**Cloud Watch & Auto Scaling**

**Cloud Watch**

Amazon CloudWatch is a monitoring and observability service provided by AWS. It lets you collect, access, and analyze operational data in the form of logs, metrics, and events from AWS services, applications, and on-premises servers.

In Amazon CloudWatch, monitoring for AWS services like EC2 comes in two levels:

1. Basic Monitoring

* Default and Free
* Data is collected at 5-minute intervals.
* Applies to services like EC2 when no custom configuration is applied.
* Limited granularity: not suitable for high-frequency monitoring or alerting.
* Good for non-critical workloads.

Example:  
For an EC2 instance, CPU Utilization is reported every 5 minutes.

2. Detailed Monitoring

* Paid (Low cost per metric)
* Data is collected at 1-minute intervals.
* Must be enabled manually (e.g., check the box in the EC2 settings or use AWS CLI).
* Allows for more responsive alarms, dashboards, and auto scaling.

Example:  
You can set alarms that react within 1–2 minutes instead of waiting up to 5 minutes.

Note :

If you don’t want to pay for **AWS CloudWatch Detailed Monitoring**, you can use third-party tools like **Prometheus** and **Grafana** for monitoring.

* When an **application or project is fully hosted on the AWS cloud**, teams often prefer using **CloudWatch** because it is tightly integrated with AWS services, easy to set up, and managed by AWS.
* When an **application's workload is distributed across multiple platforms** (e.g., AWS, on-premises, other cloud providers), tools like **Prometheus** and **Grafana** are often used.
  + **Prometheus** is responsible for collecting and storing metrics.
  + **Grafana** is used to visualize the data collected by Prometheus.

These tools offer flexibility, open-source customization, and **no additional AWS monitoring cost**, making them ideal for hybrid or multi-cloud environments.

**📊 Monitoring Tools Comparison**

| **Feature / Tool** | **AWS CloudWatch** | **Prometheus** | **Grafana** |
| --- | --- | --- | --- |
| **Type** | Managed AWS Service | Open-source metrics collection system | Open-source visualization & dashboard tool |
| **Setup Required** | Minimal (fully integrated with AWS) | Requires installation & configuration | Requires installation; connects to data sources |
| **Cost** | Basic free, detailed is paid (per metric/log) | Free (self-hosted); resource cost applies | Free (self-hosted); resource cost applies |
| **Data Collection** | AWS metrics, custom metrics | Time-series metrics (via exporters or scraping) | Pulls data from Prometheus and other sources |
| **Visualization** | CloudWatch Dashboards | Basic built-in UI | Advanced dashboards, highly customizable |
| **Alerting** | Integrated with CloudWatch Alarms, SNS | Alert manager for notifications | Grafana Alerting supports multi-source alerts |
| **Use Case** | AWS-only or full AWS workloads | Multi-cloud, hybrid, containerized environments | Unified dashboards from multiple tools |
| **Best For** | AWS-integrated, managed services | Custom environments, Kubernetes, microservices | Visualizing metrics from multiple backends |
| **Scalability** | High (AWS-managed) | Requires own scaling management | Scales with Prometheus or other sources |
| **Logging Support** | Yes (CloudWatch Logs) | No built-in logs, only metrics | No logging; metrics visualization only |

**Example Use Cases**

* **CloudWatch**: Ideal for monitoring EC2, Lambda, RDS, S3, etc., within AWS.
* **Prometheus + Grafana**: Ideal for containerized apps (like Kubernetes), hybrid cloud, or when you want open-source control and no AWS billing.

**🚀 Auto Scaling in Cloud Monitoring and Infrastructure**

Auto Scaling refers to the automatic adjustment of compute resources (like EC2 instances, containers, etc.) based on traffic, load, or performance metrics.

Auto Scaling in AWS with CloudWatch

CloudWatch integrates directly with Auto Scaling Groups (ASGs) to automatically scale EC2 instances.

How It Works:

1. Metric-based Scaling:
   * CloudWatch monitors metrics like CPU Utilization, NetworkIn, etc.
   * If thresholds are crossed (e.g., CPU > 80%), CloudWatch Alarm triggers scaling.
2. Target Tracking Policy:
   * Keeps a specific metric (like average CPU) at a target value.
   * Example: Maintain average CPU at 50%.
3. Scheduled Scaling:
   * Scale resources based on known patterns (e.g., weekdays at 9 AM).
4. Step Scaling:
   * Adds/removes instances based on how much the metric deviates from the threshold.

**🔹 Vertical Scaling (Scale Up / Scale Down)**

* Vertical scaling means increasing the capacity of a single machine. This could mean adding more CPU, memory (RAM), or storage to an existing server.
* **Example**: In AWS, upgrading an EC2 instance from t2.micro to m5.large is vertical scaling.
* **Benefits**:
  + Simpler to implement—no change in architecture is needed.
  + Suitable for monolithic applications or databases where scaling out is difficult.
* **Limitations**:
  + There is a hardware limit - you can only scale up to a certain point.
  + Usually requires restarting the server, which can cause downtime.
  + Becomes expensive at higher tiers.

**🔸 Horizontal Scaling (Scale Out / Scale In)**

* Horizontal scaling involves adding more servers or instances to handle increased load. Instead of upgrading a single machine, you add more of them.
* **Example**: In AWS, setting up an Auto Scaling Group that automatically adds EC2 instances when traffic spikes is horizontal scaling.
* **Benefits**:
  + No downtime - new instances are added automatically.
  + Highly scalable and fault-tolerant. If one instance fails, others continue running.
  + Ideal for cloud-native applications, microservices, and distributed systems.
* **Limitations**:
  + More complex to set up—requires load balancers and stateless application design.
  + Requires coordination among instances (e.g., shared storage or session management).

In short: vertical scaling makes a single machine stronger, while horizontal scaling adds more machines to share the load.

EC2 instance types

1. Compute Optimized

* Use Case: Best for compute-intensive workloads that require high-performance processors.
* Example Applications: High-performance web servers, scientific modeling, batch processing, gaming servers.
* Instance Family: C series (e.g., c5, c6g)
* Key Feature: High ratio of vCPU to memory.

1. Storage Optimized

* Use Case: Ideal for workloads that require high, fast, and consistent IOPS or large amounts of data to be read/write locally.
* Example Applications: Data warehousing, Hadoop, log processing, NoSQL databases like Cassandra.
* Instance Family: I, D, and H series (e.g., i3, d2)
* Key Feature: Local NVMe or SSD-based storage with high throughput.

1. Memory Optimized

* Use Case: Designed for workloads that process large data sets in memory.
* Example Applications: In-memory databases (e.g., Redis, Memcached), real-time big data processing, high-performance SAP workloads.
* Instance Family: R, X, z series (e.g., r5, x1e)
* Key Feature: High memory-to-vCPU ratio.

1. General Purpose

* Use Case: Balanced compute, memory, and networking resources—suitable for a wide variety of workloads.
* Example Applications: Web servers, dev/test environments, small databases, application servers.
* Instance Family: T, M series (e.g., t3, m5)
* Key Feature: Balanced resources for typical use cases.

**Practical: Vertical Scaling on EC2 (Scale Up/Down)**

**Goal:**

Upgrade or downgrade your EC2 instance to a different instance type (e.g., from t2.micro to t3.medium) to increase or decrease its compute power.

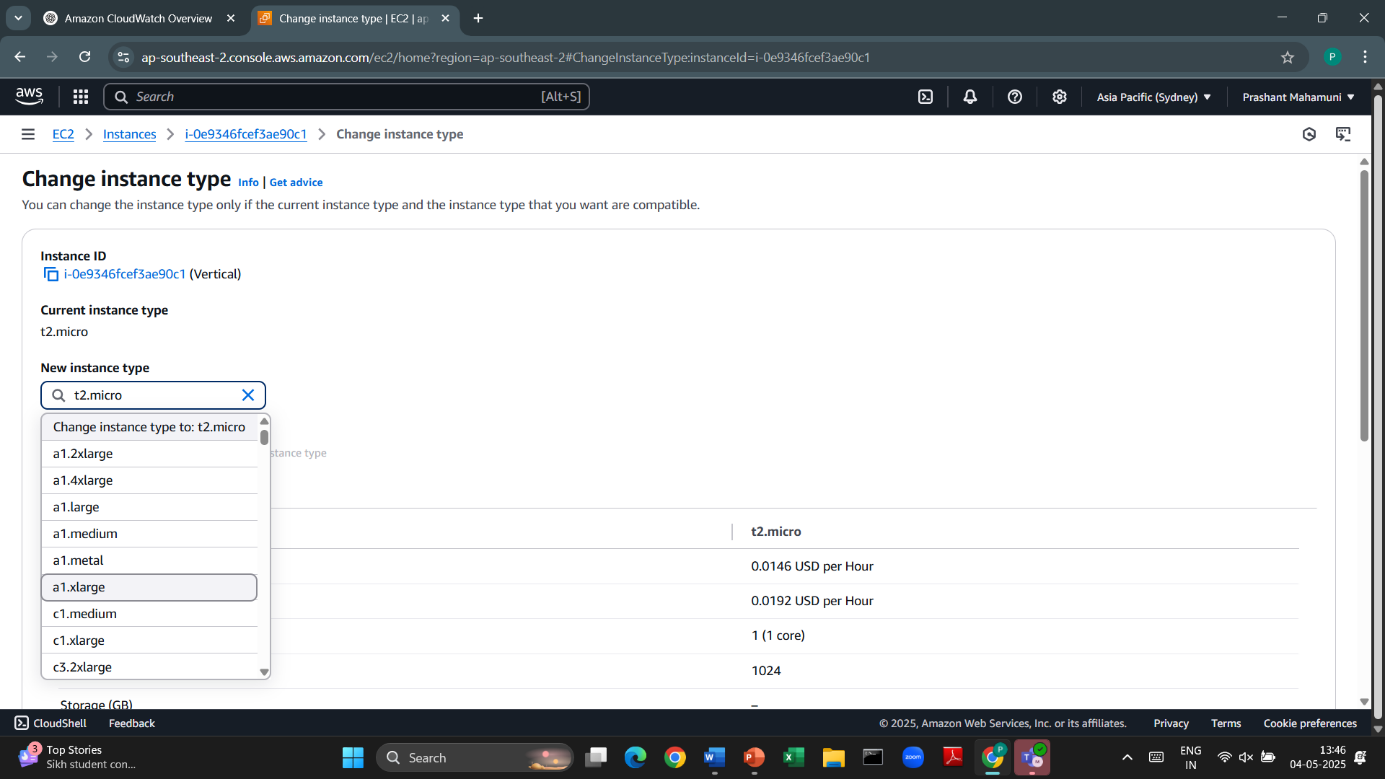
**Steps to Perform Vertical Scaling:**

**1. Stop the EC2 Instance**

* You cannot change the instance type while it is running.
* Go to the **EC2 Console** > **Instances**.
* Select your instance, then click **Instance State → Stop instance**.
  + Make sure your application can tolerate downtime.

**2. Change the Instance Type**

* Once the instance is stopped:
  + Click on **Actions → Instance Settings → Change Instance Type**.
* Choose a new instance type (e.g., t3.medium, m5.large).
  + Select based on your new **CPU, memory, or network** needs.



**3. Start the Instance**

* After changing the type, go back to **Instance State → Start instance**.
* Your instance will now run with the new configuration.

**4. Verify the Change**

* SSH into your EC2 instance and run:

$lscpu # to check CPU cores

$free -h # to check memory

* Confirm your system has scaled up (or down) as expected.

Note:

* Your data is safe if it’s stored on **EBS**. Vertical scaling does **not** affect your attached EBS volumes.
* Use **Elastic IP** so your public IP doesn't change after stopping and starting the instance.

**Practical: Horizontal Scaling with Auto Scaling Group (ASG)**

We are going to scale our EC2 instances when the CPU usage reaches 70%. This is how the process works:

1. CloudWatch continuously monitors the CPU usage of our EC2 instances.
2. When the CPU usage exceeds 70%, CloudWatch Alarm is triggered.
3. The alarm is connected to Amazon SNS (Simple Notification Service), which:
   * Sends an email alert to notify us about high usage.
   * Notifies Auto Scaling to take action.
4. Based on this alert, Auto Scaling Group automatically launches new EC2 instances to distribute the load.
5. As a result, the system handles the increased traffic by scaling horizontally (adding more instances).

Auto Scaling Group Configuration

When creating or configuring an Auto Scaling Group, you set three key parameters:

1️) Minimum Capacity

* This is the minimum number of EC2 instances that should always be running.
* Even if there's no traffic, the ASG will never scale below this number.
* Example: If Min = 2, there will always be at least 2 instances running.

2️) Desired Capacity

* This is the initial or target number of instances that the ASG will try to maintain.
* If instances fail or terminate, the ASG will automatically launch replacements to meet this desired count.
* This is also the value ASG starts with when first launched.
* Example: If Desired = 3, the group will try to keep 3 instances running.

3️) Maximum Capacity

* This is the upper limit of how many EC2 instances the ASG can launch.
* Even during peak load, the ASG will never scale out beyond this number.
* Example: If Max = 5, the group will not create more than 5 instances, no matter the load.

Example:

Let’s say you configure:

* Min = 2
* Desired = 3
* Max = 5

At launch, the ASG starts with 3 instances. If traffic increases, it can scale up to 5. If traffic drops, it can scale down, but never below 2 instances.

Note:   
When the server load suddenly increases and then quickly drops, it is called a spike. In such cases, we do not need to increase the number of servers, because the load is temporary.

However, if the high load continues for at least 5 minutes, it indicates sustained traffic, and in that case, Auto Scaling should increase the number of servers to handle the load effectively.

Summary:

* Spikes (short, sudden load) → No scaling needed.
* Sustained load (e.g., CPU > 70% for 5 minutes) → Trigger CloudWatch Alarm → Auto Scaling increases servers.

Components Needed for Auto Scaling Setup in AWS

To implement Auto Scaling, we need the following three main components, each with their specific parts:

1️) CloudWatch

Used for monitoring, alarming, and visualizing metrics.

* Alarm → Triggers scaling actions when a threshold (e.g., CPU > 70%) is breached.
* Logs → Stores application and system logs (optional for deeper analysis).
* Dashboards → Visual display of EC2 metrics (e.g., CPU, memory, network).

2️) SNS (Simple Notification Service)

Used for notifications and communication between services.

* SNS Topic → Acts as a message bus between CloudWatch and Auto Scaling.
* Email → Sends alerts to administrators or users.
* Push Notification → Sends messages to mobile or SMS (optional).

3️) Auto Scaling

Manages launching and terminating EC2 instances based on demand.

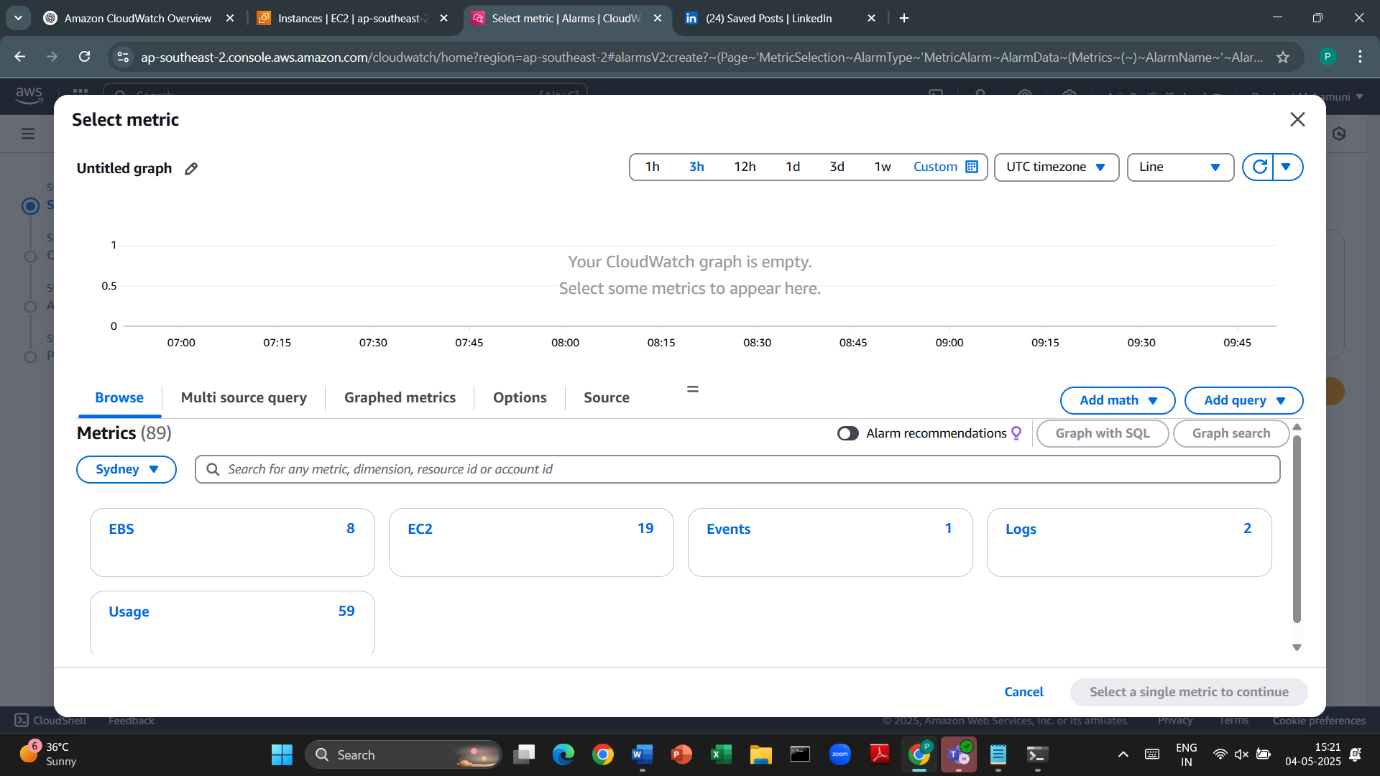
* Launch Template
  + AMI → Amazon Machine Image (OS + preinstalled software)
  + Key Pair → For SSH access
  + Security Group → Controls inbound/outbound traffic
  + EBS Size → Root volume size and storage type
* Auto Scaling Group (ASG)
  + Defines min, max, and desired number of instances
  + Tied to specific subnets and optionally a Load Balancer
* Auto Scaling Policies
  + Define when and how to scale (e.g., scale out if CPU > 70% for 5 mins)
  + Can be based on CloudWatch Alarms or scheduled actions

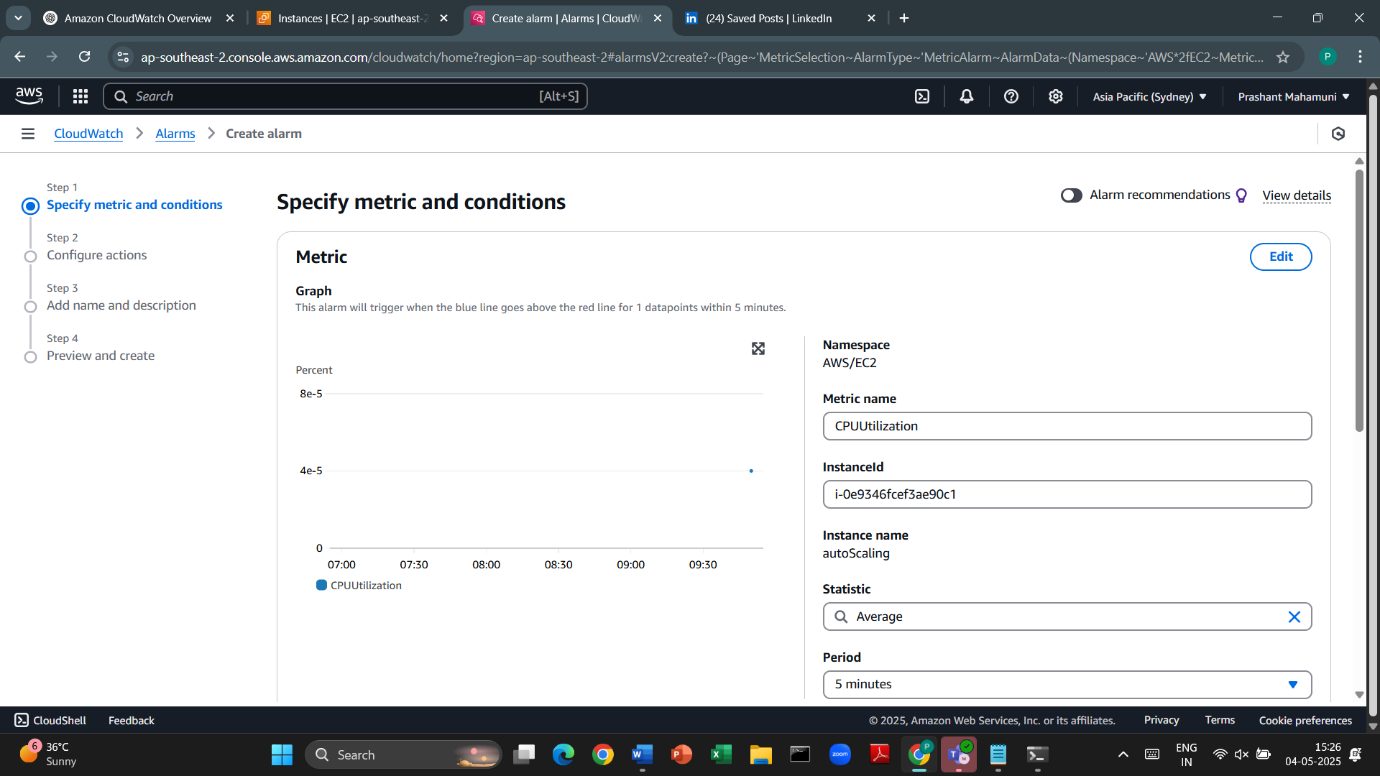
Practical: CloudWatch Alarm with EC2 (Nginx + CPU Usage)

**Goal:**

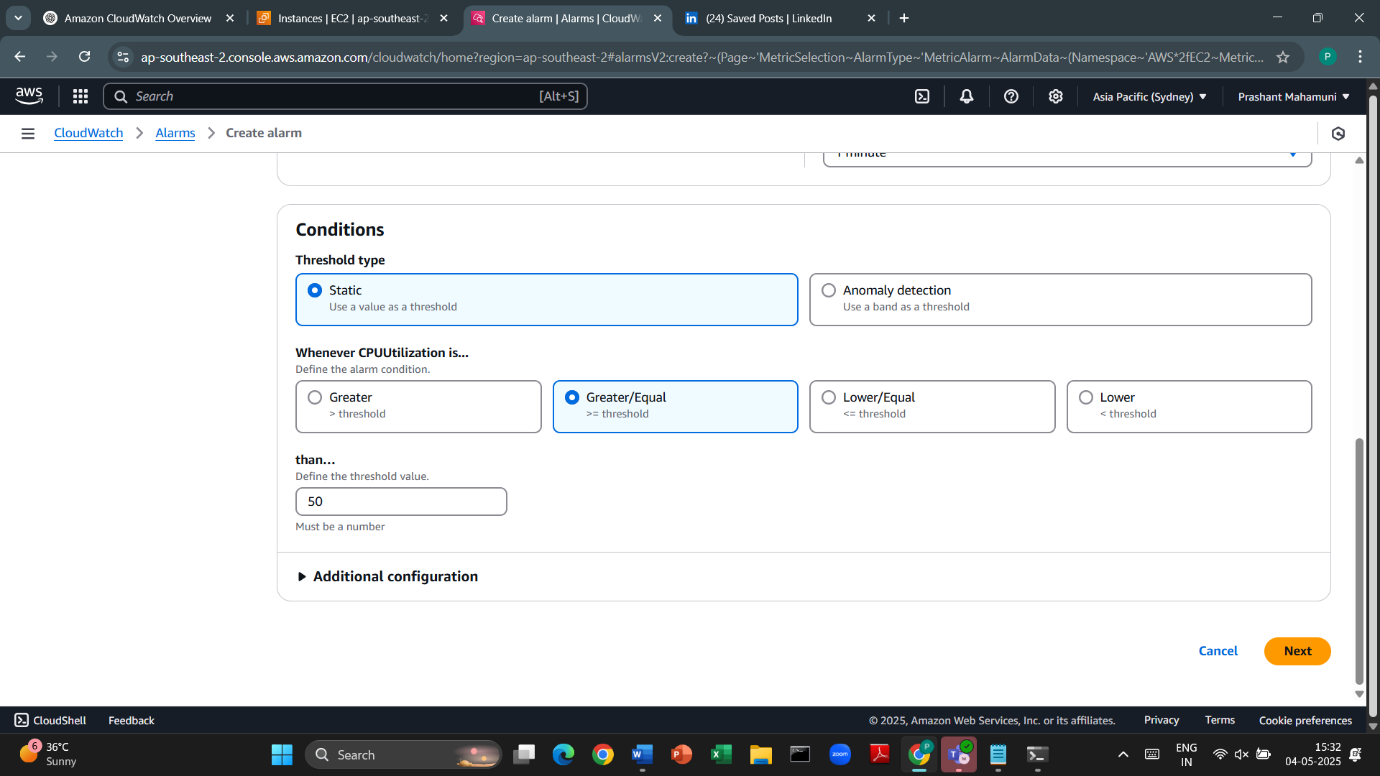
1. Launch an EC2 instance and install Nginx.
2. Use stress to simulate 50%+ CPU usage.
3. Set up a CloudWatch Alarm + SNS to send an email when CPU > 50%.
4. Create CloudWatch Alarm   
   1) Go to CloudWatch → Alarms → Create Alarm

2) Choose Metric:



3) EC2 → Per-Instance Metrics → CPUUtilization  
 4) Select your instance. 

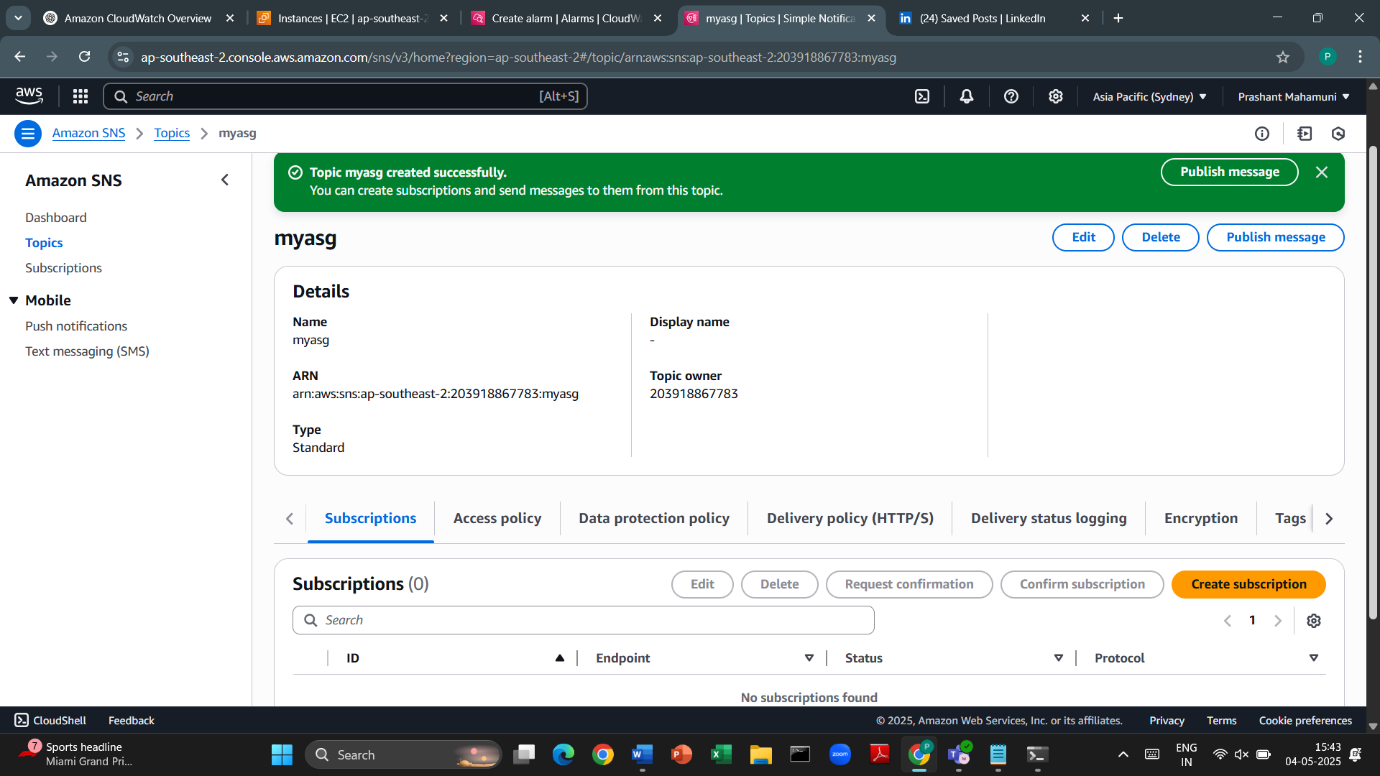
5) Set Conditions:   
 1) Threshold: Greater than 50  
 2) Period: 2 data points within 1 minutes



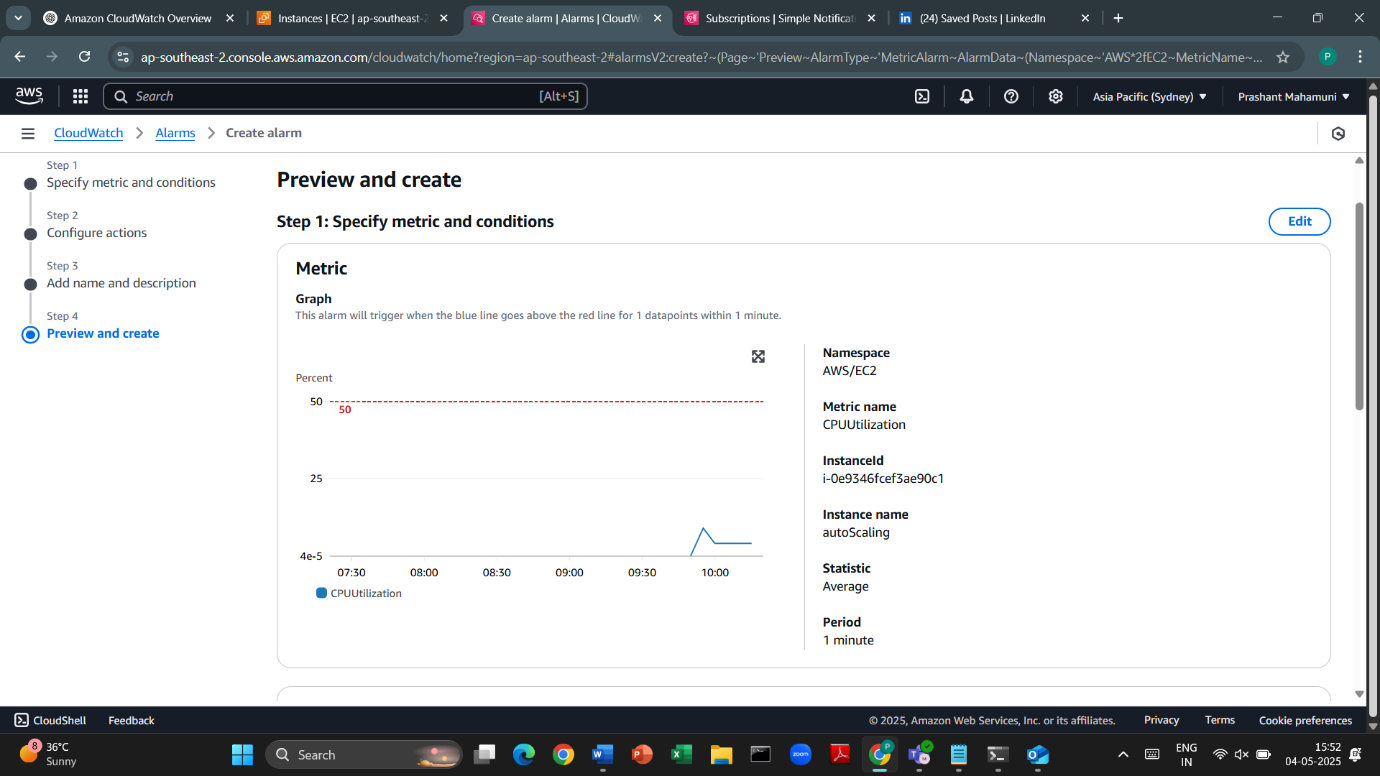
1. Create SNS Topic and Email Subscription  
   Go to **AWS Console → SNS → Topics → Create Topic** Type: Standard  
    Name: CPUAlertTopic

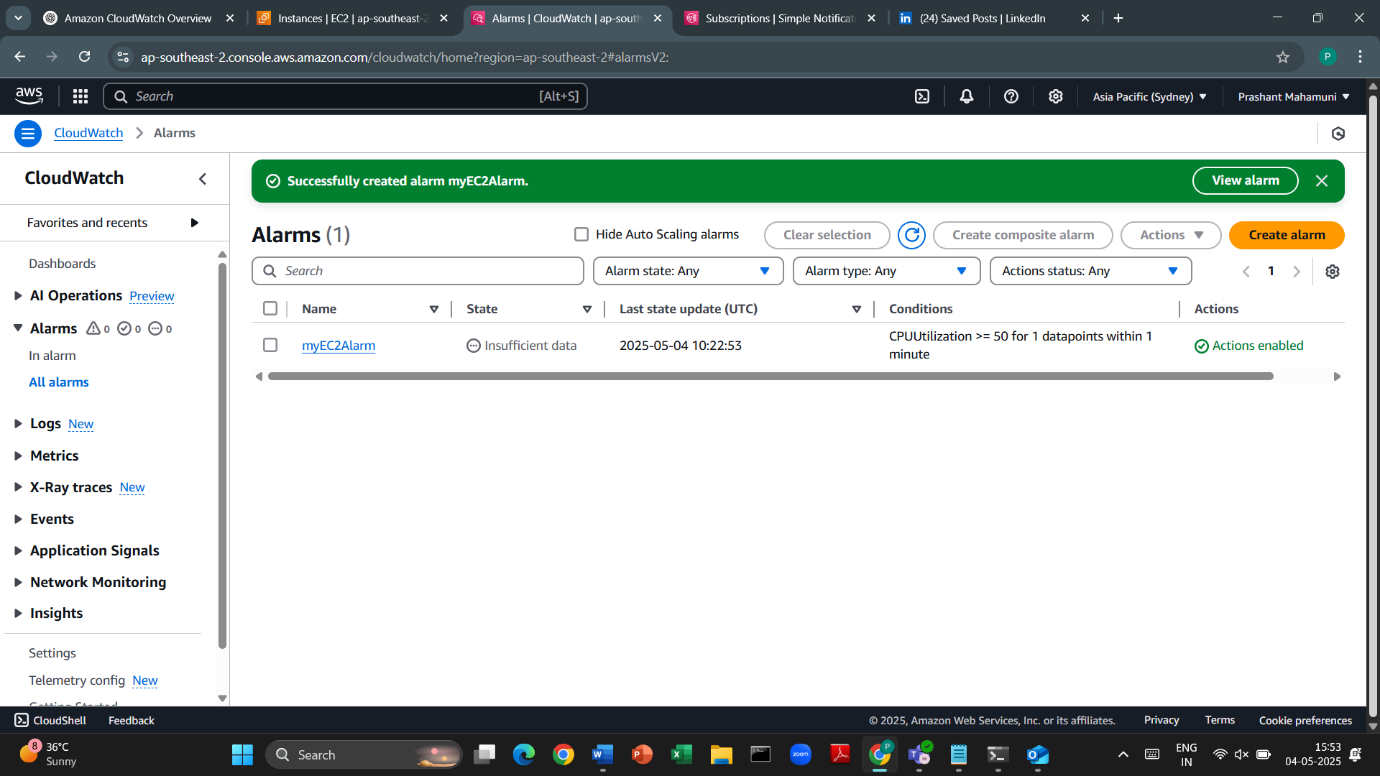
After creating the topic:  
 Click **Create subscription**  
 Protocol: Email  
 Endpoint: your email (e.g., yourname@example.com)

Go to your email inbox and confirm the subscription.

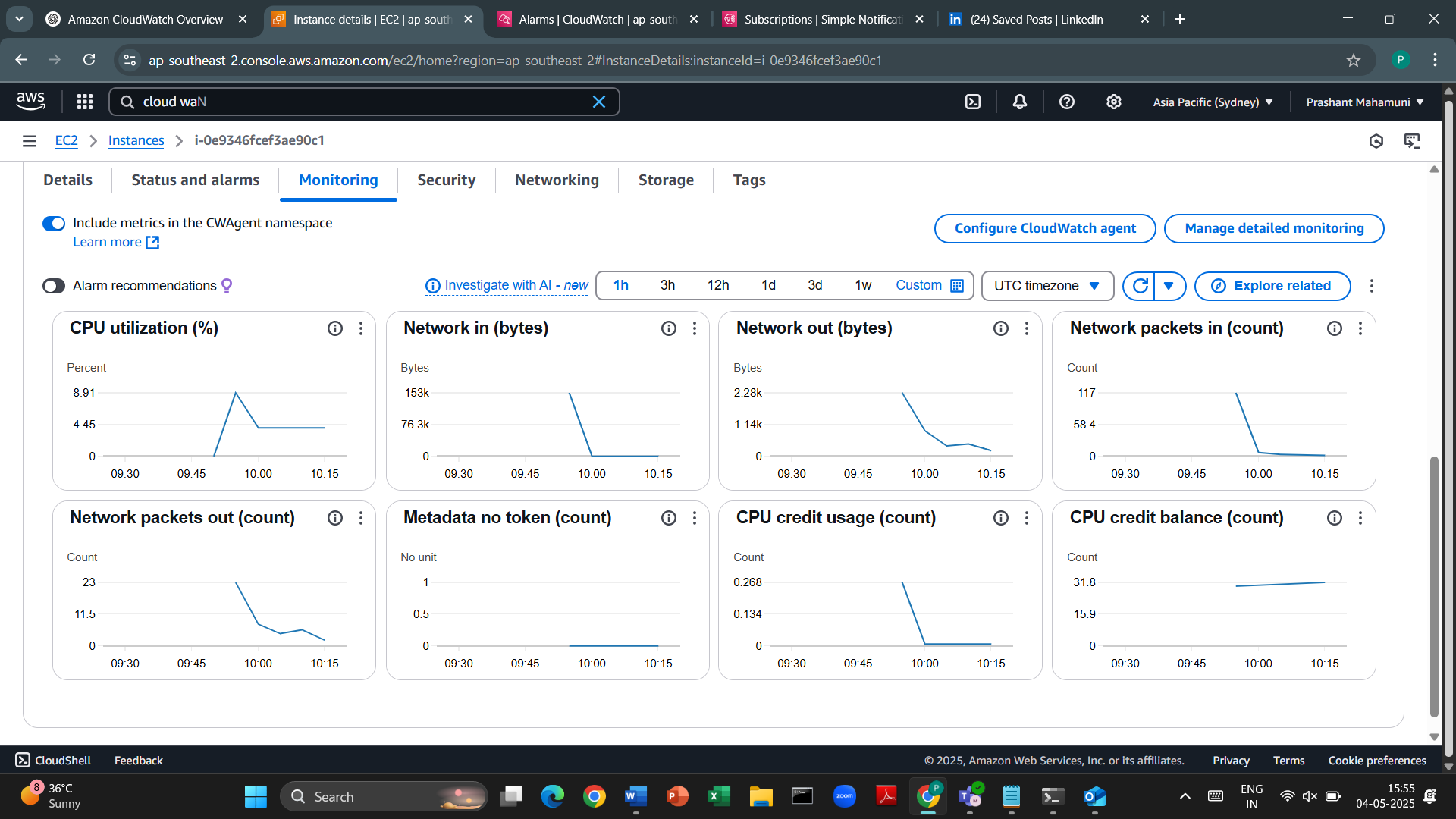


1. Select Actions:  
   Send notification to SNS topic CPUAlertTopic  
   Name your alarm and create it

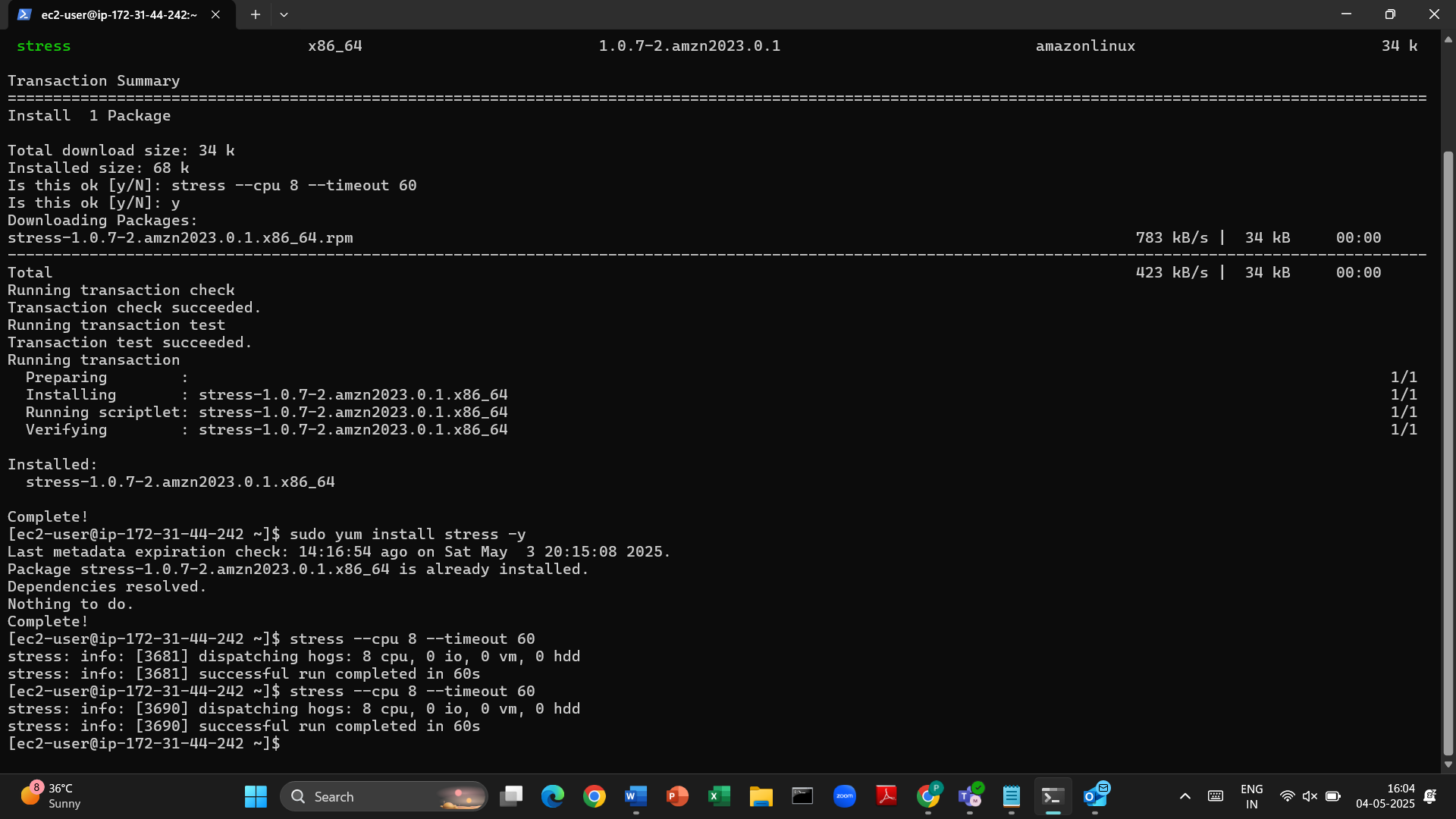




1. Enable Detailed Monitoring From EC2



1. ssh to the server install stress command



Result:

* After 2-5 minutes, CPU > 50%
* CloudWatch Alarm enters ALARM state
* SNS sends email to your inbox (check spam folder too!)

