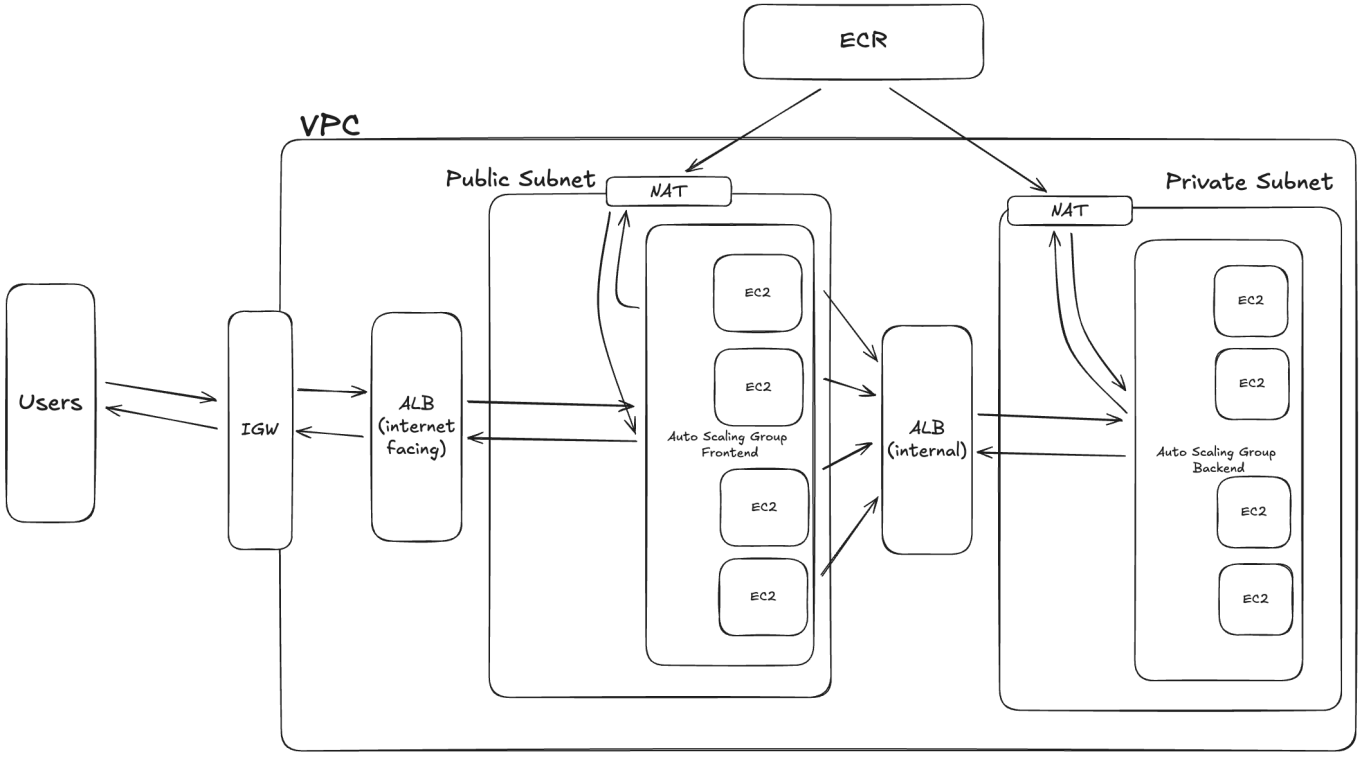
This guide, we'll see how to deploy a production ready weather application on AWS with enterprise grade architecture. We'l' use **Auto Scaling Groups** and **Application Load Balancers** for high availability, **public-private subnets** for security isolation, and **Amazon ECR** for containerized deployments. To make it production complete, we'll configure a custom domain with **Route 53** and enable HTTPS using **AWS Certificate Manager**. The result? A fully scalable, secure web application accessible via your own domain with SSL encryption exactly how professional applications should be deployed.

GitHub Repository: https://github.com/Shreyas-Yadav/weather-aws

Clone the above repo and continue.

Our weather application uses a multi-tier architecture on AWS that separates the frontend and backend for better security and scalability.



Step 1: Creating the VPC

What is a VPC?

A VPC (Virtual Private Cloud) is your isolated network in AWS. Think of it as your own private data center in the cloud where all your servers and resources will live securely.

Setting Up the VPC

Navigate to VPC Creation

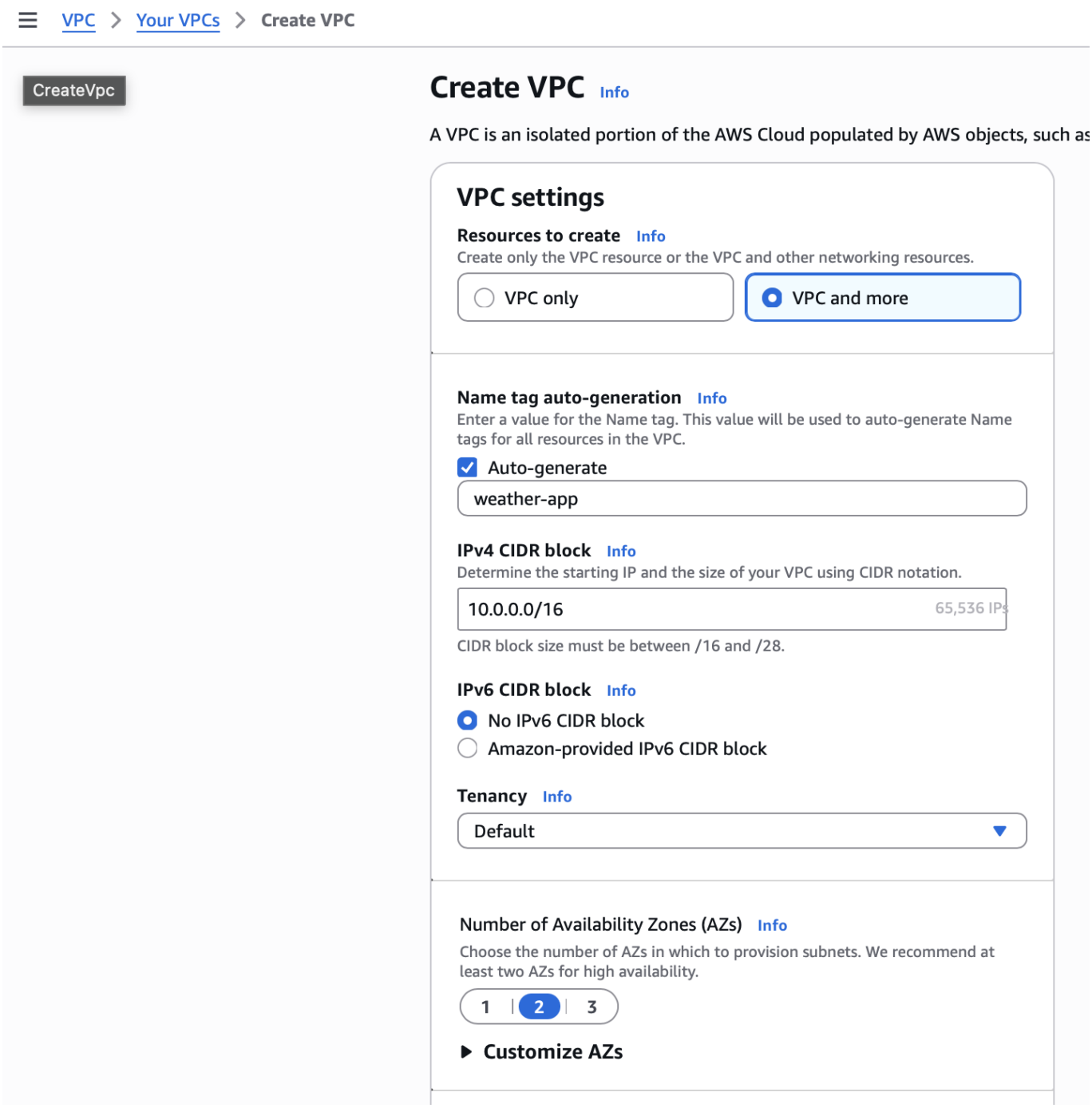
Go to AWS Console → Search "VPC" → Click "Your VPCs" → Click **"Create VPC"**

Choose "VPC and more"

You'll see two options:

- **VPC only** - Manual setup (tedious)
- **VPC and more** - Automatic setup(recommended)

Select **"VPC and more"** - AWS will automatically create all networking components for you.



Basic Configuration

Name tag auto-generation: `weather-app` This prefix will be added to all resources, making them easy to identify.

IPv4 CIDR block: `10.0.0.0/16` This is your VPC's IP address range (65,536 available IPs).

IPv6: Select "No IPv6 CIDR block" **Tenancy:** Keep as "Default"

Availability Zones and Subnets

Number of Availability Zones: Select **2** This is crucial for high availability - if one data center fails, your app continues running in the other.

Number of public subnets: 2 These will host your frontend servers (internet-accessible)

Number of private subnets: 2 These will host your backend servers (hidden from internet)

Subnet CIDR blocks: Keep the defaults:

```
Public subnet 1a:  10.0.0.0/20
Public subnet 1b:  10.0.16.0/20
Private subnet 1a: 10.0.128.0/20
Private subnet 1b: 10.0.144.0/20
```

NAT Gateways

Select **"1 per AZ"** (creates 2 total)

What's a NAT Gateway? It allows your private backend servers to access the internet (for API calls, updates) while staying hidden from incoming internet traffic.

Why 2 NAT Gateways? High availability each private subnet gets its own NAT Gateway for redundancy.

Additional Settings

VPC endpoints: None **DNS options:** Enable both

- Enable DNS hostnames
- Enable DNS resolution

Create VPC

Click **"Create VPC"** and wait 2-3 minutes.

AWS will create:

- 1 VPC
- 4 Subnets (2 public, 2 private)
- 1 Internet Gateway
- 2 NAT Gateways
- Route tables

Step 2: Creating Security Groups

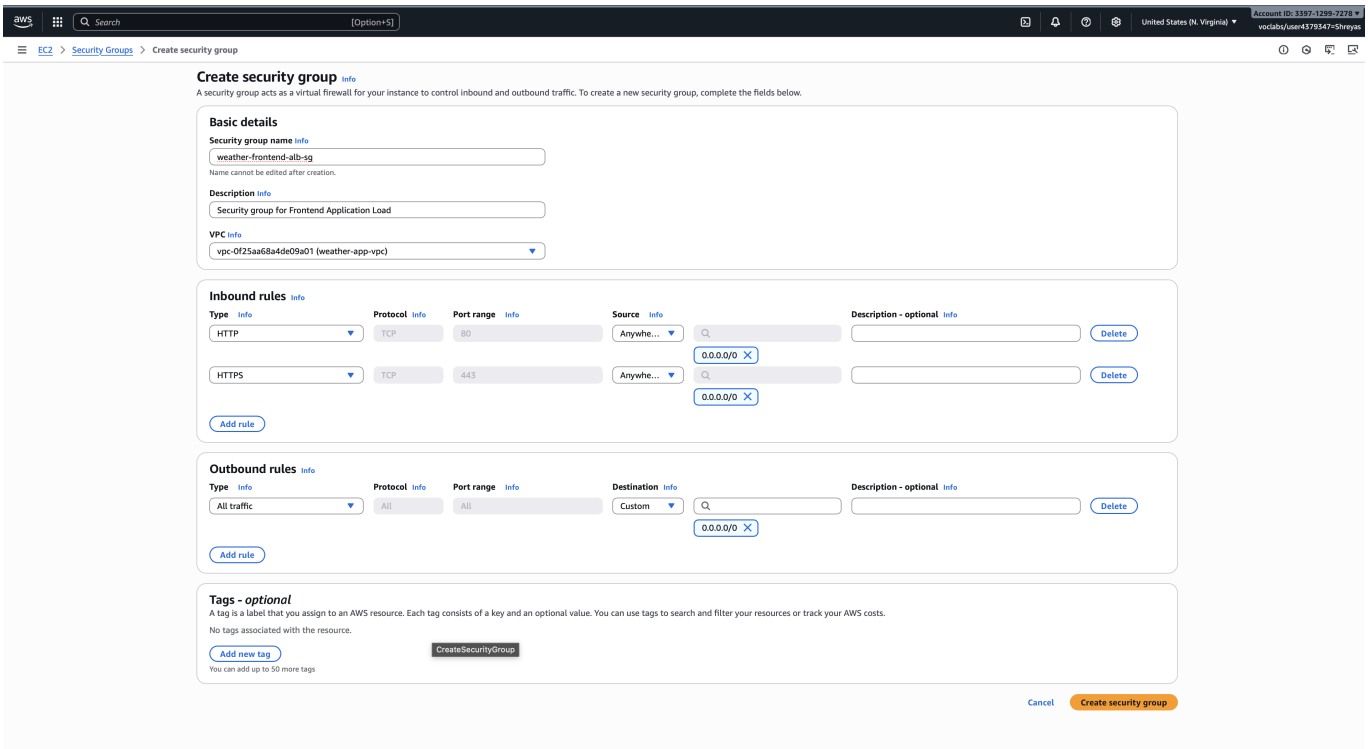
What are Security Groups?

Security Groups are virtual firewalls that control inbound and outbound traffic to your AWS resources.

We'll create 4 security groups to control traffic between components.

Security Group 1: Frontend ALB Security Group

Navigate: EC2 → Security Groups → Create security group




```
Name: weather-frontend-alb-sg
Description: Security group for Frontend Application Load Balancer
VPC: weather-app-vpc
```

Inbound Rules:

- HTTP (80) from 0.0.0.0/0
- HTTPS (443) from 0.0.0.0/0

Outbound Rules: All traffic (default)

Security Group 2: Frontend EC2 Security Group

```
Name: weather-frontend-sg
Description: Security group for Frontend EC2 instances
VPC: weather-app-vpc
```

Inbound Rules:

- HTTP (80) from Custom → `weather-frontend-alb-sg`
- SSH (22) from My IP (optional for debugging)

Outbound Rules: All traffic (default)

Security Group 3: Backend ALB Security Group

```
Name: weather-backend-alb-sg
Description: Security group for Backend Application Load Balancer
VPC: weather-app-vpc
```

Inbound Rules:

- HTTP (80) from Custom → `weather-frontend-sg`

Outbound Rules: All traffic (default)

Security Group 4: Backend EC2 Security Group

```
Name: weather-backend-sg
Description: Security group for Backend EC2 instances
VPC: weather-app-vpc
```

Inbound Rules:

- Custom TCP (3000) from Custom → `weather-backend-alb-sg`
- SSH (22) from My IP (optional for debugging)

Outbound Rules: All traffic (default)

Traffic Flow

```
Internet → Frontend ALB (80,443) → Frontend EC2 (80) → Backend ALB (80) →
Backend EC2 (3000)
```.

Step 3: IAM Role for EC2 Instances.

Note for AWS Academy/Lab Users
If you're using **AWS Academy Labs** or **AWS Learner Lab**, you cannot
create custom IAM roles. Instead, use the pre-configured role provided:

Role name: `LabRole`

This role already has the necessary permissions for ECR, Parameter Store,
and other AWS services.

When creating EC2 instances or Launch Templates, select **LabRole** as the
IAM instance profile.

For Regular AWS Accounts

````
If you're using a personal/company AWS account (not AWS Labs), follow
these steps to create a custom IAM role:

**Navigate:** IAM → Roles → Create role

**Step 1: Select Trusted Entity**
````

Trusted entity type: AWS service Use case: EC2

````


Click **Next**

**Add Permissions**

Search and select these 2 managed policies:

1. **AmazonEC2ContainerRegistryReadOnly**
2. **AmazonSSMManagedInstanceCore**

Click **Next**

**Name and Create**
````

Role name: weather-ec2-role Description: IAM role for Weather App EC2 instances

````


Click **Create role**

**Add Parameter Store Policy**

1. Click on **weather-ec2-role** → **Permissions** tab
2. Click **Add permissions** → **Create inline policy** → **JSON** tab
```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameter",
        "ssm:GetParameters"
      ],
      "Resource": "arn:aws:ssm:*:*:parameter/weather-app/*"
    }
  ]
}
````

3. Policy name: `ParameterStoreReadPolicy`
````

4. Click **Create

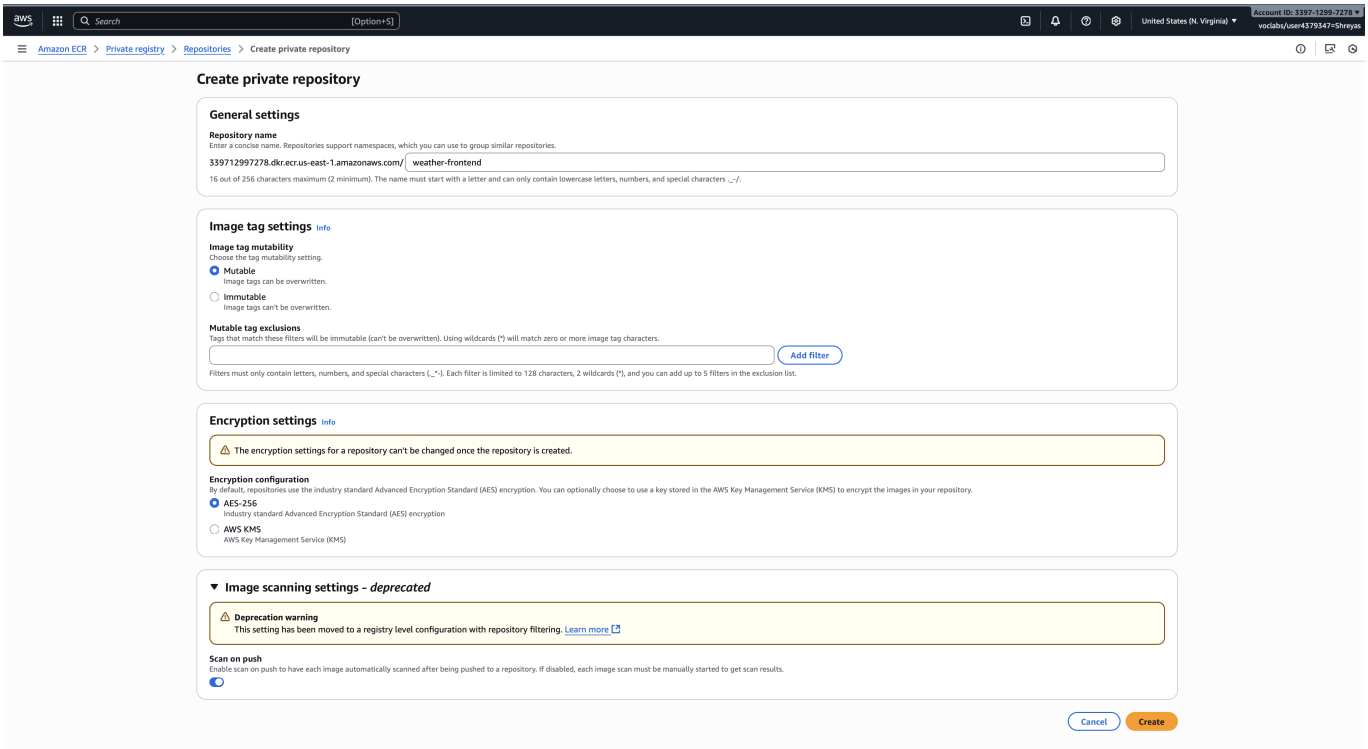
Step 4: Creating ECR Repositories

What is Amazon ECR?

Amazon Elastic Container Registry (ECR) is a Docker container registry where we'll store our frontend and backend Docker images. EC2 instances will pull images from here during deployment.

Create ECR Repositories

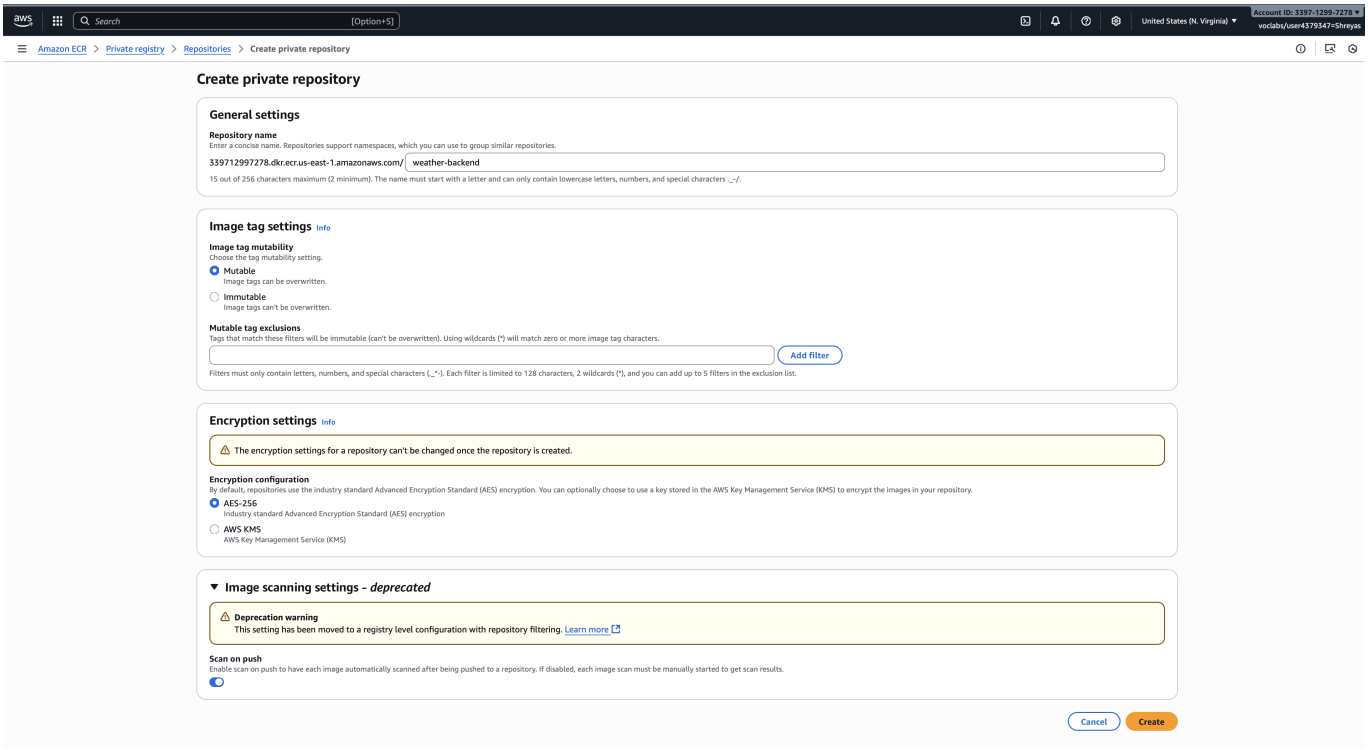
Navigate: Amazon ECR → Repositories → Create repository



Repository 1: Frontend

```
Repository name: weather-frontend
Image tag mutability: Mutable
Encryption: AES-256 (default)
Scan on push: Enable (optional)
```

Click **Create repository**



Repository 2: Backend

```
Repository name: weather-backend
Image tag mutability: Mutable
Encryption: AES-256 (default)
Scan on push: Enable (optional)
```

Click **Create repository**

Note Your Repository URIs

After creation, note down both repository URIs (format: `account-id.dkr.ecr.region.amazonaws.com/repository-name`):

```
Frontend URI: 339712997278.dkr.ecr.us-east-1.amazonaws.com/weather-frontend
Backend URI: 339712997278.dkr.ecr.us-east-1.amazonaws.com/weather-backend
```

You'll need these URIs to push Docker images and configure EC2 instances.

Step 5: Build and Push Docker Images to ECR

Prerequisites

Ensure you have installed:

- AWS CLI (latest version)
- Docker

Build Docker Images

Important: Build images compatible with EC2 architecture (Linux AMD64/ARM64).

Navigate to your project directory and build both images:

Build Backend:

```
docker buildx build --platform linux/amd64,linux/arm64 -t weather-backend
--load ./backend
```

Build Frontend:

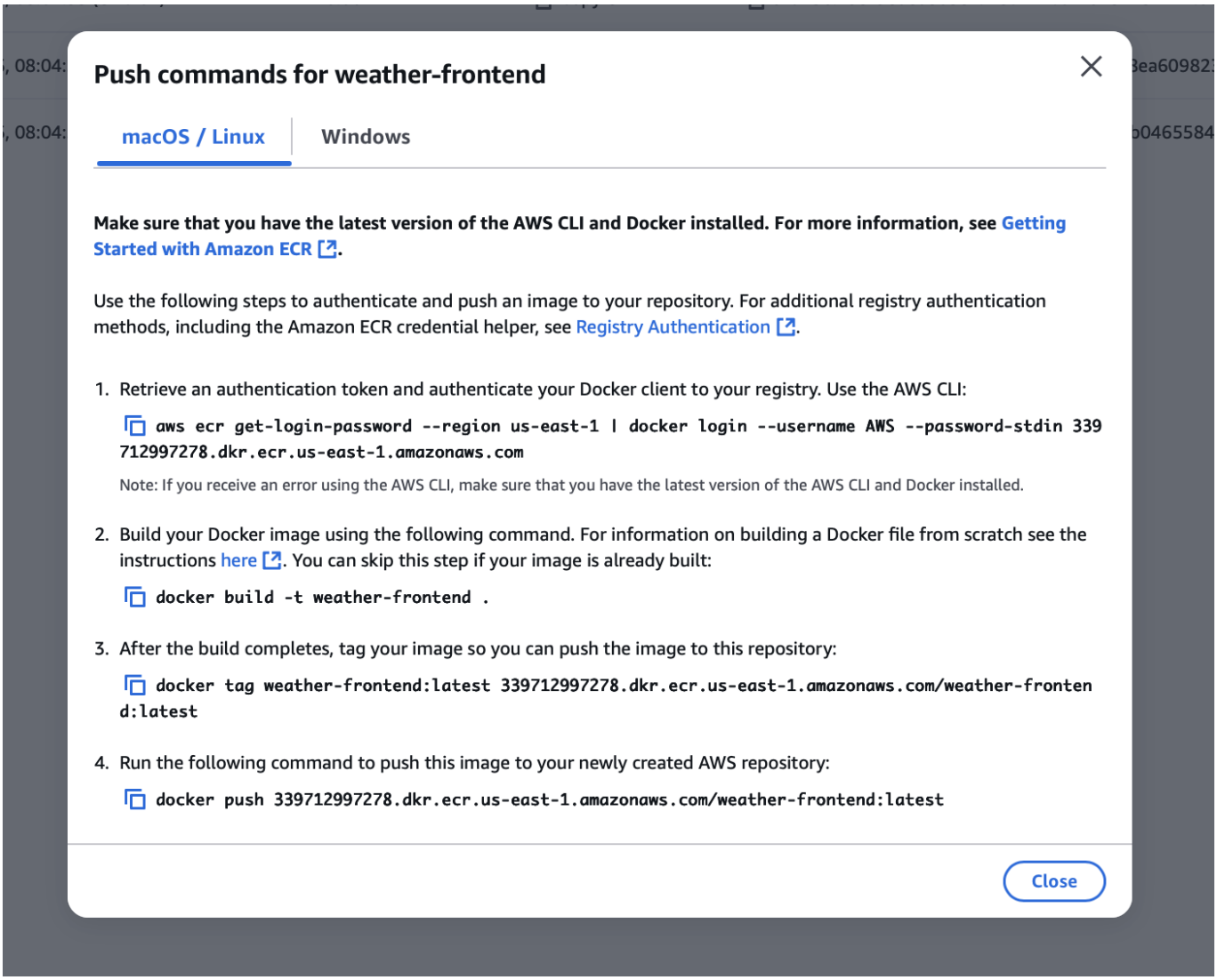
```
docker buildx build --platform linux/amd64,linux/arm64 -t weather-frontend
--load ./frontend
```

Why `--platform` flag?
EC2 instances run Linux. This ensures your Docker images are compatible with EC2's operating system and CPU architecture.

Push Images to ECR

AWS provides ready-to-use push commands for each repository.

To view push commands: ECR → Repositories → Select repository → Click **"View push commands"**



Authenticate Docker to ECR

Replace `us-east-1` with your region and `339712997278` with your AWS account ID:

```
aws ecr get-login-password --region us-east-1 | docker login --username
AWS --password-stdin 339712997278.dkr.ecr.us-east-1.amazonaws.com
```

Tag Frontend Image

```
docker tag weather-frontend:latest 339712997278.dkr.ecr.us-east-
1.amazonaws.com/weather-frontend:latest
```

Push Frontend Image

```
docker push 339712997278.dkr.ecr.us-east-1.amazonaws.com/weather-
frontend:latest
```

Tag Backend Image

```
docker tag weather-backend:latest 339712997278.dkr.ecr.us-east-
1.amazonaws.com/weather-backend:latest
```

Push Backend Image

```
docker push 339712997278.dkr.ecr.us-east-1.amazonaws.com/weather-
backend:latest
```

Verify Images in ECR

Go to **Amazon ECR → Repositories**

Check both repositories:

- `weather-frontend` - Should show `latest` tag
- `weather-backend` - Should show `latest` tag

—

Step 6: Creating Target Groups

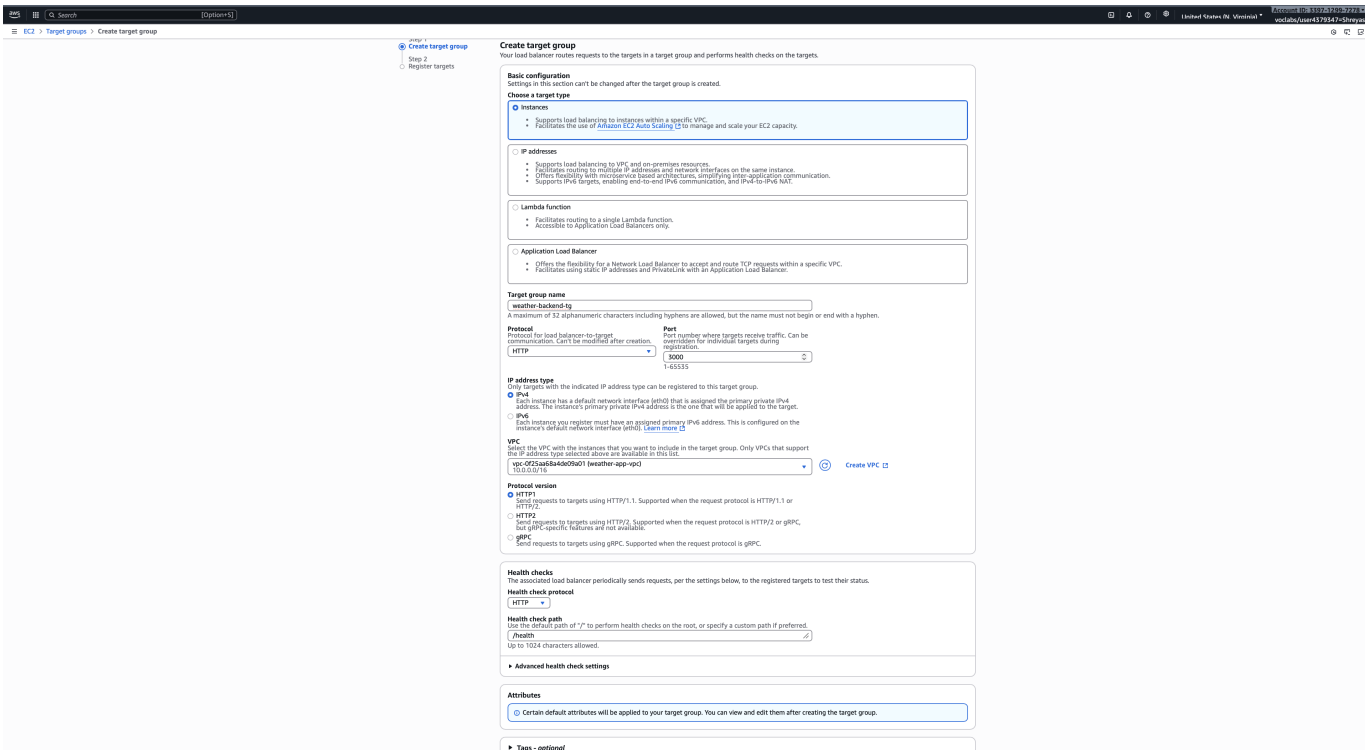
What are Target Groups?

Target Groups are used by Application Load Balancers to route traffic to registered EC2 instances. They also perform health checks to ensure traffic only goes to healthy instances.

We'll create 2 target groups - one for frontend and one for backend.

Target Group 1: Backend Target Group

Navigate: EC2 → Target Groups → Create target group



Step 1: Basic Configuration

```
Target type: Instances
Target group name: weather-backend-tg
Protocol: HTTP
Port: 3000
VPC: weather-app-vpc
Protocol version: HTTP1
```

Step 2: Health Checks

```
Health check protocol: HTTP
Health check path: /health
```

Advanced health check settings:

```
Healthy threshold: 2
Unhealthy threshold: 3
Timeout: 5 seconds
Interval: 30 seconds
Success codes: 200
```

Step 3: Register Targets

Skip this step - instances will be added automatically by Auto Scaling Group.

Click **Create target group**

Target Group 2: Frontend Target Group

Step 1: Basic Configuration

```
Target type: Instances
Target group name: weather-frontend-tg
Protocol: HTTP
Port: 80
VPC: weather-app-vpc
Protocol version: HTTP1
```

Step 2: Health Checks

```
Health check protocol: HTTP
Health check path: /health
```

Advanced health check settings:

```
Healthy threshold: 2
Unhealthy threshold: 3
Timeout: 5 seconds
Interval: 30 seconds
Success codes: 200
```

Step 3: Register Targets

Skip this step - instances will be added automatically by Auto Scaling Group.

Click **Create target group**

Step 7: Creating Application Load Balancers

What is an Application Load Balancer?

An Application Load Balancer (ALB) distributes incoming traffic across multiple EC2 instances. We'll create two ALBs:

- **Frontend ALB** (Internet-facing) - Receives traffic from users
- **Backend ALB** (Internal) - Receives traffic from frontend instances

---

ALB 1: Backend Application Load Balancer

Navigate: EC2 → Load Balancers → Create load balancer → Application Load Balancer

Basic Configuration

```
Load balancer name: weather-backend-alb
Scheme: Internal
IP address type: IPv4
```

Network Mapping

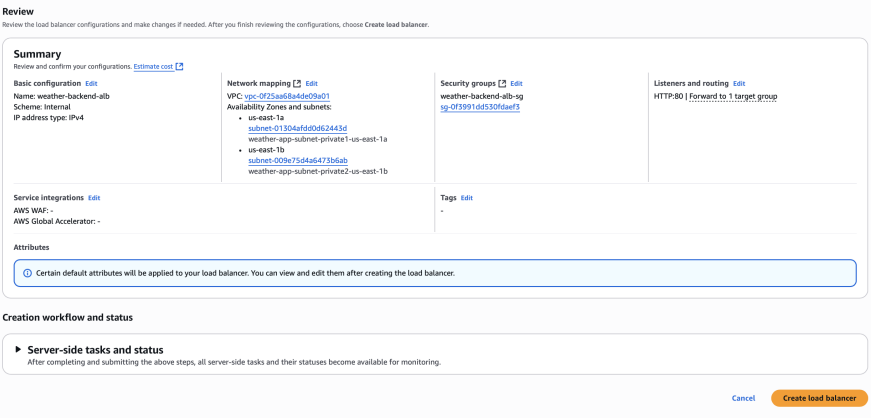
```
VPC: weather-app-vpc
Availability Zones: Select both
  ☑ us-east-1a → Select private subnet
  ☑ us-east-1b → Select private subnet
```

Security Groups

```
Remove: default
Add: weather-backend-alb-sg
```

Listeners and Routing

```
Protocol: HTTP
Port: 80
Default action: Forward to weather-backend-tg
```



Click **Create load balancer**

Wait 2-3 minutes for the ALB to become active.

Important: Copy the Backend ALB DNS name - you'll need it later!

ALB 2: Frontend Application Load Balancer

Basic Configuration

```
Load balancer name: weather-frontend-alb
Scheme: Internet-facing
IP address type: IPv4
```

Network Mapping

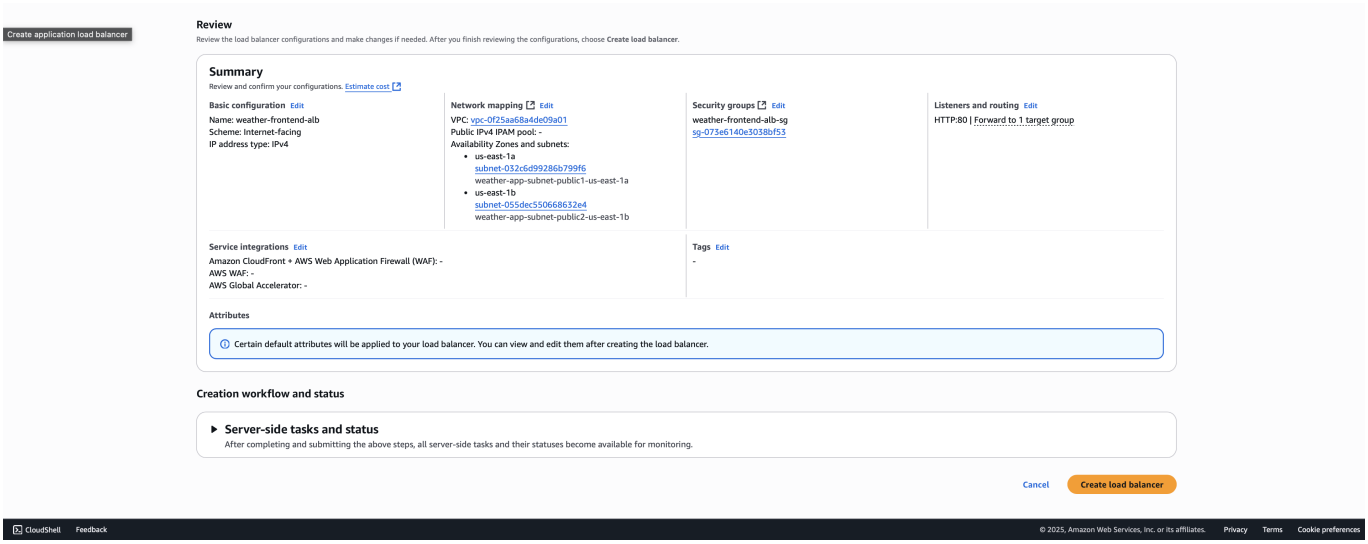
```
VPC: weather-app-vpc
Availability Zones: Select both
  ☑ us-east-1a → Select public subnet
  ☑ us-east-1b → Select public subnet
```

Security Groups

```
Remove: default
Add: weather-frontend-alb-sg
```

Listeners and Routing

```
Protocol: HTTP
Port: 80
Default action: Forward to weather-frontend-tg
```



Click **Create load balancer**

Wait 2-3 minutes for the ALB to become active.

Verify Load Balancers

Go to **EC2 → Load Balancers**

Check both ALBs show **State: Active**

Note down both DNS names:

```
Frontend ALB DNS: weather-frontend-alb-xxxxx.us-east-1.elb.amazonaws.com
Backend ALB DNS: weather-backend-alb-xxxxx.us-east-1.elb.amazonaws.com
```

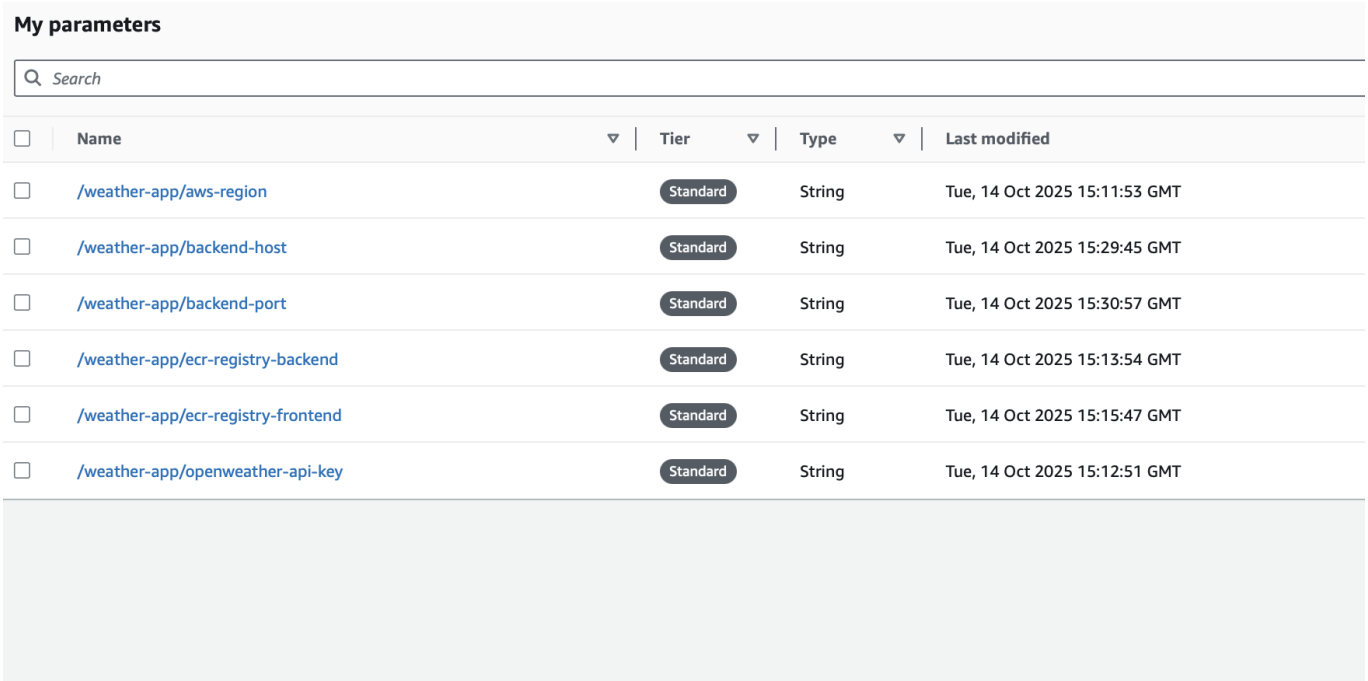
Step 8: Configure AWS Systems Manager Parameter Store

What is Parameter Store?

Parameter Store allows you to store configuration data and secrets centrally. EC2 instances will fetch these values at runtime, eliminating hardcoded configuration.

Create Parameters

Navigate: AWS Systems Manager → Parameter Store → Create parameter.



Create the following parameters:

Parameter 1: AWS Region

```
Name: /weather-app/aws-region
Type: String
Value: us-east-1 (or your region)
```

Parameter 2: Backend ALB Host (used by frontend to call an api)

```
Name: /weather-app/backend-host
Type: String
Value: <Your Backend ALB DNS without http://>
Example: internal-weather-backend-alb-xxxxx.us-east-1.elb.amazonaws.com
```

Parameter 3: Backend ALB Port (used by frontend to call an api)

```
Name: /weather-app/backend-port
Type: String
Value: 80
```

Parameter 4: Frontend ECR Registry

```
Name: /weather-app/ecr-registry-frontend
Type: String
Value: <Your account ID>.dkr.ecr.us-east-1.amazonaws.com/weather-frontend
Example: 339712997278.dkr.ecr.us-east-1.amazonaws.com/weather-frontend
```

Parameter 5: Backend ECR Registry

```
Name: /weather-app/ecr-registry-backend
Type: String
Value: <Your account ID>.dkr.ecr.us-east-1.amazonaws.com/weather-backend
Example: 339712997278.dkr.ecr.us-east-1.amazonaws.com/weather-backend
```

Parameter 6: OpenWeather API Key

```
Name: /weather-app/openweather-api-key
Type: String (or SecureString for encryption)
Value: <Your OpenWeatherMap API key>
```

Get your free API key from: https://openweathermap.org/api

Verify Parameters

All 6 parameters should now be visible in Parameter Store with the prefix `/weather-app/`

—

Step 9: Create Launch Templates

What is a Launch Template?

A Launch Template defines the configuration for EC2 instances that will be launched by Auto Scaling Groups. We'll create two templates - one for backend and one for frontend.

Launch Template 1: Backend Launch Template

Navigate: EC2 → Launch Templates → Create launch template

Launch Template Name and Description

```
Launch template name: weather-backend-lt
Template version description: Backend instances for weather app
```

Application and OS Images (AMI)

```
Quick Start: Ubuntu
Ubuntu Server 24.04 LTS (HVM), SSD Volume Type
Architecture: 64-bit (x86)
```

Instance Type

```
Instance type: t3.micro (or t2.micro for free tier)
```

Key Pair

```
Key pair name: Select your existing key pair
```

Network Settings

```
Subnet: Don't include in launch template
Firewall (security groups): weather-backend-sg
```

Storage

```
Keep default: 8 GiB, gp3
```

Advanced Details

IAM instance profile:

```
LabRole (for AWS Academy users)
OR weather-ec2-role (for regular AWS accounts)
```

Metadata accessible: Enabled

User data:

Copy the backend user data script from the repository:

Backend User Data Script →

This script:

- Installs Docker and AWS CLI
- Fetches configuration from Parameter Store
- Pulls Docker image from ECR
- Starts the backend container

Click **Create launch template**

Launch Template 2: Frontend Launch Template

Follow the same steps with these differences:

```
Launch template name: weather-frontend-lt
Template version description: Frontend instances for weather app
Security group: weather-frontend-sg
```

User data:

Copy the frontend user data script from the repository:

Frontend User Data Script →

This script:

- Installs Docker and AWS CLI
- Fetches backend ALB DNS from Parameter Store
- Pulls Docker image from ECR
- Starts the frontend container with backend configuration

Click **Create launch template**

Step 10: Create Auto Scaling Groups

What is an Auto Scaling Group?

An Auto Scaling Group (ASG) automatically manages EC2 instances - launching new instances when traffic increases and terminating them when traffic decreases. We'll create two ASGs - one for backend and one for frontend.

Note Create ASG first for backend and then create for frontend.

Auto Scaling Group 1: Backend ASG

Navigate: EC2 → Auto Scaling Groups → Create Auto Scaling group

Step 1: Choose Launch Template

```
Auto Scaling group name: weather-backend-asg
Launch template: weather-backend-lt
Version: Default (1)
```

Click **Next**

Step 2: Choose Instance Launch Options

```
VPC: weather-app-vpc

Availability Zones and subnets:
  ☑ weather-app-subnet-private1-us-east-1a
  ☑ weather-app-subnet-private2-us-east-1b
```

Click **Next**

Step 3: Configure Advanced Options

```
Load balancing:
  ☑ Attach to an existing load balancer

Choose from your load balancer target groups:
  ☑ weather-backend-tg

Health checks:
  ☑ Turn on Elastic Load Balancing health checks
  Health check grace period: 300 seconds
```

Click **Next**

Step 4: Configure Group Size and Scaling

```
Group size:
  Desired capacity: 2
  Minimum capacity: 2
  Maximum capacity: 4

Scaling:
  ☑ Target tracking scaling policy

  Scaling policy name: backend-cpu-scaling
  Metric type: Average CPU utilization
  Target value: 70
  Instance warmup: 300 seconds
```

Click **Next** → **Next** → **Create Auto Scaling group**

Wait 5-10 minutes for instances to launch and become healthy.

Auto Scaling Group 2: Frontend ASG

Follow the same steps with these differences:

Step 1: Choose Launch Template

```
Auto Scaling group name: weather-frontend-asg
Launch template: weather-frontend-lt
```

Step 2: Network

```
Availability Zones and subnets:
  ☑ weather-app-subnet-public1-us-east-1a
  ☑ weather-app-subnet-public2-us-east-1b
```

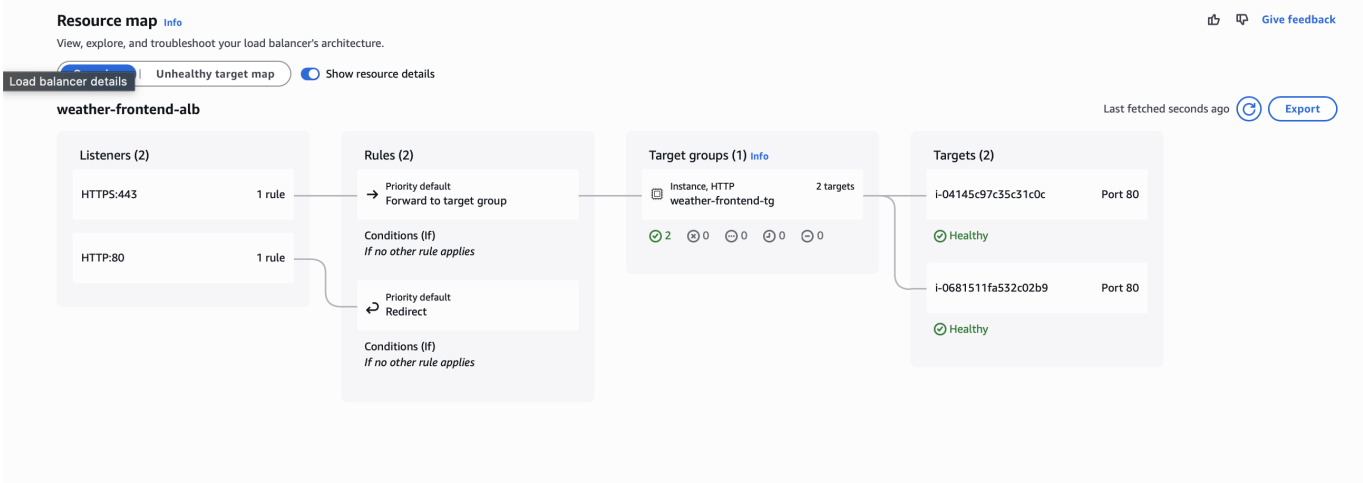
Step 3: Load Balancing

```
Target group: weather-frontend-tg
```

Step 4: Group Size and Scaling

```
Desired capacity: 2
Minimum capacity: 2
Maximum capacity: 4

Scaling policy name: frontend-cpu-scaling
Metric type: Average CPU utilization
Target value: 70
```



Click **Create Auto Scaling group**

Wait 5-10 minutes for instances to launch and become healthy.

Verify Auto Scaling Groups

Check Instances:

1. Go to **EC2 → Instances**
2. You should see 4 running instances:
   - 2 backend instances (in private subnets)
   - 2 frontend instances (in public subnets)

Check Target Health:

1. Go to **EC2 → Target Groups**
2. Select **weather-backend-tg** → Targets tab
   - Both targets should show **Healthy**
3. Select **weather-frontend-tg** → Targets tab
   - Both targets should show **Healthy**

—--

Step 11: Testing Your Application

Test Application via Frontend ALB

Get Frontend ALB DNS:

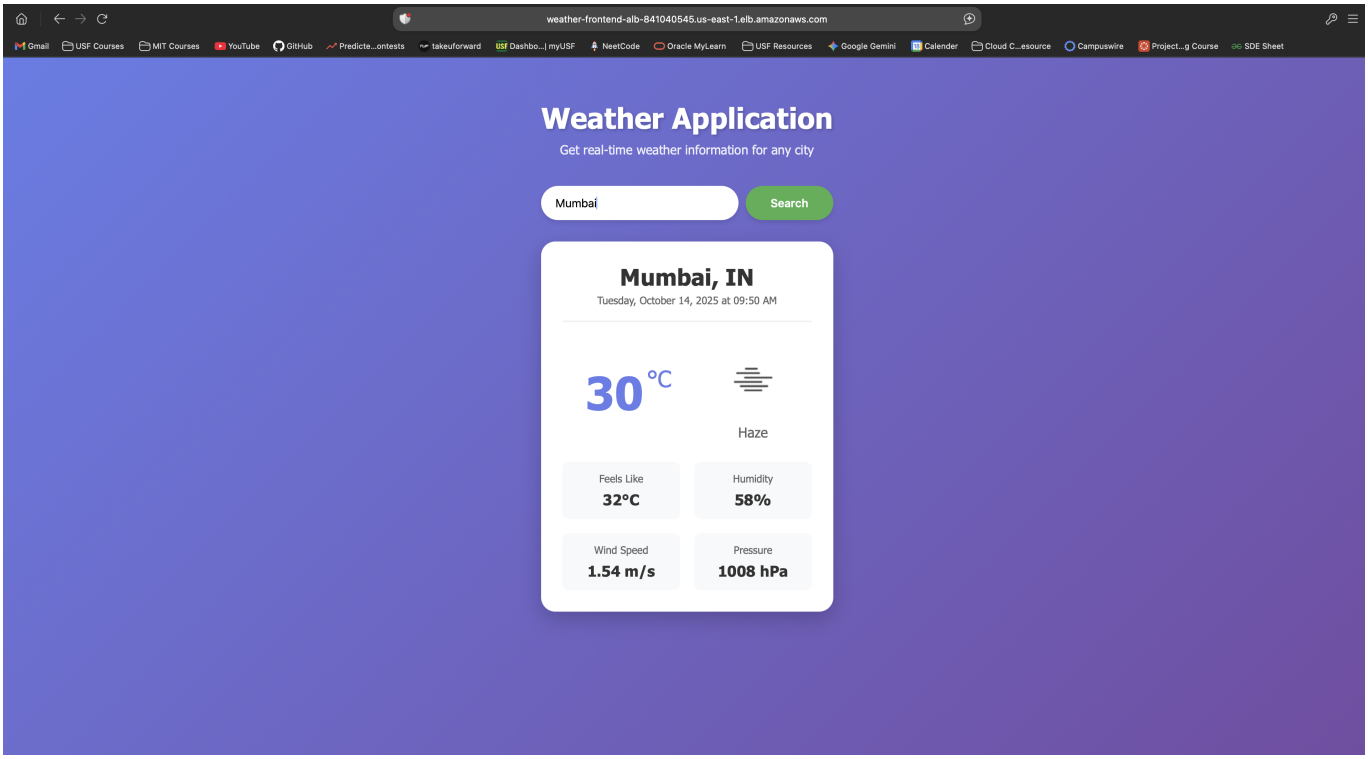
1. Go to **EC2 → Load Balancers**
2. Select **weather-frontend-alb**
3. Copy the **DNS name**

Example: `weather-frontend-alb-841040545.us-east-1.elb.amazonaws.com`

Test in Browser:

Open your browser and navigate to:

```
http://weather-frontend-alb-XXXXXXX.us-east-1.elb.amazonaws.com
```



You should see:

- Weather Application interface
- Search box to enter city name
- Real-time weather data when you search

Test the Complete Flow:

1. Enter a city name (e.g., "Mumbai", "London", "New York")
2. Click **Search**

3. Weather information should display:
    - Current temperature
    - Weather condition (Haze, Clear, etc.)
    - Feels like temperature
    - Humidity
    - Wind speed
    - Pressure

Common Issues:

If the application doesn't work:

1. **Check Target Health:**
    - EC2 → Target Groups → Check both target groups show "Healthy"
2. **Check Instance Logs:**
    - Connect to instances via Session Manager
    - View logs: `sudo docker logs weather-frontend` or `sudo docker logs weather-backend`
3. **Check Security Groups:**
    - Verify all security group rules are correct
    - Frontend ALB should allow 80 from internet
    - Backend ALB should allow 80 from frontend SG

—--

Step 12: Add Custom Domain and Enable HTTPS

Prerequisites

You need a registered domain name from any registrar (Namecheap, GoDaddy, etc.)

Part A: Create Hosted Zone in Route 53

Navigate: Route 53 → Hosted zones → Create hosted zone

Configuration:

Domain name: shri.software Type: Public hosted zone

Click **Create hosted zone**

Update Nameservers at Your Domain Registrar:

1. Copy the 4 NS (nameserver) records from Route 53
2. Go to your domain registrar (Namecheap, GoDaddy, etc.)
3. Replace existing nameservers with the 4 AWS nameservers

4. Save changes

Wait 15-30 minutes for DNS propagation

Part B: Request SSL/TLS Certificate

Navigate: AWS Certificate Manager (ACM) → Request certificate **Step 1: Request Certificate** Certificate type: Request a public certificate Click **Next**

Step 2: Domain Names Fully qualified domain name: shri.software Add another name: *.shri.software (optional, for subdomains)

Step 3: Validation Method Validation method: DNS validation - recommended

Step 4: Key Algorithm RSA 2048 (default) Click **Request**

Part C: Validate Certificate

After requesting the certificate, ACM will show validation details. **Click "Create records in Route"**

This automatically adds the required CNAME validation records to your hosted zone. Wait 5-10 minutes for validation to complete. **Certificate Status:** Will change from "Pending validation" to "Issued"

Part D: Create A Record for Your Domain

Navigate: Route 53 → Hosted zones → shri.software

Create A Record: Record name: (leave empty for root domain) Record type: A - Alias Route traffic to: Alias to Application Load Balancer Region: us-east-1 Load balancer: weather-frontend-alb

Click **Create records**

Part E: Add HTTPS Listener to Frontend ALB

Navigate: EC2 → Load Balancers → weather-frontend-alb **Add HTTPS Listener:**

1. Click **Listeners** tab
2. Click **Add listener**

Protocol: HTTPS Port: 443 Default action: Forward to weather-frontend-tg Default SSL/TLS certificate: From ACM → Select your certificate (shri.software)

Click **Add**

Part F: Redirect HTTP to HTTPS

Edit HTTP:80 Listener:

1. Select **HTTP:80** listener

2. Click **Actions** → **Edit listener**
3. Remove existing forward action
4. Add action → **Redirect to...**

Protocol: HTTPS Port: 443 Status code: 301 - Permanently moved

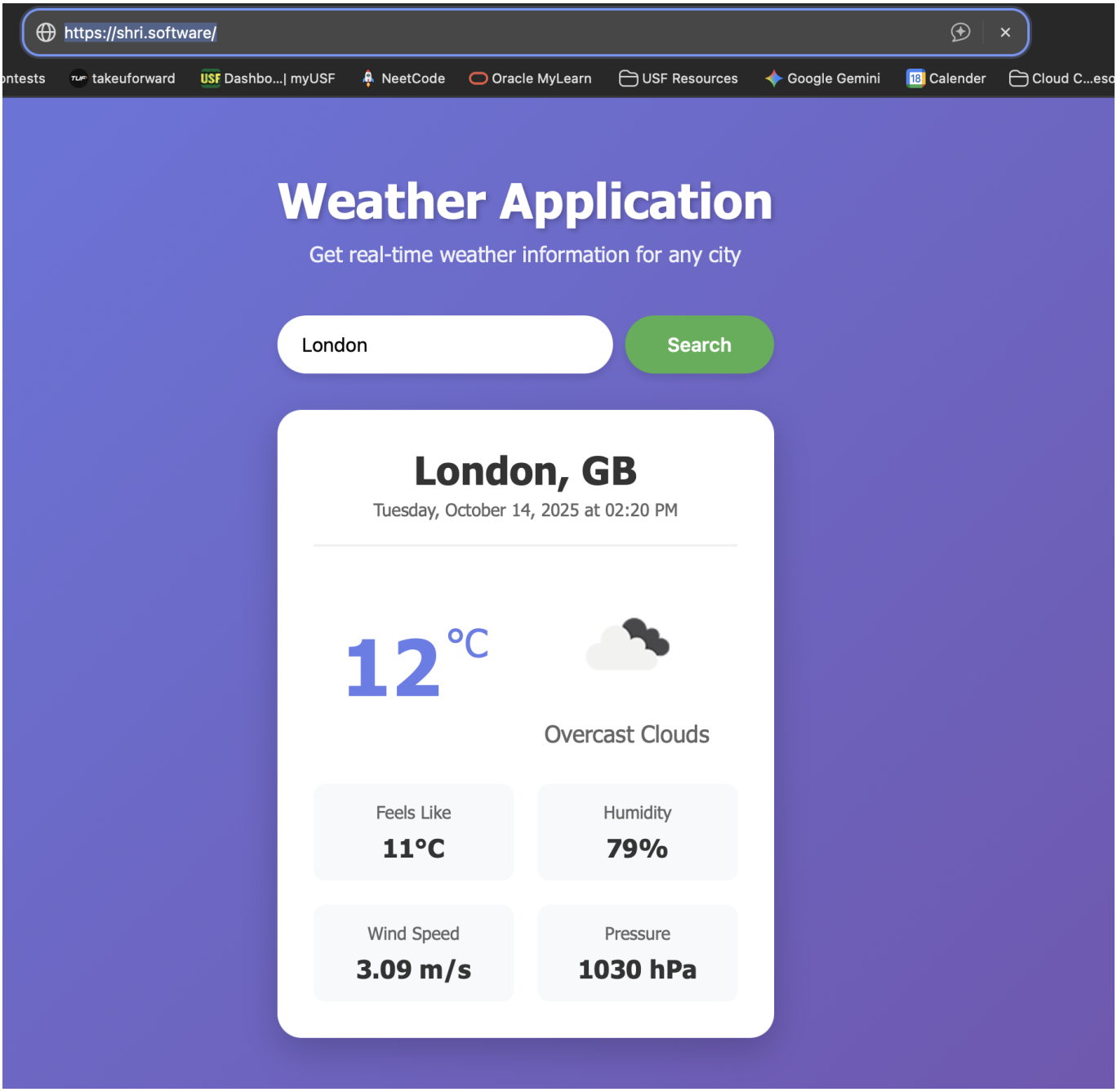
Click **Save changes**

Verify Your Setup

Test in Browser:

Open: https://your_domain_name

You should see: ✅ Padlock icon (secure connection) ✅ Weather application loads ✅ HTTP automatically redirects to HTTPS



Congratulations! Your application is live.

Step 13: Cleanup and Delete Resources

It is essential to delete all resources created in this guide to avoid incurring unnecessary charges on your AWS account.

Cleanup Order

1. **Auto Scaling Groups (ASGs):** Delete `weather-frontend-asg` and `weather-backend-asg`. (This action automatically terminates all running EC2 instances.)
2. **Application Load Balancers (ALBs):** Delete `weather-frontend-alb` and `weather-backend-alb`.
3. **Target Groups:** Delete `weather-frontend-tg` and `weather-backend-tg`.
4. **Launch Templates:** Delete `weather-frontend-lt` and `weather-backend-lt`.
5. **ECR Repositories:** Delete `weather-frontend` and `weather-backend`. (This deletes the stored Docker images.)
6. **Route 53 & AWS Certificate Manager (ACM):** Delete the **A Record** pointing to the ALB. Delete the **ACM Certificate**. Delete the **Hosted Zone** in Route 53.
7. **IAM Role & Security Groups:** Delete the **IAM Role** (`weather-ec2-role`). Delete the **4 Security Groups** (`weather-frontend-alb-sg`, `weather-frontend-sg`, etc.).
8. **Systems Manager Parameters:** Delete all **Parameters** under the `/weather-app/` path.
9. **VPC:** Delete the **VPC** (`weather-app-vpc`). (This final step cleans up Subnets, Internet Gateway, and NAT Gateways.)

Conclusion and Thank You..!

Congratulations! You have successfully deployed a highly available, secure, and production-ready weather application on AWS using enterprise-grade architecture.

By completing this guide, you've mastered key AWS services including VPC, Auto Scaling Groups, Application Load Balancers, ECR, Route 53, and ACM. This foundational knowledge is crucial for deploying any modern, scalable web application.

Thank you for following along, and I hope this guide helps you on your cloud journey!