**SECTION 14: AMAZON CLOUD WATCH METRICS**

* **159)Cloud watch metrics:-**

Cloud watch provides metrics for every services

* **What is metrics?**

Metrics is a variable monitor

* So we can create cloud watch dash boards if we want to access all cloud watch metrics at once

There are some types of cloud watch metrics like

Ec2 instacnes,auto scaling group(:if desired increased),service limits,billing etc

* **Cloud watch alarms:-**

Alarms are used to trigger notifications

Like if the cpu monitorization exceeded 90 % give alarm like that

**160)Cloud watch metrics and cloud watch alarms:-**

1. First we have to open cloud watch service
2. Then we have to create a alarm because tin cloud watch
3. Then we have to select the metric because metric will watch the services
4. So we can select the cpu utilization so that means if cpu utilization increased more than 90% then alarm will raise
5. Then static like how to calculate like static or maxi,mum or minimum etc
6. Threshold type we should select like static or what etc
7. CPU utilization greater or qal we should select that
8. Alarm state trigger
9. Then we should select an SNS topic or create new or use topic
10. We can also use autoscaling group,ec2 instances etc
11. Then we should go to ec2 instances and should attach to that
12. So thereeis another kind of alarm we can create only in one region so that is billing alarm it can be created in one region that is us-east-1a
13. **The billing alarm can be only accessible only in us east 1a(Exam point of view)**

**161) Amazon Cloud watch:-**

**Cloud watch logs:-**

* Cloud watch logs are nothing but whe you deploy a application you want to know what the process is happening , is the code every thing is right or else we have to watch step by step soo for that we use cloud watch logs
* Cloud watch logs store data in cloud watch groups.
* These can be collected from beanstalks (collection of logs in containers),ECS,AWS lambda,cloud watch log agents
* The cloud watch logs will not collect directly the adminsistrator should sert what to send to cloud watch log then the data will store there.
* It has retention if data lost we can recover
* **By default no ec2 instance will go to cloud watch**
* You need to drun a cloud watch agent in order push log files
* Make IAM Permissions are Correct
* These can connect both on premises and outside servcies

**162) Cloud watch logs hands on:-**

It seems like the text you've shared is explaining how to use **AWS CloudWatch Logs** to monitor and troubleshoot your **Lambda functions**. Let me break it down in a simpler way:

**Here's what’s happening in the text:**

1. **Log Groups**:
   * CloudWatch Logs organizes logs into **Log Groups**. In the example, there's a **Log Group** named AWS lambda, demo-lambda. This group was automatically created when you created and ran a Lambda function.
2. **Log Streams**:
   * Inside the Log Group, there are **Log Streams**. A Log Stream contains logs related to a specific instance or execution of the Lambda function. Each time the Lambda function is executed, a new Log Stream is created, and it contains the log data of that execution.
3. **Lambda Logs**:
   * When you run your Lambda function, it outputs logs. These logs can contain information like:
     + The **request ID** for the Lambda invocation.
     + Any **debug messages** you added in the code, like value one, value two, or any errors that occur during the function execution.
     + These logs appear in **CloudWatch Logs** under the corresponding Log Stream.
4. **Adding Logs to Lambda**:
   * You can **modify** your Lambda function to add extra log lines for debugging, like print an extra log line. Once you deploy the updated Lambda and run it again, the new log entries will appear in CloudWatch Logs.
5. **Handling Exceptions**:
   * If an error or exception occurs in your Lambda function (for example, by raising an exception with the raise exception line in the code), the logs will capture this error message, and you'll be able to see it in the **Log Stream** in CloudWatch Logs. This helps you troubleshoot issues.
6. **Practical Use**:
   * CloudWatch Logs is very helpful for **monitoring and troubleshooting** Lambda functions and other AWS services. You can look at the logs to understand what happened during execution, especially if there was an error.

**Key Points:**

* **CloudWatch Logs** is a service for collecting, monitoring, and analyzing logs from AWS services, like Lambda.
* Each **Lambda execution** generates logs that appear in **Log Streams** within a **Log Group**.
* You can add custom **log statements** in your Lambda code to output helpful information during execution.
* You can also capture **errors** (like exceptions) in the logs to troubleshoot your code.

**Example Walkthrough:**

1. **Run Lambda Function**:  
   You ran a Lambda function that automatically generated logs in CloudWatch Logs under a **Log Group**.
2. **Modify Lambda to Add Logs**:  
   You added a new log line (print an extra log line), deployed the Lambda, and tested it. The new log line now appears in a new **Log Stream** in CloudWatch Logs.
3. **Handle an Error**:  
   You intentionally introduced an error in your Lambda function (with a raised exception). After testing the Lambda again, the exception is logged in CloudWatch, and you can see what went wrong in the logs.

**Why is this useful?**

* **Troubleshooting**: If your Lambda function encounters an error, CloudWatch Logs lets you see the exact error and figure out why it failed.
* **Monitoring**: You can monitor your Lambda function's behavior, performance, and resource usage over time by analyzing the logs in CloudWatch.

**In Summary:**

This lesson is explaining how you can use **AWS CloudWatch Logs** to monitor and debug Lambda functions, by storing and viewing log entries generated by Lambda during execution. You can use these logs to troubleshoot, monitor, and analyze your functions, which is crucial for DevOps and software development tasks.

Let me know if you need any further clarification!

**163)Event bridge Over view:-**

No problem! Let me break down **Amazon EventBridge** (formerly **CloudWatch Events**) and explain it in simpler terms.

**What is Amazon EventBridge?**

Amazon EventBridge is a **serverless event bus** that lets you **react to events** that happen in your AWS environment, or from external sources. In simple words, it lets you trigger actions automatically based on specific events.

**EventBridge vs. CloudWatch Events**

* **EventBridge** is just the **new name** for **CloudWatch Events**. They are the same service, but AWS rebranded it and added more features to make it more powerful.

**What is an Event?**

An **event** is any change or occurrence in your system, like:

* **EC2 instance starts** or **stops**.
* **S3 bucket is created** or **updated**.
* **Lambda function is invoked**.
* **Security alert** (like a root user logging in).

**Use Cases for EventBridge**

1. **Scheduled Events (Cron Jobs)**:
   * **Example**: You want a script to run every hour (like a cron job).
     + **How**: With EventBridge, you can **create a rule** that triggers an event every hour. This event can then trigger a Lambda function to run the script.
     + **Result**: This is like having a **serverless cron job**, but without needing a physical server to handle the scheduling.
2. **Reacting to AWS Service Events**:
   * **Example**: You want to get an alert if someone logs in as the **root user** of your AWS account (this is a security risk because the root user should only be used rarely).
     + **How**: EventBridge can **capture this login event** and send it to an **SNS topic** (Simple Notification Service). SNS can then send an **email notification** to the security team.
     + **Result**: Whenever a root user signs in, the security team gets an **email alert**.
3. **Integrating AWS Services**:
   * EventBridge can react to events from various AWS services, such as:
     + EC2
     + S3
     + CodeBuild
     + IAM (for user activities like logins)
     + And many others…
   * For example, if an **S3 bucket is created**, you can trigger an action like notifying a team, or starting an EC2 instance.

**EventBridge Components**

1. **Event Bus**:
   * This is the central hub where events are collected.
   * There are **default event buses** (for AWS services) and **custom event buses** (where you can send your own events or events from external sources).
2. **Sources**:
   * Sources are the places where events come from.
   * For example, the source could be:
     + An **AWS service** like EC2, S3, Lambda.
     + A **third-party service** like Zendesk, Datadog (partners with AWS).
     + **Custom events** from your own applications.
3. **Targets**:
   * A **target** is what happens when the event is triggered.
   * For example, the target could be:
     + A **Lambda function** that runs a script.
     + An **SNS topic** that sends an email.
     + **SQS queue**, **Step Functions**, etc.

**Examples to Understand Better**

1. **Scheduled Event (Cron Job)**:
   * You want to back up data from an S3 bucket every day at 10 AM.
     + You set up a rule in EventBridge to trigger every day at 10 AM.
     + EventBridge triggers a Lambda function that copies the data from the S3 bucket to a backup location.
2. **Security Alert (Root User Login)**:
   * You create a rule in EventBridge to watch for any login events involving the **root user**.
   * When this event occurs, EventBridge sends the event to an SNS topic that sends an email to the security team, alerting them about the root user login.

**Extra Features in EventBridge**

1. **Partner Event Buses**:
   * EventBridge can also integrate with **third-party services** (like Zendesk or Datadog). These services can send events into your EventBridge event bus, and you can trigger actions based on those events as well.
2. **Custom Event Buses**:
   * You can create your own **custom event bus** to handle events from your own applications or other sources outside of AWS.
3. **Schema Registry**:
   * EventBridge also has a **Schema Registry** to manage and validate the data structure (schema) of events. This helps ensure the events are consistent and correctly formatted.
4. **Event Archiving and Replay**:
   * EventBridge allows you to **archive** events (store them for a period of time), and you can later **replay** those events if needed.

**Summary**

* **Amazon EventBridge** is a service that lets you react to events (changes or actions) in AWS or third-party systems.
* You can use it to **schedule tasks**, like running a script every hour, or **trigger actions** like sending an email when a specific event occurs (e.g., a security-related root user login).
* It can handle events from **AWS services**, **third-party sources**, or even your **own custom events**.
* **Targets** are where the event data goes to (like a Lambda function or SNS topic).
* It's a **serverless** way to handle automation and integrations in your AWS environment.

**165)Event bridge HandsOn:-**

Let's break down what was explained in the video, step by step, and simplify the concepts for you.

**1. Creating an EventBridge Rule to Invoke Lambda Every Hour:**

The goal here is to **schedule** a Lambda function to run every hour automatically using **Amazon EventBridge**.

**Steps to Create the Rule:**

* **Create an EventBridge Rule**:
  + The rule is named **InvokeLambdaEveryHour**.
  + This rule will trigger every hour and invoke a Lambda function.
  + To create the rule, go to the **EventBridge console**.
  + Choose **Create rule**, give it a name, and choose **Schedule** as the event type.
  + For the schedule, you select **Rate-based** to specify how often the rule will run. In this case, it's set to **every 1 hour**.
  + Then, you choose the **Lambda function** you want to invoke when the rule runs. You select an existing Lambda function, e.g., demo Lambda.
  + You can then configure additional settings like retry options and permissions (a role is automatically created for you).
  + Finally, you create the rule, and **EventBridge Scheduler** is responsible for making sure your Lambda function runs every hour.

**What did we do here?**

* We set up an **automatic scheduler** using EventBridge that triggers a Lambda function **every hour**. This is useful for tasks that need to run periodically without any manual intervention.

**2. Creating a Rule for Login Notifications:**

In this case, the goal is to create a rule that triggers a notification whenever **someone logs into your AWS account** using the **AWS Console**.

**Steps to Create the Rule:**

* **Create EventBridge Rule**:
  + The rule is named **send notification for login**.
  + This rule will be triggered by **AWS Console Sign-In Events**.
  + In the EventBridge rule creation, you choose an **Event Pattern**, and specifically, you're looking for the **Sign-in events** that happen when a user logs into the AWS Console.
  + **Target**: The event will trigger an **SNS Topic** (Simple Notification Service), like demo-ccp topic. SNS is a service that sends notifications, and in this case, it will send an **email notification** to let you know when someone logs into your AWS account.
  + Finally, you create the rule, and whenever someone logs in, the rule will trigger, and you'll get an email notification.

**What did we do here?**

* We created a rule that listens for **AWS Console login events** and sends **email notifications** to alert us when someone logs in.

**3. Creating a Rule for EC2 Instance Termination:**

This rule will notify you whenever an **EC2 instance** is **terminated** (stopped or deleted).

**Steps to Create the Rule:**

* **Create EventBridge Rule**:
  + The rule is named **EC2InstanceTerminateNotifications**.
  + This rule listens for an **EC2 instance state change** event, specifically when an instance is **terminated**.
  + The event pattern is filtered for instances where the state changes to **terminated** (indicating that the instance was deleted or stopped).
  + **Target**: Just like the previous rule, the target here is an **SNS Topic** (e.g., demo-ccp topic) to send a notification when the EC2 instance is terminated.
  + Once you create the rule, you will be notified whenever an EC2 instance is terminated.

**What did we do here?**

* We created a rule that listens for when an **EC2 instance is terminated** and sends a notification (email or other methods) when this happens.

**Key Terminologies Explained:**

* **EventBridge Rule**: A rule in Amazon EventBridge that defines how events should be handled. For example, it could define a schedule (like "run every hour"), or it could define an event pattern (like "when an EC2 instance is terminated").
* **Event Pattern**: A way to filter events that match certain criteria. For example, you can filter for events that occur when a user logs in or when an EC2 instance changes state.
* **Target**: The action or service that will be triggered when the event happens. For example, when someone logs in, the target might be an SNS topic that sends an email.
* **SNS Topic**: A Simple Notification Service topic. It allows you to send notifications (like emails or SMS) when certain events occur.
* **EventBridge Scheduler**: A service within EventBridge that allows you to create scheduled events. For example, you can schedule a Lambda function to run every hour, or on any specific schedule.

**Summary of What Was Done:**

1. **Scheduled Lambda Every Hour**:
   * Set up a rule to invoke a Lambda function every hour using EventBridge Scheduler. This allows you to automate periodic tasks without worrying about manual execution.
2. **Notification for Console Logins**:
   * Set up a rule that reacts to AWS Console sign-in events. Whenever someone logs in, an SNS notification is sent to notify the user (via email).
3. **EC2 Termination Notifications**:
   * Set up a rule to track when an EC2 instance is terminated. You receive a notification whenever an EC2 instance is stopped or deleted.

**Why would you use EventBridge?**

* **Automation**: Automatically trigger actions like running Lambda functions, sending emails, or managing resources.
* **Event-driven architecture**: React to events from AWS services (EC2, Lambda, etc.) or external services.
* **Scheduling**: Automate tasks on a set schedule (e.g., running scripts every hour).

EventBridge is powerful because it allows you to react to specific events in your AWS account and automate responses (notifications, function invocations, etc.).

**165)AWS Cloud Trail:-**

Let's break it down and simplify what **AWS CloudTrail** is, how it works, and its purpose.

**What is AWS CloudTrail?**

**AWS CloudTrail** is a **service** that helps you **track** and **log all API calls** and events happening in your AWS account. This means any action or request made on AWS (like using the console, SDK, CLI, etc.) is **recorded** by CloudTrail.

**Why is CloudTrail Important?**

* **Governance**: CloudTrail helps in managing your AWS environment by tracking who is doing what and when.
* **Compliance**: It keeps a record of all activities for compliance and security audits (for example, when someone logs into AWS and what they do).
* **Audit**: It allows you to check the history of API calls made within your AWS account (like which service was used, by whom, and when).

**What Types of Events Are Logged by CloudTrail?**

CloudTrail logs a variety of actions:

* **Console actions**: Actions taken via the AWS Management Console (UI).
* **SDK actions**: Actions taken via the AWS SDK (Software Development Kit).
* **CLI actions**: Actions taken via the AWS Command Line Interface.
* **Service activity**: Any API calls made within AWS services.

For example:

* If someone logs into the AWS Console, every action they take will be recorded in CloudTrail.
* If someone makes an API call (e.g., creating an EC2 instance), that will be logged.

**How CloudTrail Works**

1. **CloudTrail Records API Calls**:
   * Every action made using the AWS Console, AWS SDK, AWS CLI, or other AWS services is recorded by CloudTrail.
2. **Storage of Logs**:
   * These logs are then stored and can be sent to:
     + **CloudWatch Logs**: For real-time monitoring and analysis.
     + **Amazon S3**: For long-term storage and retention of logs.
3. **Multi-Region Monitoring**:
   * CloudTrail can be set up to monitor **all AWS regions** or just **specific regions**.
   * You can create a **CloudTrail trail** that records events from all regions or just one, based on your needs.

**Real-Life Example:**

Imagine someone deleted an S3 bucket in your AWS account. You would want to know:

* **Who deleted the bucket?**
* **When did they delete it?**
* **Which API call did they use to delete it?**

CloudTrail would record all of this information, allowing you to track and investigate the deletion event.

**Long-Term Retention and Auditing:**

* **CloudWatch Logs**: If you want to analyze and monitor logs in real-time, you can send CloudTrail logs to **CloudWatch Logs**.
* **Amazon S3**: For **long-term storage**, CloudTrail logs can be stored in an **S3 bucket**. This is useful if you need to keep logs for a long time (for example, for compliance purposes).

**Summary:**

1. **CloudTrail** records all the actions that happen in your AWS account, like API calls, SDK or CLI commands, and activities in the AWS Console.
2. You can **monitor** CloudTrail logs in **CloudWatch Logs** or store them for **long-term retention** in **Amazon S3**.
3. You can create a **CloudTrail trail** to record events from **all regions** or a **specific region**.
4. CloudTrail is essential for **auditing** and **tracking** who did what in your AWS account and is useful for investigating security or operational issues.

**Key Terminology:**

* **API Calls**: Requests made to AWS services. For example, creating an EC2 instance is an API call.
* **Trail**: A configuration in CloudTrail that records events from your account.
* **CloudWatch Logs**: A service where you can send logs to monitor and analyze them in real-time.
* **Amazon S3**: A storage service where CloudTrail logs can be stored for long-term retention.

**Example Use Case:**

* **Who deleted an S3 bucket?**
  + Check CloudTrail logs to see which **user** or **role** made the **API call** to delete the bucket and when it happened.

In essence, **CloudTrail** is a **logging and auditing tool** that tracks everything that happens in your AWS account, making it a crucial service for security, troubleshooting, and compliance purposes. Let me know if you want further clarification!

**167)AWS X-Ray:-**

We can see visual representation of our application

* Trouble shooting or understand dependeies,review request behaviour,find errors and expectations,identify users that are impacted

Let's break down **AWS X-Ray** in simple terms and explain the concepts so that it's easier to understand.

**What is AWS X-Ray?**

**AWS X-Ray** is a service that helps you **debug**, **analyze**, and **troubleshoot** your **distributed applications**. It's particularly useful when you have **microservices** (services that are spread across different systems and need to communicate with each other) because it provides a way to track requests and visualize how data flows through your entire system.

**The Problem It Solves:**

When you have a **monolithic application** (a single, large app), debugging can be relatively simple because everything is in one place. You can check logs and try to identify the problem.

However, when you move to **distributed systems** (where different services or applications communicate over networks using queues like **SQS** or topics like **SNS**), debugging becomes much more challenging. You can't see everything in one place. Different services might fail at different times, and tracing what caused the issue across the different services is difficult.

**How Does AWS X-Ray Help?**

AWS X-Ray helps you trace the flow of a request across multiple **distributed services**. It lets you see **exactly where the problems** are occurring and helps you **visualize** how different services are interacting.

Here’s how AWS X-Ray can be useful:

1. **Full Visibility of Requests**:
   * If a request passes through multiple services (for example, from a Lambda function to an SQS queue and then to an EC2 instance), X-Ray tracks it and gives you a **visual view** of its path.
2. **Performance Monitoring**:
   * X-Ray helps you spot **performance bottlenecks** in your system. For example, if one part of the application is slow, you can see which specific service is causing the delay.
3. **Troubleshooting**:
   * If something goes wrong (like an error or failure), X-Ray can help you **trace the issue** and understand exactly where and why it happened.
4. **Service Dependencies**:
   * X-Ray shows you the **relationships** between services. You can see how each service is dependent on others, and if one service is having problems, you can quickly understand what other services might be impacted.
5. **Service Level Agreement (SLA) Monitoring**:
   * X-Ray helps you track if your application is meeting its **performance standards**. You can see if your services are responding on time or if they are being **throttled** or **slowed down**.
6. **User Impact**:
   * You can identify **which users** are being affected by performance issues or failures in your system, helping you prioritize which issues to fix first.

**How Does AWS X-Ray Work?**

1. **Enable X-Ray**:
   * First, you need to enable X-Ray on your AWS services. This can be done in services like **AWS Lambda**, **EC2**, **API Gateway**, etc. Once enabled, these services will automatically start sending data to X-Ray.
2. **Trace Requests**:
   * X-Ray starts tracing requests that flow through your services. For example, when a user makes a request to your API, X-Ray tracks this request as it moves through different services (like Lambda, SQS, etc.).
3. **Generate Visual Representation**:
   * X-Ray gives you a **visual graph** (called a **service map**) that shows the relationship between all your services and how they are performing. This allows you to pinpoint any issues quickly.
4. **Analyze Data**:
   * You can use the X-Ray console to analyze the traced data. You can filter by **request ID**, **error rates**, **latency**, etc., to focus on the issues you're investigating.

**Key Terminologies in AWS X-Ray:**

* **Trace**: A trace represents a single request as it travels through multiple services. It captures details about what happened at each step of the request's journey.
* **Segment**: A segment represents a unit of work in a service (e.g., an HTTP request, a database query). Each service involved in the trace creates segments.
* **Service Map**: A visual representation of your architecture showing how different services are connected and how requests flow through them. It also shows where failures or performance issues occur.
* **Sampling**: AWS X-Ray does not need to trace every single request. It can sample a percentage of requests (e.g., only trace 10% of the traffic) to avoid performance overhead.
* **Faults and Errors**: X-Ray helps you track **faults** (requests that failed) and **errors** (e.g., exceptions thrown in the code).

**Use Cases for AWS X-Ray:**

1. **Microservice Debugging**:
   * If you have multiple microservices (e.g., a service that calls another service via SQS), X-Ray helps you track the journey of requests and debug problems like slow response times or failures.
2. **Performance Optimization**:
   * By tracking requests and services, you can see where the system is slowing down and identify which specific service is causing delays. This can help you improve the performance of your application.
3. **Tracking Errors and Failures**:
   * If an error occurs in one service, X-Ray helps you pinpoint the error and identify what caused it, helping you fix issues faster.
4. **Service-Level Monitoring**:
   * You can use X-Ray to monitor service performance and ensure your services meet the performance expectations set in your **Service Level Agreements (SLAs)**.

**Summary:**

* **AWS X-Ray** is a tool for debugging and analyzing distributed applications, providing you with a **visual representation** of how services interact and where issues are occurring.
* It helps track **requests**, pinpoint performance **bottlenecks**, troubleshoot **errors**, and optimize your microservices.
* By tracing the requests across multiple services, X-Ray provides **full visibility** and helps you understand where your system is failing or slowing down.

**Example Scenario:**

Imagine you have an application that uses multiple microservices:

1. A **web frontend** that talks to an **API Gateway**.
2. The API Gateway triggers a **Lambda function**.
3. The Lambda function writes data to an **SQS queue**, which is processed by an **EC2 instance**.

With X-Ray enabled:

* When a user makes a request to your application, X-Ray will trace that request across all these services.
* You’ll be able to see if one of these services is causing delays or errors.
* If the Lambda function is slow or the EC2 instance is failing, X-Ray will show you exactly where the problem is happening.

By using **AWS X-Ray**, you get a clear, visual overview of how your services are performing and where problems might lie, which is especially useful for troubleshooting complex, distributed systems.

Let me know if you'd like more details on any part of this!

**Amazon Code Guru:**

An amazon code guru will do two things

1. **Automated code review**: when develeoprt pshes cod the anpther developer mjust review
2. **Application permoforamnce recommendation**: So when developers push our code,there is usually another developer that does a code review. And then pushes to production after that we should monitor it will be also done.

Functionalities:

**Code guru reviewer :will review the code ang give necessary suggestions**

**Code guru profiler:-to give you visibility or recommendationsabout your application performance**

during runtimes or in production.

Let's break down **Amazon CodeGuru** and how it works in simpler terms.

**Overview of Amazon CodeGuru:**

Amazon **CodeGuru** is an **AI-powered** service that helps developers improve their code by automating code reviews and providing performance optimization suggestions. It works in two main areas:

1. **CodeGuru Reviewer**: Automated code review tool.
2. **CodeGuru Profiler**: Application performance monitoring tool for runtime analysis.

**1. CodeGuru Reviewer (Automated Code Reviews):**

**What is it?**

* CodeGuru Reviewer is a service that automatically reviews the **source code** you push to repositories like **GitHub**, **CodeCommit**, or **Bitbucket**.
* It looks for **bugs**, **security vulnerabilities**, and **code inefficiencies** in your code, giving suggestions for improvements.

**How does it work?**

* When you commit your code, CodeGuru Reviewer runs **static code analysis**, meaning it inspects the code without running it.
* It uses **machine learning** (trained with tons of code samples from both open-source projects and Amazon’s codebase) to identify issues that might be hard to detect manually.
* It can find:
  + **Bugs** that might cause errors in your application.
  + **Memory leaks** that can slow down your app.
  + **Security vulnerabilities** like unprotected user inputs.
  + **Coding best practices**, helping you write cleaner, more efficient code.

**Supported Languages:**

* CodeGuru currently supports **Java** and **Python**.
* It can integrate with platforms like **GitHub**, **Bitbucket**, and **AWS CodeCommit**.

**Why is it useful?**

* It helps you **find issues early** before your code goes live.
* Saves you time by detecting problems automatically (faster than waiting for a human reviewer).
* Increases **code quality** and reduces human error in reviews.

**2. CodeGuru Profiler (Performance Recommendations):**

**What is it?**

* **CodeGuru Profiler** monitors the performance of your **running application** (in production or in pre-production).
* It helps you identify performance bottlenecks in your code, like **CPU usage** or **memory leaks**, which can be costly in terms of resources.

**How does it work?**

* **Pre-production**: It helps you optimize the code before you deploy it. For example, it can tell you which parts of the code are consuming a lot of resources, allowing you to optimize it early.
* **Production**: Once your app is running in production, CodeGuru Profiler continuously monitors your app and helps you understand how resources are being used. It will recommend improvements to help reduce resource usage and lower costs.

**Key Features of CodeGuru Profiler:**

* **CPU utilization analysis**: Identifies which parts of the code are using too much CPU.
* **Memory usage**: Helps you find parts of your app that are consuming too much memory (like large objects or memory leaks).
* **Cost reduction**: If your app is consuming excessive compute resources, CodeGuru Profiler will help you optimize it, reducing your AWS costs.
* **Anomaly detection**: It detects when your app is behaving abnormally, helping you catch issues early.
* **Heap summaries**: Helps identify what data objects in memory are using the most space.

**Why is it useful?**

* Helps you improve the **efficiency** of your application.
* Helps you **optimize your costs**, by detecting parts of your application that consume too many resources.
* Allows you to **monitor in real-time** once your app is live.

**How CodeGuru Fits Together:**

| **Service** | **CodeGuru Reviewer** | **CodeGuru Profiler** |
| --- | --- | --- |
| **What it does** | Automatically reviews your code for bugs, security issues, and best practices. | Monitors your application’s performance and suggests optimizations. |
| **When to use** | During development when you push new code or make changes. | When your app is running (in pre-production or production). |
| **Focus area** | Code quality (bugs, security, best practices). | Performance (CPU, memory, cost optimization). |
| **Use case example** | Reviewing code for security vulnerabilities or fixing memory leaks before deployment. | Monitoring and optimizing performance of a live app in production. |

**How to Use CodeGuru:**

1. **Set up CodeGuru Reviewer:**
   * Link CodeGuru with your **code repository** (e.g., GitHub, CodeCommit).
   * When you commit your code, CodeGuru will automatically analyze it and give you recommendations.
   * It will highlight lines of code that might have issues and suggest improvements.
2. **Set up CodeGuru Profiler:**
   * Integrate CodeGuru Profiler into your application.
   * It will monitor your application’s performance in real-time.
   * After deployment, it will continue analyzing how resources like CPU and memory are being used, offering suggestions on improving performance and reducing costs.

**Key Terminologies:**

* **Static Code Analysis**: A process of analyzing the source code without running the application.
* **Machine Learning**: CodeGuru uses ML to learn from thousands of open-source and internal repositories to identify potential issues in your code.
* **Performance Bottlenecks**: Sections of code that slow down the application or consume excessive resources (CPU, memory, etc.).
* **Heap Summaries**: Data on what objects are using the most memory in your application.

**Summary:**

* **CodeGuru Reviewer** helps you with **automated code reviews** before deployment, looking for bugs, security issues, and inefficiencies in your code.
* **CodeGuru Profiler** helps you monitor your app’s **performance** and **resource usage** once it's live, identifying areas where you can improve efficiency and reduce costs.

Both services help you improve **code quality** and **performance** at different stages of the development lifecycle.

**170)AWS health dash board:-**

Let’s break down what was explained in the lecture about the **AWS Health Dashboard** in a simple and step-by-step way.

**AWS Health Dashboard Overview:**

The AWS Health Dashboard is a service that provides you with information about the health of AWS services, specifically focusing on any events or issues that might affect your account. It helps you stay informed about service disruptions, maintenance, or performance problems that could impact the resources or applications running in your AWS account.

**Two Main Parts of the AWS Health Dashboard:**

1. **Service History**:
   * This part provides general health information for **all AWS services** across all regions.
   * It shows you **the status** of services (e.g., EC2, S3) in all regions and displays any **issues** that may have occurred (outages, performance problems, etc.) over time.
   * You can see which **AWS services were affected** and when these issues happened.
   * It also gives you an option to subscribe to an **RSS feed** to get updates about these health events.
   * **Example**: If AWS has an issue in a specific region, like US East (Ohio), it will show up in Service History, and you can track the status over time.
2. **Account Health Dashboard**:
   * This is **specific to your AWS account**.
   * It provides **alerts and updates** for events that **directly impact your resources and services**. For example, if your EC2 instances or S3 buckets are affected by an issue, you’ll get information about it here.
   * The Account Health Dashboard shows you **relevant, timely information** and provides **notifications** for:
     + **Scheduled maintenance** (e.g., planned updates to a service).
     + **Performance issues** (e.g., EC2 instance failures, S3 latency problems).
     + **Service outages** that are directly impacting the resources in your account.
   * You can view past health events (like an EC2 issue that happened last week) and get **remediation guidance** on how to fix or mitigate problems.
   * It aggregates data for the entire **AWS Organization** (if you have multiple AWS accounts under one organization).
   * You get **alerts and proactive notifications** to help you stay ahead of potential issues that may arise.

**Important Terminologies:**

* **AWS Service Health Dashboard**: This used to be the name for the general status of AWS services (now part of the Service History section in AWS Health).
* **AWS Personal Health Dashboard (PHD)**: This was the previous name for the **Account Health Dashboard**, which now provides detailed alerts and updates specific to your AWS account.
* **Scheduled Maintenance**: These are events planned by AWS, such as updates or maintenance tasks that might impact the availability of a service.
* **Remediation Guidance**: AWS provides instructions or suggestions on what actions to take if your resources are affected by an issue.
* **RSS Feed**: A way to receive updates automatically. You can subscribe to an RSS feed to receive updates on the general health status of AWS services.
* **Aggregated Data for Organization**: If you have multiple AWS accounts under one organization, you can view health events and information across all of them in one place.

**How to Access AWS Health Dashboard:**

1. Go to the **top right corner of the AWS Management Console** (next to the notification bell).
2. Click to access the AWS Health Dashboard.

**Use Cases:**

* **Example 1**: If an EC2 instance in your region (say US East 2) goes down, the **Account Health Dashboard** will show an alert about this issue, with additional details on how this might affect your application.
* **Example 2**: If AWS is doing scheduled maintenance on a service like S3 or EC2, you’ll receive **proactive notifications** telling you when the maintenance will happen and if it will impact your resources.

**Summary:**

* **Service History** shows the general health of AWS services and any past issues.
* **Personal Health Dashboard** is personalized to your account, showing real-time alerts for issues affecting your specific AWS resources.
* It helps you stay informed about any events or outages that might disrupt your services, and provides guidance on how to handle them.

This tool is especially useful for staying on top of AWS-related issues and for proactive management of the health of your AWS environment.

**Summary:-**

Let's break down and simplify everything you've learned in the monitoring section:

**1. AWS CloudWatch**

**CloudWatch** helps monitor and manage AWS resources and applications in real-time.

* **CloudWatch Metrics**:
  + Monitors the performance of AWS services (e.g., EC2, S3).
  + It tracks things like CPU usage, memory, disk space, and network traffic.
  + **Example**: Monitoring how much data is being transferred by an S3 bucket.
* **CloudWatch Alarms**:
  + Sends **notifications** when a metric goes outside a certain threshold.
  + You can also trigger actions like rebooting an EC2 instance when a certain metric is reached.
  + **Example**: If an EC2 instance CPU usage goes over 90%, an alarm can trigger an email via SNS or automatically reboot the instance.
* **CloudWatch Logs**:
  + Collects and stores logs from different sources like EC2, Lambda, or servers.
  + Logs help in debugging, troubleshooting, and tracking errors in applications.
  + **Example**: Collecting logs from Lambda functions to troubleshoot performance or errors.
* **CloudWatch Events (EventBridge)**:
  + Allows you to react to events (e.g., when a new EC2 instance is launched) or schedule actions.
  + **Example**: Automatically triggering a Lambda function when an S3 object is uploaded.

**2. AWS CloudTrail**

* **CloudTrail** logs **API calls** made within your AWS account.
* It tracks actions like who created an EC2 instance, deleted a bucket, or changed permissions.
* **Example**: If you want to see who deleted an S3 bucket or modified IAM roles, CloudTrail will show you that information.
* **CloudTrail Insights**:
  + It automatically analyzes CloudTrail logs to help identify unusual API activity, helping with **security audits**.
  + **Example**: Detecting unusual login activity or API calls that may indicate a security breach.

**3. Amazon X-Ray**

* **X-Ray** helps trace requests through **distributed applications**.
* It is useful for **root cause analysis** of performance issues or errors when your application consists of multiple services (e.g., Lambda, EC2, SQS).
* **Example**: If a user experiences slow load times, X-Ray helps identify which service is causing the delay.

**4. AWS Health Dashboard**

* **Service Health Dashboard**:
  + Shows the overall health of AWS services across all regions.
  + **Example**: You can check if there’s an ongoing issue with EC2 in the US-East-1 region.
* **Account Health Dashboard**:
  + Focuses on services **specifically used** in your account and shows any issues or maintenance that affect your infrastructure.
  + **Example**: If you’re using EC2 in US-East-1, this dashboard will notify you if there’s a problem with your EC2 instances in that region.

**5. AWS CodeGuru**

* **CodeGuru Reviewer**:
  + Uses machine learning to review your **code** automatically.
  + It can identify bugs, memory leaks, and security issues before you deploy your code to production.
  + **Example**: When you push new code to a repository, CodeGuru checks for coding errors or inefficiencies.
* **CodeGuru Profiler**:
  + Analyzes the performance of your application in **production**.
  + Helps identify performance bottlenecks and cost optimizations, such as reducing CPU usage.
  + **Example**: After deploying an application, CodeGuru can suggest optimizations to reduce the amount of CPU time used.

**Summary of What You Can Do With These Services:**

1. **Monitor your AWS services** in real-time (e.g., CloudWatch Metrics, CloudWatch Logs).
2. **Set up alerts and automated actions** based on specific metrics (e.g., CloudWatch Alarms).
3. **Track and audit API calls** and **get logs** for your AWS resources (CloudTrail).
4. **Analyze the health** of your AWS resources and services (AWS Health Dashboard).
5. **Detect errors and performance issues** in your code and applications, both pre-deployment and in production (CodeGuru and X-Ray).

**Where and How You Use These Tools:**

* **CloudWatch**: Use it to monitor **performance** and **logs** in real-time.
* **CloudTrail**: Use it for **security auditing** and tracking **API calls**.
* **X-Ray**: Use it for **debugging and tracing** requests through distributed applications.
* **Health Dashboard**: Use it to check the status of **AWS services** or **your account-specific resources**.
* **CodeGuru**: Use it to **automate code reviews** and improve **application performance**.

These services together form a robust monitoring and troubleshooting ecosystem, ensuring that you can track, debug, and optimize your applications and AWS resources efficiently.