STEP-BY-STEP IMPLEMENTATION

Step 1: Setting Up the VPC — shloka-canary-task-vpc (Oregon, 10.0.0.0/16)

What was done:

A dedicated VPC was created with four subnets (two public, two private), routing tables, an Internet Gateway, and a NAT Gateway to support a highly available canary deployment architecture.

Why was it done:

Canary deployments require reliable, isolated network environments where private EC2 instances run behind an ALB.

- **Public subnets** → ALB + Bastion
- **Private subnets** → Application servers (v1 and v2)

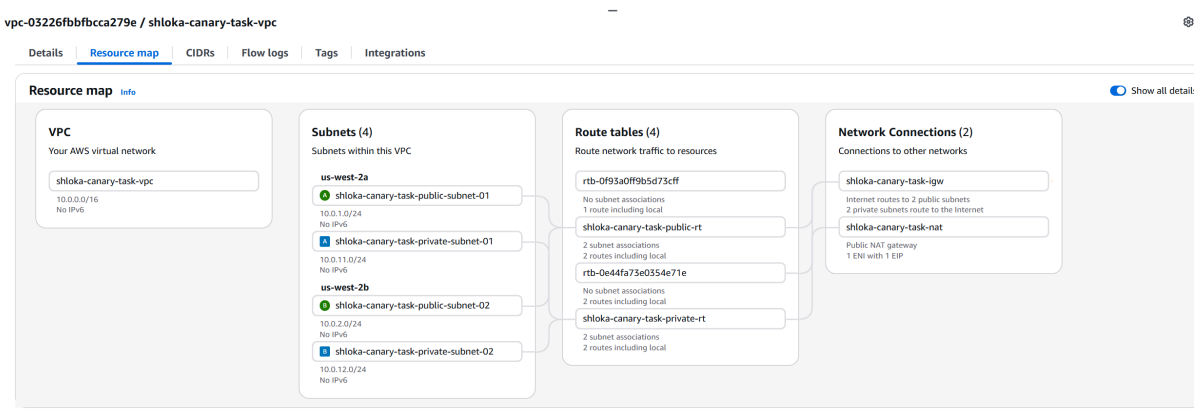
  NAT + IGW are needed for controlled internet access while keeping servers private.

How was it done:

- Created a VPC: `shloka-canary-task-vpc` with CIDR **10.0.0.0/16**
- Created four subnets:

Subnet Name	AZ	CIDR	Type
shloka-canary-task-public-subnet-01	us-west-2a	10.0.1.0/24	Public
shloka-canary-task-private-subnet-01	us-west-2a	10.0.11.0/24	Private
shloka-canary-task-public-subnet-02	us-west-2b	10.0.2.0/24	Public
shloka-canary-task-private-subnet-02	us-west-2b	10.0.12.0/24	Private

- Created route tables:
  - `shloka-canary-task-public-rt` → route to **IGW**
  - `shloka-canary-task-private-rt` → route to **NAT Gateway**
- Attached:
  - Internet Gateway → `shloka-canary-task-igw`
  - NAT Gateway → `shloka-canary-task-nat`
- Associated all subnets to correct RTs.



VPC resource map

Step 2 - Launching EC2 Instances (Version 1 — Stable App Servers)

What was done:

Launched three EC2 instances:

- One **Bastion host** in public subnet
- Two **application servers (v1)** in private subnets

  Nginx was installed and configured to serve Version 1 of the app.

Why was it done:

- Bastion → secure SSH access into private instances
- v1 instances → represent the *current production* version
- These servers form the “old TG” for canary comparison

How was it done:

Instances launched:

- `shloka-canary-task-bastion` → public-subnet-01
- `shloka-canary-task-server01` → private-subnet-01
- `shloka-canary-task-server02` → private-subnet-02

User data used to configure Nginx (Version 1 HTML):

```
#!/bin/bash
# version 1

yum -y update
yum -y install nginx

systemctl enable nginx
systemctl start nginx

cat << 'EOF' > /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Shloka Canary Page v1</title>
    <style>
        body {
            margin: 0;
            padding: 0;
            height: 100vh;
            background: linear-gradient(135deg, #0f2027, #203a43, #2c5364);
            font-family: "Inter", Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            color: #fff;
        }
```

```
        .card {
            background: rgba(255,255,255,0.1);
            padding: 50px 70px;
            border-radius: 18px;
            backdrop-filter: blur(8px);
            box-shadow: 0 10px 35px rgba(0,0,0,0.3);
            text-align: center;
        }
        h1 {
            font-size: 3rem;
            margin: 0 0 10px 0;
            font-weight: 700;
            letter-spacing: 1px;
        }
        p {
            margin: 0;
            font-size: 1.25rem;
            opacity: 0.9;
        }
        .tag {
            margin-top: 20px;
            display: inline-block;
            padding: 6px 14px;
            background: rgba(255,255,255,0.2);
            border-radius: 12px;
            font-size: 0.9rem;
            letter-spacing: 0.5px;
        }
    </style>
</head>
<body>
    <div class="card">
        <h1>Canary Deployment</h1>
        <p>Instance is live and serving.</p>
        <div class="tag">Version 1</div>
    </div>
```

```
</body>
</html>
EOF

systemctl restart nginx
```

Step 3 - Creating Target Group for Version 1

What was done:

Created `shloka-canary-task-v1-TG` and registered both private server instances.

Why was it done:

This target group will serve **90% traffic** during canary deployment and acts as the stable production environment.

How was it done:

- Created Target Group → name: `shloka-canary-task-v1-TG`
- Protocol: HTTP 80
- Health check path: `/`
- Registered:
  - server01
  - server02

Step 4 - Creating the Application Load Balancer

What was done:

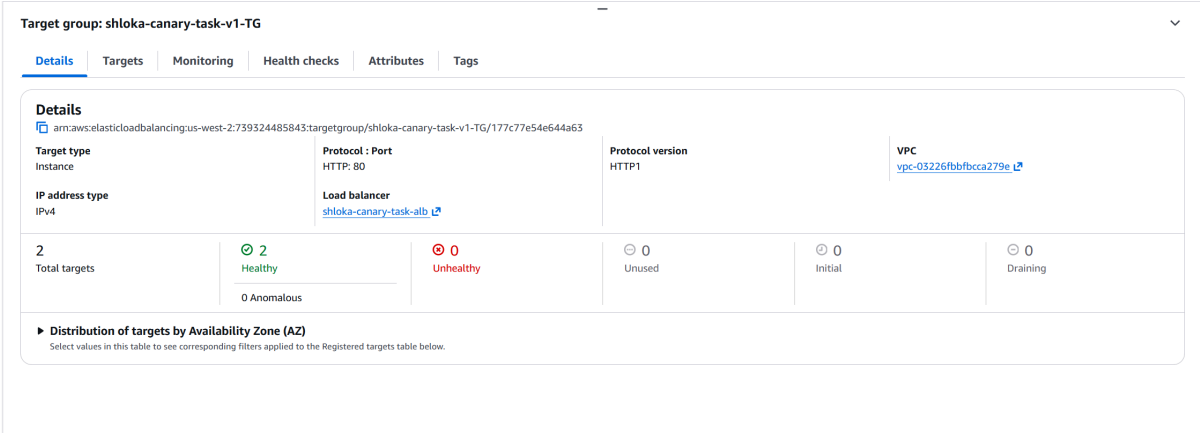
Created ALB with listeners, attached public subnets, configured SG, and forwarded traffic to TG-v1.

Why was it done:

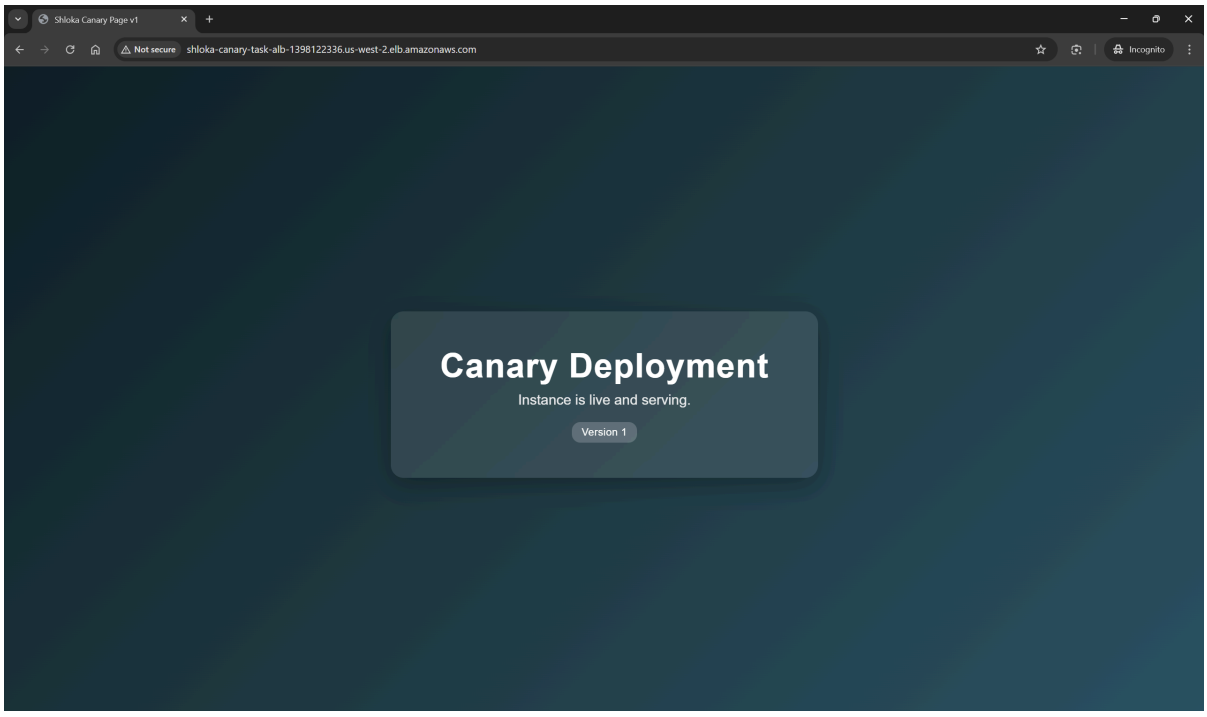
The ALB distributes traffic across v1 and (later) v2 instances and enables weighted canary deployment.

How was it done:

- ALB name → `shloka-canary-task-alb`
- Subnets → both public subnets
- Security group → allow inbound **HTTP:80**
- Default action → forward to `shloka-canary-task-v1-TG`
- Updated private instance SG → allow inbound **80 from ALB SG**



Healthy Instances - SUCCESS



Web page loading on ALB DNS - SUCCESS

Step 5 - Launching Version 2 Instance (Canary Instance)

What was done:

Launched a separate EC2 instance configured to serve Version 2 of the application.

Why was it done:

This instance represents the **canary version**, receiving only 10% of ALB traffic to safely test changes

How was it done:

Instance name → `shloka-canary-task-server-test`

- Launched in private-subnet-01
- User data for Nginx serving **Version 2** HTML:

```
#!/bin/bash
# version 2

yum -y update
yum -y install nginx

systemctl enable nginx
systemctl start nginx

cat << 'EOF' > /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Shloka Canary Page v2</title>
    <style>
        body {
            margin: 0;
            height: 100vh;
```

```
        background: radial-gradient(circle at center, #ffe29f, #ffa99f, #f
f719a);
        font-family: "Inter", Arial, sans-serif;
        display: flex;
        justify-content: center;
        align-items: center;
        color: #222;
      }
      .panel {
        background: rgba(255,255,255,0.7);
        padding: 45px 65px;
        border-radius: 20px;
        backdrop-filter: blur(10px);
        box-shadow: 0 8px 25px rgba(0,0,0,0.2);
        text-align: center;
      }
      h1 {
        margin: 0 0 12px;
        font-size: 2.8rem;
        font-weight: 700;
      }
      p {
        margin: 0;
        font-size: 1.2rem;
        opacity: 0.85;
      }
      .version {
        margin-top: 25px;
        display: inline-block;
        padding: 8px 18px;
        background: #222;
        color: #fff;
        border-radius: 14px;
        font-size: 0.95rem;
      }
    </style>
```

```
</head>
<body>
  <div class="panel">
    <h1>Canary Deployment</h1>
    <p>Instance is running updated build.</p>
    <div class="version">Version 2</div>
  </div>
</body>
</html>
EOF

systemctl restart nginx
```

Step 6 - Creating Version 2 Target Group

What was done:

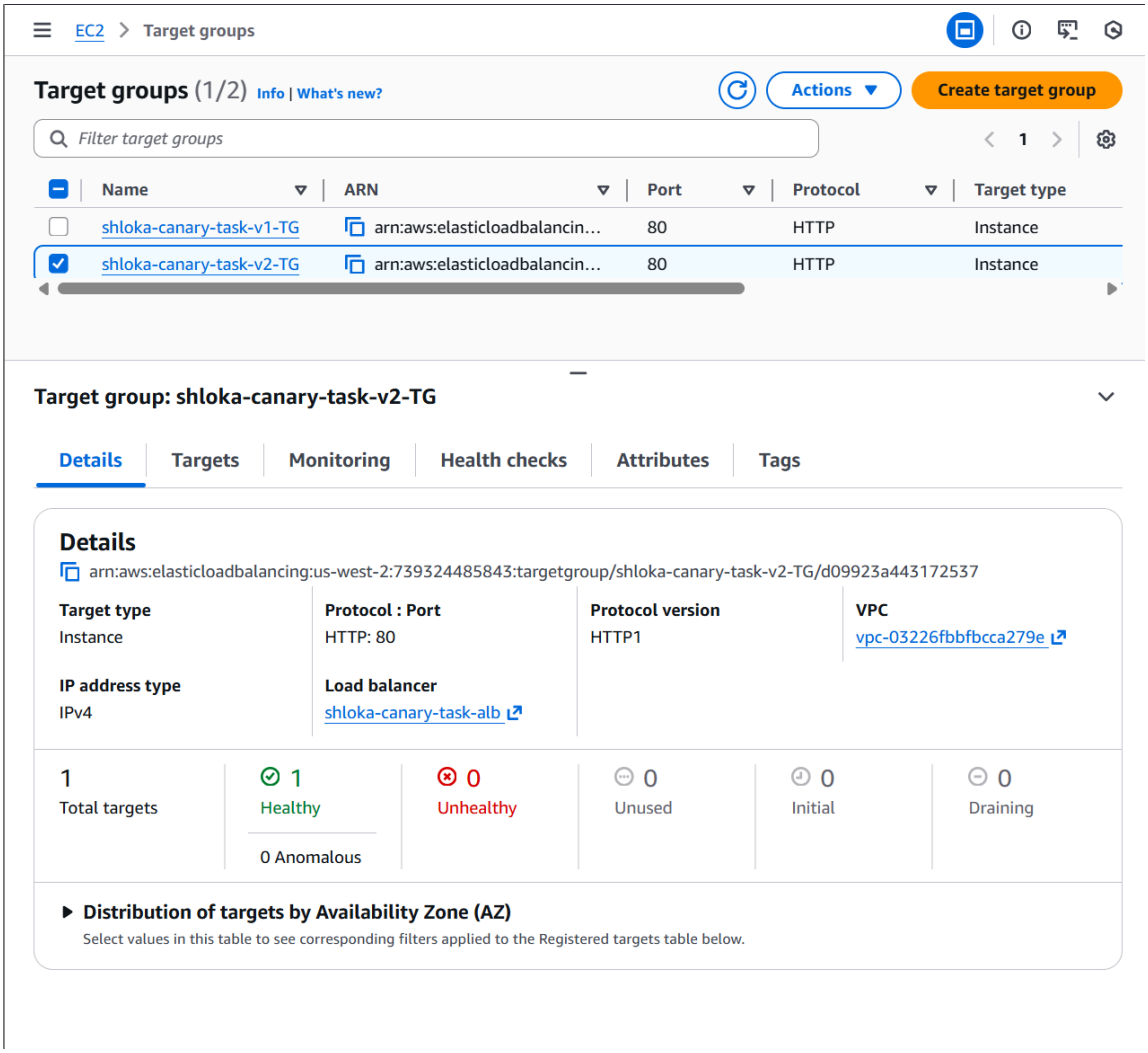
A new target group `shloka-canary-task-v2-TG` was created and the v2 instance was attached.

Why was it done:

This TG is used in the weighted ALB listener rule to route 10% traffic to the new build of the app.

How was it done:

- TG type: instance-based
- Protocol: HTTP
- Health check: `/`
- Registered canary instance (`server-test`)



Healthy instance verification of TG v2

Step 7 - Editing ALB Listener Rules for Canary Deployment

What was done:

Weighted forwarding was configured so the ALB sends 90% traffic to v1 and 10% to v2.

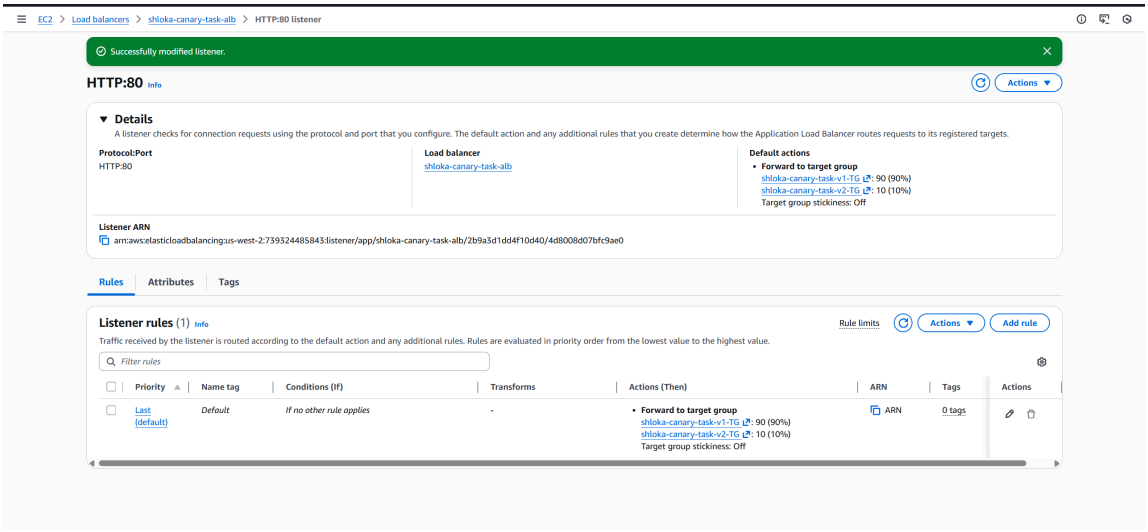
Why was it done:

To perform a safe canary deployment where only a fraction of real traffic hits the new version before full rollout.

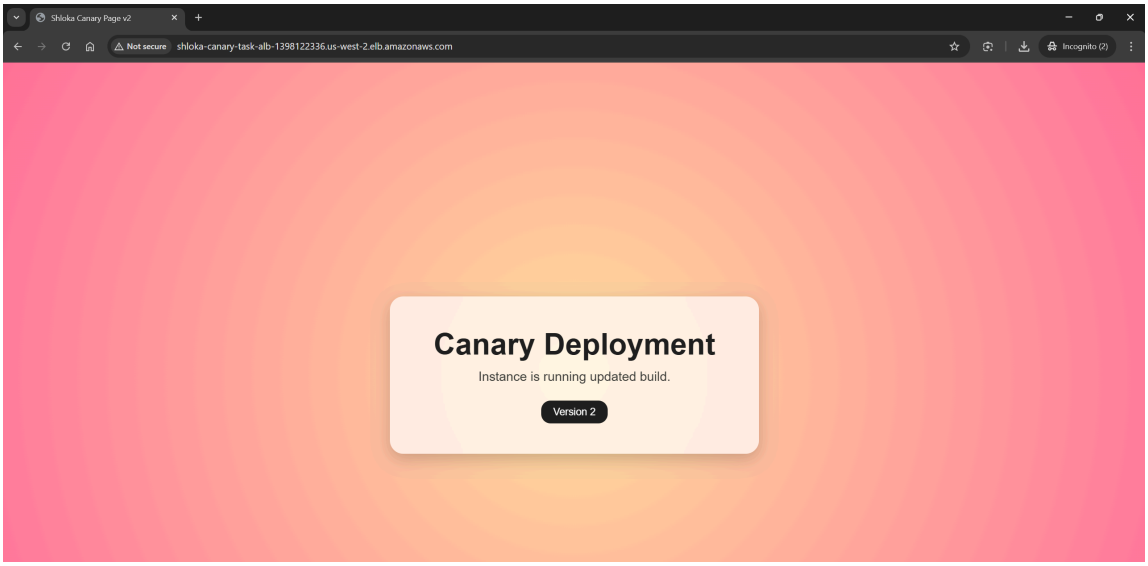
How was it done:

- Edited **default rule**

- Changed action:
  - v1-TG → 90%
  - v2-TG → 10%



Updated ALB listening rules



verifying canary deployment after multiple refreshes

Step 8 - Setting Up S3 Bucket for ALB Access Logs

What was done:

Created an S3 bucket + added necessary bucket policy for ALB to write logs.

Why was it done:

To validate the canary setup by checking real distribution of requests across both target groups.

How was it done:

- Created bucket: `shloka-alb-logs-canary-task`
- Added required bucket policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::797873946194:root"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::shloka-alb-logs-canary-task/AWSLogs/
*"
    },
    {
      "Sid": "AWSLogDeliveryAclCheck",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "s3:GetBucketAcl",
      "Resource": "arn:aws:s3:::shloka-alb-logs-canary-task"
    }
  ]
}
```

- In ALB → **Attributes → Access Logs**

  Added bucket → enabled logging.

Step 9 - Validating Canary Traffic Distribution Using ALB Access Logs

What was done:

Analyzed ALB access logs stored in S3 to confirm that traffic was successfully distributed between Version 1 (v1) and Version 2 (v2) target groups according to the 90/10 canary weighting.

Why was it done:

A canary deployment must **prove** that:

- Only a controlled fraction of traffic reached the canary version
- Most traffic remained on the stable version

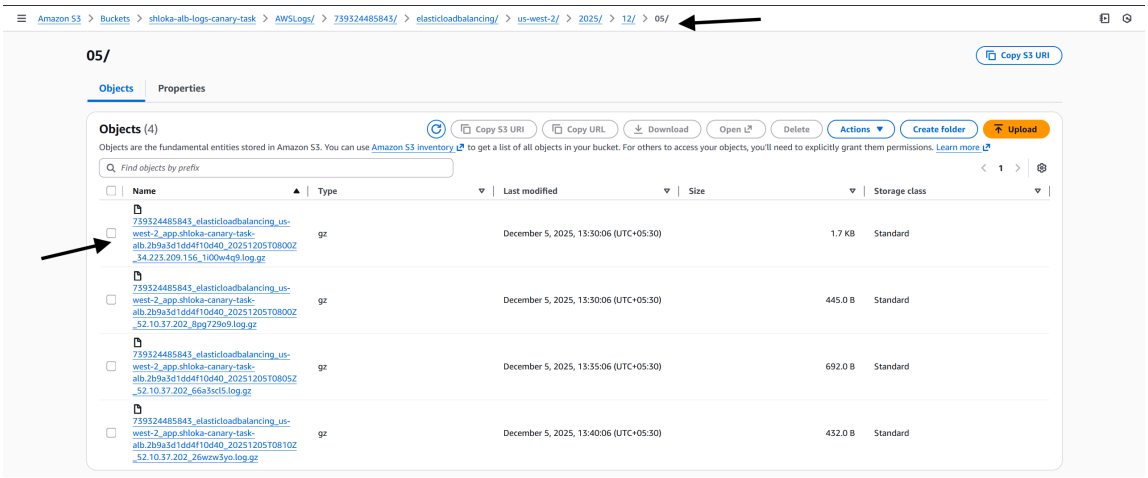
  ALB access logs provide irrefutable backend evidence of request routing.

This serves as:

- Deployment validation
- Documentation for the assignment
- Safety confirmation before rolling out fully to v2

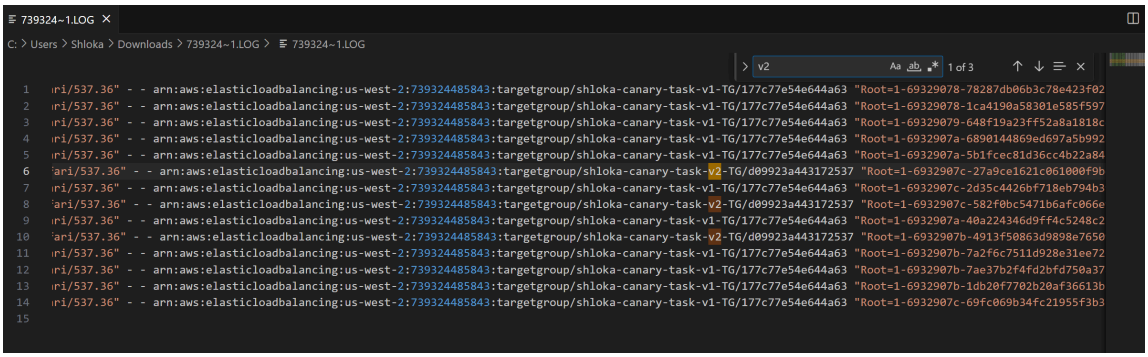
How was it done:

- After enabling logging, generated traffic by refreshing ALB DNS repeatedly.
- Navigated to:



breadcrumb path and log file in s3

- Opened each `.log.gz` file → used S3 “Open” to downloaded locally.
- Inside each log entry, searched for v2:



Step 10 - Implementing the Rollback Plan (Reset Traffic to 100% v1)

What was done:

Performed a full rollback by updating the ALB listener rules to send **100% traffic** to the stable Version 1 target group and **0% to Version 2**.

Why was it done:

A rollback plan is crucial in production-style canary deployments.

If performance issues, bugs, or instability are detected in the canary build, teams must immediately shift all traffic back to the stable version.

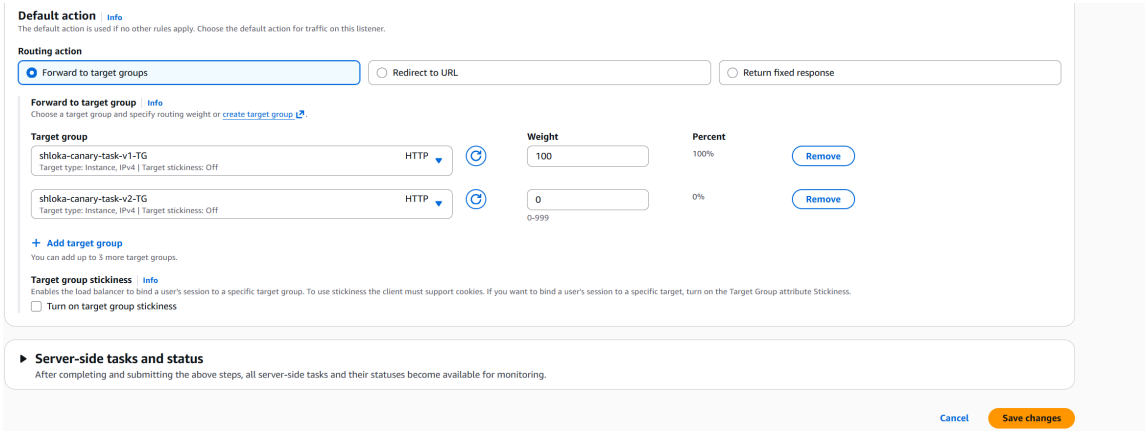
Rollback ensures:

- Zero downtime
- No impact on end users
- Rapid recovery without any infrastructure teardown

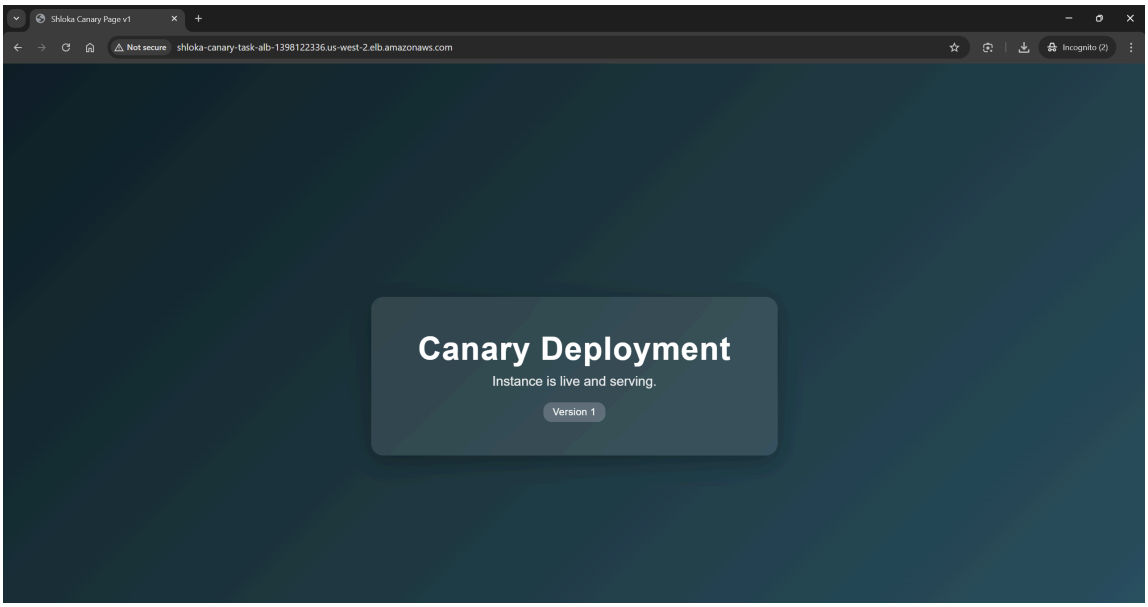
How was it done:

Modified weighted forwarding:

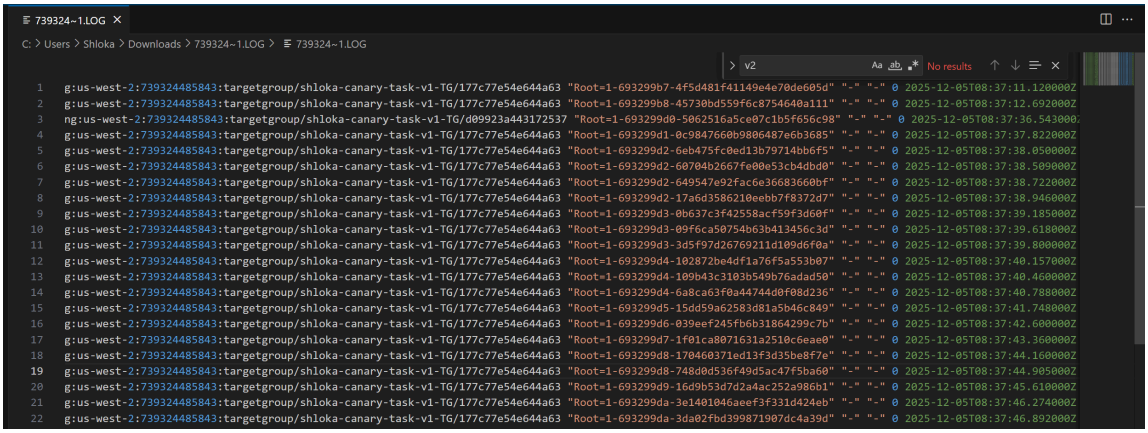
- Changed **v1-TG → 100%**
- Changed **v2-TG → 0%**
- Saved rule changes
- Waited a few seconds for ALB propagation
- Validated rollback:
  - Refreshed ALB DNS repeatedly
  - Verified all responses were Version 1
  - Checked ALB logs (optional) confirming all requests routed to v1



updated listening rules



showing v1 only. SUCCESSFULLY CHANGED Listening rules



updated logs after changes no v2