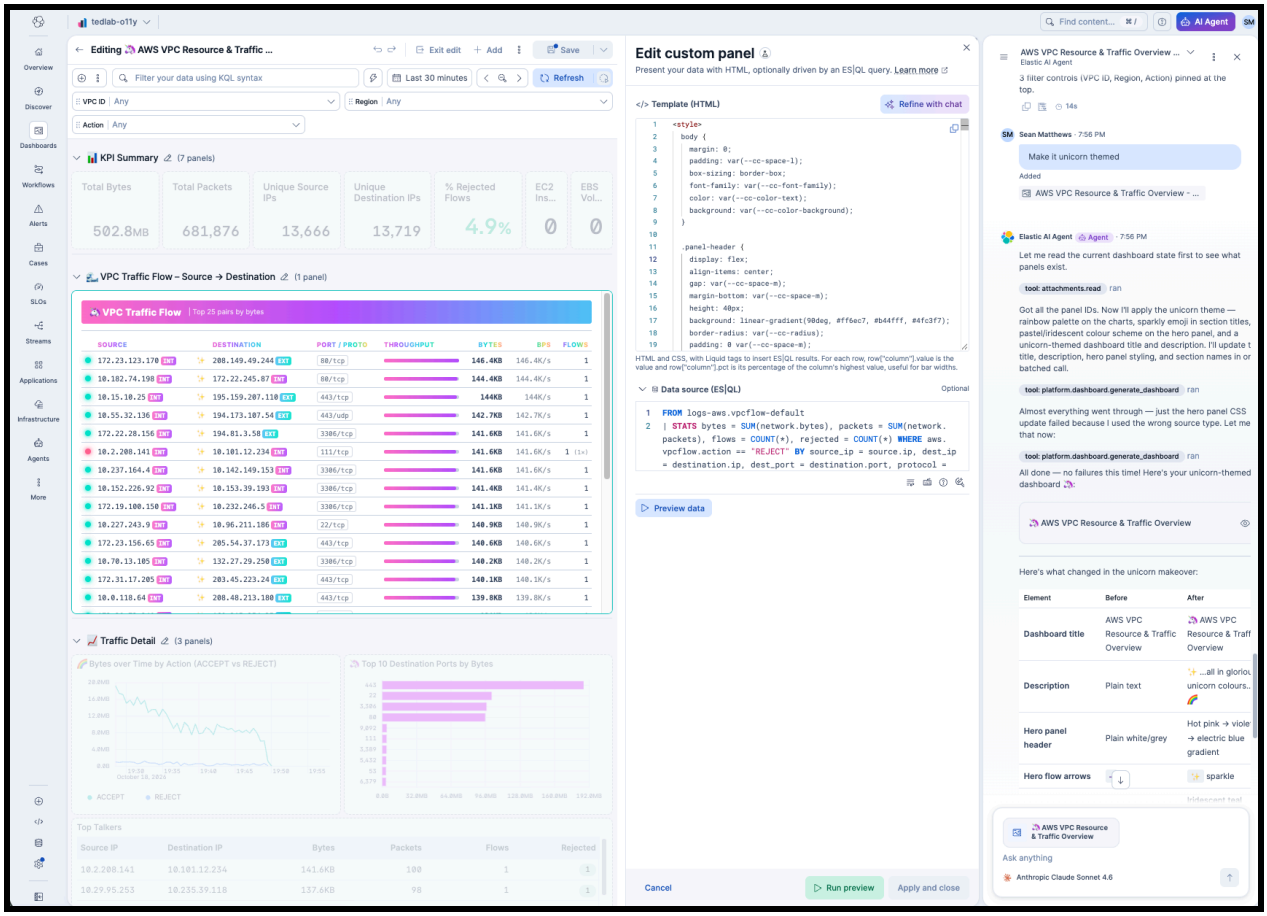
LAYOUT
1. KPIs: total bytes, total packets, unique source IPs, unique destination IPs, %
    rejected, EC2 instance count, EBS volume count.
2. Hero (full width, Vega node-link; fall back to custom_content HTML): "VPC Traffic Flow
    — Source → Destination". Top 25 named source→destination pairs by bytes, sources on
    the left, destinations on the right. Line thickness = bytes; colour green = all
    ACCEPT, amber = some REJECT, red = mostly REJECT. Tooltip: bytes (B/KB/MB/GB), flows,
    rejected, port/protocol.
3. Traffic: bytes over time by action; top 10 destination ports by bytes; top-talkers
    table (source name, destination name, bytes, packets, flows, rejected).
4. EC2: inventory table (Name, ID, type, state, AZ, private/public IP, latest CPU %);
    line charts by Name for CPU, NetworkIn/Out and DiskRead/Write; metric for count of
    instances with failed status checks.
5. EBS: volume table (ID, Name tag if present, avg read/write bytes and ops, queue
    length, throughput %, min burst balance); read/write bytes over time; queue length
    over time (flag values > 1); burst balance by volume, lowest first.

Use human-readable byte units and one colour scheme throughout. Save the dashboard and return
the link.
```

- You should see an output similar to the below:



- You can now make adjustments to it, like changing the theme, style or by adding something else in
    - Maybe a custom HTML panel?
    - How about a unicorn theme?
- **Simply ask the agent, and it will do it for you!**



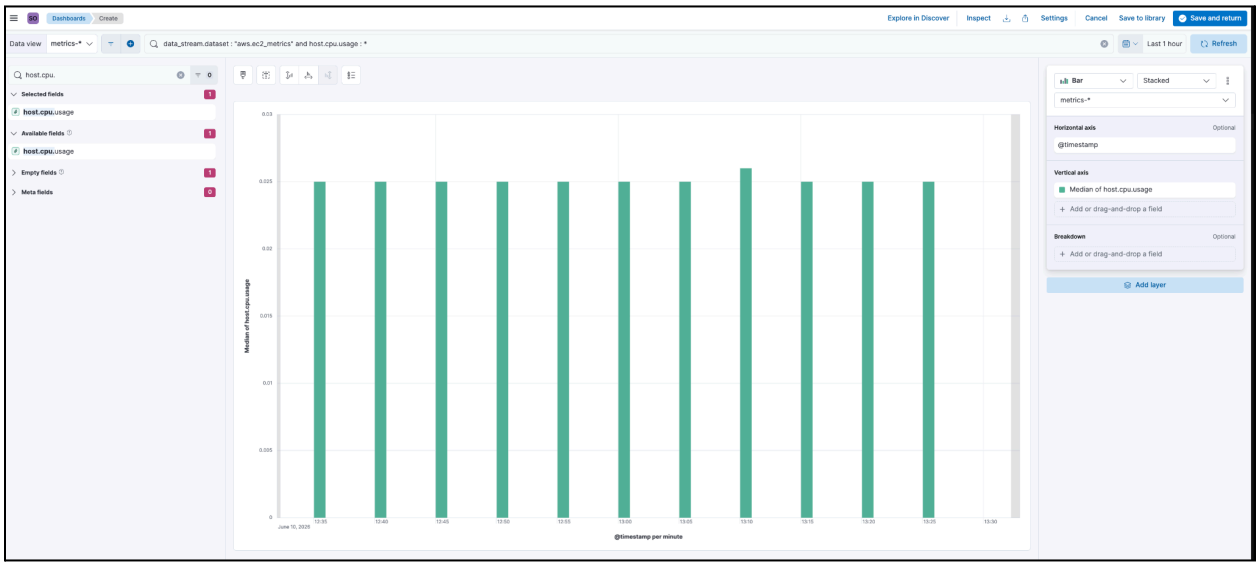
Option 2 - Alternatively, create it by hand:

1. **Create a new dashboard:**
    - In **Kibana** navigate to **Dashboards**
    - **Select:** Create a dashboard
    - **Select:** Save
        - **Name**: AWS Monitoring Dashboard
2. **Create a visualization for EC2 CPU Usage:**
    - Next navigate to **Visualisation** and **Select: Create Visualization**
        - This shall take you to a visualization builder
    - Filter the Data with the following items:
        - **Data View:** metrics-*

- **Search Filter/KQL syntax:** data_stream.dataset : "aws.ec2_metrics" and host.cpu.usage : *
    - Then select **Refresh** (This will filter the available data to your query)



- Format the Visualization to a line Graph:
    - In the left-hand side, filter the available fields for **host.cpu.usage**
    - Select this field and drag it inter the center of the visualization, **you should now see a bar chart**
        - Kibana will infer the data and present suggestions

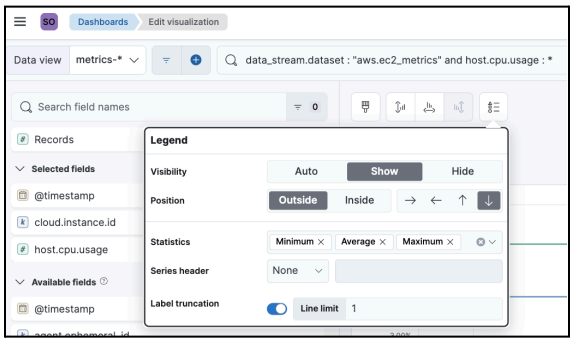


    - We are going to change this to a **line graph**
        - Simply select where it says “**Bar**” in the top right corner, and change this to “**Line**”
- Change the Vertical Axis:
    - Select the Vertical axis where it currently says “Median of host.cpu.usage”
    - Change the following:
        - **Functions**: Average
        - **Field**: host.cpu.usage

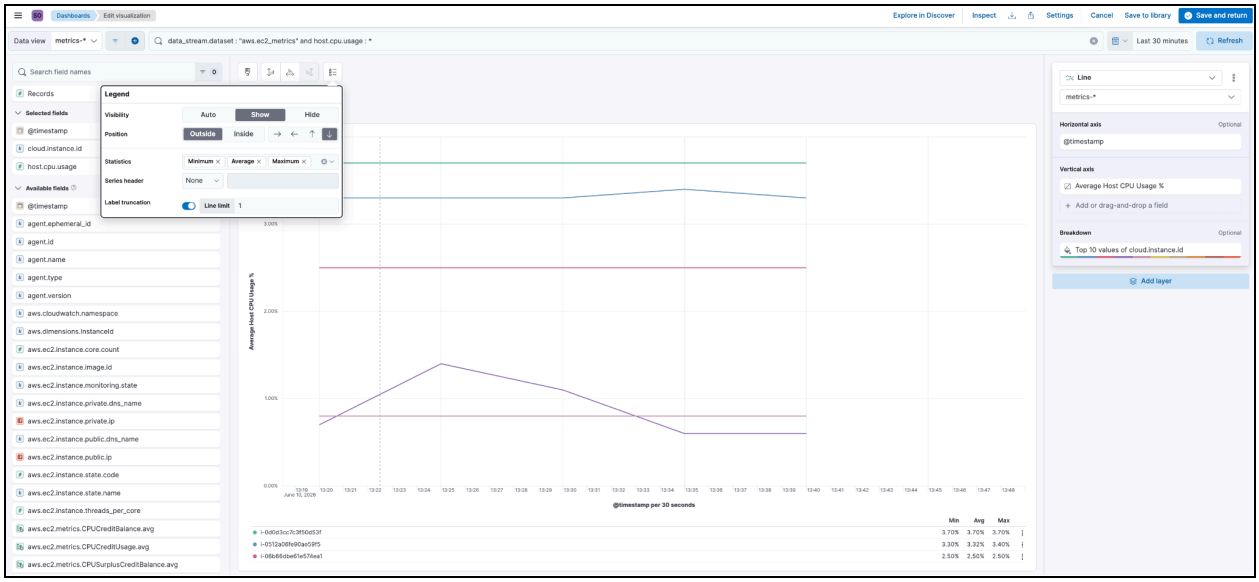
- **Appearance**:
    a. **Name**: Average Host CPU Usage %
    b. **Value format:** Percent

- Select **close** in the bottom left corner

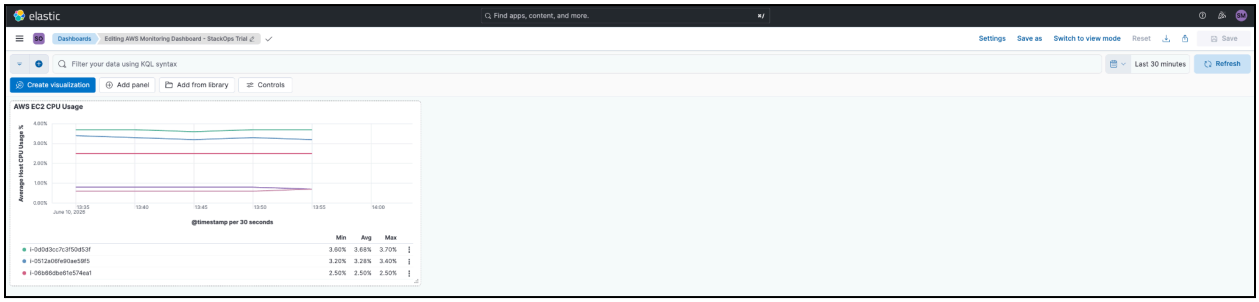
- Add a breakdown - this will help us to change the visualization from showing an aggregate of all EC2 instances, to showing each individual one (Or the top 10 in our case):
    - Select the "**+ Add or drag-and-drop a field"** in the Breakdown section on the right hand side
    - Change the following:
        - **Data**: Top values
        - **Fields**: cloud.instance.id
        - **Select**: Aggregate by this dimension first
        - **Number of Values:** 10
        - **Rank by**: Average Host CPU Usage %
        - **Rank direction**: Descending
    - Select **close** in the bottom left corner
- Lets now edit the legend to fir our needs
    - Click on the Legend icon
    - Set the Legend as per the below image



- You should now see a visualization that looks like this:



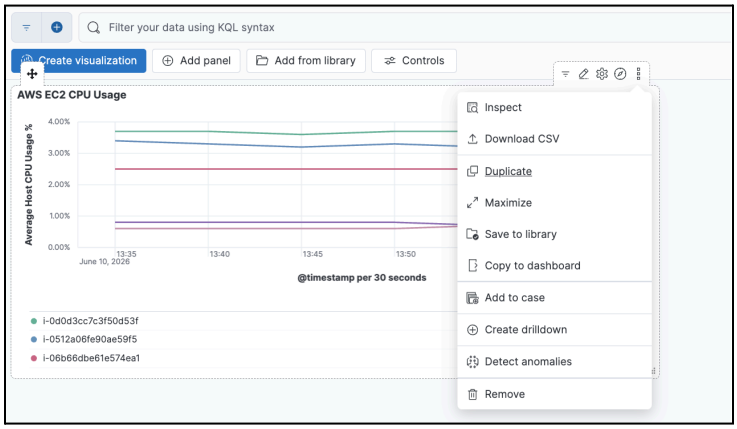
- Select Save and return, this shall take you back to our dashboard
  - We are just missing a couple of things now for this visualization
    - Lets give it a name by hovering over the top right corner of the visualization → Select the **Settings** cog
      a. Give the Visualization a **name**: AWS EC2 CPU Usage
      b. **Save**
    - By hovering over the bottom right corner you will reveal a resize option, lets make the visualization slightly smaller to roughly ⅓ of the width of the dashboard
- **Save** the dashboard in the **top right corner** of the screen. Your dashboard should now look like this



3. **Create a visualization for Ingress Network Bytes per EC2 instance:**
   - **Note:** Now we know how to create a visualization, this will be much quicker as we

will leverage some of the settings from step 2

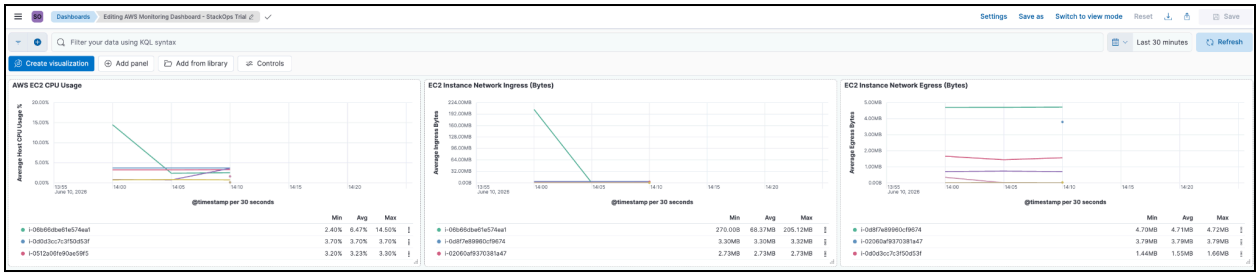
- However over the **top right corner** of the AWS EC2 CPU Usage visualization ⟶ select the **3 dots** to reveal actions ⟶ select **Duplicate**
    - This shall create a duplicate visualization so we don't have to re-invent the wheel



- Edit this **NEW** visualization by hovering over the top right of the visualization to reveal a **pencil icon** ⟶ select the **pencil icon** (**Edit lens**) ⟶ **Edit in lens** (This shall reveal the same edit window we used earlier)
- Apply the following changes:
    - **KQL syntax:** data_stream.dataset : "aws.ec2_metrics" and host.network.ingress.bytes : *
    - Vertical Axis:
        - **Field**: host.network.ingress.bytes
        - **Appearance**:
            a. **Name**: Average Ingress Bytes
            b. **Format**: Bytes (1024)
    - Breakdown:
        - **Rank by:** Average Ingress Bytes
    - Everything else remains the same
- **Select:** Save and return
- Finally, update the **Name:** EC2 Instance Network Ingress (Bytes)

- Save the dashboard

4. **Create a visualization for Egress Network Bytes per EC2 instance:**
    - **Repeat the same steps in Step 3, however apply the following:**
        - Apply the following changes:
            - **KQL syntax:** data_stream.dataset : "aws.ec2_metrics" and host.network.egress.bytes : *
            - Vertical Axis:
                a. **Field**: host.network.egress.bytes
                b. **Appearance**:
                    i. **Name**: Average Egress Bytes
                    ii. **Format**: Bytes (1024)
            - Breakdown:
                a. **Rank by:** Average Egress Bytes
            - Everything else remains the same
        - **Select:** Save and return
        - Finally, update the **Name:** EC2 Instance Network Egress (Bytes)
            - Save the dashboard



5. Add in the VPC Flow Logs visualizations
    - **Note:** For this bit we shall leverage the out-of-the-box dashboards to make life easier
    - Navigate to the [Logs AWS] VPC Flow Log Overview dashboard
        - Hover over the top right of the **VPC Flow Total Requests** visualization to

reveal the actions icon (3 dots)

- **Select:** Copy to dashboard
- **Select**: Existing dashboard → Enter your "**AWS Monitoring Dashboard**"
- **Select:** Copy and go to dashboard

- This shall add the visualization to your custom dashboard
- Now repeat the same steps but copy the VPC Flow Reject Logs visualization to your AWS Monitoring Dashboard
- Your dashboard should now look like this



6. **Add a control to allow easy filtering:**

- **Remember that we have used cloud.instance.id in the breakdown for our visualizations?** Well, we did this for a reason.
  - StackOps and Elastic Observability normalizes fields into a common schema, for example aws InstanceId's are added to a field called cloud.instance.id so it can be easily searched with a consistent naming convention
- Lets use this field to our advantage by adding a control:
  - At the top of your dashboard click **Add → Controls → Add Control**
  - In the Field search for "cloud.instance.id" → Select this so it is **highlighted → Save**

- This will add the control to the top of the dashboard. You can now use this to quickly select an EC2 instance and filter the whole dashboard to this InstanceId



Dopamine Moment:

- **Congratulations!** In this section you have managed to create the following:
    - A number of personalized visualizations based on your requirements
        - In each of the visualizations useful clear labels have been added to show what exactly is being displayed (%, Bytes, names etc..)
    - A personalized dashboard to show key information required for monitoring your resource usage in AWS
    - Combined multiple data sources to produce this dashboard
        - Remembering that these data sources all leverage the out-of-the-box integrations which normalize the data into a common and consistent schema allowing us to slice and dice the data displaced on the dashboard
    - Created simple logical controls to help you filter the data shown on this dashboard
- Whilst this is a simple dashboard and a simple use case, you can create any dashboard for any use case

- The key here is having the data in a consistent and clean format

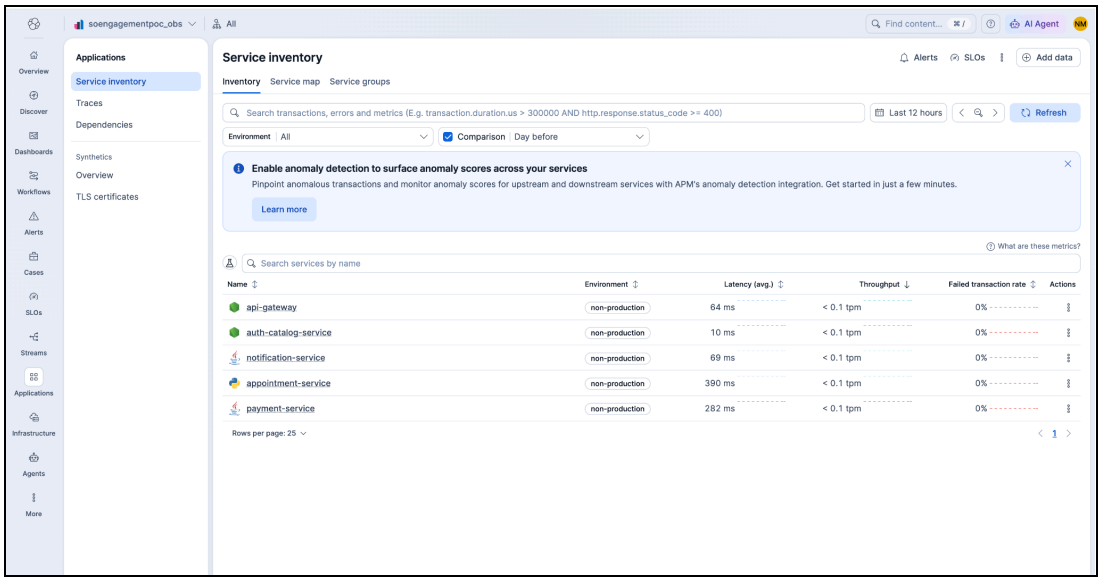
Stage 4: Tracing your Application flow

Objective:

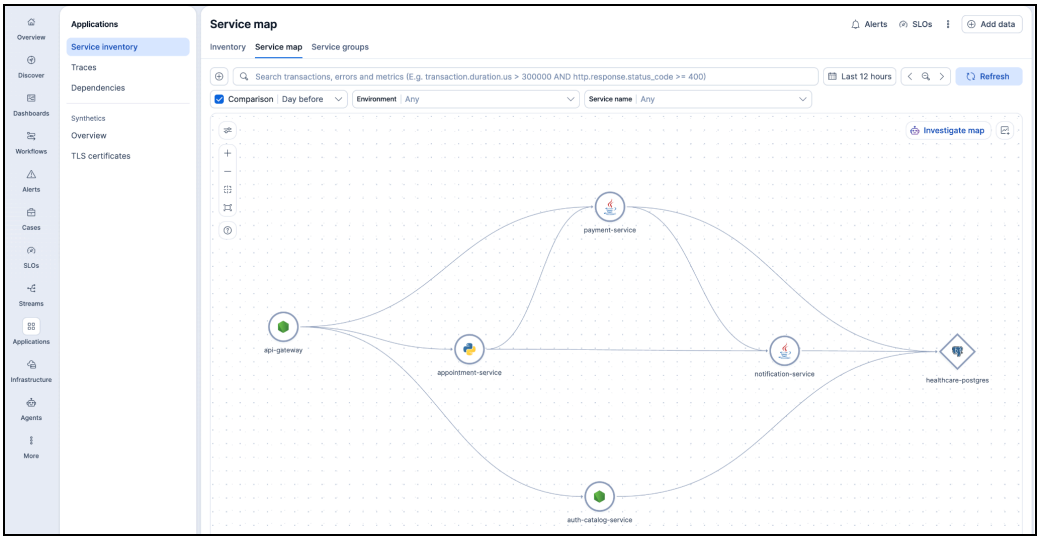
- Use the out-of-the-box StackOps Elastic Observability dashboards to check application health and troubleshoot failures by linking transactions back to their source logs.

Steps:

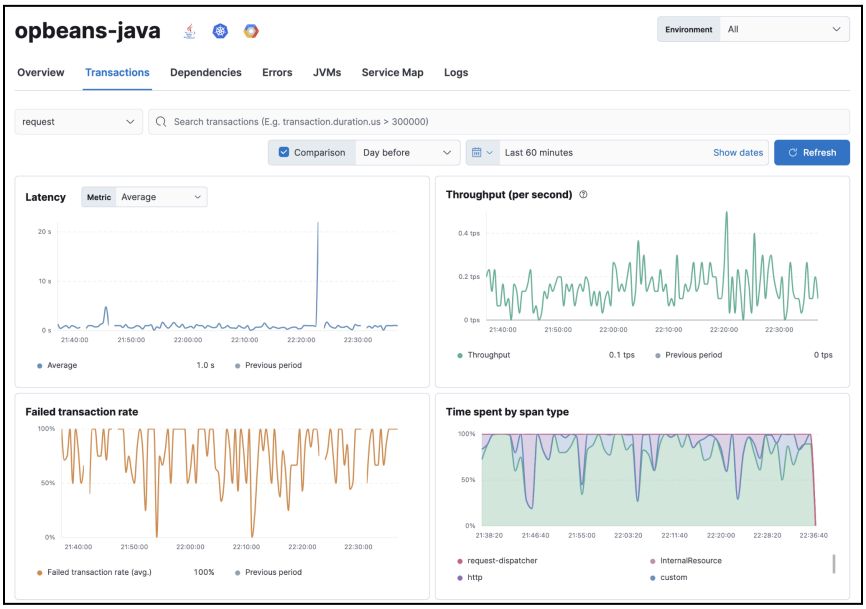
1. In **Kibana**, navigate to **Applications → Service Inventory** within Kibana
    - If you have instrumented an application with OpenTelemetry as per Stage 3 in the Setting Up Monitoring section, then you should see some applications listed here - (example below).



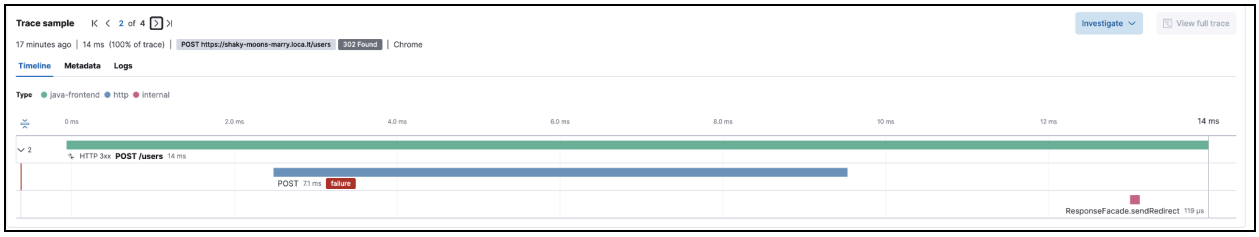
    - You can also select the Service Map to get an idea of how the various services connect together within the application.



2. Select one of the services here
    - A Java app is always a good one to choose and will be what we use in the example
3. As soon as you open the service, you shall be able to view key filtered dashboards.
    - **Select** the Transactions tab, you'll see something similar to the below

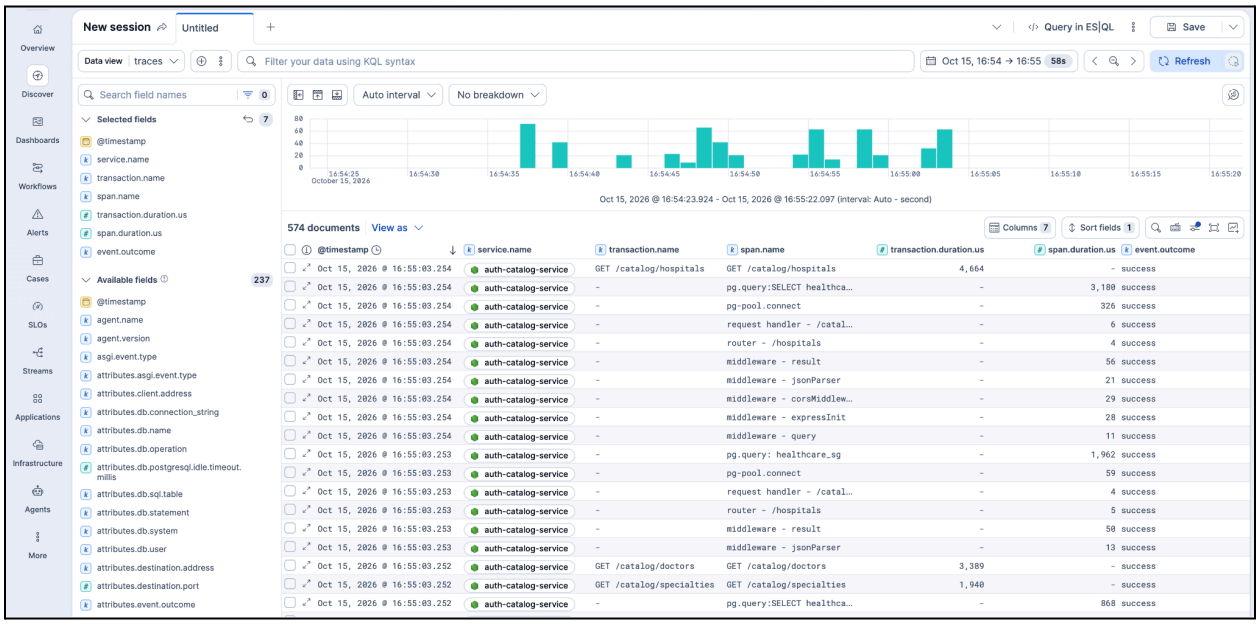


    - You can also select the other tabs to view Errors, Dependencies and Logs
4. Scroll to the bottom of the Transaction page, this will show you individual transactions
5. Select one of the transactions, this will expand and show you a timeline of the Transaction including any spans included within the transaction



- This shows you where your time is being spent within the application, based on the services and the relevant functions
- This will also show you any failures or non-successful actions within the application

6. You can pivot straight to the logs associated with this transaction by selecting **Investigate → Trace Logs**



- **FYI:** If you want to try this out with a sample application, you can leverage this GitHub repo which is used for some of the StackOps in-person workshops

Dopamine Moment:

- You have seen how to view the health of your individual services within an application
- The Service Inventory and Service Map are extremely useful when trying to understand the health of your application
- As you instrument more and more applications with OpenTelemetry (EDOT) you shall see more and more services appear on these dashboards

    - Each language you instrument will show slightly different metrics on the dashboards based on the types of telemetry signals they output
    - For example; Java will be slightly different to Go
- You can get a clear insight into the overall performance of the service from the **Overview** tab, with the other tabs being used to give you more granular insights

Annex

AWS Monitoring Dashboard

To import the pre-created dashboard from the Day 2 Operations → Personalize your data section, follow the below steps:

1. In Kibana, go to **Admin and Settings** → **Saved Objects**
2. **Select**: Import
3. Upload the **aws-monitoring-dashboard.ndjson** and select create new objects with random IDs
4. You should now see this dashboard listed in the **Dashboards**

Additional Learning

StackOps Resiliency Accelerator (SORA)

To facilitate rapid deployment, StackOps provides the SORA, a specialized tool designed to automate the setup of monitoring configurations. SORA can discover your AWS resources (like S3, VPC, etc) and generate the necessary Elastic Agent configuration files. This helps to reduce the complexity of configuration and deployment.

[Learn more]:
To use SORA, you may refer to the following learning resource:

- Video guide
- Online documentation

Elastic Agent Builder

Elastic Agent Builder is a conversational environment for building AI agents that answer questions and take actions over your data using natural language. Agents combine large language model (LLM) reasoning and context engineering with built-in and custom tools that query your indices, so responses stay grounded in your data.

[Learn more]:
https://www.elastic.co/elasticsearch/agent-builder
https://www.elastic.co/search-labs/blog/elastic-ai-agent-builder-context-engineering-introduction
Elastic Agent Builder: Build your most relevant AI agents with Elasticsearch

Elastic Workflows

Elastic Workflows is the native automation engine built into the Elasticsearch Platform. It provides reliable, scripted automation for straightforward tasks and AI-driven automation to solve complex problems that require human-like reasoning. Trigger actions from events, schedules, or on-demand requests. No external tools or messy integrations required.

[Learn more]:
https://www.elastic.co/docs/explore-analyze/workflows
https://www.elastic.co/blog/elastic-workflows-technical-preview
Using Elastic Workflows to create a business impact detector

Conclusion

Continuous Learning

We hope the 14-days trial has allowed you to quickly understand how to set up Elastic monitoring to kick-start your observability journey. You are encouraged to refer to StackOps online documentation to explore other capabilities and integration beside those listed in this document.

In additional, you are also highly encouraged to attend the hands-on workshops curated by StackOps, in partnership with Elastic, to learn more on other capabilities like Machine Learning, Universal Profiling, ES|QL for complex log analysis, etc and improve your observability implementation.