



Department of Computer Science Engineering

## **CLOUD COMPUTING TOOLS AND TECHNIQUES LAB**

### **PROJECT REPORT**

**"Deploying a scalable 3 tier Inventory Management Web Application on AWS using AWS Amplify and EC2"**

Presented By:

Sachin Mhetre - 22070122119

Mihir Hebalkar - 22070122120

Onkar Mendhapurakar - 22070122135

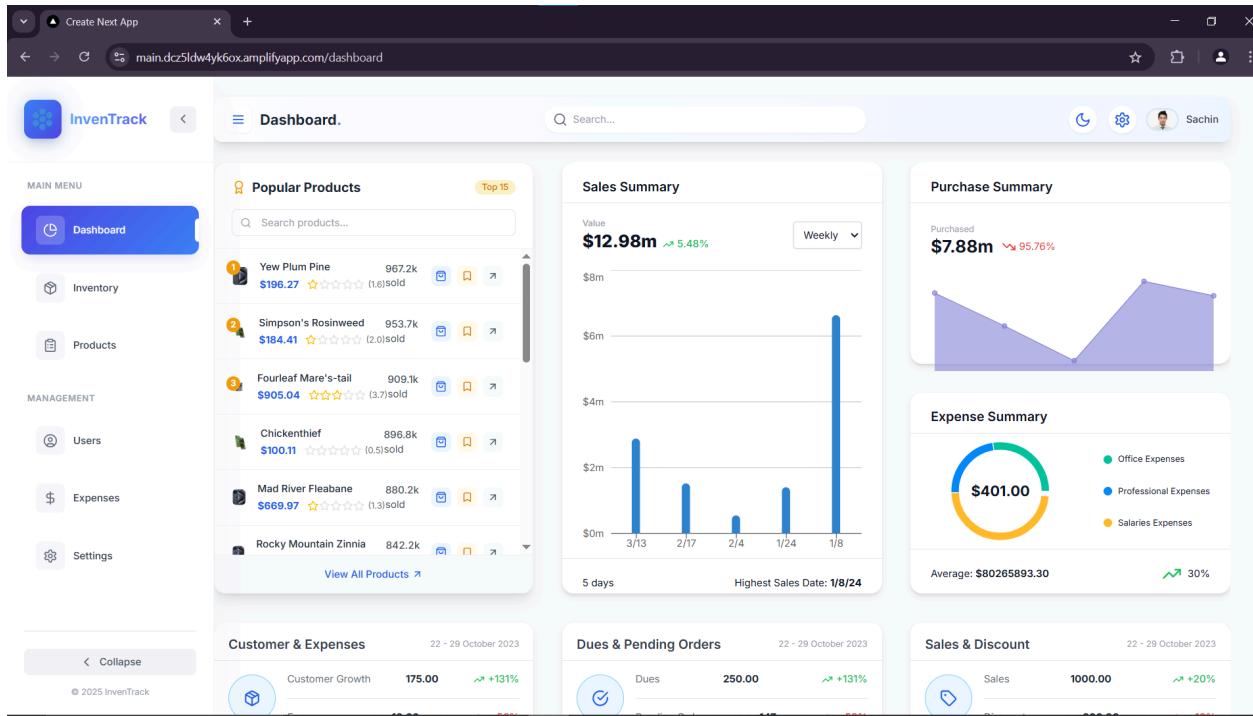
**Under the guidance of:**

Dr. Aditi Sharma

# Index

S. No.	Title	Page No.
1.	Introduction	1
2.	Problem Statement	2
3.	Tech Stack	3
4.	Objectives	5
5.	Key Features	6
6.	AWS Cloud Architecture	7
7.	Project Setup and Steps Followed	8
	7.1 VPC Network Setup	8
	7.2 Compute Layer Deployment	9
	7.3 Configuring AWS RDS	10
	7.4 Deploying Frontend using AWS Amplify	11
	7.5 S3 Bucket for Files Storage	12
	7.6 Load Balancing	13
	7.7 Auto Scaling	13
	7.8 CloudWatch Implementation	14
8.	Conclusion	15

# Introduction



Modern digital business operations need advanced and expandable inventory systems which can handle product management across multiple platforms and locations. This project solves this need by creating a specific full-stack cloud-based inventory system which manages smartphone product inventories.

The application implements a contemporary 3-tier architecture which divides frontend from backend and database to achieve modular development and scalability and maintainability. The Next.js frontend component operates from AWS Amplify while GitHub handles continuous deployment to support efficient development processes. The backend runs on Node.js and Express.js and operates from an Amazon EC2 server within a protected scalable structure. The secure API Gateway service from AWS enables front-to-back communication by exposing APIs.

The system stores its data in Amazon RDS (PostgreSQL) relational database which operates from a private subnet to boost security measures. The system achieves high availability and scalability through the implementation of AWS Auto Scaling Groups alongside Application Load Balancer (ALB) in multiple Availability Zones.

The system operates at peak efficiency because Amazon CloudWatch enables centralized monitoring and logging functions which track EC2, RDS, ALB, and API Gateway systems. The infrastructure provides both support for growth along with traffic spikes and delivers high reliability and cost efficiency.

Our solution builds a professional inventory management platform on AWS cloud services to create an application which scales well and ensures security while being maintainable for enterprise-class operations.

## Problem Statement

Businesses that sell physical goods must manage inventory as their core operational function especially when they operate in the electronics retail sector where smartphone inventory needs precise management. A growing business demands increasingly complex inventory management which needs real-time monitoring of stock amounts alongside sales patterns and purchase activities and user permissions and expense tracking between different departments and sites.

Traditional inventory management systems show multiple significant shortcomings in their operations:

- **Scalability:** The performance of legacy systems suffers when they must handle high volumes of concurrent users and frequent data updates or sudden spikes in traffic which results in performance slowdowns and system downtime.
- **Data Consistency and Accuracy:** A disorganized system without centralization and proper architecture design leads to frequent stock discrepancies and delayed order fulfillment along with human errors.
- **Security and Access Control:** A disorganized system without centralization and proper architecture design leads to frequent stock discrepancies and delayed order fulfillment along with human errors.
- **Lack of Real-Time Monitoring:** Business loss occurs from unidentified problems such as overselling and stockouts and infrastructure failures because automated dashboards and alerts are not implemented.
- **Limited Flexibility:** Systems which do not support modular updates or cloud-native CI/CD methods become difficult to maintain and adapt to evolving

business needs.

Businesses that operate across various regions or platforms including web and mobile and Point of Sale systems need scalable cloud-native inventory management systems. This includes:

- Elastic scaling to meet changing workloads with solutions such as Auto Scaling Groups (ASG).
- High fault tolerance and availability through multi-AZ deployment and load balancing (ALB).
- Constant deployment pipelines for assured delivery of latest features or bug fixes without downtime.
- Seamless interoperability with cloud-native monitoring and logging platforms (e.g., Amazon CloudWatch) for real-time system administration.

A 3-tier inventory management web application development on AWS serves to resolve essential operational requirements. The system implements a modular structure that places Next.js through AWS Amplify in the frontend layer and Node.js on EC2 connected to API Gateway for backend services and PostgreSQL on Amazon RDS as the database. The system architecture supports both quick feature deployments through CI/CD pipelines and automatic horizontal scaling and dynamic response to business needs in real time.

## Tech Stack

### Frontend

- **Framework:** Next.js (via AWS Amplify Hosting)
- **Hosting Platform:** AWS Amplify
- **Deployment:** Continuous deployment from GitHub



### Backend

- **Server:** Amazon EC2 (Ubuntu/Linux)
- **Language/Framework:** Node.js / Express.js
- **API Exposure:** AWS API Gateway (for HTTPS proxy to EC2 backend)



## Database

- **Type:** Amazon RDS (PostgreSQL)
- **Access:** Private subnet access from backend EC2



## Infrastructure & Scalability

- **Load Balancer:** AWS Application Load Balancer (ALB)
- **Auto Scaling:** Amazon EC2 Auto Scaling Group (ASG)
- **Subnets:** Public subnets across multiple Availability Zones (AZs)



## Monitoring & Logging

- **Monitoring:** Amazon CloudWatch (for EC2, ALB, RDS, and API Gateway)
- **Log Management:** CloudWatch Logs and Custom Dashboards



## Cloud Platform

- **Provider:** Amazon Web Services (AWS)
- **Region:** ap-south-1 (Mumbai)



## Version Control & CI/CD

- **Version Control:** Git (via GitHub)
- **CI/CD:** GitHub to AWS Amplify (auto-deploy on push)



## Objectives

The primary goal of this project is to **design, develop, and deploy a scalable 3-tier inventory management web application** on Amazon Web Services (AWS) that supports efficient stock tracking, real-time analytics, and seamless scalability for growing businesses. The application is intended to serve both end-users and administrators with optimized performance, high availability, and ease of management.

Specifically, the objectives are:

1. **To build a modular and scalable architecture** using AWS services, enabling smooth handling of increasing user and data loads.

2. **To provide an intuitive frontend interface** using Next.js hosted via AWS Amplify, supporting responsive design and seamless user interaction.
3. **To implement a secure and reliable backend** with Node.js and Express.js hosted on Amazon EC2, exposed via AWS API Gateway.
4. **To ensure secure data storage and efficient queries** through Amazon RDS (PostgreSQL) with access limited to backend servers.
5. **To leverage cloud-native tools** like AWS Auto Scaling, Load Balancer, and CloudWatch for dynamic scaling, availability, and monitoring.
6. **To automate deployment workflows** using GitHub and AWS Amplify for rapid iteration and continuous delivery.
7. **To offer essential inventory management features** that address real-world business challenges such as sales monitoring, stock tracking, and purchase summaries.

## Key Features

### Inventory Tracking

- Add, update, and manage product listings with pricing, stock, and ratings.
- View summaries of sales, purchases, and stock levels in real-time.

### Dashboard & Analytics

- Interactive dashboard with key metrics: sales trends, stock status, customer growth, and expenses.
- Visual reports for better decision-making.

### Scalable & Secure Cloud Setup

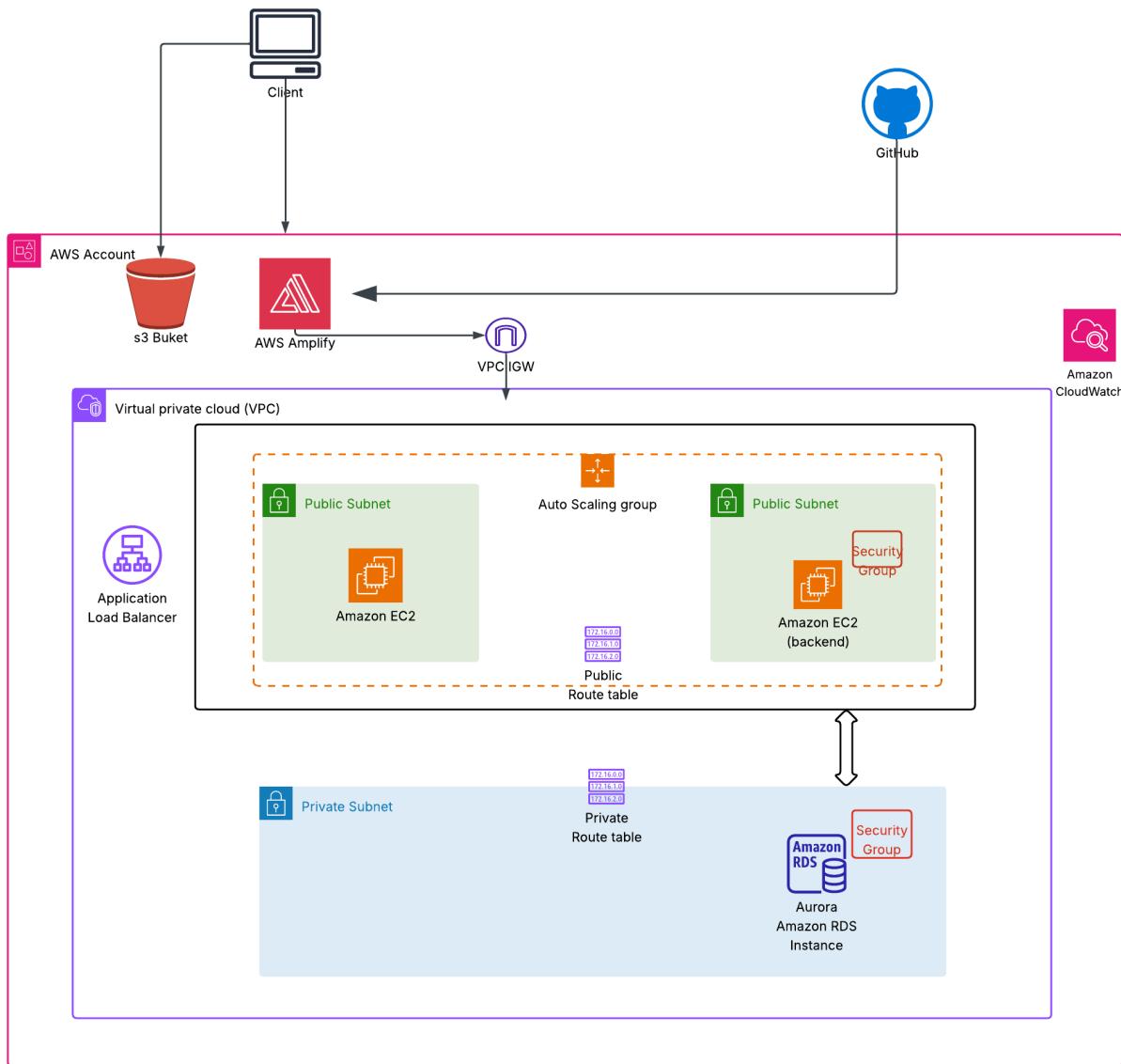
- Auto-scaling backend (EC2) with load balancer for high availability.
- Secure database access via private subnet (Amazon RDS).

### Monitoring & Reliability

- Real-time performance tracking and error alerts via Amazon CloudWatch.

- CI/CD integration for seamless deployment using GitHub and AWS Amplify.

## AWS Cloud Architecture



## Project Setup and Steps followed:

### Network Setup

#### 1. Setup budget

Setting up zero spend budget so that we get notified once our spending exceeds \$0.01 which is above the AWS free tier limits.

**Billing and Cost Management > Budgets > Create budget**

**Choose budget type**

**Budget setup**

- Use a template (simplified)  
Use the recommended configurations. You can change some configuration options after the budget is created.
- Customize (advanced)  
Customize a budget to set parameters specific to your use case. You can customize the time period, the start month, and specific accounts.

**Templates - new**

Choose a template that best matches your use case.

- Zero spend budget  
Create a budget that notifies you once your spending exceeds \$0.01 which is above the AWS Free Tier limits.
- Monthly cost budget  
Create a monthly budget that notifies you if you exceed, or are forecasted to exceed, the budget amount.
- Daily Savings Plans coverage budget  
Create a coverage budget for your Savings Plans that notifies you when you fall below the defined target.
- Daily reservation utilization budget  
Create a utilization budget for your reservations that notifies you when you fall below the defined target.

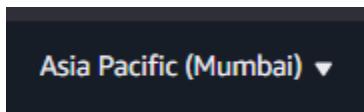
**Zero spend budget - Template**

**Budget name**

Provide a descriptive name for this budget.  
  
Names must be between 1-100 characters.

## 2. Selecting Region

We chose the Asia Pacific (Mumbai) region to ensure low latency for Indian users and comply with local data residency norms. It also offers cost-effective services with all required AWS features for our architecture.



## 3. Creating VPC

### VPC creation with 10.0.0.0/16 CIDR block

**VPC > Your VPCs > vpc-0ac334e711a51bf31**

**You successfully created vpc-0ac334e711a51bf31**

**vpc-0ac334e711a51bf31**

**Details**

<b>VPC ID</b> <input type="text" value="vpc-0ac334e711a51bf31"/>	<b>State</b> <input checked="" type="radio"/> Available	<b>Block Public Access</b> <input type="radio"/> Off	<b>DNS hostnames</b> Disabled
<b>DNS resolution</b> Enabled	<b>Tenancy</b> default	<b>DHCP option set</b> <a href="#">dopt-04aa3c05eb24712c6</a>	<b>Main route table</b> <a href="#">rtb-07d74496dbe9337bc</a>
<b>Main network ACL</b> <a href="#">acl-0f0be9e94613d37ad2</a>	<b>Default VPN</b> No	<b>IPv4 CIDR</b> 10.0.0.0/16	<b>IPv6 pool</b> -
<b>IPv6 CIDR (Network border group)</b> -	<b>Network Address Usage metrics</b> Disabled	<b>Route 53 Resolver DNS Firewall rule groups</b> -	<b>Owner ID</b> <a href="#">971422672601</a>

**Resource map** | **CIDRs** | **Flow logs** | **Tags** | **Integrations**

- Public subnet (10.0.1.0/24) for internet-facing components

VPC > Subnets > Create subnet

### Subnet settings

Specify the CIDR blocks and Availability Zone for the subnet.

**Subnet 1 of 2**

**Subnet name**  
Create a tag with a key of 'Name' and a value that you specify.  
  
The name can be up to 256 characters long.

**Availability Zone** Info  
Choose the zone in which your subnet will reside, or let Amazon choose one for you.

**IPv4 VPC CIDR block** Info  
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.

**IPv4 subnet CIDR block**  
 256 IPs  
< > ^ ^

b. Private subnet (10.0.2.0/24) for database isolation

VPC > Subnets > Create subnet

**Subnet 2 of 2**

**Subnet name**  
Create a tag with a key of 'Name' and a value that you specify.  
  
The name can be up to 256 characters long.

**Availability Zone** Info  
Choose the zone in which your subnet will reside, or let Amazon choose one for you.

**IPv4 VPC CIDR block** Info  
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.

**IPv4 subnet CIDR block**  
 256 IPs  
< > ^ ^

**Tags - optional**

Key	Value - optional
<input type="text" value="Name"/>	<input type="text" value="private-subnet"/> <small>X</small> <small>Remove</small>

Add new tag  
You can add 49 more tags.  
Remove

Add new subnet

⌚ You have successfully created 2 subnets: subnet-0adec40412c76337e, subnet-0ceb772c77c87c5a0

Last updated less than a minute ago C Actions Create subnet

**Subnets (2)** Info

<input type="checkbox"/>	Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR	IPv6 CIDR
<input type="checkbox"/>	private-subnet	subnet-0ceb772c77c87c5a0	<small>Available</small>	vpc-0ac334e711a51bf31	<input type="checkbox"/> Off	10.0.2.0/24	-
<input type="checkbox"/>	public-subnet	subnet-0adec40412c76337e	<small>Available</small>	vpc-0ac334e711a51bf31	<input type="checkbox"/> Off	10.0.1.0/24	-

c. Created a VPC internet gateway to access subnets that were created

**Create internet gateway** Info

An internet gateway is a virtual router that connects a VPC to the internet. To create a new internet gateway specify the name for the gateway below.

**Internet gateway settings**

**Name tag**  
Creates a tag with a key of 'Name' and a value that you specify.

vpc-internet-gateway

**Tags - optional**  
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

**Key**  **Value - optional**

You can add 49 more tags.

#### d. Attaching the created Internet gateway to VPC

**Attach to VPC (igw-068ed3bde54220db2)** Info

**VPC**  
Attach an internet gateway to a VPC to enable the VPC to communicate with the internet. Specify the VPC to attach below.

**Available VPCs**  
Attach the internet gateway to this VPC.

vpc-0ac334e711a51bf31

#### e. Route table configuration

**Create route table** Info

A route table specifies how packets are forwarded between the subnets within your VPC, the internet, and your VPN connection.

**Route table settings**

**Name - optional**  
Creates a tag with a key of 'Name' and a value that you specify.

public-route-table

**VPC**  
The VPC to use for this route table.

vpc-0ac334e711a51bf31

**Tags**  
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

**Key**  **Value - optional**

You can add 49 more tags.

Same for Private route table

#### f. Subnet associations

**Route tables (1/3) Info**

Name	Route table ID	Explicit subnet associ...	Main	VPC	Owner ID
-	rtb-07d74496dbe9337bc	-	Yes	vpc-0ac334e711a51bf31	971422672601
<input checked="" type="checkbox"/> public-route-table	rtb-0c1807326188390e4	-	No	vpc-0ac334e711a51bf31	971422672601
<input type="checkbox"/> private-route-table	rtb-0ff5828df7836caae	-	No	vpc-0ac334e711a51bf31	971422672601

**Subnet associations**

No subnet associations  
You do not have any subnet associations.

**Subnets without explicit associations (2)**

The following subnets have not been explicitly associated with any route tables and are therefore associated with the main route table:

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
private-subnet	subnet-0ceb772c77c87c5a0	10.0.2.0/24	-
public-subnet	subnet-0adec40412c76337e	10.0.1.0/24	-

- g. Private route table has only local access, Public route table is configured to 0.0.0.0/0 allowing access from anywhere.

**Edit routes**

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No
0.0.0.0/0	local	-	No
	Internet Gateway	-	
	igw-068ed3bde54220db2	-	

**Add route** Cancel Preview Save changes

## Compute Layer Deployment

### 4. EC2 backend, Creating an EC2 Instance

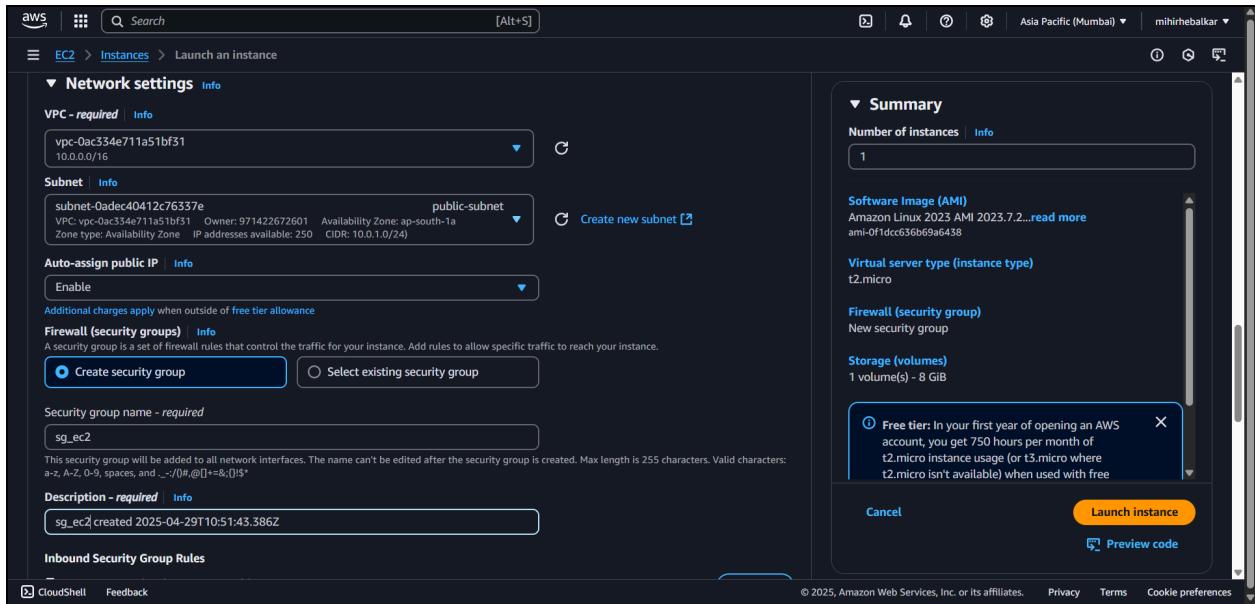
- a. Creating an EC2 instance

Selecting the created VPC and public subnet in network settings.

Selected T2.micro for free tier eligibility

Creating an EC2 instance

And Launching Instance

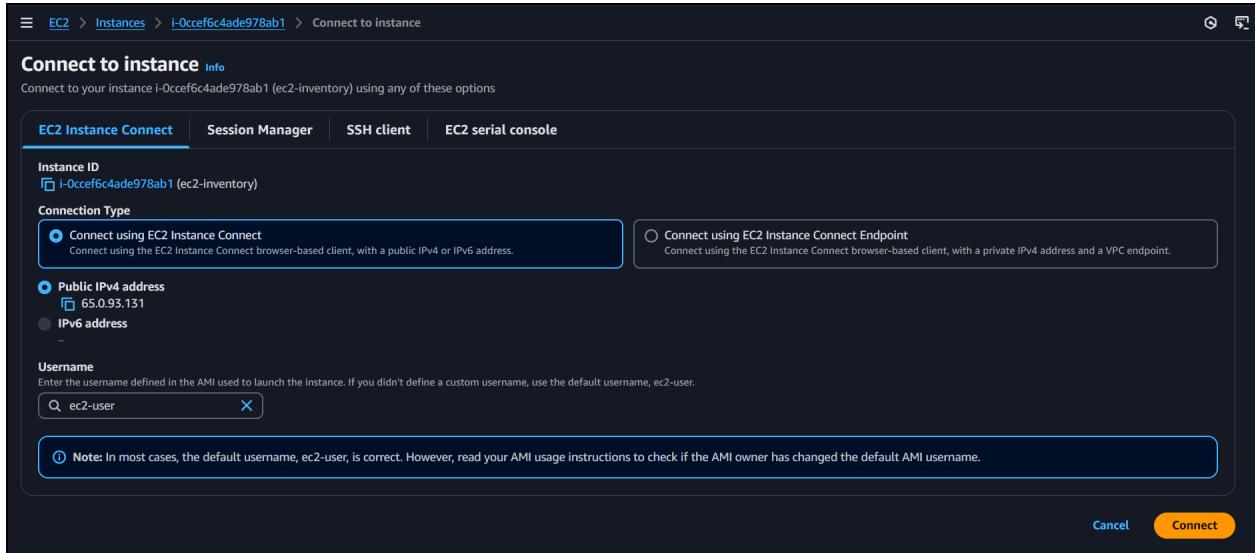


## Instance information

The screenshot shows the AWS EC2 Instances summary page for instance **i-0cccef6c4ade978ab1**. The left sidebar navigation includes 'EC2', 'Dashboard', 'EC2 Global View', 'Events', 'Instances' (selected), 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', 'Capacity Reservations', 'Images', 'AMIs', 'AMI Catalog', 'Elastic Block Store', 'Volumes', 'Snapshots', and 'Lifecycle Manager'. The main content area displays the 'Instance summary for i-0cccef6c4ade978ab1 (ec2-inventory)'. Key details include:

- Instance ID:** i-0cccef6c4ade978ab1
- Public IPv4 address:** 65.0.93.131
- Private IPv4 address:** 10.0.1.61
- Instance state:** Running
- Private IP DNS name (IPv4 only):** ip-10-0-1-61.ap-south-1.compute.internal
- Instance type:** t2.micro
- VPC ID:** vpc-0ac334e711a51bf31
- Subnet ID:** subnet-0adec40412c76337e (public-subnet)
- Instance ARN:** arn:aws:ec2:ap-south-1:971422672601:instance/i-0cccef6c4ade978ab1
- Auto Scaling Group name:** Managed

## b. Connecting to instance using EC2 Instance Connect



## c. Deploying Backend on ec2 instance

### Installing node, npm and git

```

~/m/
[ec2-user@ip-10-0-1-61 ~]$ sudo su -
[root@ip-10-0-1-61 ~]# curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload Total spent Left Speed
100 13226  100 13226    0     0  21832      0 ---:--- ---:--- ---:--- 21825
=> Downloading nvm as script to '/root/.nvm'

=> Appending nvm source string to /root/.bashrc
=> Appending bash_completion source string to /root/.bashrc
=> Close and reopen your terminal to start using nvm or run the following to use it now:

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion" # This loads nvm bash_completion
[root@ip-10-0-1-61 ~]# ./.nvm/nvm.sh
[root@ip-10-0-1-61 ~]# nvm install node
Downloading and installing node v23.11.0...
Downloading https://nodejs.org/dist/v23.11.0/node-v23.11.0-linux-x64.tar.xz...
#####
Computing checksum with sha256sum
Checksums matched!
Now using node v23.11.0 (npm v10.9.2)
Creating default alias: default -> node (> v23.11.0)
[root@ip-10-0-1-61 ~]# node -v
v23.11.0
[root@ip-10-0-1-61 ~]# npm -v
10.9.2
[root@ip-10-0-1-61 ~]# 

```

i-0ccefc4ade978ab1  
Public IPs: 65.0.93.131 Private IPs: 10.0.1.61

After cloning repo, install packages and run backend on port 80

```

10:38:06 AM - Starting compilation in watch mode...
[0]
[1] Server running on port 80
[0]
[0] 10:38:11 AM - Found 0 errors. Watching for file changes.
[1] 103.197.74.172 - - [29/Apr/2025:10:39:03 +0000] "GET / HTTP/1.1" 404 139
^C[0] npx tsc -w exited with code SIGINT
[1] nodemon --exec ts-node src/index.ts exited with code 130
[root@ip-10-0-1-61 server]# npm i pm2 -g
added 134 packages in 7s

```

Installing pm2 which is a production process manager, particularly for Node.js applications, that helps ensure applications run continuously by automatically restarting them if they crash, and provides features like log management, monitoring, and load balancing

And running backend using pm2

```

[root@ip-10-0-1-61 server]# pm2 start ecosystem.config.js
[PM2] Spawning PM2 daemon with pm2_home=/root/.pm2
[PM2] PM2 Successfully daemonized
[PM2] [WARN] Applications inventory-management not running, starting...
[PM2] App [inventory-management] launched (1 instances)

```

<b>id</b>	<b>name</b>	<b>namespace</b>	<b>version</b>	<b>mode</b>	<b>pid</b>	<b>uptime</b>	<b>⌚</b>	<b>status</b>	<b>cpu</b>	<b>mem</b>	<b>user</b>	<b>watching</b>
0	inventory-management	default	N/A	fork	29912	0s	0	online	0%	35.1mb	root	disabled

```

[root@ip-10-0-1-61 server]# pm2 status

```

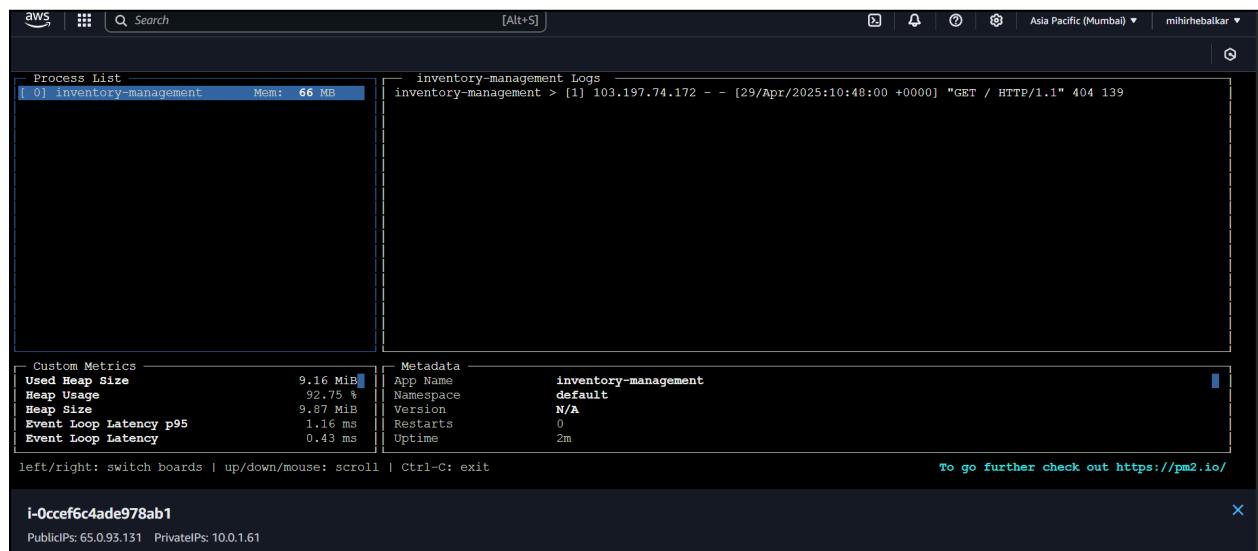
<b>id</b>	<b>name</b>	<b>namespace</b>	<b>version</b>	<b>mode</b>	<b>pid</b>	<b>uptime</b>	<b>⌚</b>	<b>status</b>	<b>cpu</b>	<b>mem</b>	<b>user</b>	<b>watching</b>
0	inventory-management	default	N/A	fork	29912	14s	0	online	0%	68.8mb	root	disabled

```

[root@ip-10-0-1-61 server]# pm2 monit

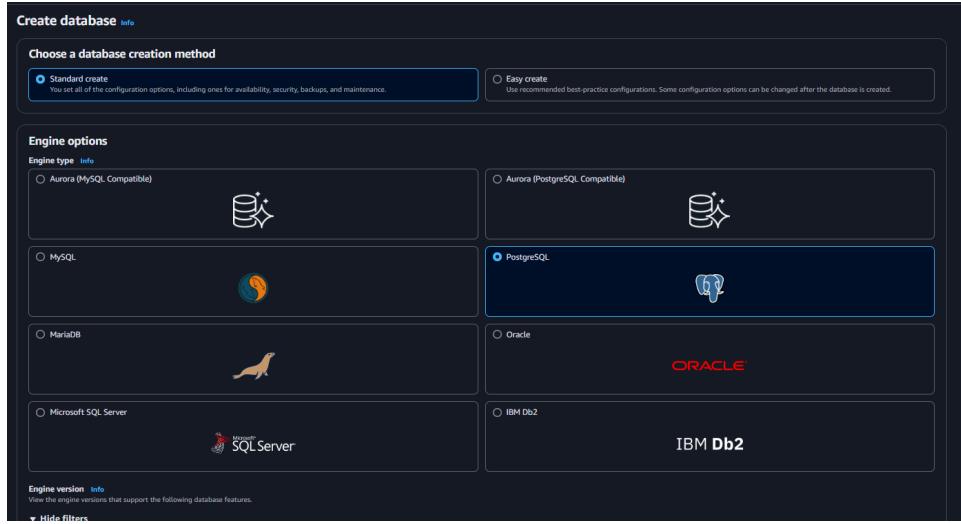
```

Now backend has been successfully deployed on ec2 and can be used by the frontend



## 5. Configuring AWS RDS - Relational database

## a. Selecting type of database which we have used - postgresql



## b. Settings Credentials

The screenshot shows the 'Settings' page for a PostgreSQL database instance. It includes fields for 'DB instance identifier' (set to 'rds-Inventorymanagement'), 'Master username' (set to 'postgres'), and 'Master password'. Under 'Credentials management', 'Self managed' is selected. There are also sections for 'Auto generate password' and 'Password strength' (set to 'Neutral'). A note at the bottom states: 'Minimum constraints: At least 8 printable ASCII characters. Can't contain any of the following symbols: / \ \* @'.

## c. Assigning VPC and Subnet Groups to the DB

The screenshot shows the configuration page for assigning VPC and Subnet Groups to the database. It includes sections for 'Virtual private cloud (VPC)', 'DB subnet group', 'Public access', and 'VPC security group (firewall)'. Under 'Virtual private cloud (VPC)', 'vpc-0ac334e71a51bf31' is selected. Under 'DB subnet group', 'rds-subnet-group' is selected. Under 'Public access', 'No' is selected. Under 'VPC security group (firewall)', 'Choose existing' is selected. A note at the bottom states: 'Existing VPC security groups'.

## e. Database has been successfully Created

The screenshot shows the AWS RDS console with the 'Databases' page selected. On the left, there's a sidebar with options like Dashboard, Databases (which is selected), Query editor, Performance insights, Snapshots, Exports in Amazon S3, Automated backups, Reserved instances, Proxies, Subnet groups, Parameter groups, Option groups, Custom engine versions, Zero-ETL integrations, Events, and Event subscriptions. The main area displays a table titled 'Databases (1)'. The table has columns: DB identifier, Status, Role, Engine, Region ..., Size, and Recommendations. One row is visible for 'rds-inventorymanagement', which is marked as 'Available', an 'Instance', running 'PostgreSQL', in the 'ap-south-1b' region, and 'db.t4g.micro' size.

## f. Prisma schema is synchronized with your AWS RDS PostgreSQL database (inventorymanagement)

```
[root@ip-10-0-1-236 server]# nano .env
[root@ip-10-0-1-236 server]# npx prisma migrate dev --name init
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": PostgreSQL database "inventorymanagement", schema "public" at "rds-inventorymanagement.ct8ay0em2c0e.us-east-2.rds.amazonaws.com:5432"
Already in sync, no schema change or pending migration was found.

✓ Generated Prisma Client (v5.16.2) to ./node_modules/@prisma/client in 147ms

[root@ip-10-0-1-236 server]# npm run seed
```

The database was successfully seeded with initial data for all tables (Products, Sales, Purchases, Users, Expenses, etc.) using JSON files

```
[root@ip-10-0-1-236 server]# npm run seed

> server@1.0.0 seed
> ts-node prisma/seed.ts

Cleared data from Products
Cleared data from ExpenseSummary
Cleared data from Sales
Cleared data from SalesSummary
Cleared data from Purchases
Cleared data from PurchaseSummary
Cleared data from Users
Cleared data from Expenses
Cleared data from ExpenseByCategory
Seeded products with data from products.json
Seeded expenseSummary with data from expenseSummary.json
Seeded sales with data from sales.json
Seeded salesSummary with data from salesSummary.json
Seeded purchases with data from purchases.json
Seeded purchaseSummary with data from purchaseSummary.json
Seeded users with data from users.json
Seeded expenses with data from expenses.json
Seeded expenseByCategory with data from expenseByCategory.json
[root@ip-10-0-1-236 server]#
```

i-0ae964ba1ff8b4e8b (ec2-inventorymanagement-backend)

PublicIPs: 3.15.29.53 PrivateIPs: 10.0.1.236

The inventory-management application is running in online status with minimal resource usage (0% CPU, 22.6MB memory) under PM2 process manager.

```
[root@ip-10-0-1-61 CloudProject]# cd server
[root@ip-10-0-1-61 server]# pm2 start ecosystem.config.js
[PM2] Applying action restartProcessId on app [inventory-management](ids: [ 0 ])
[PM2] [inventory-management] (0) ✓


| <b>id</b> | <b>name</b>          | <b>namespace</b> | <b>version</b> | <b>mode</b> | <b>pid</b> | <b>uptime</b> | <b>ø</b> | <b>status</b> | <b>cpu</b> | <b>mem</b> | <b>user</b> | <b>watching</b> |
|-----------|----------------------|------------------|----------------|-------------|------------|---------------|----------|---------------|------------|------------|-------------|-----------------|
| 0         | inventory-management | default          | N/A            | fork        | 33591      | 0s            | 1        | online        | 0%         | 22.6mb     | root        | disabled        |


[root@ip-10-0-1-61 server]#
```

## App is running with successful /dashboard access and healthy performance

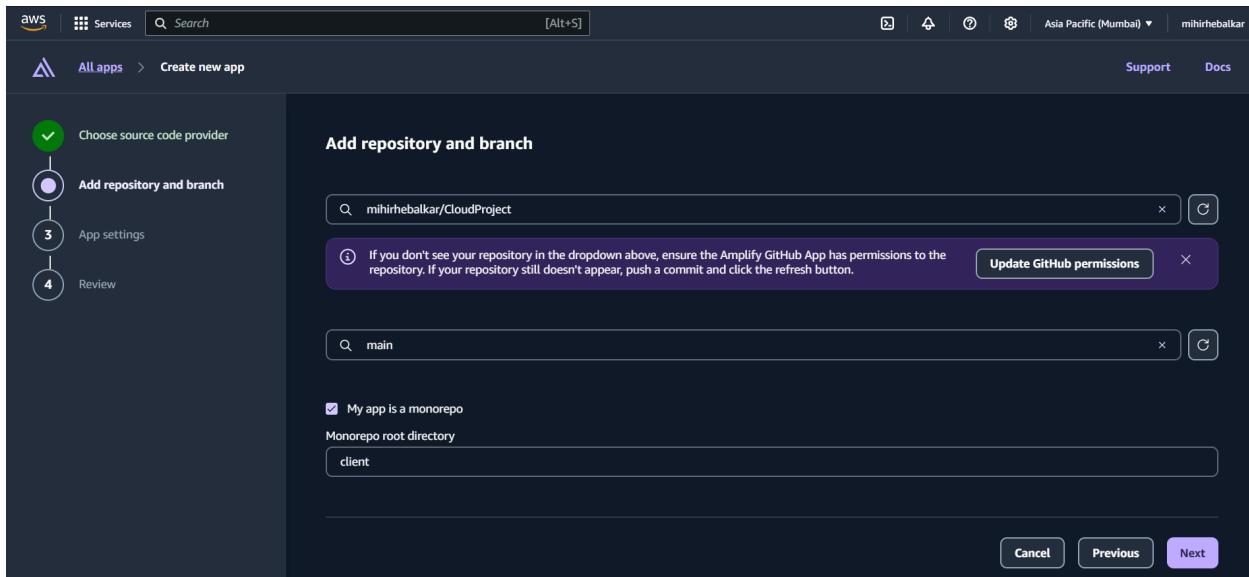
The screenshot shows the AWS CloudWatch Metrics and Logs interface. On the left, a process list shows 'inventory-management' with 64 MB of memory. The main area displays logs for the 'inventory-management' application, showing two log entries: one for a GET request to '/' and another for a GET request to '/dashboard'. Below the logs, there is a section for 'Custom Metrics' and 'Metadata'. The 'Custom Metrics' table includes columns for 'Used Heap Size', 'Heap Usage', 'Heap Size', 'Event Loop Latency p95', and 'Event Loop Latency', with values like 8.72 MiB, 88.34 %, 9.87 MiB, 1.20 ms, and 0.35 ms respectively. The 'Metadata' table includes columns for 'App Name', 'Namespace', 'Version', 'Restarts', and 'Uptime', with values like 'inventory-management', 'default', 'N/A', 0, and 6m. At the bottom, there is a note to 'To go further check out https://pm2.io/'.

Using <http://65.0.93.131> public ec2 IP

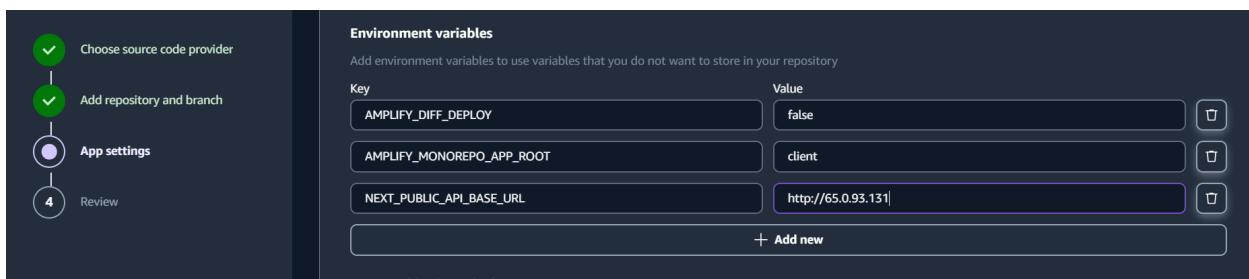
## 5. Deploying Frontend using AWS Amplify

The screenshot shows the AWS Amplify 'Create new app' wizard. Step 1, 'Choose source code provider', is selected. It shows four options: GitHub (selected), BitBucket, CodeCommit, and GitLab. Below the options, it says 'Amplify requires read-only access to your repository.' and 'To manually deploy an app from Amazon S3 or a Zip file, select "Deploy without Git"'. There is also a 'Start with a template' button at the bottom.

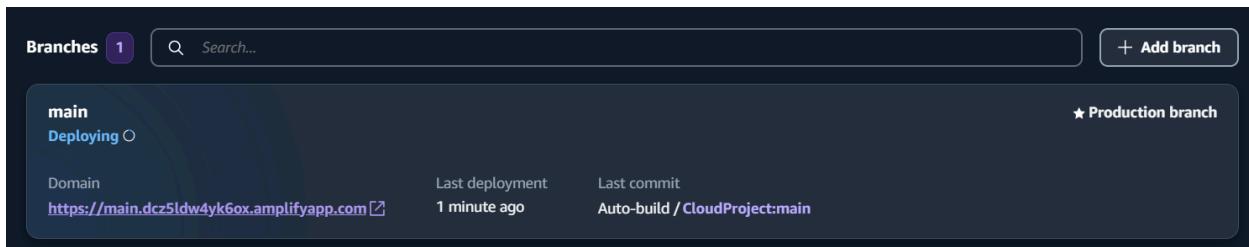
### a. Select repo and as monorepo select root directory for frontend



### b. Add env to access backend



### c. Deploying Nextjs application



Backend requires http but amplify gives https hence creating api gateway

### d. API Gateways

## Create http API

Add HTTP integrations for endpoints like

```
/* ROUTES */
app.use("/dashboard", dashboardRoutes); // http://localhost:8000/dashboard
app.use("/products", productRoutes); // http://localhost:8000/products
app.use("/users", userRoutes); // http://localhost:8000/users
app.use("/expenses", expenseRoutes); // http://localhost:8000/expenses

/* SERVER */
const port = Number(process.env.PORT) || 3001;
app.listen(port, "0.0.0.0", () => {
  console.log(`Server running on port ${port}`);
});
```

**API details**

**API name**  
An HTTP API must have a name. The name is a non-unique value you use to identify and organize your APIs. To programmatically refer to this API, use the API ID that API Gateway generates for you.

**IP address type** | [Info](#)  
Select the type of IP addresses that can invoke the default endpoint for your API. You don't need to redeploy your API for the update to take effect.

**IPv4**  
Includes only IPv4 addresses.

**Dualstack**  
Includes IPv4 and IPv6 addresses.

**Integrations (4)** | [Info](#)  
Specify the backend services that your API will communicate with. These are called integrations. For a Lambda integration, API Gateway invokes the Lambda function and responds with the response from the function. For an HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.

Method	URL endpoint	Remove
GET	http://65.0.93.131/dashboard	<a href="#">Remove</a>
HTTP		<a href="#">Remove</a>
Method	URL endpoint	Remove
GET	http://65.0.93.131/users	<a href="#">Remove</a>
HTTP		<a href="#">Remove</a>
Method	URL endpoint	Remove
GET	http://65.0.93.131/expenses	<a href="#">Remove</a>
HTTP		<a href="#">Remove</a>
Method	URL endpoint	Remove
ANY	http://65.0.93.131/products	<a href="#">Remove</a>

**Define stages - optional**

**Configure stages** Info

Stages are independently configurable environments that your API can be deployed to. You must deploy to a stage for API configuration changes to take effect, unless that stage is configured to autodeploy. By default, all HTTP APIs created through the console have a default stage named \$default. All changes that you make to your API are autodeployed to that stage. You can add stages that represent environments such as development or production.

Stage name	Auto-deploy
\$default	<input checked="" type="checkbox"/>
prod	<input checked="" type="checkbox"/>
<a href="#">Add stage</a>	<a href="#">Remove</a>
	<a href="#">Remove</a>

[Cancel](#) [Previous](#) [Next](#)

### e. Add prod stage with autodeploy

Stages prod -> <https://unp7x0fnp0.execute-api.ap-south-1.amazonaws.com/prod>

### f. Add into env of Amplify

Hosting -> Environment variables -> Manage variables

Variable Value Branch Actions

NEXT\_PUBLIC\_API\_BASE\_URL execute-api.ap-south-1.amazonaws.com/prod All branches

**Redeploy**

The dashboard displays the following key metrics:

- Popular Products:** Yew Plum Pine (\$186.27), Simpson's Rosinweed (\$184.41), Fourleaf Mare's-tail (\$905.04), Chickenthief (\$101.11), Mad River Fleabane (\$669.97), Rocky Mountain Zinnia (842.2k).
- Sales Summary:** Total value \$12.98m, growth 5.48% over the last 5 days.
- Purchase Summary:** Total value \$7.88m, growth -95.76%.
- Expense Summary:** Total value \$401.00, showing a donut chart for Office Expenses, Professional Expenses, and Salaries Expenses.
- Customer & Expenses:** Customer Growth 175.00, +131%.
- Dues & Pending Orders:** Dues 250.00, +131%.
- Sales & Discount:** Sales 1000.00, +20%.

## 6. S3 Bucket for Files storage

## a. Create s3 bucket

The screenshot shows the 'Create bucket' wizard. Under 'General configuration', the 'AWS Region' is set to 'Asia Pacific (Mumbai) ap-south-1'. The 'Bucket type' is set to 'General purpose', which is described as recommended for most use cases. A 'Bucket name' field contains 's3-inventorymanagement'. Below it, a note says 'Bucket names must be 3 to 65 characters and unique within the global namespace.' A 'Copy settings from existing bucket - optional' section includes a 'Choose bucket' button. The URL format is shown as 'Format: s3://bucket/prefix'.

## b. Bucket has been successfully created

The screenshot shows the 'General purpose buckets' list. It displays one bucket named 'cl4-inventorymanagement' located in 'Asia Pacific (Mumbai) ap-south-1'. The bucket was created on 'April 29, 2025, 17:01:11 (UTC+05:30)'. Action buttons include 'Copy ARN', 'Empty', 'Delete', and 'Create bucket'.

The screenshot shows the 'Upload' page for the 'cl4-inventorymanagement' bucket. It lists five files: 'product2.png', 'product3.png', 'profile.jpg', 'logo.png', and 'product1.png', all in 'image/png' format with sizes ranging from 12.7 KB to 23.3 KB. A 'Find by name' search bar is at the top. Action buttons include 'Remove', 'Add files', and 'Add folder'.

## c. Upload assets

## d. Edit bucket policy

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "PublicReadGetObject",  
      "Effect": "Allow",
```

```

        "Principal" : "*",
        "Action": "s3:GetObject",
        "Resource" : "arn:aws:s3:::cl4-inventorymanagement/*"
    }
]
}

```

The screenshot shows the AWS S3 'Edit bucket policy' interface. At the top, there's a navigation bar with the AWS logo, search bar, and region selection (Asia Pacific (Mumbai)). Below it, the breadcrumb path is 'Amazon S3 > Buckets > cl4-inventorymanagement > Edit bucket policy'. The main area has tabs for 'Bucket policy' (selected), 'Policy examples', and 'Policy generator'. A note says 'The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts.' Below this is a 'Bucket ARN' field with the value 'arn:aws:s3:::cl4-inventorymanagement'. The 'Policy' section contains the JSON policy code shown in the previous snippet. To the right, there's a sidebar with 'Edit statement' and 'Select a statement' sections, and a button '+ Add new statement'.

```

1  {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "PublicReadGetObject",
6       "Effect": "Allow",
7       "Principal": "*",
8       "Action": "s3:GetObject",
9       "Resource": "arn:aws:s3:::cl4-inventorymanagement/*"
10    }
11  ]
12 }

```

### e. Now object URLs are visible

<https://cl4-inventorymanagement.s3.ap-south-1.amazonaws.com/>

### f. Update next.config.mjs to allow traffic from s3 bucket

The screenshot shows a code editor with two tabs: 'page.tsx' and 'next.config.mjs'. The 'next.config.mjs' tab is active, showing the following code:

```

client > next.config.mjs > ...
1  /** @type {import('next').NextConfig} */
2  const nextConfig = {
3    images: {
4      remotePatterns: [
5        {
6          protocol: "https",
7          hostname: "cl4-inventorymanagement.s3.ap-south-1.amazonaws.com",
8          port: "",
9          pathname: "**",
10        },
11      ],
12    },
13  };
14
15 export default nextConfig;

```

## 7. Load balancing

Creating load balancer,

Selected current vpc and 2 availability zones

Ap-south-1b and ap-south-1a for 2 separate instances which run our backend, now load balancer will choose instance based on availability and balance load

The screenshot shows the AWS CloudFormation console with a new stack named "im-lb" being created. The "Details" tab is selected, displaying the stack's ARN, status (CREATE\_IN\_PROGRESS), VPC ID, and two availability zones (ap-south-1b and ap-south-1a). The "Listeners and rules" tab shows one listener rule defined for port 80. The "Load balancers" tab lists the single "im-lb" load balancer. A browser window at the bottom shows a failed attempt to access the load balancer's DNS endpoint.

**Details**

- Load balancer type: Application
- Status: Active
- VPC: vpc-0ac334e711a51bf31
- Availability Zones:
  - subnet-08e450ab166bbc290 ap-south-1b (ap-s1-aZ3)
  - subnet-0ade40412c76337e ap-south-1a (ap-s1-aZ1)
- Load balancer IP address type: IPv4
- Date created: April 29, 2025, 20:01 (UTC+05:30)

**Listeners and rules (1)**

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

**Load balancers (1/1)**

Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
im-lb	im-lb-1817131851.ap-sout...	Active	vpc-0ac334e711a51bf31	2 Availability Zones	application	April 29, 2025, 20:01 (UTC+05:30)

**Browser Screenshot:**

Cannot GET /

```

[{"productId": "d35623ee-beff-42b2-8776-2f99f8bb4782", "name": "Pinkscale Blazing Star", "price": 456.04, "rating": 2.25, "stockQuantity": 124834}, {"productId": "baclac77-7358-425e-be16-0bddd9f02e59", "name": "Gila Milkvetch", "price": 899.05, "rating": 3.56, "stockQuantity": 799402}, {"productId": "1afc136b-4d9f-4e8e-aace-81edf908a404", "name": "Rocky Mountain Zinnia", "price": 261.37, "rating": 3.25, "stockQuantity": 842192}, {"productId": "af84cc12-4fea-4f58-aece-f2ce92ca9580", "name": "Guadalupe Suncup", "price": 555.93, "rating": 4.09, "stockQuantity": 236333}, {"productId": "86e3bb1c-2f5d-4774-98f3-4df7cddd0a0f", "name": "Saline Phlox", "price": 82.62, "rating": 4.8, "stockQuantity": 601208}, {"productId": "26b017c6-06d8-443f-9b4a-d6b1ce6f4c0", "name": "Common Brighteyes", "price": 435.44, "rating": 0.27, "stockQuantity": 124068}], 
```

Now aws amplify can use the load balancer url im-lb.. (inventory management)

## 8. Auto Scaling

### a. Created autoscaling group and used existing load balancer

Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max	Availability Zones
inventorymg-asg	inventorymgmt-asg   Version Default	0	Updating capacity...	1	1	3	ap-south-1b, ap-south...

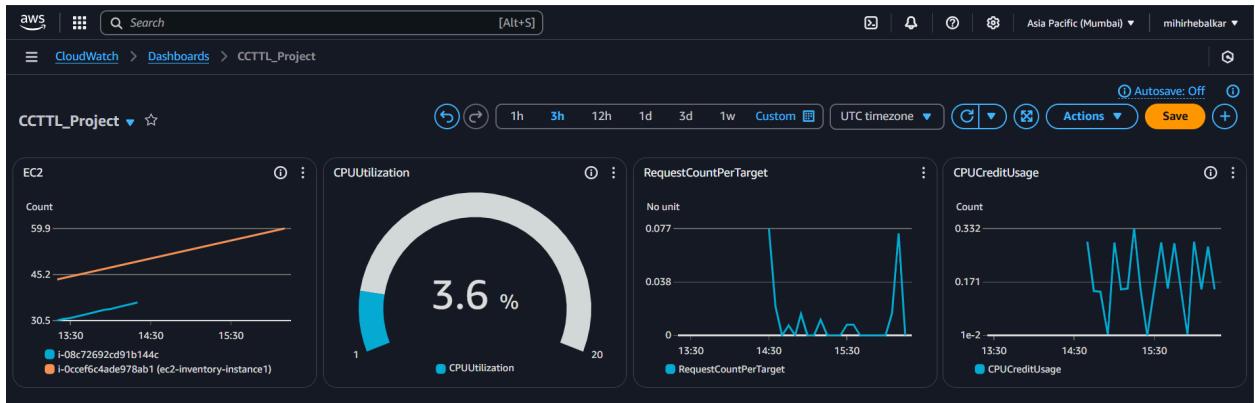
The screenshot shows the AWS EC2 Auto Scaling Groups page. The top navigation bar includes the AWS logo, search bar, and region selection (Asia Pacific (Mumbai)). The main header is "inventorymg-asg". Below it, the "Capacity overview" section displays the ARN: arn:aws:autoscaling:ap-south-1:971422672601:autoScalingGroup:12ac885a-f1c7-46fb-bedc-cd136f4691b\$autoScalingGroupName/inventorymg-asg. It shows Desired capacity (1), Scaling limits (Min - Max: 1 - 3), Desired capacity type (Units (number of instances)), and Status (-). A note indicates the date created: Tue Apr 29 2025 20:07:08 GMT+0530 (India Standard Time).

The navigation tabs include Details (selected), Integrations - new, Automatic scaling, Instance management, Instance refresh, Activity, and Monitoring.

The "Launch template" section details the configuration: Launch template (lt-06f99fd5af97bbd8 inventorymgmt-asg), AMI ID (ami-0da823d47691f050e), Instance type (t2.micro), Owner (arn:aws:iam::971422672601:root), Version (Default), Security groups (-), Security group IDs (sg-04504c40a666a1f9e), Key pair name (cloudproject), Create time (Tue Apr 29 2025 19:23:56 GMT+0530 (India Standard Time)), Description (template for auto scaling grp), Storage (volumes) (-), and Request Spot Instances (No). A link to "View details in the launch template console" is provided.

## 9. CloudWatch implementation

Implemented CloudWatch dashboard which shows EC2 instance utilization and CPU utilization, along with it our cloudwatch dashboard shows Request count per target and CPU Credit Usage. Implementing this allows us to monitor usage of our project on AWS



## Conclusion

The report discusses a very strong and up-to-date cloud-native implementation for inventory management. By strategically utilizing AWS cloud services coupled with an infrastructure that is entirely cloud-based, the system efficiently solves shortcomings of traditional inventory systems and provides significant benefits in operations. The benefits of the architecture include:

- Elastic scalability – Enabling the system to automatically cope with different workloads, assuring optimal performance at peak and off-peak periods.
- High availability – By ensuring that the system will be resilient and have a minimum failure time.
- Enhanced security – Follow best practices for network isolation, access control, and data protection to maintain system integrity and compliance.
- Continuous deployment – Rapid, reliable updates are achieved via CI/CD pipelines, reducing the time taken for new features and fixes to be deployed.
- Comprehensive monitoring – Includes state of the art monitoring and alerting tools for proactive management and keeping problems at bay.

In short, this solution serves as a reliable, scalable, and secure infrastructure, founded on microservices for inventory management, which can be used for both vibrant and growing business environments.