Book review – Infrastructure Monitoring with Amazon CloudWatch: Effectively monitor your AWS infrastructure to optimize resource allocation, detect anomalies, and set automated actions

By Ewere Diagboya (Author)

Summary: A good beginner's guide to AWS Monitoring with CloudWatch with many fundamental concepts, step by step walkthrough and setup examples, tips for monitoring many common and important AWS services !

This is a good book for System Administrators/SREs to set up useful and important monitoring/alarms/dashboards of AWS services. It has 2 sections. Section 1 is more an introduction to AWS CloudWatch, Events, Alarms, Logs, Metrics and Dashboards. Section 2 is the centerpiece of this book with many step by step guide to set up useful monitoring for different AWS services, e.g. EC2, ECS, EKS, Fargate, DB services (RDS, Dynamo, ElastiCache, Document DB, RedShift), Serverless (Lambda, API Gateways, SQS, SNS, Steps, tracing with X-Ray), Big Data (ETL, Kinesis, CloudTrail, Glue), Storages (EBS, S3, EFS, FSx), Networking (VPC, flowlogs, AppMesh, CloudMap, Route 53). The last Chapter is a summary of best practices.

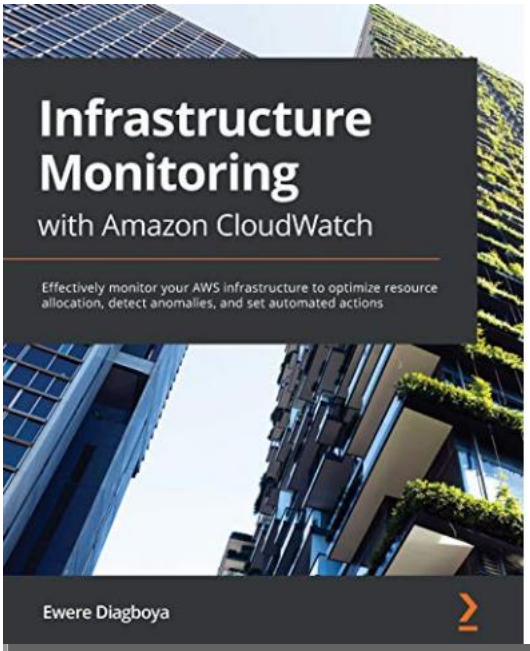
It shows important metrics to be monitored for different services, e.g. Average Queue Length (AQL) for EBS volume.

The only suggestion is some links to https://www.packtpub.com/product/aws-masterclass-monitoring-and-devops-with-aws-cloudwatch-video will result in 404 because this course is no longer available in Packt site.

1/ buy it at Amazon: https://www.amazon.com/Infrastructure-Monitoring-Amazon-CloudWatch-infrastructure-ebook/dp/B08YS2PYKJ/ref=sr_1_1 (kindle $17.19)

2/ authors book demo videos 1-9: https://www.youtube.com/playlist?list=PLeLcvrwLe186RQpGRXrruQ-zWFPYrFS5E

3/ sample code in GitHub - https://github.com/PacktPublishing/Infrastructure-Monitoring-with-Amazon-CloudWatch



Section 1: Introduction to Monitoring and Amazon CloudWatch

This part is focused on introducing the concepts of monitoring, logging, metrics, and alerts in a monitoring system. It will also explain the importance of monitoring infrastructure and applications and the advantages of using CloudWatch as a managed service for collecting metrics and alerting.

The following chapters are included in this section:

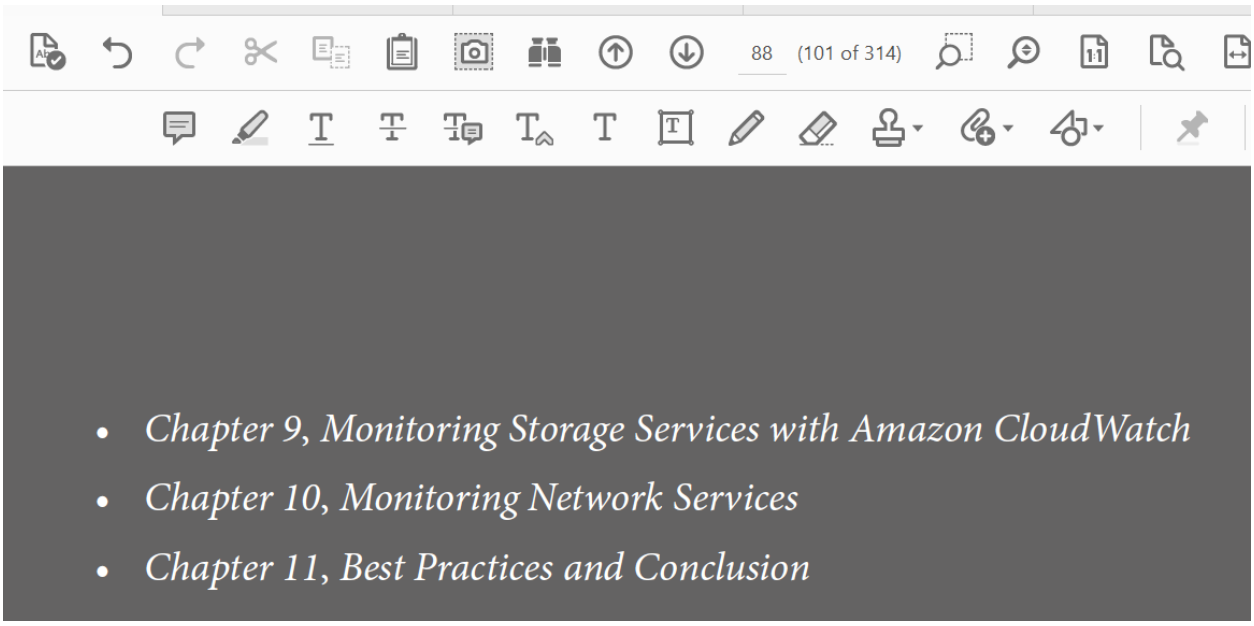
- *Chapter 1, Introduction to Monitoring*
- *Chapter 2, CloudWatch Events and Alarms*
- *Chapter 3, CloudWatch Logs, Metrics, and Dashboards*

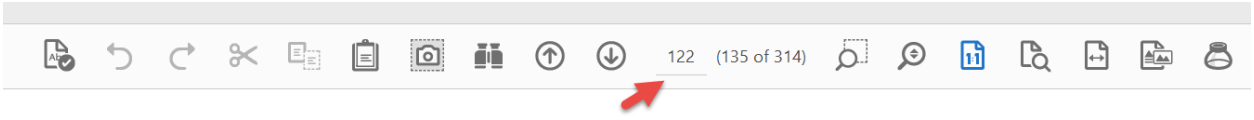
Section 2: AWS Services and Amazon CloudWatch

After whetting your appetite in the first part with the meaning of monitoring and the various pillars that make up monitoring, we shift gears and move faster to see real-life applications of monitoring using CloudWatch as our monitoring tool. Every AWS service can be monitored and tracked and understanding can be drawn from the behavior of the infrastructure. Every service is unique in its own way. A better understanding of its metrics will help improve the availability of the system, improve system performance, and cut down unnecessary costs in other applications.

The following chapters are included in this section:

- *Chapter 4, Monitoring AWS Compute Services*
- *Chapter 5, Setting Up Container Insights on Amazon CloudWatch*
- *Chapter 6, Performance Insights for Database Services*
- *Chapter 7, Monitoring Serverless Applications*
- *Chapter 8, Using CloudWatch for Maintaining Highly Available Big Data Services*





We will then connect this knowledge to how CloudWatch exclusively manages container applications with its unique feature called CloudWatch Container Insights. This makes it easier to monitor, manage, and work with container monitoring, because this feature is specifically tailored to containers both within the AWS ecosystem and any other kind of system that runs containers for their applications. We will corroborate all this understanding by doing a practical setup of metrics, monitors, and dashboards based on the different AWS container services.

We will cover the following topics in this chapter:

- Introducing the concept of containers
- Orchestrating container services in AWS
- Triggering ECS events on Amazon EventBridge
- Configuring CloudWatch Container Insights
- Monitoring of EKS applications and services
- Setting up custom dashboards and metrics for containers
- Case study of Amazon EKS logging and monitoring

For this EventBridge example, using ECS, we shall configure Amazon EventBridge to respond to a state change in ECS. This can be helpful to know when something happens in an ECS cluster. We already explained the different states a task can be in. But it isn't just tasks that change in ECS. There are other components of an ECS cluster that also change. Some other components of an ECS cluster that have state changes include the following:

- **Services**: Some state changes include `SERVICE_STEADY_STATE`, `TASKSET_STEADY_STATE`, `CAPACITY_PROVIDER_STEADY_STATE`, and `SERVICE_DESIRED_COUNT_UPDATED`.
- **Deployment**: Some state changes include `SERVICE_DEPLOYMENT_IN_PROGRESS`, `SERVICE_DEPLOYMENT_COMPLETED`, and `SERVICE_DEPLOYMENT_FAILED`.

For this EventBridge scenario, we will be using a custom **event pattern**, which is not in EventBridge by default. This pattern is designed to read a deployment state change event from Amazon ECS. It is going to check whether a service deployment fails, is in progress, or is completed, and will trigger an SNS topic to send an email of the status of the deployment. This event will be targeted to a particular ECS Fargate cluster we have already created. The name of the cluster is `samplefargate`. To link up the event to the ECS cluster, we will be using the **ARN** of the ECS Fargate cluster.

Monitoring of EKS applications and services

We already mentioned earlier the different aspects of an EKS cluster. This time, we will be talking about how we activate and configure monitoring for the master node (also called the control plane), and then we will talk about how to collect logs for the pods that are deployed in the worker node. But first, let's talk about how to configure monitoring for the EKS control plane.

Monitoring the control plane

First, let's see how to enable monitoring on Amazon EKS. When an EKS cluster is set up, the monitoring features are usually not activated by default. We shall activate them on the EKS console when we have a control plane running. To do this, we go to the EKS console of the cluster that has been created.

To activate the different components of the control plane, we take the following steps:

1. Log in to the AWS management console and navigate to **Amazon EKS** on the services menu.
2. Click on the **Clusters** link on the side menu to view the list of EKS clusters that have been created.
3. Click on the cluster that needs the logging features enabled. This will take us to the cluster configuration page:

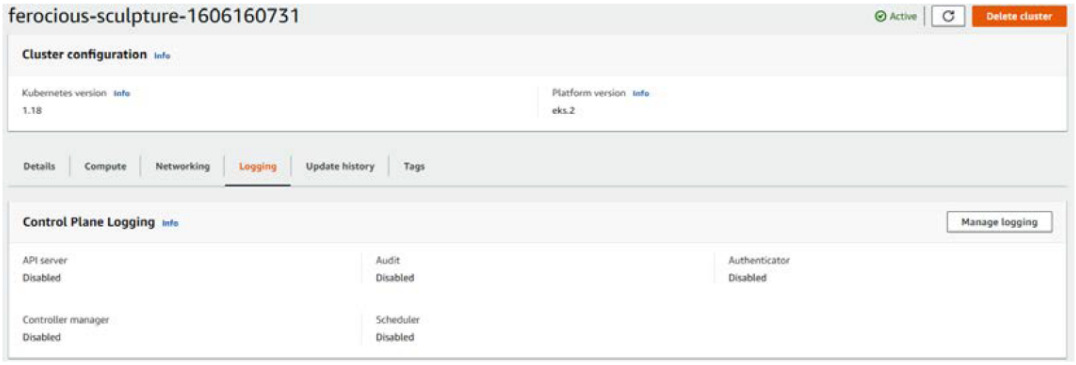


Figure 5.11 – Configuring logs for the EKS control plane

4. Click on the **Logging** option to take us to the **Control Plane Logging** tab. This will show the different components of the control plane that can be monitored. In the screenshot in *Figure 5.11*, they are disabled by default.

5. Click on the **Manage logging** button to change the settings for the logging option:

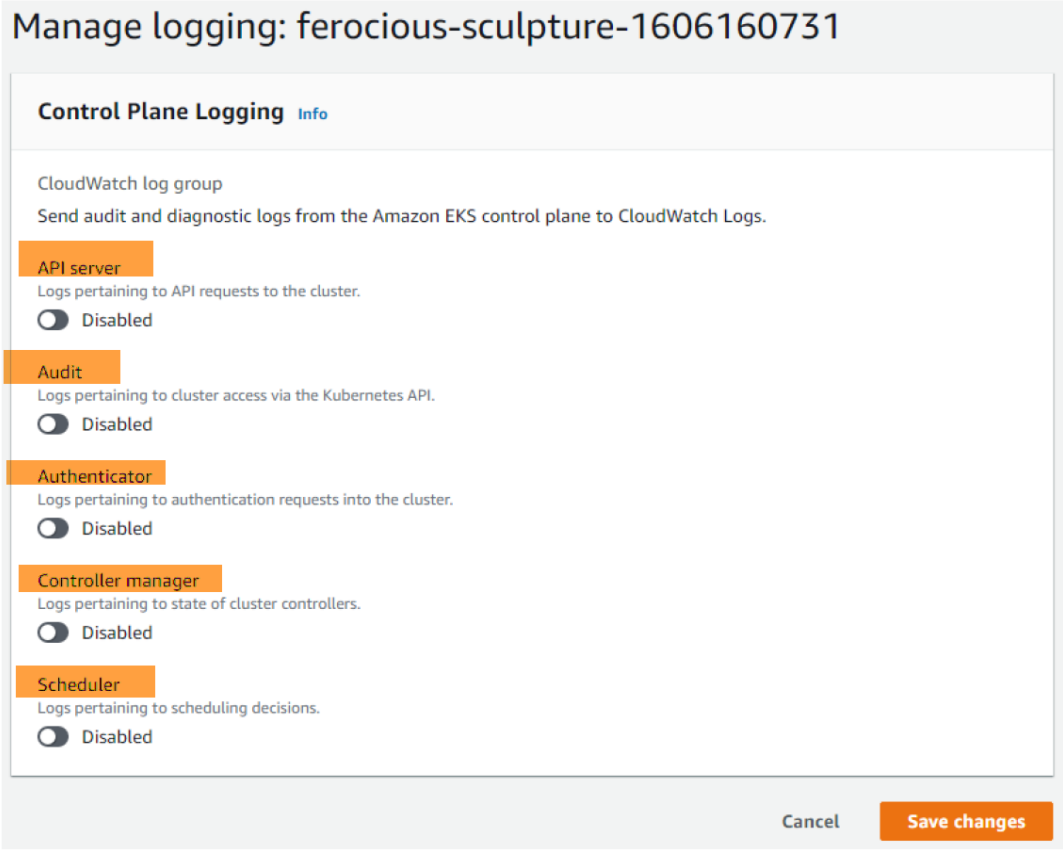


Figure 5.12 – Activating logging for different control plane components

6. To activate logging for any of the components, click on **Disabled** (or the switch beside **Disabled**) to make it **Enabled**, and then click on the **Save changes** button. For our example, we only activated **API server**.

   The logs will get written to a CloudWatch log group in a specific format, as shown in the following screenshot:

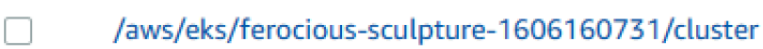


Figure 5.13 – Log group for the EKS control plane

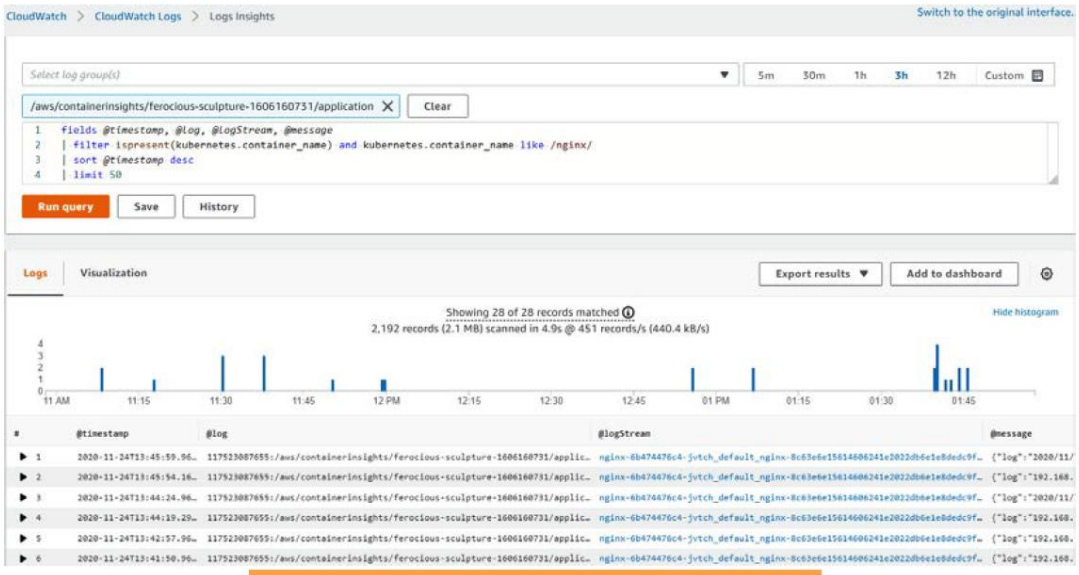


Figure 5.21 – Pod logs in CloudWatch Logs Insights

> **Important Note**
>
> The `kubeconfig` file in Kubernetes is a configuration file that contains both authorization and authentication information used to communicate with the Kubernetes cluster. The `kubeconfig` file is made up of certificates and the Kubernetes API server URL. The certificates serve the password used to authenticate every request made to the Kubernetes API server to perform all types of Kubernetes administration operations.

We have successfully configured Container Insights for EKS. Let's help to streamline the automated dashboard(s) from EKS and create a more specific dashboard. The next section is about how to use the CloudWatch dashboard to configure a more specific dashboard.

Setting up custom dashboards and metrics for containers

Just like every other dashboard in CloudWatch, CloudWatch dashboards are usually generated automatically, when Container Insights has been activated. Be it Amazon ECS or Amazon EKS, CloudWatch dashboards are automatically generated that take note of the top relevant information of the cluster in which the Container Insights agent is actively running. It is also possible to create a custom dashboard from the logs that have been collected from the containers running within the cluster setup.

10. The next step is to add the ECS dashboard. We shall follow the same procedure, but this time, instead of EKS clusters, we shall click on the dropdown and select **ECS Clusters**:

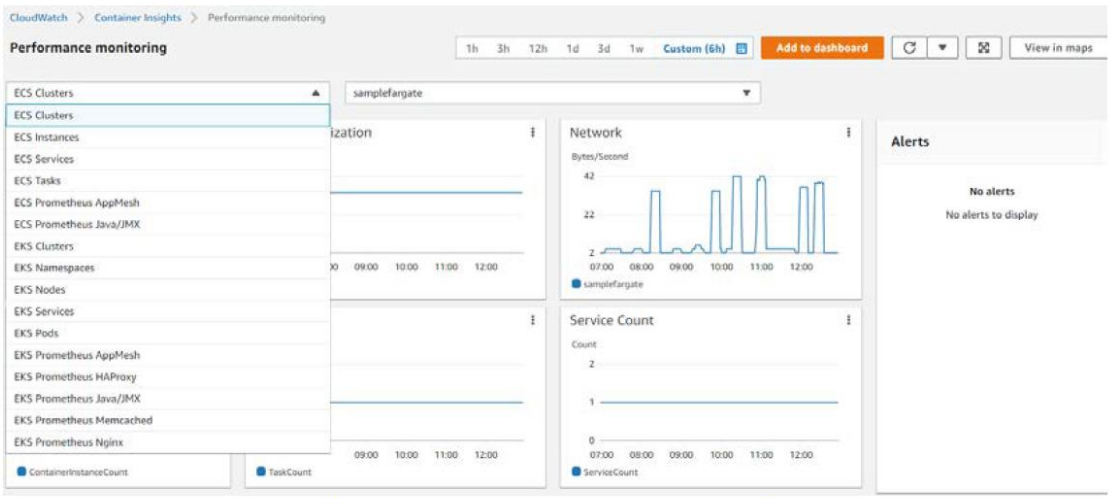


Figure 5.24 – Selecting ECS Clusters

11. Click on **Add to dashboard** again to add the ECS widgets to the same dashboard, which is the `ContainersHub` dashboard.

    This will give us an aggregated dashboard that contains both ECS and EKS metrics.

12. Click on the **Save dashboard** button to save the widgets that have been added to the dashboard.

We can modify the dashboard by removing some widgets that are not very crucial for us to monitor at all times. For this scenario, I will be removing a couple of widgets, such as **Task Count**, **Container Instance Count**, **Cluster Failures**, **Service Count**, and **Number of Nodes**, as I just want the CPU, memory, disk, and network widgets. We can delete a widget by clicking on the menu button on the widget and clicking on the **Delete** option, as shown in the following screenshot. When done, click the **Save dashboard** button to save all the changes that have been made to the dashboard:

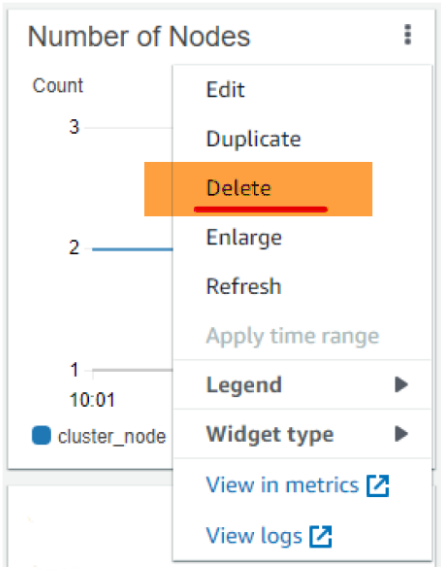
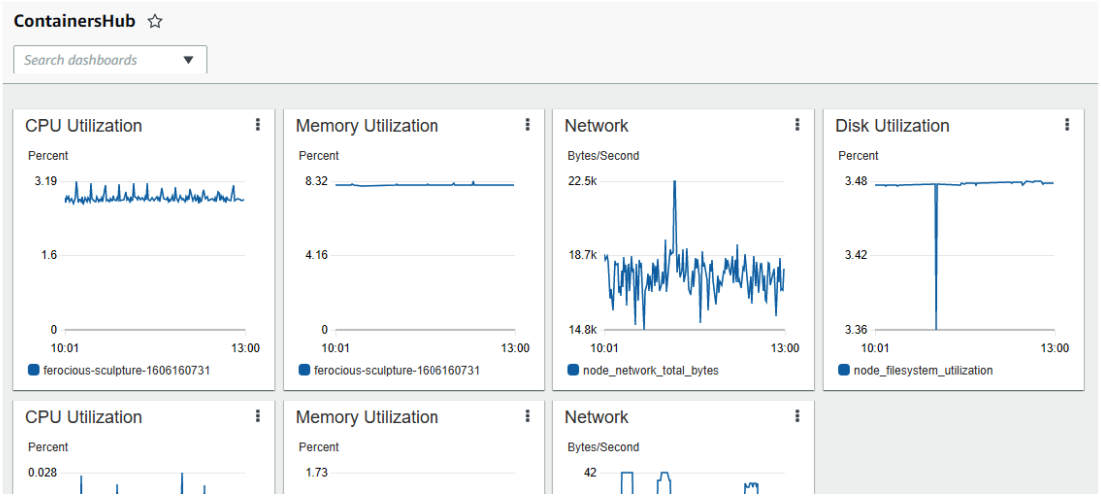


Figure 5.25 – Deleting a widget

We shall delete all the other widgets and then we'll have a final dashboard that is more concise and looks like this:



Now that we have configured a proper dashboard to monitor our ECS and EKS clusters, let's look at a case study that will cover all that we have learned so far.

Case study of Amazon EKS logging and monitoring

You are a DevOps engineer who has just been employed to help migrate a couple of monolithic applications into microservices. The choice made for managing the microservices is to use a container-based system due to the advantages containers have. The company has gone further to say that they will be using Amazon EKS as the service for orchestration of the Docker images that have been built. Your company has just got the ISO 27001 certification for information security. So, things such as auditing applications and services are taken very seriously. Logs of applications are to be kept and audited from time to time to ensure that the company continues to remain compliant. Added to that, close monitoring of the applications and notification alarms needs to be configured to receive notifications when applications have any kind of behavior that is not normal.

Solution

From the preceding case study, a couple of things need to be taken into consideration to make this setup successful and meet the requirements that have been stated:

- An EKS cluster needs to be set up with monitoring enabled on the EKS cluster.
- Container Insights needs to be able to collect logs and be stored with a long retention period, or logs to be sent over to Amazon S3 for later auditing.
- Metrics and alarms will need to be configured for the logs collected to trigger an alarm when anything that has to do with `ERROR` is found in the logs.

This case study helps us to mimic a real-life scenario of the kind of task that will be expected of a DevOps engineer or a cloud engineer when we have acquired monitoring skills. We shall be covering more real-life scenario case studies of what will be required as an engineer with monitoring knowledge.

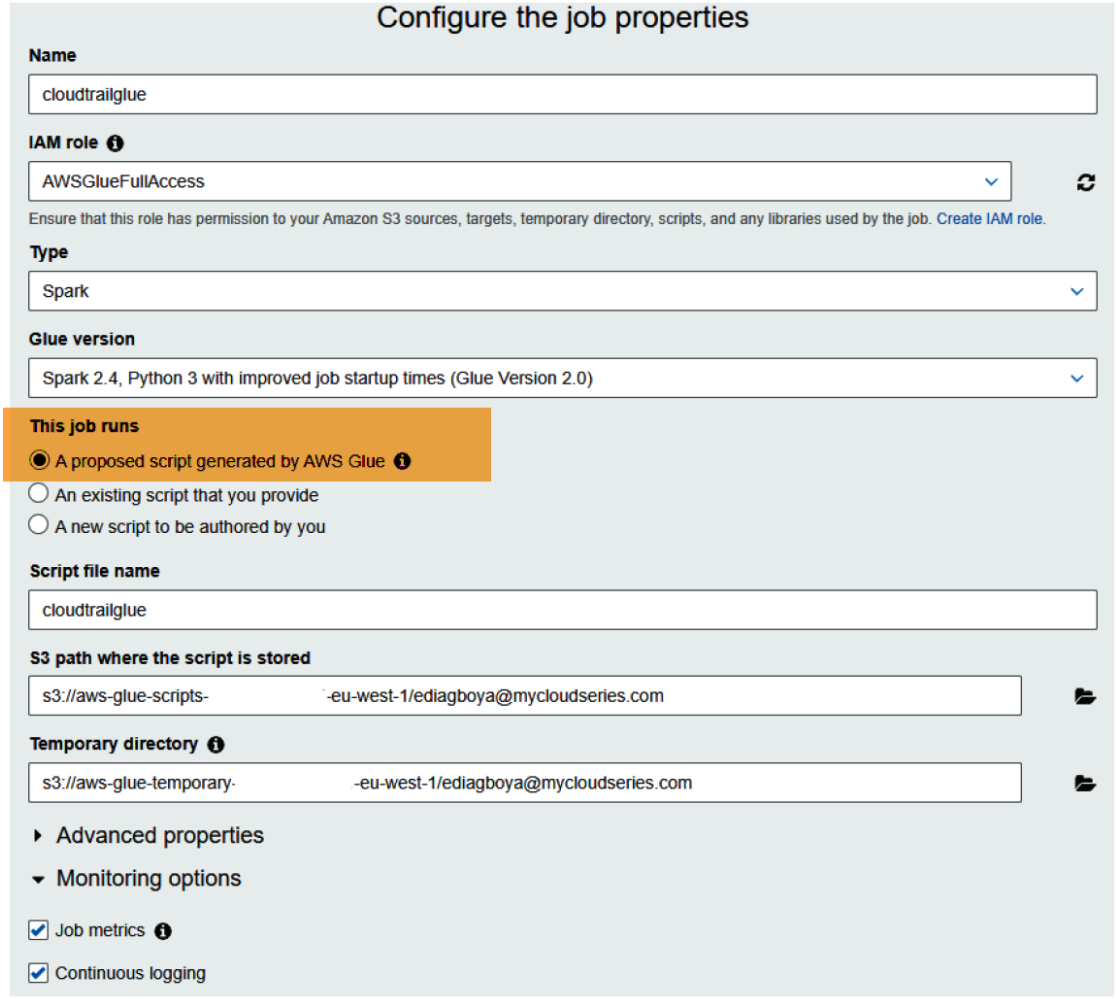


Figure 8.8 – Configuring the Glue job

5. Select the data source, which will be **cloudtrail_logs_athena_cloudtrail_log_book**, which we created in the previous section. Then, click **Next** to continue.
6. For the transform type, leave the default option selected, which is **Change schema**. Click **Next** to continue.

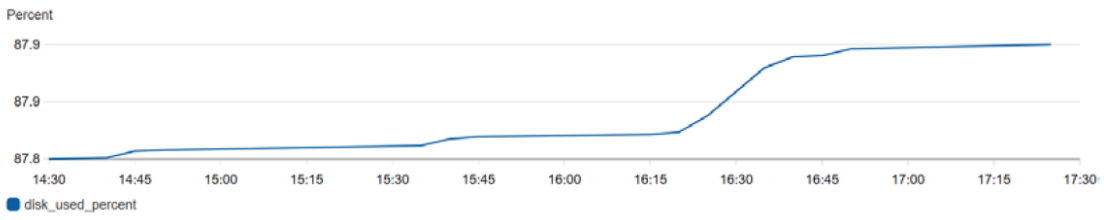
Another important metric to look at is **Average Queue Length**. There is a standard that needs to be maintained for an EBS volume; this standard is an **Average Queue Length** (**AQL**) of at least one per minute. From the graph in the preceding screenshot, we can see that the AQL is less than one per minute, which means that everything is running fine with the EBS volume. The read and write throughput also gives you an idea of the read/write speed of the EBS volume.

However, there is a metric that is not visible in this dashboard, even when you try to view this dashboard of the EBS volume in CloudWatch. That metric is the percentage of disk space consumed. This metric can only be obtained when the Unified CloudWatch agent is installed in the EC2 instance where the EBS volume is attached. Then, that metric is activated during the installation and is sent to CloudWatch. Most of the time, it is impossible to know when the disk space is filled up because, by default, there is no dashboard for that. We explained how to configure the Unified CloudWatch agent in *Chapter 4*, *Monitoring AWS Compute Services*, in the *Collecting custom metrics on EC2* section. In that section, in Table 4.1, we also identified a list of metrics that can be captured by the Unified CloudWatch agent, among them `disk_used_percent`, which is stored as a metric in the CloudWatch metrics, as shown in the following screenshot:

	Instance Nam...	ImageId	InstanceId	InstanceType	device	fstype	path	Metric Name
✔	Monitoring_Instance	ami-00514a528eadb	i-0d7	t2.small	xvda1	ext4	/	disk_used_percent
	Monitoring_Instance	ami-00514a528eadb	i-0d7	t2.small	tmpfs	tmpfs	/dev/shm	disk_used_percent
	Monitoring_Instance	ami-00514a528eadb	i-0d7	t2.small	devtmpfs	devtmpfs	/dev	disk_used_percent
	Monitoring_Instance	ami-00514a528eadb	i-0d7.	t2.small	xvda1	ext4	/var/lib/docker	disk_used_percent

Figure 9.3 – EBS volume custom metrics in CloudWatch

Clicking on the first metric will generate a graph that shows the percentage of disk space that has been used on the EBS volume, as shown in the following screenshot:



Once the load balancer is created, click on it and scroll down to view the details. This contains more information about the load balancer. On the tabs that are displayed, click on the **Monitoring** tab to show graphs and metrics about the load balancer, as shown in the following screenshot:

Figure 10.5 – Application load balancer graphs

The preceding screenshot shows different graphs with different metric information. We shall be talking about a few of these metrics that are very key to help you understand what is happening with the load balancer. The first graph in the preceding screenshot has the title **Unhealthy Hosts** and tells us the number of hosts that the load balancer is not connected to. We explained that a load balancer usually had servers configured behind it. AWS load balancers have a feature that does a health check to ensure that the load balancer can connect to the servers configured behind it. If it cannot connect for any reason (could be that the backend server is down or the port mapping from the load balancer to the service is incorrect), the load balancer counts the number of hosts it cannot connect to as unhealthy hosts. The next metric shows **Healthy Hosts**, which means the servers that the load balancer can connect to. The **Requests** metric is a count of the number of requests that have hit the load balancer.

There is a set of HTTP status metrics such as **ELB 5XXs**, **HTTP 4XXs**, and **HTTP 2XXs**, which are all used to know the status of the backend application. The 5XX and 4XX stand for errors. Any these found as counts in the graph data in *Figure 10.5* means that the server behind the load balancer is reachable but the application in that server has errors.

There is also the metric called **Surge Queue Length**. This metric helps in understanding

Application observability with App Mesh and Cloud Map

The goal of **observability** is knowing what is going on internally. In a microservice application, there are various moving parts and components of your application, so it is quite important to understand the flow of traffic within the different microservices that are interconnected. This is one of the pillars of observability called tracing. As the application grows, the number of microservices in the system grows. If there is no service mesh in the system, it becomes near impossible to understand the inner workings of the system and how traffic flows from service to service.

> **Important Note**
>
> Microservices are an architecture used to develop applications, where the application is broken down into different pieces that work independently, run independently, are developed independently, and are also deployed independently. During runtime, these systems then connect and share information via different media, APIs, message queuing, networks, and so on.

This is where AWS App Mesh shines. With App Mesh, you can understand, monitor, and control communication within your microservices. It does this by installing a sidecar for every application deployed in your container orchestration service. With that, it is able to get network information of traffic flowing in and out of the running container. When every container in the cluster has a sidecar, it gives end-to-end visibility of the logical flow of the containers running within the cluster. This concept is also known as a **service mesh**.

> **Important Note**
>
> A service mesh is a network infrastructure layer that works side by side with applications running in a microservice architecture to help with a centralized method of administering, securing, tracing, and monitoring all network service communication between the different microservices.

So, we can rightfully say that AWS App Mesh is a managed service for service mesh, allowing deeper integration of the service to be used for both Amazon EKS and

AWS Cloud Map

From the name, you can tell that Cloud Map helps you create a map of the AWS services that you are using. Most times, these services are used as separate entities and there is usually no way of understanding the changes that occur in the services when scaling operations are made, or other types of service changes. With Cloud Map, the AWS service(s) you are using are registered to it, and it can help to map the services based on what is registered to it. Since the services are registered on Cloud Map, it can automatically discover them and knows the most up to date and healthy service resources.

When a Cloud Map namespace is created in Cloud Map (a namespace is the logical way that Cloud Map groups all the services of a single application), it creates a Route 53 private hosted zone.

There are three options Cloud Map uses to discover registered services/instances:

- API calls
- API calls and DNS queries in VPCs
- API calls and public DNS queries

Each of these options will require a service instance to be registered with its IP and by sending intermittent pings as health checks, Cloud Map is able to get a sense of the different services that your application is using. Cloud Map is mostly used for **containerized** microservices workloads.

> **Important Note**
>
> Containerized means that the application is designed to run in a Docker container. See more information about Docker containers in *Chapter 5, Setting Up Container Insights on Amazon CloudWatch*, in the *Container services in AWS* section.

In AWS, the two services for running containerized applications are Amazon ECS and Amazon EKS. They can be registered and kept up to date in AWS Cloud Map. When an application is deployed in an ECS or EKS cluster, a service is created, and that service is registered with Cloud Map. With that, Cloud Map is able to help you maintain the service

The following link is a simple Terraform script that can be edited to apply a Route 53 health check to any URL of an application that is deployed into the AWS environment (`https://github.com/PacktPublishing/Infrastructure-Monitoring-with-Amazon-CloudWatch/blob/main/Chapter10/terraform/route53healthchecker.tf`). To run the Terraform template, use the following instructions: `https://learn.hashicorp.com/tutorials/terraform/automate-terraform`.

Another way to automate monitoring configurations is using AWS **Simple Systems Manager** (**SSM**).

Audit monitoring configurations from time to time

In fast-moving systems, there are bound to be a lot of changes, configurations, and deployments happening at the same time, and quickly. It is important that services and applications are still monitored during these changes. There are situations where after a service is created or deleted, the monitoring setup does not follow suit. You could have an EC2 instance that has been terminated and the dashboard of the custom metrics from the EC2 instance is still up. This could cost some money, especially if it is not really in use.

Performing periodic audits on the monitoring and observability infrastructure is quite essential. It could also help spot possible issues. An example is logs might stop being sent to CloudWatch Logs for one reason or another. This could be an oversight and when those logs are needed at any time, you might discover that the instance has not been sending logs to CloudWatch. With periodic auditing, this can be quickly spotted and fixed.

Measuring workload performance

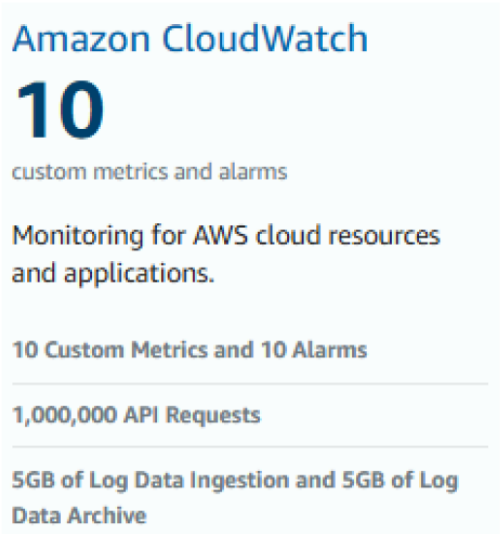
Logs and metrics are not only used to know when something has gone wrong but can also be used to envisage when a problem is about to occur. In *Chapter 1*, *Introduction to Monitoring*, we spoke about proactive and reactive monitoring. Being proactive is knowing when something will go wrong and acting on it before your attitude to the incident becomes reactive. Being reactive means that you do not know when something occurs but react when it has occurred. Measuring workload performance could mean understanding how much time it took for a request to move around the system from start

Ensuring important metrics go with alarms

There are lots of logs that can be collected for CloudWatch from various services. But it is important to focus on the most important logs and create metrics for them. The next thing that should follow the metrics is threshold alarms. Now, alarms on their own can be too much and when there are too many alarms, it reduces the importance of alarms, which should indicate important incidents and draw your attention to something important. Using composite alarms in AWS is a good way to reduce the number of notifications sent by multiple alarms that have been configured. Use the following link to see how to create composite alarms in Amazon CloudWatch: `https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Create_Composite_Alarm.html`.

An explanation of CloudWatch billing

With all the interesting features we have explained in this book, it is quite important to know that CloudWatch, just like every other AWS service, is not free of charge. Although, AWS has a free tier with the following features, which AWS promises will be free forever based on the free tier link (`https://aws.amazon.com/free`). The following screenshot shows what CloudWatch can give you for free every month:



From the screenshot, we can see that you can get 10 custom metrics and 10 alarms for free. Custom metrics are metrics that you create manually from the logs that are being sent to CloudWatch Logs. This means you can create 10 of those for free. Anything more than that you will be charged for. Now, these charges vary from metrics to dashboards, alarms, logs, events, contributor insights, to canaries. The prices for these components usually vary from region to region. For example, the cost of collecting logs and sending them to CloudWatch Logs to store costs $0.50 per GB in the Ohio region, and it costs $0.76 in the Tokyo region. These prices are not usually the same from region to region for the different components. But some components are the same across all the regions. Dashboards, for example, cost $3 per dashboard per month no matter the region you are in. For more details on CloudWatch pricing, check the CloudWatch pricing page. It gives more information based on custom metrics, container insights on both ECS and EKS, Lambda insights, and more. This is the link to the CloudWatch pricing page (`https://aws.amazon.com/cloudwatch/pricing/`). It is quite important to understand some of these metrics as they could incur a lot of charges if not well monitored and audited from time to time as many dashboards and metrics might be needed to understand our application.

Summarizing everything learned so far

In this section, we shall do a recap of everything we have learned in the last 10 chapters of this book, going from one chapter to another:

1. *Chapter 1*, *Introduction to Monitoring*, was an introduction to monitoring. We looked at monitoring from its first principles and explained how it has evolved over time and taken in software and cloud monitoring. Then we moved on to the types of monitoring and the importance of monitoring in a system. Then, we introduced Amazon CloudWatch and talked about how the well-architected AWS framework is helpful as a guide in performing monitoring more efficiently in AWS.
2. For *Chapter 2*, *CloudWatch Events and Alarms*, we started going deeper into more monitoring concepts, such as CloudWatch Events and Amazon EventBridge, comparing them and showing the differences. We also looked at the components of an event and moved straight on to look at what a CloudWatch alarm is and how

4. In *Chapter 4*, *Monitoring AWS Compute Services*, we started looking at monitoring in relation to different AWS services. Specifically, we focused on compute services such as EC2 and Elastic Beanstalk and practiced configuring logging to CloudWatch Logs using the CloudWatch Unified Agent.
5. In *Chapter 5*, *Setting Up Container Insights on Amazon CloudWatch*, we continued with compute services, but this time our focus was on containers. We introduced the concept of containers and talked about AWS services used for container orchestration. We then looked at Container Insights for monitoring on both ECS and EKS.
6. *Chapter 6*, *Performance Insights for Database Services*, took a turn into the monitoring of database services in AWS. We started by introducing database management, talking about the types of database technologies and different database technologies. We then looked at the different database technologies in AWS, such as RDS, Redshift, and how to configure monitoring and logging for these services.
7. *Chapter 7*, *Monitoring Serverless Applications*, was focused more on serverless applications. To help get a better understanding, we first explained the concept of serverless and the different serverless services in AWS. Then we went further to explain how monitoring works on each of the serverless services, such as Lambda, and introduced endpoint monitoring via CloudWatch Synthetics canaries.
8. In *Chapter 8*, *Using CloudWatch for Maintaining Highly Available Big Data Services*, we looked at monitoring big data services in AWS. We first identified the different operations in big data and the AWS services used for the different big data operations such as data analytics and data engineering. We then looked at how to monitor and track issues when running workloads in big data services.
9. *Chapter 9*, *Monitoring Storage Services with Amazon CloudWatch*, was focused on storage services and the monitoring of EBS volumes, EFS, and FSx storage services using Amazon CloudWatch to get deep insights into the behavior of the storage services.
10. *Chapter 10*, *Monitoring Network Services*, introduced more on monitoring network