Infrastructure Documentation – Elastic Beanstalk Node.js Deployment

Table of Contents

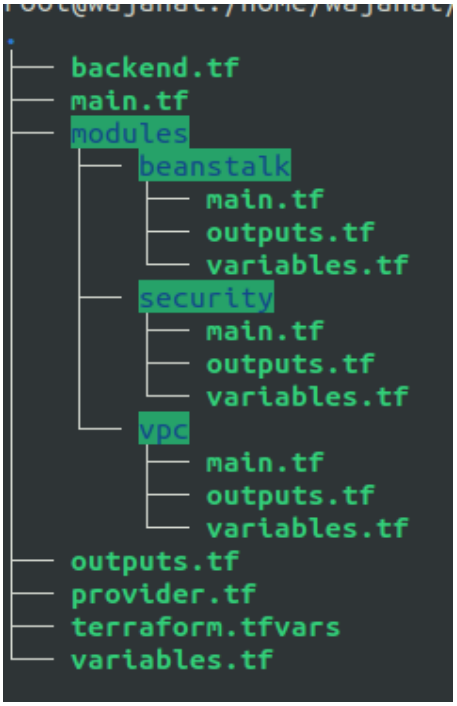
Prerequisites

- 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`)
- Terraform installed.
- S3 backend bucket for terraform state files
- S3 bucket with application zip and versions uploaded.

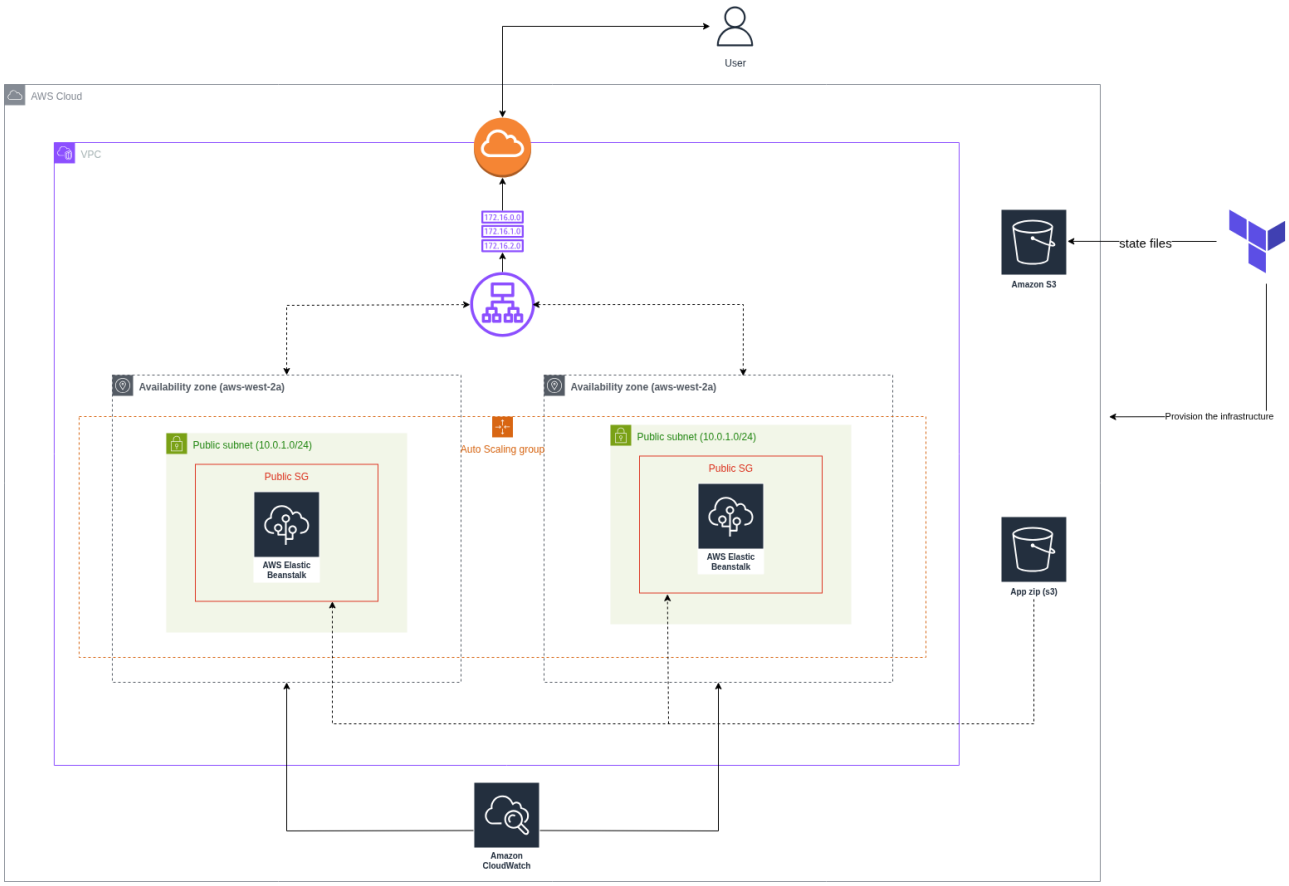
Objective

Deploy a NodeJS application on Elastic Beanstalk using Terraform. Use the s3 bucket for sourcing the application code. Deploy it, and verify it runs correctly via the given environment URL. Provision the entire infrastructure using Terraform.

Project Structure



Architecture Diagram



1. Module Documentation

Module 1 : VPC

This module creates the networking foundation for the Elastic Beanstalk environment.

Resources Created

- **VPC (10.0.0.0/16)**
  Provides isolated networking space for EB resources.
- **Two Public Subnets**
  - Subnet A: 10.0.1.0/24 (us-west-2a)
  - Subnet B: 10.0.2.0/24 (us-west-2b)
    These subnets host:
  - Elastic Beanstalk EC2 instances
  - Elastic Load Balancer (ALB)

- **Internet Gateway (IGW)**
  Required for public internet access.
- **Public Route Table**
  - Adds route `0.0.0.0/0` → IGW
  - Associates with both public subnets

Configuration Logic

- EB in your configuration **only requires public subnets**, not private ones.
- ALB **must** span at least **two subnets in two AZs** for HA.
- EC2 instances require internet access for:
  - NPM install
  - EB agent updates
  - Package downloads

This is the simplest network that satisfies EB requirements without NAT or private networks.

Variables / Outputs

- Variables: `vpc_cidr`, `public_subnets`, `region`, naming variables
- Outputs: `vpc_id`, `public_subnet_ids`

Module 2 : Security

This module creates dedicated security groups for ALB and EB instances.

Resources Created

ALB Security Group (wu-alb-sg)

- **Inbound:**
  - 80 → 0.0.0.0/0 (public access)
- **Outbound:**
  - Allow all (default)

EC2 Instance Security Group (wu-ec2-sg)

- **Inbound:**
  - 80 only from `wu-alb-sg`
    (ensures traffic flows only through the ALB)
- **Outbound:**
  - Allow all

Configuration Logic

- ALB needs to accept requests from the internet.

- EC2 instances must **not** be exposed to the internet directly.
- ALB → EC2 controlled ingress ensures:
    - Security
    - Traceability
    - Compliance

Variables / Outputs

- Variables: `vpc_id`, names
- Outputs: `alb_sg_id`, `ec2_sg_id`

Module 3 : Elastic Beanstalk

This is the **core module**, responsible for deploying the Node.js app.

Resources Created

1. Elastic Beanstalk Application

A logical container for application versions.

`aws_elastic_beanstalk_application`

2. Application Version

This references your ZIP package stored in S3:

`s3://wu-nodejs-s3/v3.zip`

Later deployments point to v4.zip.
EB requires every deployment to be an “application version” stored in S3.

Module Explanation

1. EB Application

```
resource "aws_elastic_beanstalk_application" "this"
```

Purpose:
Creates the *top-level container* for all versions of your Node.js application.

Key points:

- Name comes from `var.app_name`.
- Tags added for identification.
- No runtime is defined here — only the logical application.

2. EB Application Version

```
resource "aws_elastic_beanstalk_application_version" "version"
```

Purpose:
Defines a *versioned release* using ZIP stored in S3.

Key points:

- Loads code from `var.s3_bucket` + `var.s3_key`.
- EB requires each deployment to reference an application version.
- Version label becomes `"${app_name}-v4"` (or v3 depending on the ZIP).
- Switching ZIP later (v3 → v4) automatically triggers a redeploy.

3. EB Environment

```
resource "aws_elastic_beanstalk_environment" "env"
```

Purpose:
Creates the actual running infrastructure:

- EC2 instances
- Auto Scaling Group
- ALB
- Health checks
- Deployment behavior

3.1 Basic environment setup

- Uses `solution_stack_name` → Node.js 24 on Amazon Linux 2023.
- Connects to EB Application + Application Version.

3.2 Scaling settings

```
aws:autoscaling:launchconfiguration → InstanceType
aws:autoscaling:asg → MinSize / MaxSize
```

Controls how many EC2 instances run and what type they are.

```
setting {
  namespace = "aws:autoscaling:launchconfiguration"
  name      = "InstanceType"
  value     = var.instance_types[0]
}

setting {
  namespace = "aws:autoscaling:asg"
  name      = "MinSize"
  value     = var.min_size
}

setting {
  namespace = "aws:autoscaling:asg"
  name      = "MaxSize"
  value     = var.max_size
}
```

3.3 Security groups

```
SecurityGroups  → EC2 SG
SecurityGroups  → ALB SG
```

Attaches:

- EC2 instances → internal SG
- ALB → public SG

Ensures traffic flows **ALB → EC2 only**.

```
# Assign security groups
setting {
  namespace = "aws:autoscaling:launchconfiguration"
  name      = "SecurityGroups"
  value     = var.ec2_sg_id
}

setting {
  namespace = "aws:elb:loadbalancer"
  name      = "SecurityGroups"
  value     = var.alb_sg_id
}
```

3.4 Subnet placement

```
aws:ec2:vpc → Subnets
aws:ec2:vpc → ELBSubnets
```

Places:

- EC2 instances in both public subnets

- Load balancer in same two AZs (required for HA)

```
# Subnets for ALB and EC2 instances
setting {
  namespace = "aws:ec2:vpc"
  name      = "Subnets"
  value     = join(",", var.public_subnets)
}

setting {
  namespace = "aws:ec2:vpc"
  name      = "ELBSubnets"
  value     = join(",", var.public_subnets)
}
```

3.5 IAM

```
IamInstanceProfile → EC2 profile
ServiceRole        → Beanstalk service role
EC2KeyName         → SSH key
```

These provide:

- Permissions for EB to manage resources
- Permissions for EC2 to install packages
- SSH access if needed

```
# IAM roles
setting {
  namespace = "aws:elasticbeanstalk:environment"
  name      = "ServiceRole"
  value     = var.service_role_arn
}

setting {
  namespace = "aws:autoscaling:launchconfiguration"
  name      = "IamInstanceProfile"
  value     = var.instance_profile
}

setting {
  namespace = "aws:autoscaling:launchconfiguration"
  name      = "EC2KeyName"
  value     = var.keypair
}
```

4. Root Module

The root module ties together all submodules.

Workflow

- Calls `vpc` module → gets VPC + subnets
- Calls `security` module → gets SGs
- Calls `beanstalk` module → deploys EB infra
- Provides variable wiring (no logic inside root)

a. terraform.tfvars

This file defines all variable values required by modules.

Typical contents:

```
region              = "us-west-2"
app_name            = "wu-beanstalk-app"
env_name            = "wu-beanstalk-env"
s3_bucket           = "wu-nodejs-s3"
s3_key              = "v3.zip"
instance_profile    = "aws-elasticbeanstalk-ec2-role-wu"
service_role        = "aws-elasticbeanstalk-service-role-wu"
vpc_cidr            = "10.0.0.0/16"
public_subnets      = ["10.0.1.0/24", "10.0.2.0/24"]
environment_tag     = "sandbox"
```

This maps directly to variables consumed by modules.

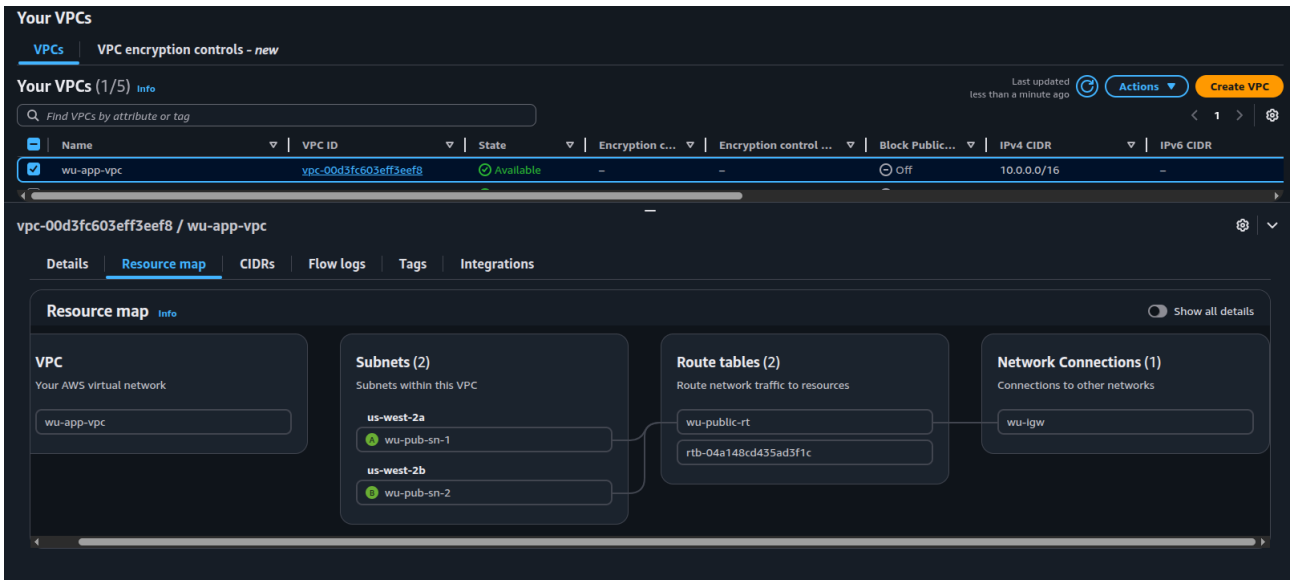
When you change `s3_key` to `"v4.zip"` → Terraform creates a new EB version & redeploys.

5. Testing & Validation (Short)

After `terraform apply`:

1. Verify VPC

- VPC exists
- Two public subnets in 2 AZs
- Route table → IGW route exists

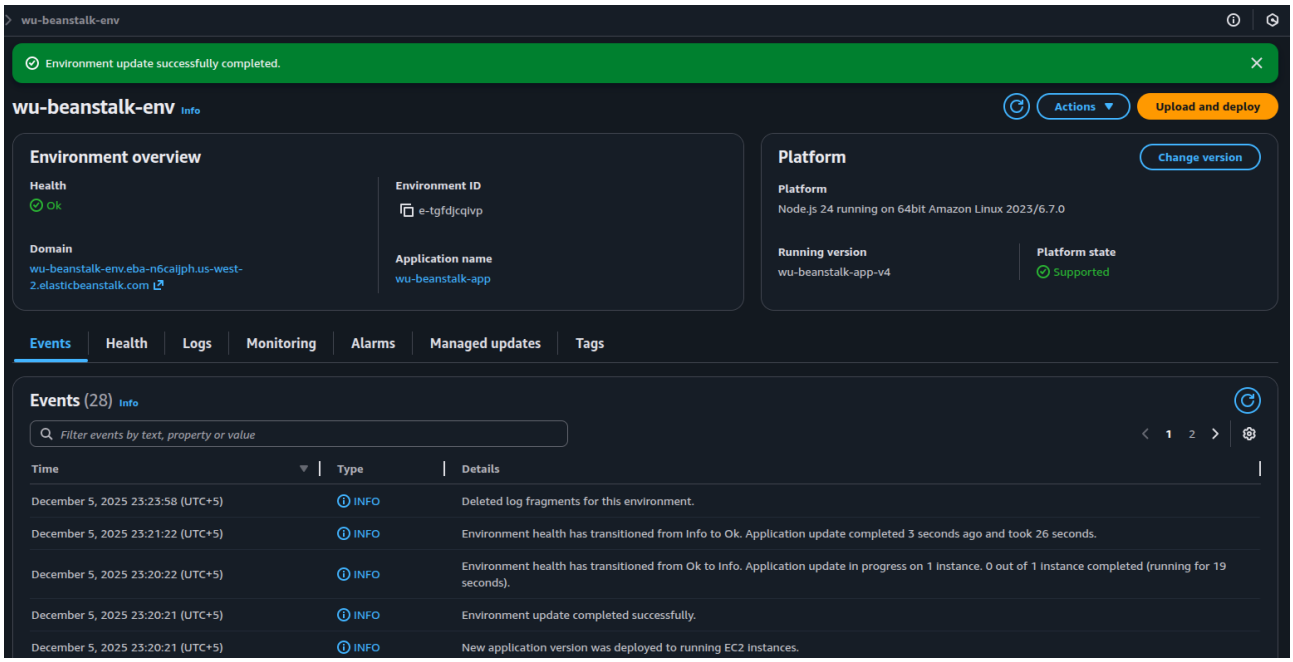


2. Verify Security Groups

- ALB SG: inbound 80 from anywhere
- EC2 SG: inbound 80 from ALB SG only

3. Verify Beanstalk Deployment

- Application
- Application versions (v3)
- Environment status = **Healthy**
- ALB created
- EC2 instance launched



4. Test Application

Go to:

`http://<env-cname>.elasticbeanstalk.com`

You should see your Node.js response.

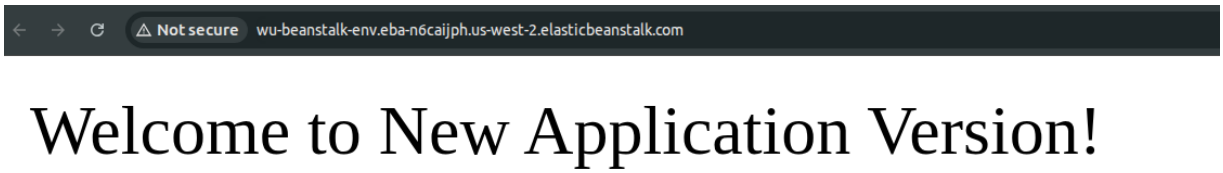
5. Redeploy New Version

- Change `s3_key = "v4.zip"`
- Run:

  ```
  terraform plan
  ```

  ```
  terraform apply
  ```

- Validate new version appears in EB console
- Check ALB URL again



Conclusion

This Terraform setup:

- Builds a clean VPC suitable for EB
- Deploys a Node.js application to Beanstalk using S3 application versions
- Implements ALB, scaling, monitoring, and health checks
- Makes future deployments trivial (change S3 key → apply)
- Ensures security by isolating EC2 instances behind ALB
- Reproduces console configuration exactly in IaC form