

Elastic Beanstalk Deployment Documentation

Table of Contents

| | |
|---|----|
| Prerequisites..... | 1 |
| Objective..... | 1 |
| Architecture Diagram..... | 2 |
| Step 1 : Environment Configuration..... | 2 |
| Step 2 : Service Access..... | 3 |
| Step 3 : Networking, Database, and Tags..... | 4 |
| Step 4 : Instance, Scaling & Load Balancer Configuration..... | 4 |
| Step 5 : Updates, Monitoring & Logging..... | 6 |
| Testing the Deployment..... | 7 |
| Redeployment (Updating Code)..... | 9 |
| Conclusion..... | 10 |
| Troubleshooting Guide..... | 10 |

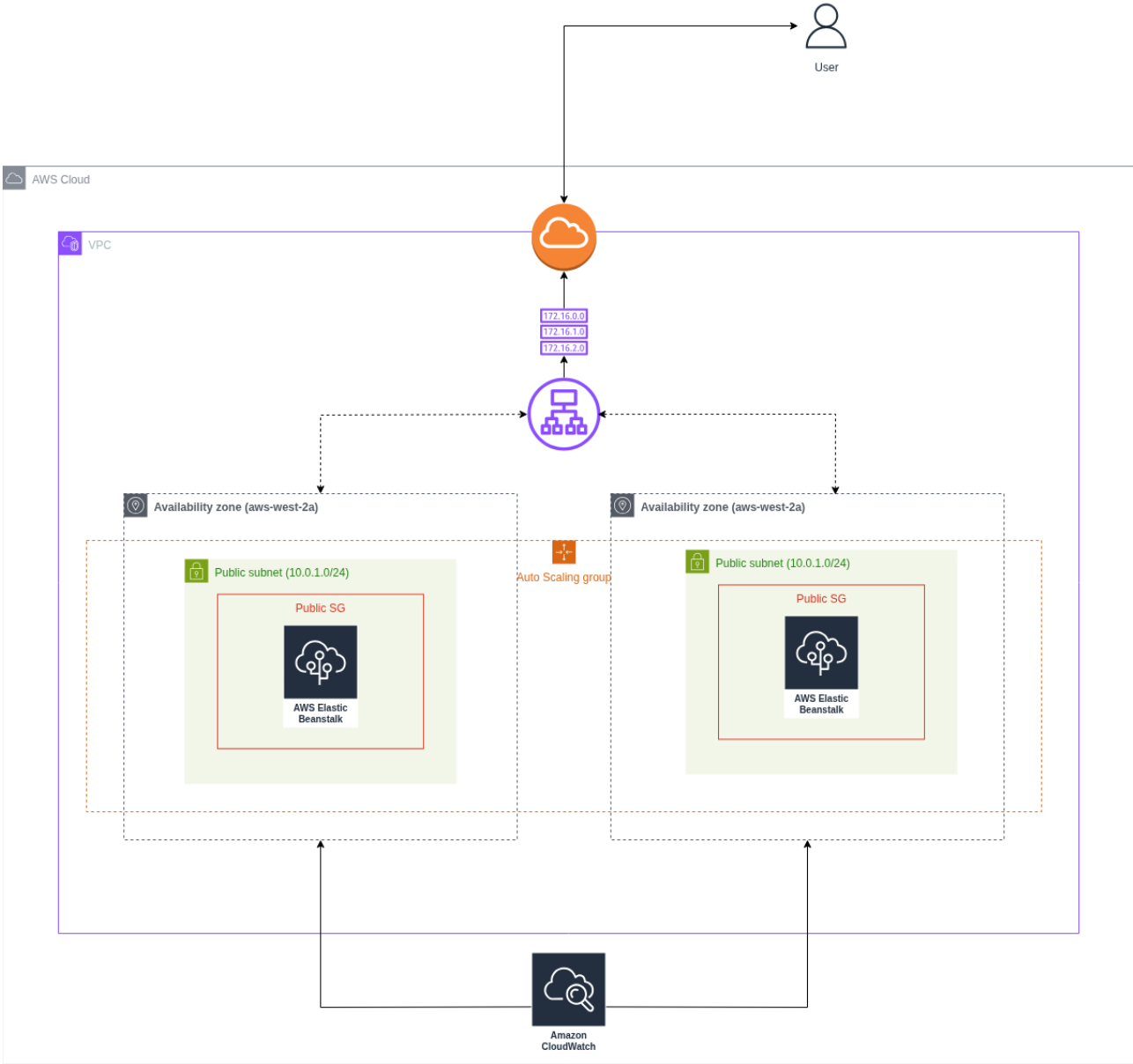
Prerequisites

- 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

IAM Roles

- **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

EC2 Key Pair

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

Step 3 : Networking, Database, and Tags

VPC Configuration

- **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.

The screenshot displays the AWS Elastic Beanstalk console's configuration wizard. On the left, a vertical sidebar lists six steps: 1. Configure environment, 2. Configure service access, 3. Set up networking, database, and tags, 4. Configure instance traffic and scaling, 5. Configure updates, monitoring, and logging, and 6. Review. Step 3 is currently selected and highlighted in blue. The main content area is titled 'Review' and shows the configuration details for the three steps. Step 1, 'Configure environment', includes fields for 'Environment tier' (Web server environment), 'Environment name' (Task09-env), 'Platform' (arn:aws:elasticbeanstalk:us-east-1:platform/Node.js 24 running on 64bit Amazon Linux 2023/6.7.0), 'Application name' (task09), and 'Application code' (wu-node-v1.zip). Step 2, 'Configure service access', shows 'Service role' (arn:aws:iam::504649076991:role/aws-elasticbeanstalk-service-role-wu), 'EC2 key pair' (wu-key), and 'EC2 Instance profile' (aws-elasticbeanstalk-ec2-role-wu). Step 3, 'Set up networking, database, and tags', shows 'VPC' (vpc-08f693a2ffd0b0f4a), 'Public IP address' (true), and 'Instance subnets' (subnet-0006b3a6fdd8db21c, subnet-01c00c9befb5ae287). Each step has an 'Edit' button in the top right corner.

| Step | Configuration |
|---|---|
| Step 1: Configure environment | Environment information Environment tier: Web server environment Environment name: Task09-env Platform: arn:aws:elasticbeanstalk:us-east-1:platform/Node.js 24 running on 64bit Amazon Linux 2023/6.7.0 Application name: task09 Application code: wu-node-v1.zip |
| Step 2: Configure service access | Service access Service role: arn:aws:iam::504649076991:role/aws-elasticbeanstalk-service-role-wu EC2 key pair: wu-key EC2 Instance profile: aws-elasticbeanstalk-ec2-role-wu |
| Step 3: Set up networking, database, and tags | Networking, database, and tags Network: VPC: vpc-08f693a2ffd0b0f4a Public IP address: true Instance subnets: subnet-0006b3a6fdd8db21c, subnet-01c00c9befb5ae287 |

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

- sg-012e1f9418ddf351f

- sg-03fbfb62ff5e84630
- These control incoming traffic to EC2 instances.

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.

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

Capacity

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

Load balancer

| | | |
|--------------------------|---|--------------------|
| Load balancer visibility | Load balancer subnets | Load balancer type |
| public | subnet-0006b3a6fdd8db21c,subnet-01c00c9befb5ae287 | application |

Load balancer is shared

false

- Health checks are done by ALB → Instance → Nginx → Node.js

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

Step 5: Configure updates, monitoring, and logging Edit

Updates, monitoring, and logging Info
Define when and how Elastic Beanstalk deploys changes to your environment. Manage your application's monitoring and logging settings, instances, and other environment resources.

| Monitoring | | |
|---|--|---|
| System enhanced | Cloudwatch custom metrics - Instance — | Cloudwatch custom metrics - environment — |
| Log streaming Disabled | Retention 7 | Lifecycle false |
| Updates | | |
| Managed updates Enabled | Deployment batch size 100 | Deployment batch size type Percentage |
| Command timeout 600 | Deployment policy AllAtOnce | Health threshold Ok |
| Ignore health check false | Instance replacement false | |
| Platform software | | |
| Lifecycle false | Log streaming Disabled | Proxy server nginx |
| Logs retention 7 | Rotate logs Disabled | Update level minor |
| X-Ray enabled Disabled | | |
| Environment properties | | |
| <div> <div>Source ▼</div> <div>Key ▲</div> <div>Value ▼</div> </div> <p>No environment properties There are no environment properties defined</p> | | |

Cancel Previous Create

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**

Task09-env

Environment update successfully completed.

Task09-env Info

Actions Upload and deploy

Environment overview

Health Ok

Domain task09-env.eba-azgya2mv.us-west-2.elasticbeanstalk.com

Environment ID `e-4xprd2vgrp`

Application name `task09`

Platform Change version

Platform Node.js 24 running on 64bit Amazon Linux 2023/6.7.0

Running version `v3`

Platform state Supported

Events (35) Info

Filter events by text, property or value

| Time | Type | Details |
|-----------------------------------|------|--|
| December 5, 2025 00:16:53 (UTC+5) | INFO | Deleted log fragments for this environment. |
| December 5, 2025 00:12:52 (UTC+5) | INFO | Environment health has transitioned from Info to Ok. |
| December 5, 2025 00:11:53 (UTC+5) | INFO | Environment health has transitioned from Severe to Info. Application update in progress. 1 out of 1 instance completed (running for 44 seconds). |
| December 5, 2025 00:11:07 (UTC+5) | INFO | Environment update completed successfully. |
| December 5, 2025 00:11:07 (UTC+5) | INFO | Successfully deployed new configuration to environment. |
| December 5, 2025 00:11:07 (UTC+5) | INFO | New application version was deployed to running EC2 instances. |
| December 5, 2025 00:10:47 (UTC+5) | INFO | Instance deployment completed successfully. |

2. Access Application

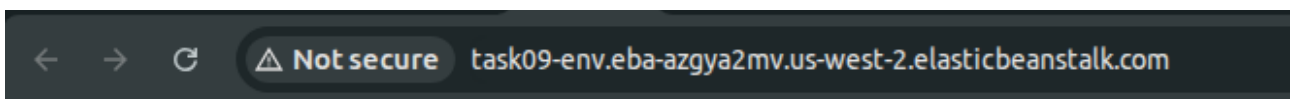
- Open the **Environment URL**

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

The screenshot displays the AWS Management Console interface for an Auto Scaling group. At the top, there's a header for 'Auto Scaling groups (1/2)' with a search bar and navigation buttons like 'Launch configurations', 'Launch templates', and 'Actions'. Below this is a table listing the Auto Scaling groups. The selected group is 'awseb-e-4xprd2vgrp-stack-AWSEBAutoScalingGroup-Yrzye4Q1wWg0'. The details section below the table shows the 'Capacity overview' with a 'Desired capacity' of 1, 'Scaling limits (Min - Max)' of 1 - 2, and 'Desired capacity type' as 'Units (number of instances)'. It also shows the 'Date created' as 'Thu Dec 04 2025 23:49:30 GMT+0500 (Pakistan Standard Time)'. The 'Launch template' section shows the 'Launch template' as 'lt-0f4ab605c94c8a', 'AMI ID' as 'ami-0b202734842659fbf', 'Instance type' as 't3.micro', and 'Security group IDs' as 'sg-012e1f9418ddf351f' and 'sg-03fbfb62ff5e84630'. The 'Owner' is listed as 'arn:aws:sts::504649076991:assumed-role/AWSReservedSSO_CE_Internship2_cf66d30bb5fd0a/f2/wajahat.ullah' and the 'Create time' is 'Thu Dec 04 2025 23:49:24 GMT+0500 (Pakistan Standard Time)'.

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.