Elastic Beanstalk Deployment Documentation

Table of Contents

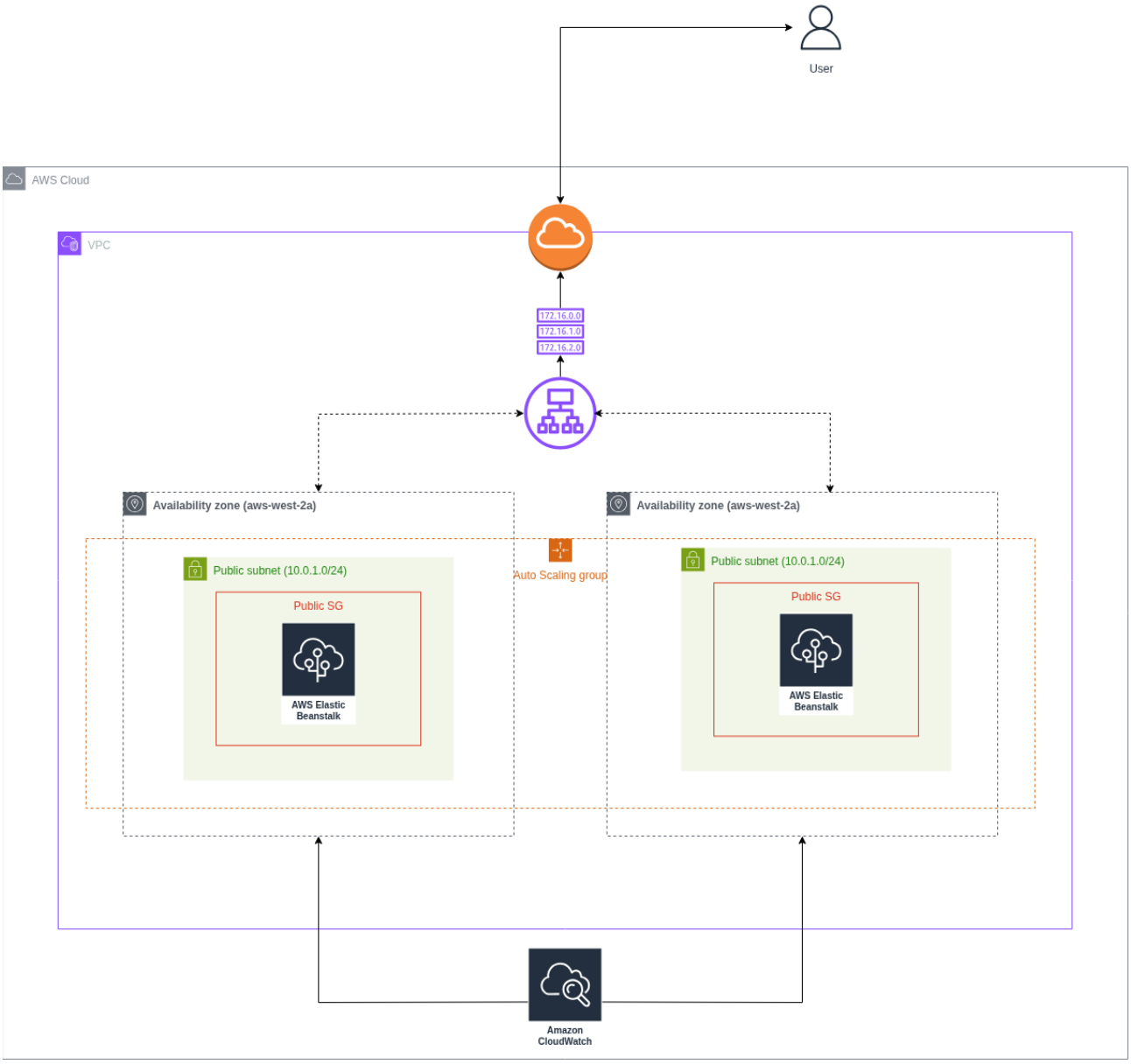
Prerequisites

- AWS account with sufficient IAM permissions (EB, EC2, S3, VPC, IAM, CloudWatch)
- A ready Node.js app, packaged as a ZIP containing `index.js`, `package.json`, and `package-lock.json` (no `node_modules`)

Objective

Deploy the Node.js app to AWS Elastic Beanstalk: upload the ZIP, configure a Node.js environment, deploy it, and verify it runs correctly via the given environment URL.

Architecture Diagram



Step 1 : Environment Configuration

Environment Information

- **Application Name:** `task09`
    - Logical grouping for your deployments.
- **Environment Name:** `Task09-env`
    - A single running environment under the application.
- **Environment Tier:** *Web server environment*
    - Ideal for web apps responding over HTTP/HTTPS.
- **Application Code:** `wu-node-v1.zip`

    - The uploaded ZIP containing your Node.js app.
- **Platform:**
    - **Node.js 24 running on 64bit Amazon Linux 2023 (Platform ARN shown)**
    - This manages Node.js runtime, NPM installation, health checks, and proxy (Nginx).

Notes

- There are multiple ways you can upload your application either upload the .zip file, use Docker, or push your project files in a public s3 and provide its URL. In this project, the zip file method is used.
- The platform automatically runs `npm install`, so you **don't need node_modules in your ZIP**.
- The platform uses **Nginx as reverse proxy**, so your port must match the proxy expectations (defaults to port 8081/8080 mapping through config).

Step 2 : Service Access

Create IAM roles specific to your configuration and attach them in the beanstak console.

Elastic Beanstalk Service Role

- Create an **IAM Role** for Elastic Beanstalk.
- Attach the following **managed policies**:
    - `AWSElasticBeanstalkEnhancedHealth`
    - `AWSElasticBeanstalkServiceRolePolicy`
- Assign this role in the **Elastic Beanstalk environment configuration** under **Service Access**.

EC2 Instance Profile

- Create an **IAM Instance Profile** for EC2 instances.
- Attach the following **managed policies**:
    - `AWSElasticBeanstalkWebTier`
    - `AWSElasticBeanstalkWorkerTier`
    - `AWSElasticBeanstalkMulticontainerDocker`
- Attach the **Instance Profile** to your Elastic Beanstalk environment.

Attach IAM Roles

Attach the roles you just created.

- **Service Role:**
    - `aws-elasticbeanstalk-service-role-wu`
    - Allows Beanstalk to:

        - Create load balancers
        - Manage auto scaling
        - Pull logs
        - Update configuration
- **EC2 Instance Profile:**
    - `aws-elasticbeanstalk-ec2-role-wu`
    - Allows EC2 instances to:
        - Pull logs to CloudWatch
        - Access Elastic Beanstalk APIs
        - Perform basic system updates

This ensures Elastic Beanstalk and EC2 instances have the necessary permissions to manage resources and interact with AWS services.

EC2 Key Pair

- **Key Pair:** `wu-key`
    - Enables SSH access to EC2 instances (optional but recommended).

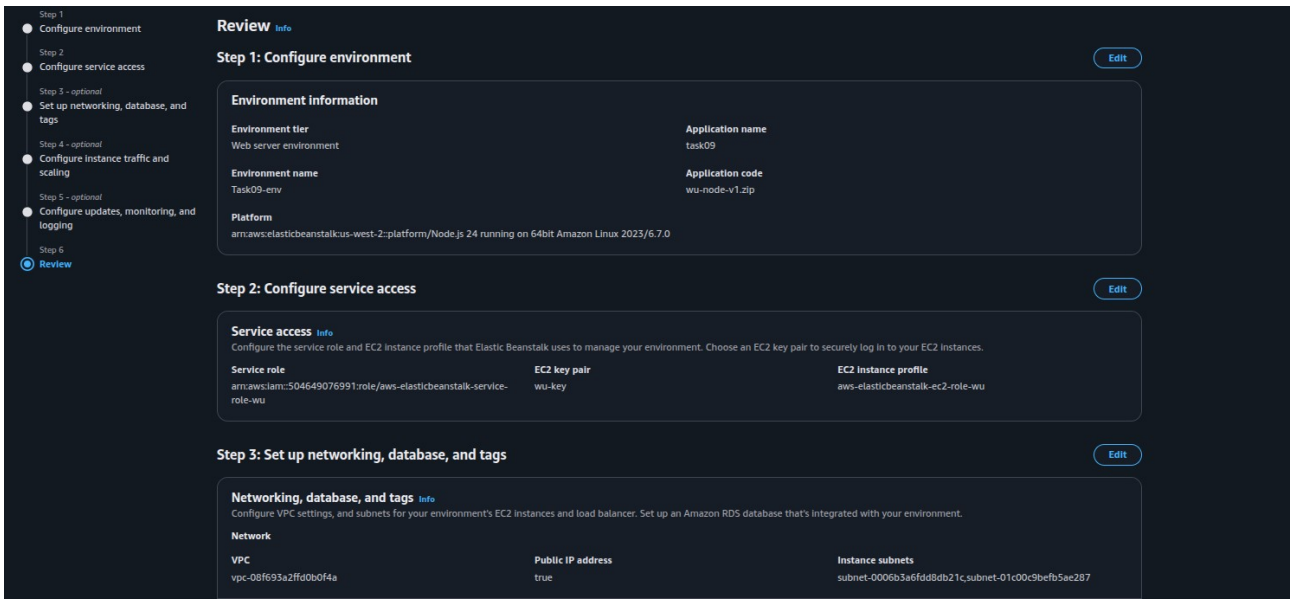
Step 3 : Networking, Database, and Tags

VPC Configuration

Create a vpc with the following configurations

VPC and Subnets

- Create a **VPC** with CIDR block: `10.0.0.0/16`.
- Create **2 public subnets**, each in a different Availability Zone (AZ):
    - **Subnet 1**: `10.0.1.0/24` (e.g., us-west-2a)
    - **Subnet 2**: `10.0.2.0/24` (e.g., us-west-2b)
- Create a **route table** for the public subnets:
    - Add a route to the internet via an **Internet Gateway (IGW)** with destination `0.0.0.0/0`.
    - Associate the route table with both public subnets.
- **VPC Selected:** `vpc-08f693a2ffd0b0f4a`
- **Public IP Assignment:** Enabled
    - Each EC2 instance receives a public IP.
- **Instance Subnets:**
    - `subnet-0006b3a6fdd8db21c`
    - `subnet-01c00c9befb5ae287`
    - Placed in **two different AZs** for high availability.



Step 4 : Instance, Scaling & Load Balancer Configuration

Instance Settings

- **Root Volume**
    - Type: `gp3`
    - Size: `10 GB`
    - IOPS: `3000`
    - Throughput: `125 MiB/s`
- **IMDSv1:** Disabled
    - Only IMDSv2 allowed → more secure.

EC2 Security Groups

Create your custom Sgs for EC2 and ALB and attach them in the Beanstalk console. Use the following configuration

ALB Security Group

- **Inbound Rules**:
    - Allow **HTTP (Port 80)** from `0.0.0.0/0`.
- **Outbound Rules**:
    - Allow all outbound traffic.

EC2 Security Group

- **Inbound Rules**:
    - Allow **HTTP (Port 80)** only from the **ALB Security Group** (use the ALB Security Group ID).
- **Outbound Rules**:
    - Allow all outbound traffic.

This ensures the **ALB can reach your application** while keeping EC2 instances secure by restricting direct access.

Capacity Configuration (Auto Scaling)

Environment Type

- **Load balanced**
    - Uses an ALB to distribute traffic across instances.

Capacity

- **Minimum Instances:** 1
- **Maximum Instances:** 2

Fleet Composition

- On-Demand Base: 0
- On-Demand Above: 70%
- (Good cost-optimized setup)

Scaling

- **Cooldown:** 360s
- **Metric:** `NetworkOut`
    - Average data sent per instance
- **Thresholds:**
    - Scale **up** when > 6,000,000 bytes
    - Scale **down** when < 2,000,000 bytes
- **Period:** 5 min
- **Breach Duration:** 5 min
- **Adjustments:** +1 / -1

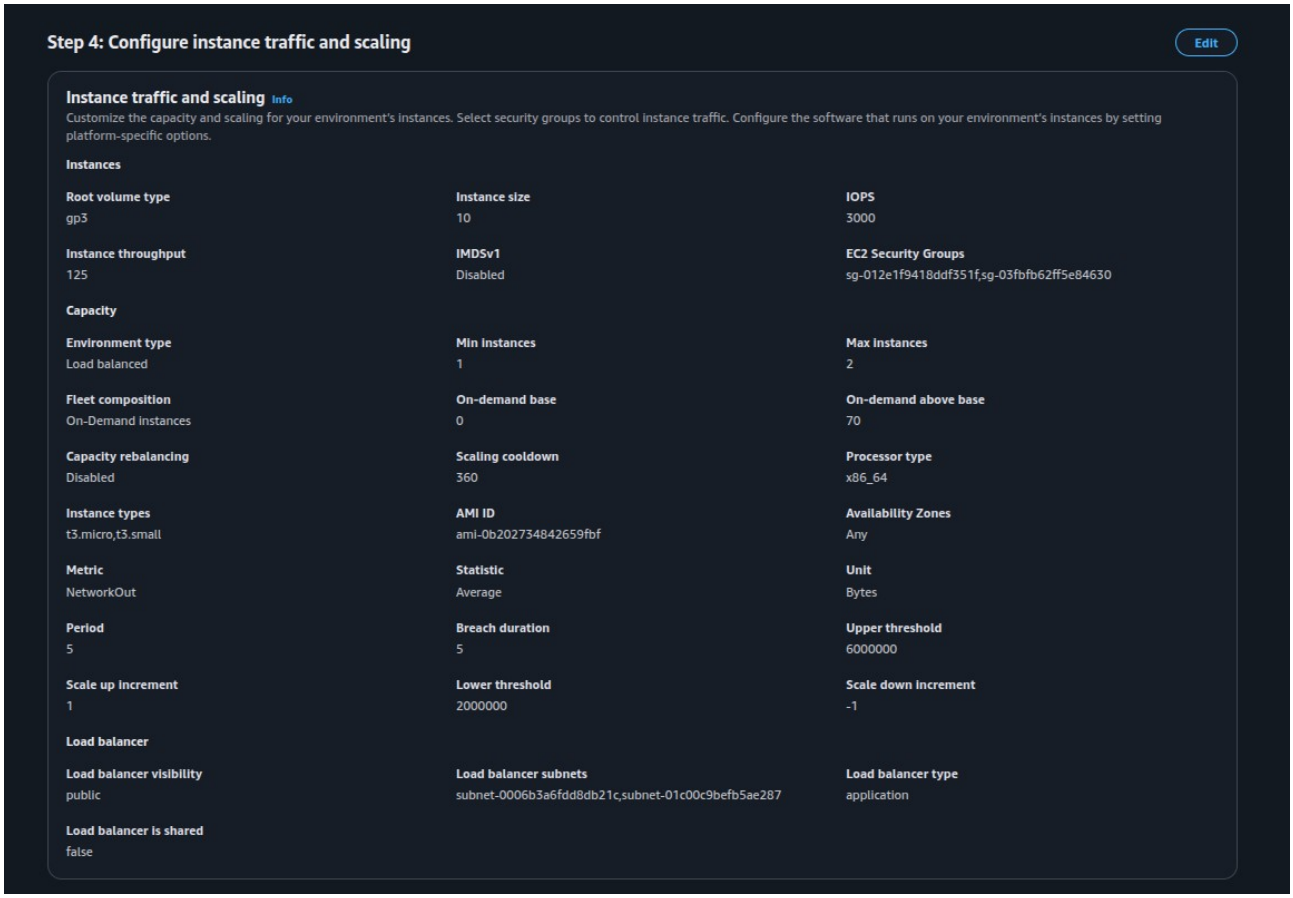
Load Balancer Configuration

General

- **LB Type:** Application Load Balancer (ALB)
- **Visibility:** Public
- **Subnets:**
    - `subnet-0006b3a6fdd8db21c`
    - `subnet-01c00c9befb5ae287`
- **Shared:** No (dedicated to this environment)

Notes

- ALB → EC2 instances via Nginx proxy inside Beanstalk.
- Health checks are done by ALB → Instance → Nginx → Node.js



Step 4: Configure instance traffic and scaling — Edit

Instance traffic and scaling Info

Customize the capacity and scaling for your environment's instances. Select security groups to control instance traffic. Configure the software that runs on your environment's instances by setting platform-specific options.

Instances

Root volume type gp3	**Instance size** 10	**IOPS** 3000
Instance throughput 125	**IMDSv1** Disabled	**EC2 Security Groups** sg-012e1f9418ddf351f,sg-03fbfb62ff5e84630

Capacity

Environment type Load balanced	**Min instances** 1	**Max instances** 2
Fleet composition On-Demand instances	**On-demand base** 0	**On-demand above base** 70
Capacity rebalancing Disabled	**Scaling cooldown** 360	**Processor type** x86_64
Instance types t3.micro,t3.small	**AMI ID** ami-0b202734842659fbf	**Availability Zones** Any
Metric NetworkOut	**Statistic** Average	**Unit** Bytes
Period 5	**Breach duration** 5	**Upper threshold** 6000000
Scale up increment 1	**Lower threshold** 2000000	**Scale down increment** -1

Load balancer

Load balancer visibility public	**Load balancer subnets** subnet-0006b3a6fdd8db21c,subnet-01c00c9befb5ae287	**Load balancer type** application
Load balancer is shared false		

Step 5 : Updates, Monitoring & Logging

Monitoring

- **System Monitoring:** Enhanced
    - Provides detailed health reports, CPU, load, memory.
- **Custom Metrics:** Not configured
    - Optional for advanced monitoring.

Logging

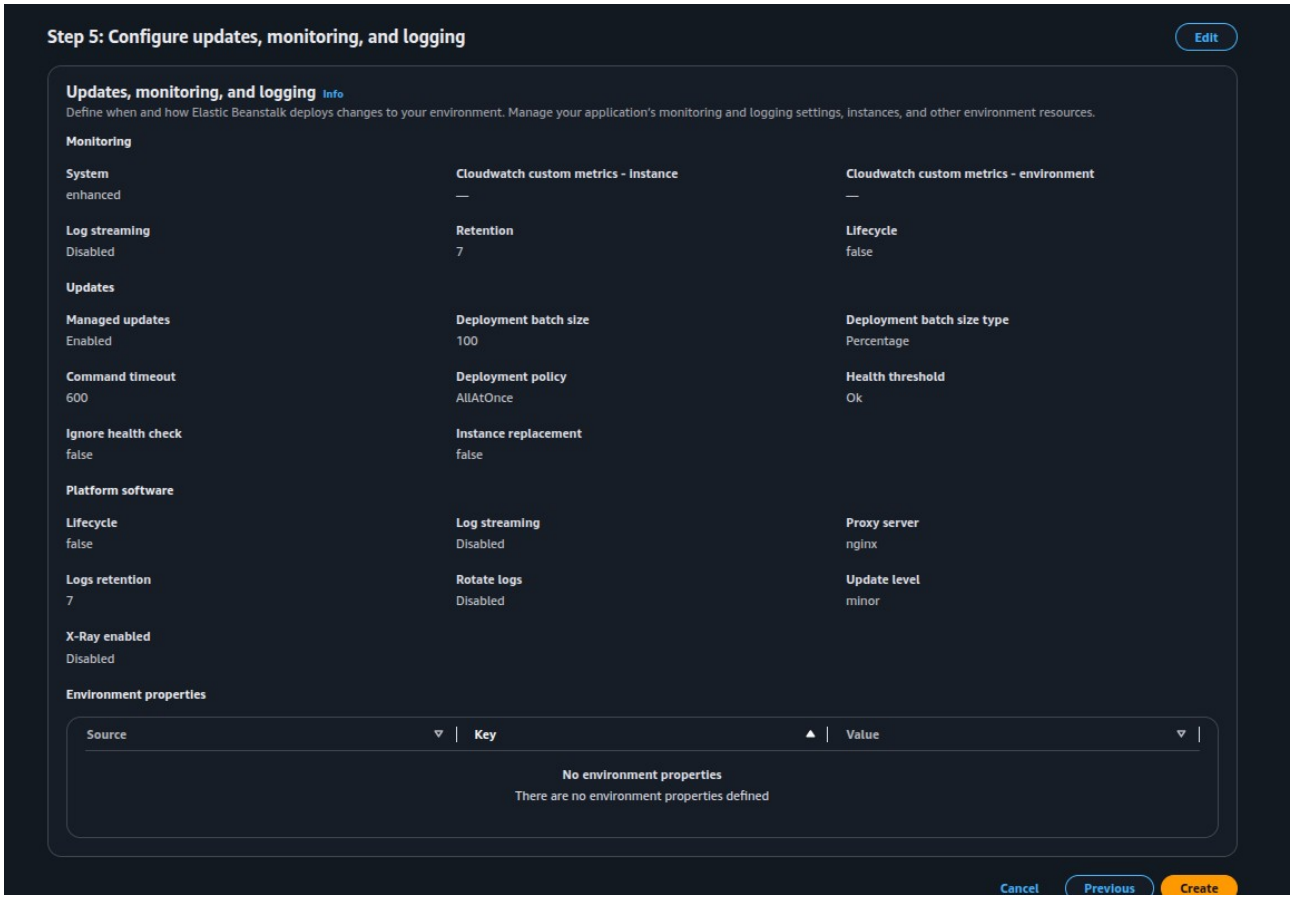
- **Log Streaming:** Disabled
    - Could be enabled to push logs to CloudWatch.
- **Retention:** 7 days
- **Rotate Logs:** Disabled

Updates

- **Managed Updates:** Enabled
    - Automated platform patching.
- **Deployment Policy:** AllAtOnce
- **Batch Size:** 100%
    - Deploy all instances at once.
- **Timeout:** 600 seconds

Platform Software

- **Proxy:** nginx
- **X-Ray:** Disabled
- **Log Retention:** 7 days



Environment Properties

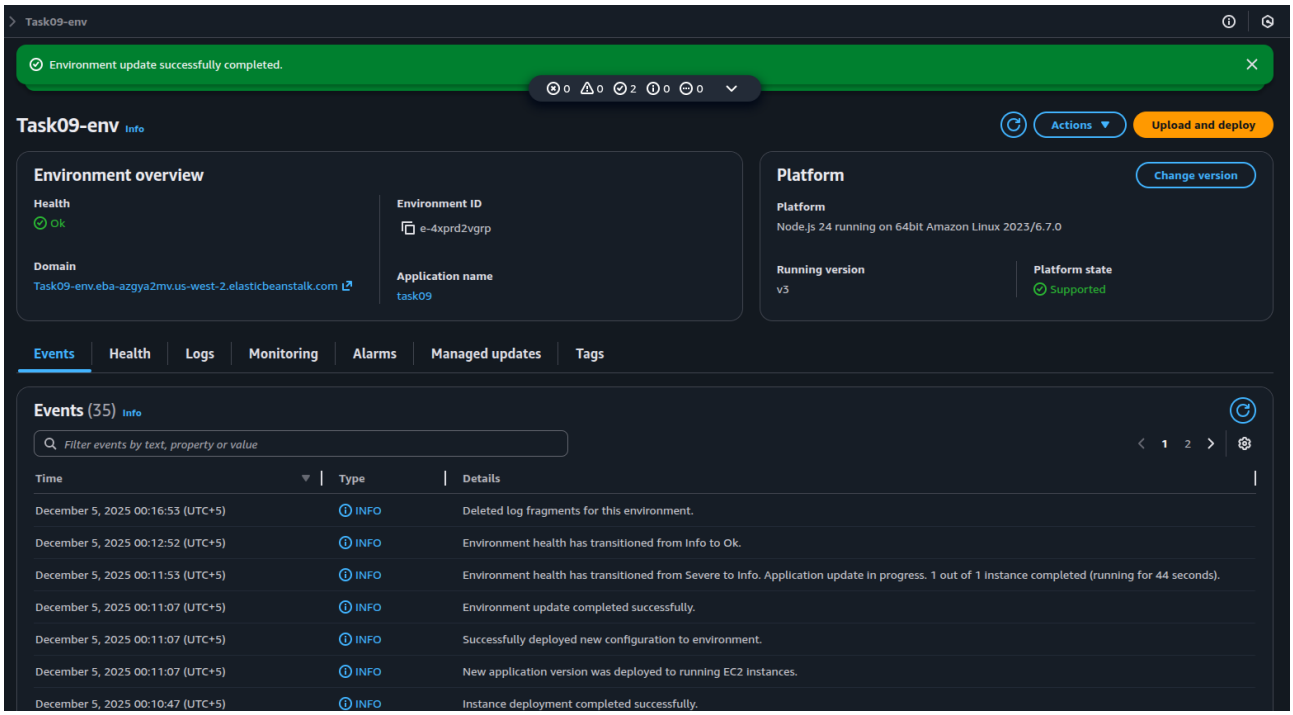
- None defined
    - For Node.js apps, environment variables can be set here (e.g., DB credentials, API keys)

Testing the Deployment

1. Validate Environment Health

From AWS Console → Elastic Beanstalk:

- Check **Environment Health = Green**
- Check **Recent Events**
- Check **EC2 instances running**
- Check **ALB target health**



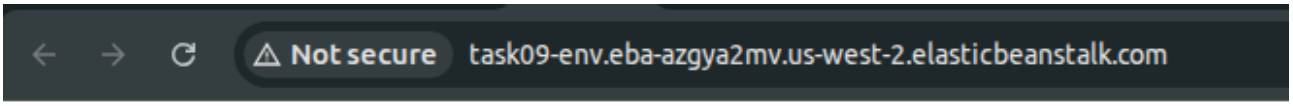
2. Access Application

- Open the **Environment URL**
  Usually something like:

```
http://task09-env.abc123.us-west-2.elasticbeanstalk.com
```

- You should see:

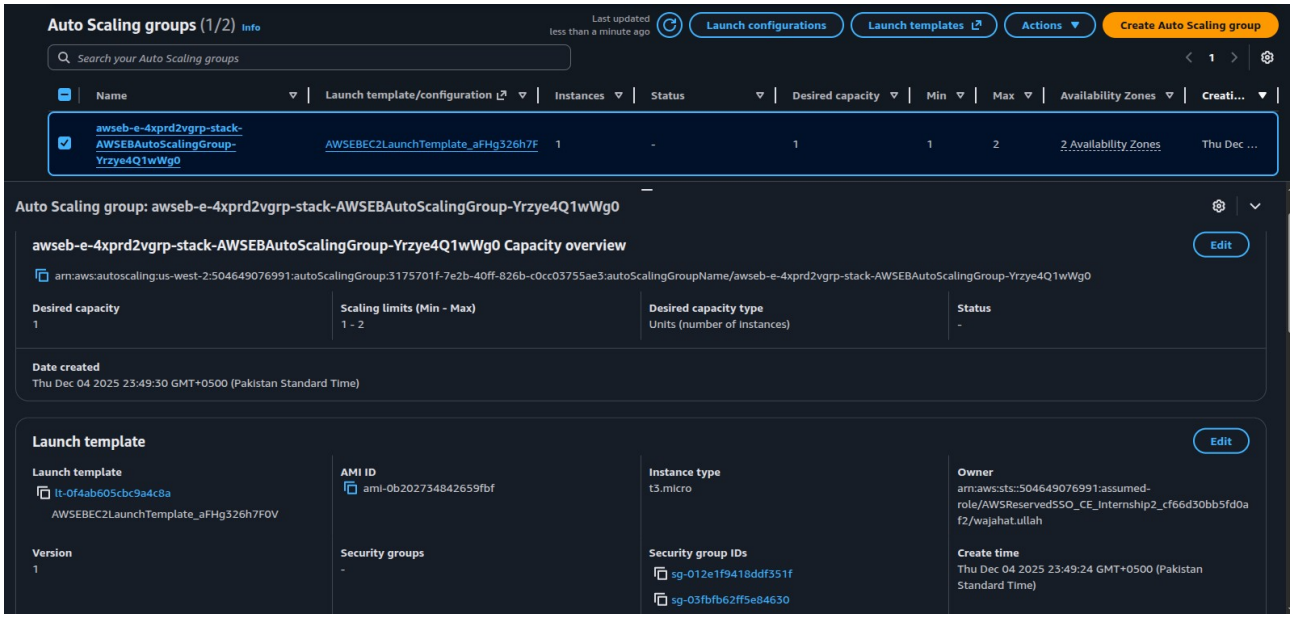
```
Hello World!
```



Hello World!

3. Validate Scaling

- Under EC2 → Auto Scaling Groups
    - Confirm min=1, max=2
    - Confirm ALB is registered



Redeployment (Updating Code)

Steps

1. Make changes to your Node.js code
2. Zip **only the required files** (NO node_modules)

```
index.js
public/
package.json
package-lock.json
```

3. Upload ZIP in:
    - Elastic Beanstalk → Application → Upload and Deploy
4. Watch:
    - Events
    - Health
    - Logs

Verification

- Re-open the environment URL
- Confirm the updated app is running

Welcome to New Application Version!

Conclusion

You successfully deployed a Node.js application using Elastic Beanstalk with:

- A load-balanced, auto-scaled environment
- Node.js running behind Nginx on Amazon Linux 2023
- Auto scaling based on real metrics
- ALB distributing traffic across instances in multiple AZs
- Automated platform & instance management by Elastic Beanstalk

Troubleshooting Guide

Issue 1: Node.js Application Listening on the Wrong Port

Cause

Elastic Beanstalk assigns a **dynamic internal port** (commonly 8081) through the `PORT` environment variable.
Your Node.js app was hardcoded to port **5000**, so NGINX → EC2 → Node.js failed to connect.
This leads to failing health checks, 502 errors, and deployment rollback.

Solution

Modify your application to listen on the port provided by Elastic Beanstalk:

```
const port = process.env.PORT || 5000;
app.listen(port);
```

This ensures the app responds correctly to the load balancer and NGINX proxy.

Issue 2: Incorrect ZIP Package Structure for Deployment

Cause

Elastic Beanstalk expects application source code and dependency definitions only.
Including unnecessary directories like `node_modules` or missing required files can break deployment, cause slow uploads, or prevent EB from running `npm install`.

Solution

ZIP only these essential files:

- `index.js`
- `package.json`
- `package-lock.json`
- `/public` directory
- `app.json`
- `Procfile` (only if needed)

Do NOT include:

- `node_modules/`
  EB installs dependencies itself during deployment.

This keeps deployment clean and prevents build-time errors.

Issue 3: Procfile Parsing Failure

Cause

Elastic Beanstalk attempted to detect a startup command from your Procfile, but the file contained incorrect formatting (extra spaces, Windows line endings, or invalid syntax).
This caused EB to reject the startup configuration and fail environment creation.

Solution

Since your app already uses the standard Node.js structure, the simplest fix is to **remove the Procfile entirely**.
Elastic Beanstalk will automatically run:

```
npm start
```

Issue 4: Incorrect Security Group Configuration Between ALB and EC2

Cause

The EC2 instance must accept traffic **only from the load balancer**, not publicly.
If the EC2 security group doesn't allow inbound port 80 from the ALB's security group, the ALB cannot perform health checks or forward traffic.
This results in unhealthy targets and 502 errors.

Solution

ALB Security Group

- Inbound:
    - HTTP 80 → 0.0.0.0/0
    - HTTPS 443 → 0.0.0.0/0 (optional)
- Outbound: Allow all

EC2 Security Group

- Inbound:
    - HTTP 80 → **from ALB security group only**
- Outbound:
    - Allow all

This ensures the ALB can reach your app while keeping EC2 instances secure.

Issue 5: Load Balancer Attempting to Use Subnets in the Same AZ

Cause

ALBs require **at least two subnets in different Availability Zones** for redundancy.
Attempting to assign multiple subnets from the same AZ leads to this EB error:

```
A load balancer cannot be attached to multiple subnets in the same Availability Zone
```

Solution

Select two **public** subnets in **different AZs**, such as:

- subnet A → us-west-2a
- subnet B → us-west-2b

This satisfies ALB's high availability requirement.

Issue 6: Missing or Incorrect IAM Roles and Permissions

Cause

If the Beanstalk service role or EC2 instance profile is missing required AWS permissions, the environment cannot:

- configure EC2
- pull logs
- report health metrics
- run EB agent commands
- finish provisioning

This can trigger:

```
Instance deployment failed
AWSEBInstanceLaunchWaitCondition timed out
```

Solution

EC2 Instance Profile Must Include atleast:

- `AWSElasticBeanstalkWebTier`
- `AWSElasticBeanstalkWorkerTier`
- `AWSElasticBeanstalkMulticontainerDocker`

Elastic Beanstalk Service Role Must Include:

- `AWSElasticBeanstalkEnhancedHealth`
- `AWSElasticBeanstalkServiceRolePolicy`

Correct roles ensure Elastic Beanstalk can fully manage EC2, logs, scaling, and deployments.