Cloud Security Project

🚀 \*\*Cloud Security Hands-On Project: Deploying a Vulnerable Web App & Secure AWS Networking\*\* 🔐

Excited to share my latest \*\*cloud security project\*\*, where I dive deep into AWS networking and security while deploying a \*\*vulnerable web application using Docker\*\*. This project integrates multiple AWS services, demonstrating real-world security principles while encouraging hands-on experimentation.

### \*\*Project Overview\*\*

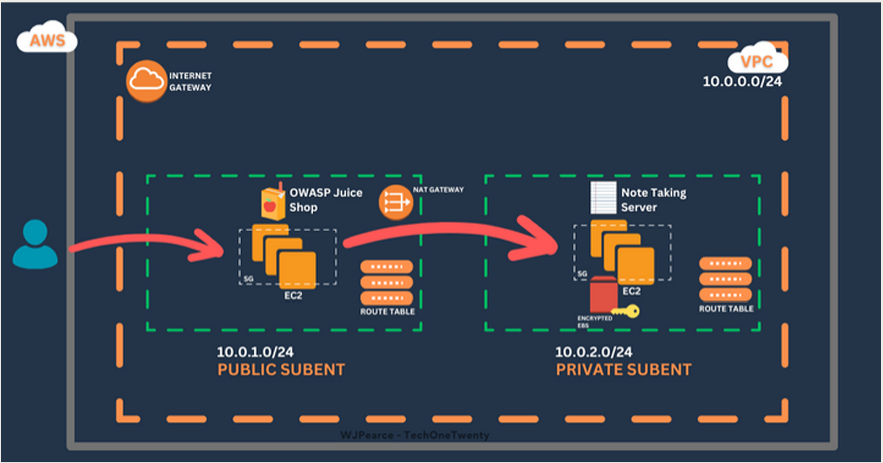
🔹 \*\*Deploy OWASP Juice Shop\*\*: A vulnerable web app running on an \*\*EC2 instance in the public subnet\*\*, exposing security flaws for ethical hacking practice.

🔹 \*\*Secure AWS Networking\*\*: Two subnets (\*\*public & private\*\*) configured within a \*\*VPC\*\*, each with tailored \*\*security groups\*\* controlling inbound/outbound traffic.

🔹 \*\*Internet & NAT Gateways\*\*: Public subnet resources can access the internet, while private subnet resources remain isolated but can \*\*reach external services securely\*\*.

🔹 \*\*Encrypted EBS Volume\*\*: Attached to the private instance for storing sensitive information, secured with \*\*AWS KMS encryption\*\*.

🔹 \*\*Terraform, Automation & Version Control\*\*: While the project initially uses the \*\*AWS console\*\*, there's room for further enhancement through \*\*Terraform deployment and automation\*\*.

**Project Architecture **

**AWS Services Overview**

**VPC:** Configured with two subnets, one public and one private.

Security Groups: Configured in the private and public subnets for the deployed instances.

**EC2:** Two Virtual machines, one inside the public subnet and the other in the private subnet. **Internet Gateway:** Allows resources in the public subnet to connect to the internet.

**NAT Gateway:** Enables instances in the private subnet to connect to external services while preventing external services from initiating connections.

**Route Table:** Directs traffic appropriately.

**Docker:** Installed and running the OWASP Juice Shop App, on the public instance.

**EBS Volume:** Attached to the EC2 instance and encrypted with KMS on the private instance for secret note-taking.

**Let us get started**

**VPC Creation:**

**Step 1:** **Create a VPC**

1. Click on "Create VPC": On the VPC Dashboard, click the "Create VPC" button.
2. Choose the VPC-only option: Depending on the requirements, you can select "VPC and more" for additional configurations like subnets, gateways, and route tables but we will do those separately after.

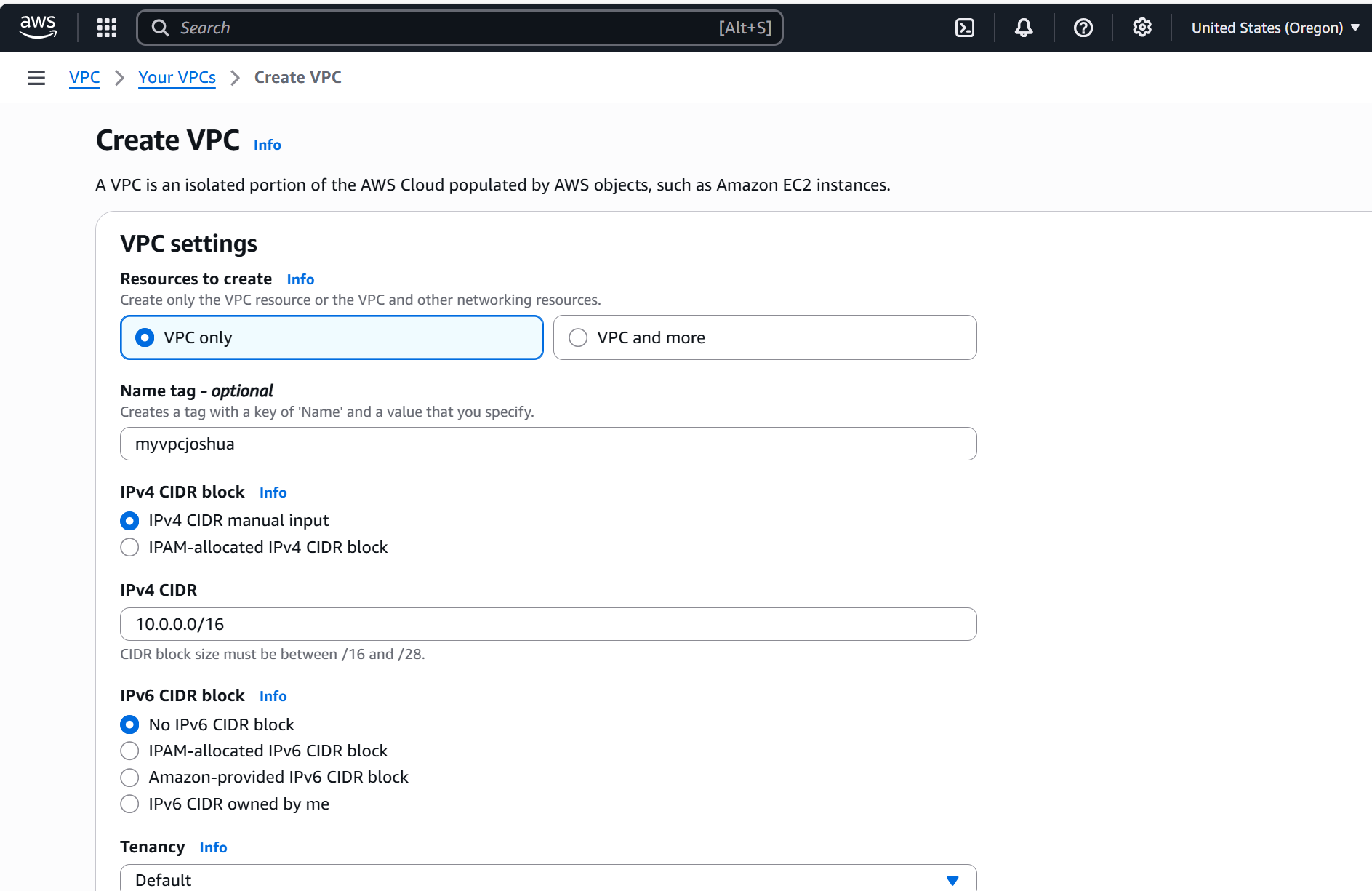
**Step 2: Configure VPC Settings**

1. Name Tag: Provide a name for your VPC to easily identify it.
2. IPv4 CIDR block: Specify the IPv4 CIDR block: 10.0.0.0/16
3. IPv6 CIDR block: Optionally, you can associate an IPv6 CIDR block - I am not doing this for the first time.
4. Tenancy: Select "Default" for shared tenancy or "Dedicated" for dedicated hardware. We want "Default" here

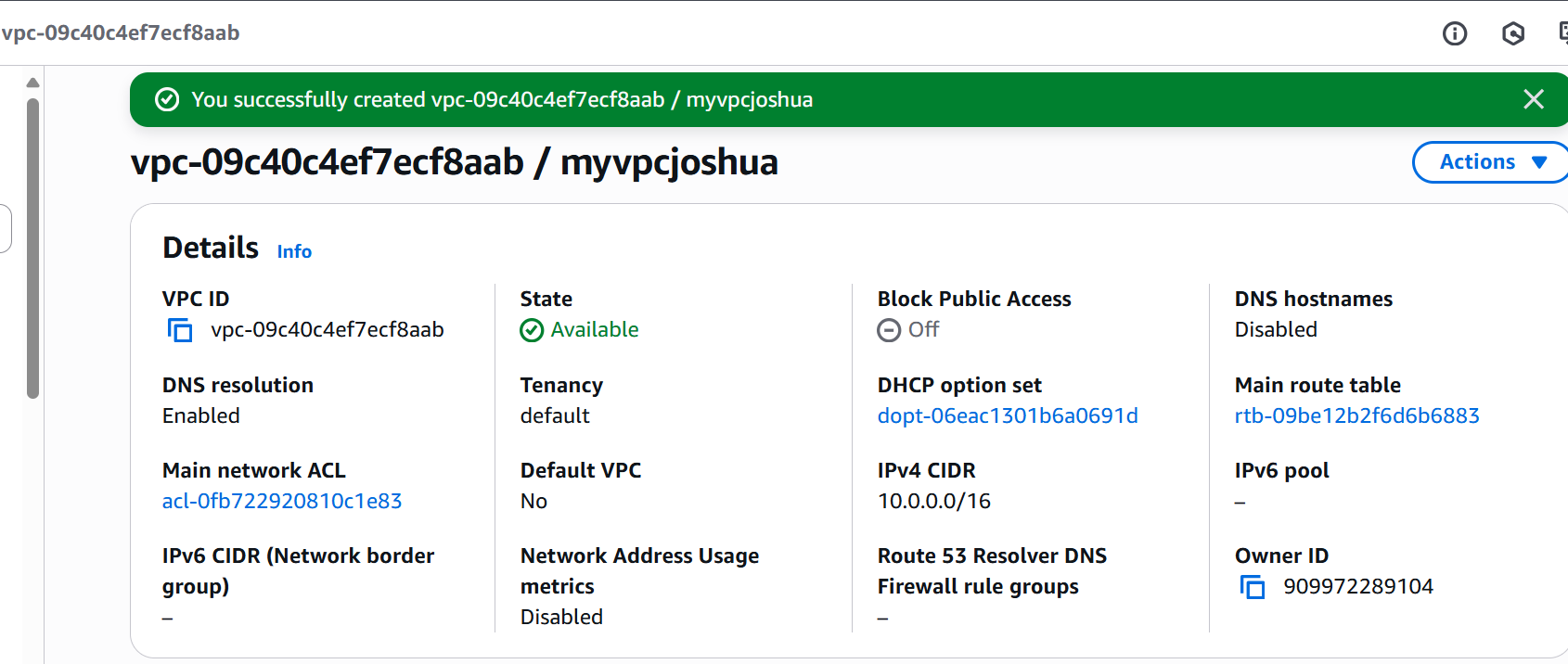
**Step 5: Review and Create**

1. Review Configuration
2. Create VPC: Click the "Create VPC" button to create your VPC.

**Here is how my configuration looks:**



**VPC Created**



**Public & Private Subnet Creation + Route Table Configuration**

**Step 1: Create Public Subnet**

1. Navigate to Subnets: In the VPC Dashboard, click on "Subnets" in the left-hand menu.
2. Configure Subnet Settings:
   1. Name Tag: Provide a name for the subnet, such as "Public Subnet".
   2. VPC: Select the VPC where you want to create the subnet.
   3. Availability Zone: For this project we will use us-west-2
   4. Create Subnet

**Step 2: Modify Public Subnet to Associate with Route Table**

1. Navigate to Route Tables: In the VPC Dashboard, click on "Route Tables" in the left-hand menu.
2. Select the route table associated with your VPC’s public subnet. If none exists, create a new route table.
3. Edit Routes
4. Add Route:
   1. Destination: Specify 0.0.0.0/0 to route all traffic.
   2. Target: Select the Internet Gateway ID.
5. Save Changes: Click "Save routes" to apply the changes.
6. Associate Route Table: Go to the "Subnet Associations" tab, click "Edit subnet associations," and select your public subnet.

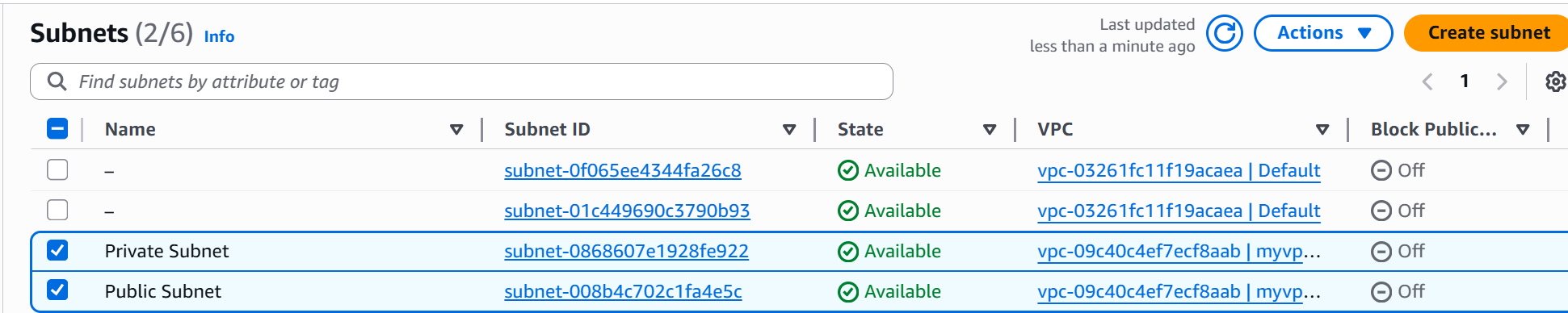
**Step 3: Create Private Subnet**

1. Navigate to Subnets: In the VPC Dashboard, click on "Subnets" in the left-hand menu.
2. Create Subnet
3. Configure Subnet Settings:
   1. Name Tag: "Private Subnet".
   2. VPC: Select the VPC where you want to create the subnet.
   3. Availability Zone: For this project we will use us-west-2
   4. IPv4 CIDR Block: Specify an IPv4 CIDR block (e.g., 10.0.2.0/24).
   5. Create Subnet

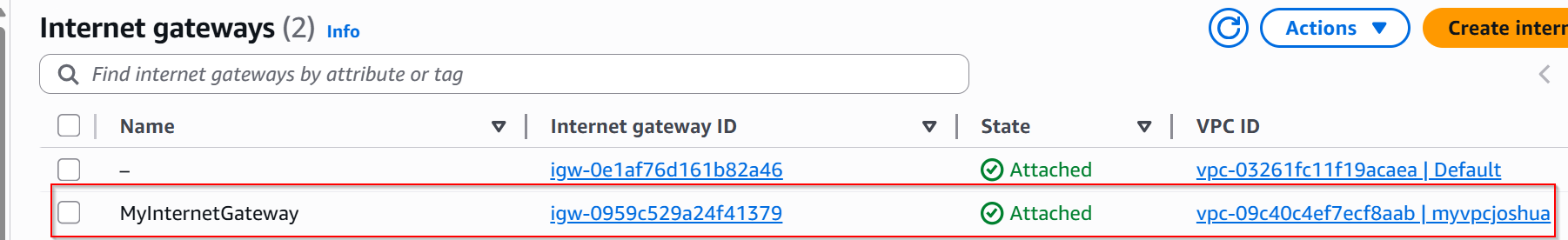
**Step 4: Update Private Subnet Route Table to Use NAT Gateway**

1. In the VPC Dashboard, click on "Route Tables" in the left-hand menu.
2. Select Private Route Table: Select the route table associated with your private subnet. If none exists, create a new route table.
3. Edit Routes
4. Add Route:
   1. Destination: Specify 0.0.0.0/0 to route all traffic.
   2. Target: Select the NAT Gateway ID.
5. Click "Save routes" to apply the changes.
6. Associate Route Table: Go to the "Subnet Associations" tab, click "Edit subnet associations," and select your private subnet.

**Below is a view of the subnets, one public and the other private:**

****

**Internet Gateway Creation + Route Table Configuration**

**Step 1: Create an Internet Gateway**

1. Navigate to Internet Gateways: In the VPC Dashboard, click on "Internet Gateways" in the left-hand menu.
2. Click on "Create Internet Gateway": Click the "Create internet gateway" button.
3. Configure Internet Gateway Settings:
4. Name Tag: Provide a name for the Internet Gateway (e.g., "MyInternetGateway").
5. Create Internet Gateway: Click the "Create internet gateway" button.

**Step 2: Attach Internet Gateway to VPC**

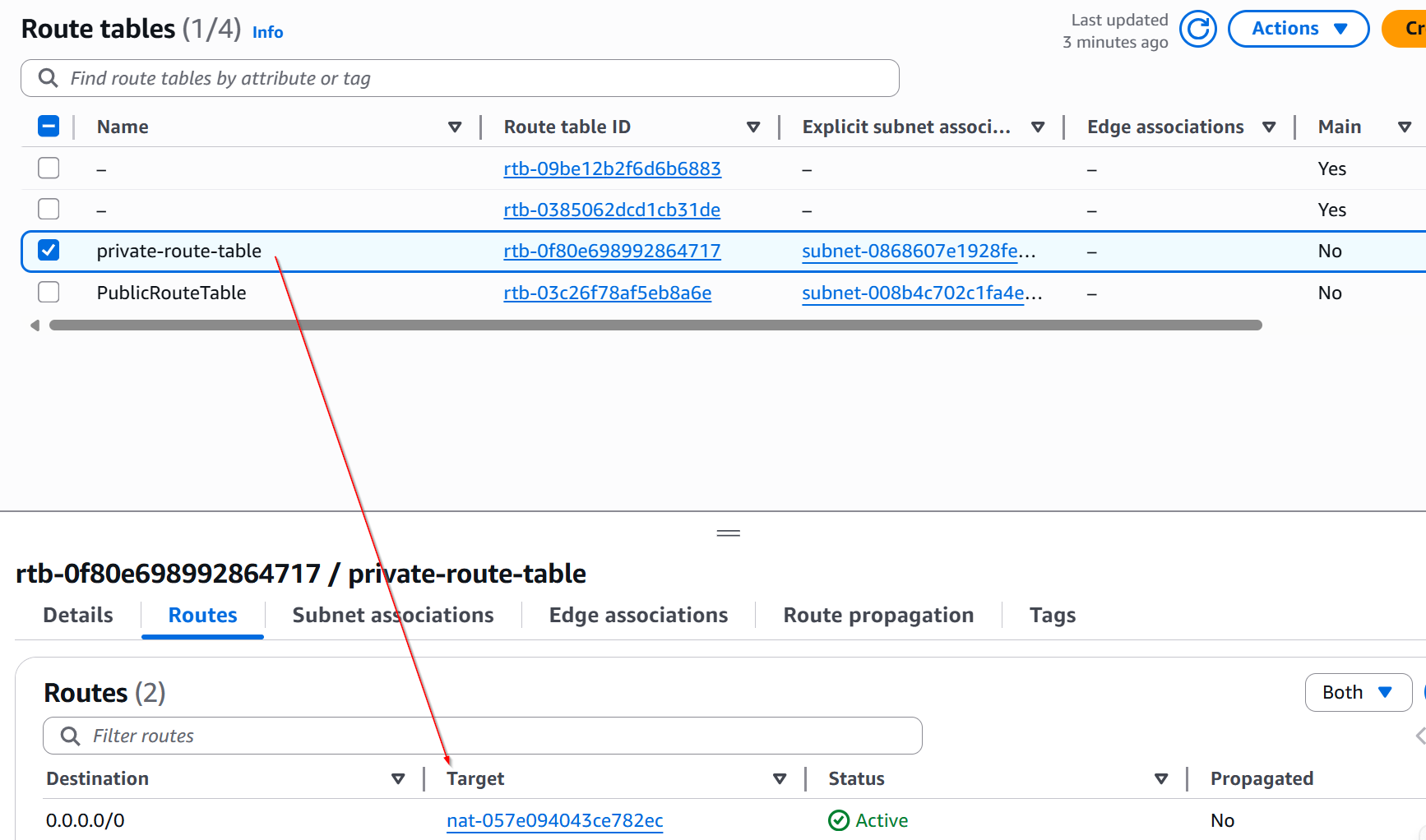
1. Select Internet Gateway: Select the newly created Internet Gateway.
2. Click on "Actions": Choose "Attach to VPC".
3. Select VPC: Choose your VPC (10.0.0.0/16) from the dropdown menu.
4. Attach Internet Gateway: Click the "Attach internet gateway" button.

**Step 3: Create Route Table for Public Subnet**

1. Navigate to Route Tables: In the VPC Dashboard, click on "Route Tables" in the left-hand menu.
2. Click on "Create Route Table": Click the "Create route table" button.
3. Configure Route Table Settings:
   1. Name Tag: Provide a name for the route table (e.g., "PublicRouteTable").
   2. VPC: Select your VPC (10.0.0.0/16).
4. Create Route Table

**Step 4: Add Route to Public Route Table**

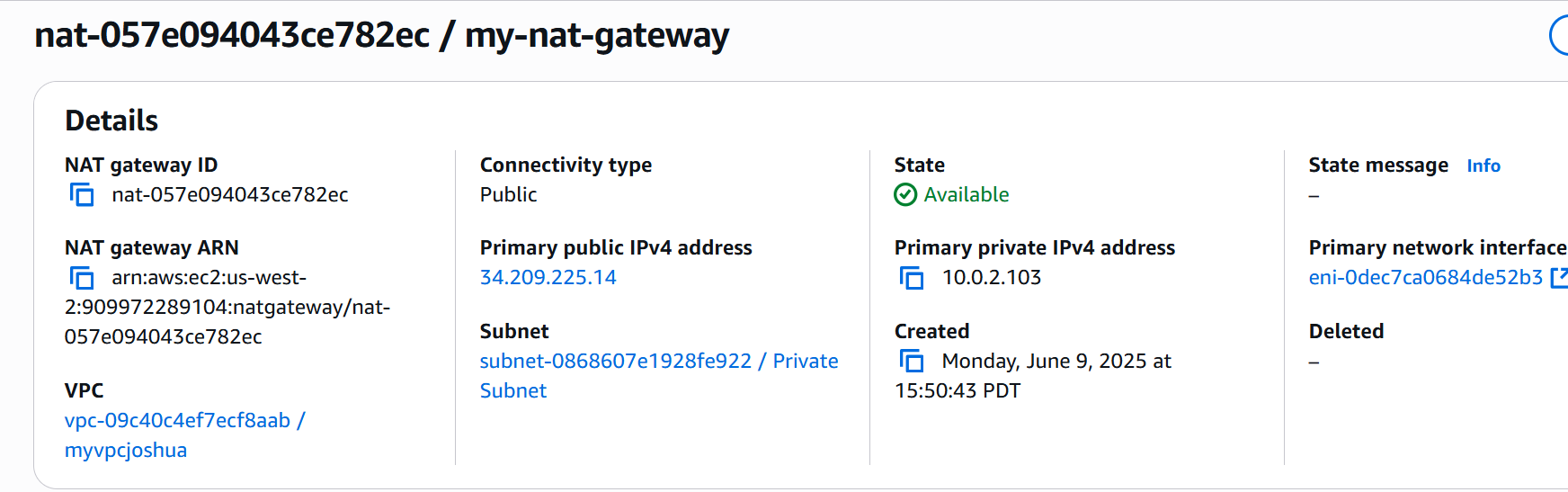
1. Select the route table created for the public subnet.
2. Edit Routes
3. Add Route
   1. Destination: Specify 0.0.0.0/0 to route all traffic.
   2. Target: Select the Internet Gateway ID.
4. Save routes
5. Associate Route Table with Public Subnet:
   1. Go to the "Subnet Associations" tab.
   2. Click "Edit subnet associations."
   3. Select your public subnet (10.0.1.0/24).
   4. Save

**Step 5: Create Route Table for Private Subnet**

1. Navigate to Route Tables: In the VPC Dashboard, click on "Route Tables" in the left-hand menu.
2. Click the "Create route table"
3. Configure Route Table Settings:
   1. Name Tag: Provide a name for the route table (e.g., "PrivateRouteTable").
   2. VPC: Select your VPC (10.0.0.0/16).
4. Create Route Table

**Step 6: Add Route to Private Route Table**

1. Select the route table created for the private subnet.
2. Associate Route Table with Private Subnet
   1. Edit subnet associations.
   2. Select your private subnet (10.0.2.0/24).
   3. Save

**NAT Gateway Creation + Route Table Configuration**

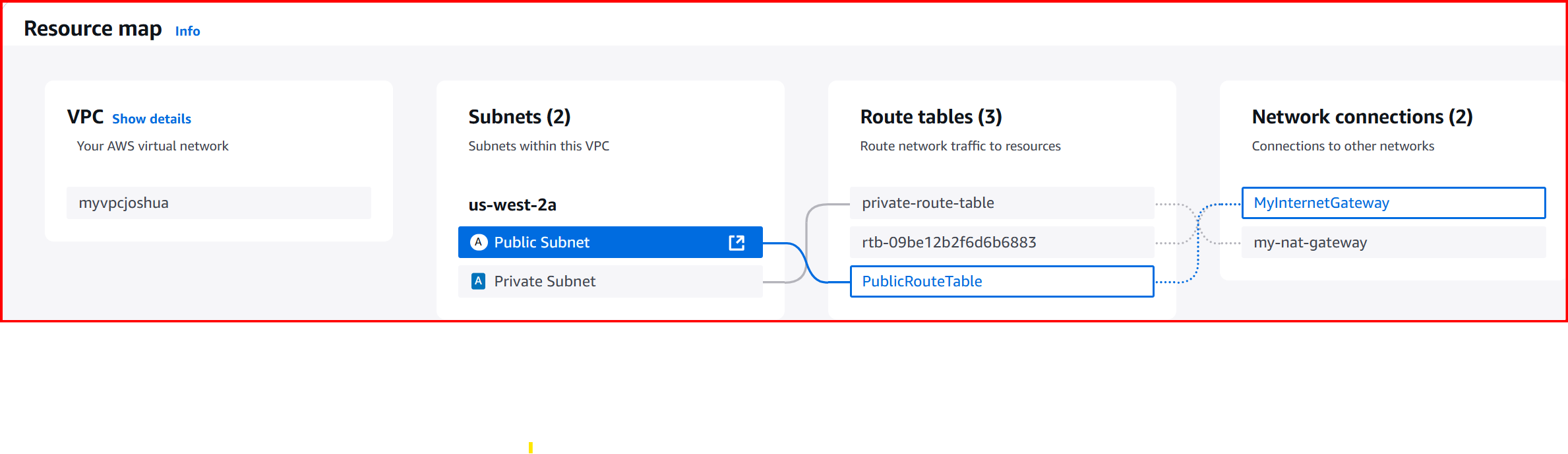
**Step 1: Create a NAT Gateway**

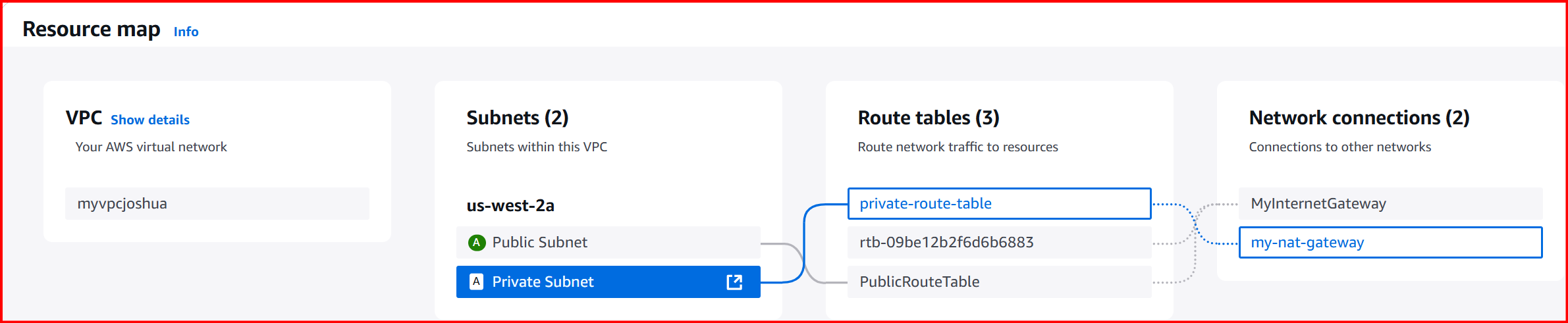
1. Navigate to NAT Gateways: In the VPC Dashboard, look for the "NAT Gateways" option in the left-hand menu and click on it. 2. 3. 4. 1.
2. Click on "Create NAT Gateway": Click the "Create NAT Gateway" button.
3. Configure NAT Gateway Settings:
   1. Subnet: Select the public subnet (10.0.1.0/24) in which you want to place the NAT Gateway.
   2. Elastic IP Allocation: Click "Allocate Elastic IP" to assign a new Elastic IP to your NAT Gateway or select an existing Elastic IP.
4. Create NAT Gateway: Click the "Create NAT Gateway" button.

**Step 2: Update Route Table for Private Subnet**

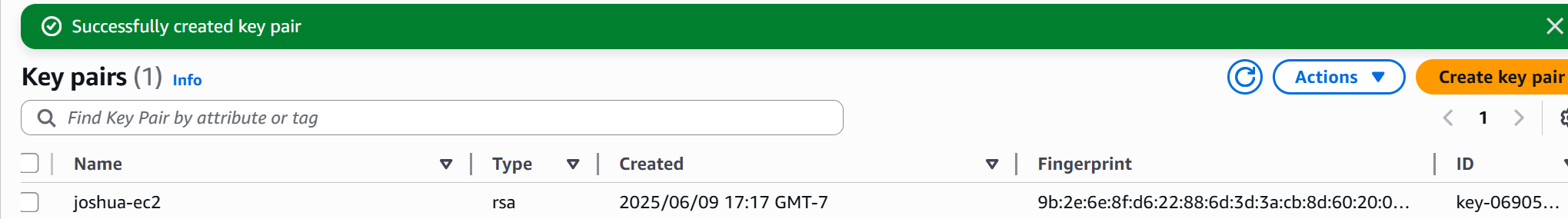
1. Navigate to Route Tables: In the VPC Dashboard, click on "Route Tables" in the left-hand menu. 2. 3. 4. 5.
2. Select Private Route Table: Select the route table associated with your private subnet (e.g., "PrivateRouteTable").
3. Edit Routes: Click on the "Routes" tab and then click the "Edit routes" button.
4. Add Route:
   1. Destination: Specify 0.0.0.0/0 to route all traffic.
   2. Target: Select the NAT Gateway ID that you just created.
5. Save Changes: Click "Save routes" to apply the changes.

**Before we start launching resources, here is how it should all look together.**

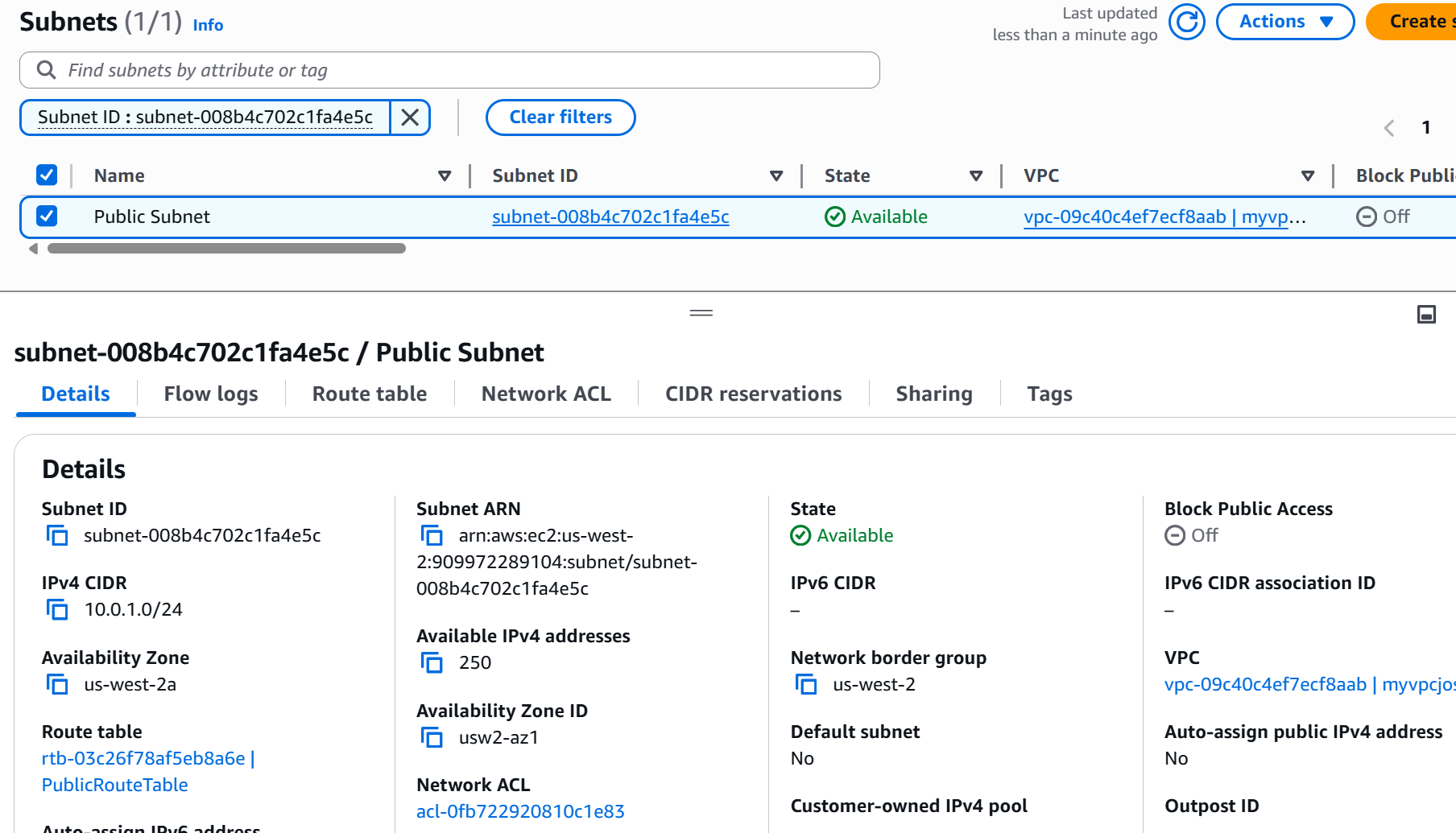


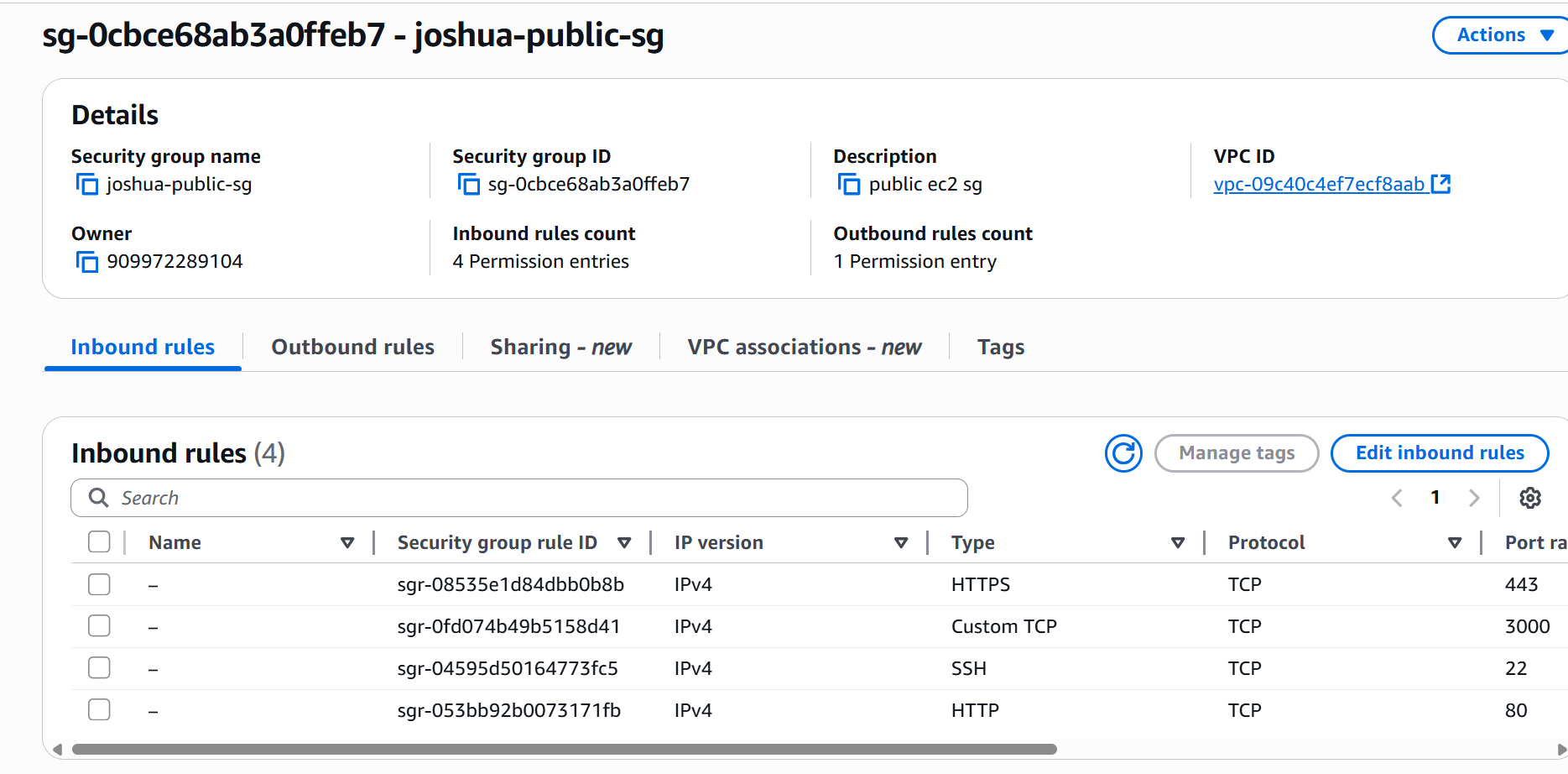


**Next we will work on Public EC2 Creation + SSH Key Pair**

**Step 1: Prepare an SSH Key Pair**

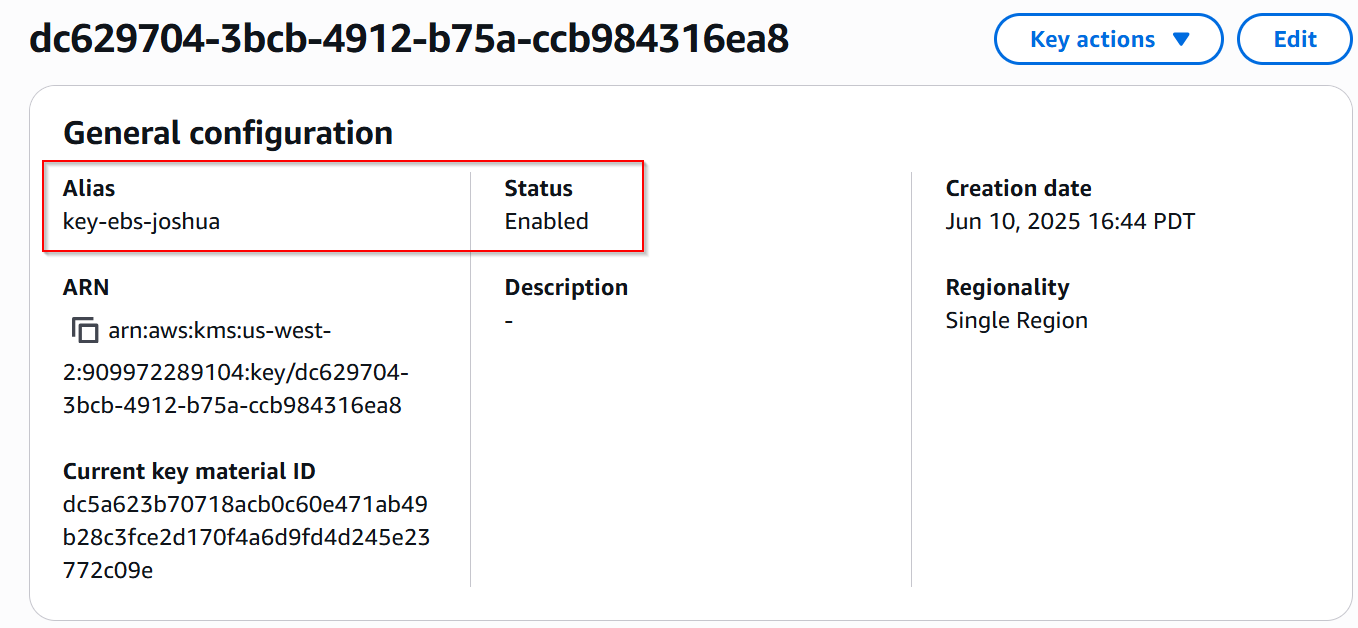
1. Create or Use an Existing Key Pair:
   1. If you do not already have a key pair, you can create one in the AWS Management Console.
   2. Navigate to EC2 Dashboard: In the AWS Management Console, type "EC2" in the search bar and select "EC2" from the results.
   3. Click on "Key Pairs": In the left-hand menu, click on "Key Pairs" under the "Network & Security" section.
   4. Click on "Create Key Pair": Click the "Create key pair" button.
   5. Enter Key Pair Name: Provide a name for your key pair (tech-one-ec2).
   6. Select Key Pair File Format: Choose .pem for Linux/macOS or .ppk for Windows (PuTTY).
   7. Create Key Pair: Click "Create key pair". The private key file will be automatically downloaded to your computer. Store it securely, very securely.

**Step 2: Launch EC2 Instance in Public Subnet**

1. Navigate to EC2 Dashboard:
   1. In the AWS Management Console, type "EC2" in the search bar and select "EC2" from the results.
2. Click on "Launch Instance":
3. Configure Instance Details:
   1. Name: Set the name to tech-one-ec2-public.
   2. AMI: Choose "Ubuntu Server 24.04 LTS"
   3. Instance Type: Select t2.micro.
   4. Key Pair: Choose the key pair you created earlier (e.g., my-ec2-keypair). This allows you to SSH into your instance later.
4. Network Settings:
   1. VPC: Select your VPC (10.0.0.0/16).
   2. Subnet: Select your public subnet (10.0.1.0/24).
   3. Auto-assign Public IP: Enable this option to assign a public IP address.
   4. IAM Role: This field can be left as "None"
5. Configure Storage: Leave the default storage settings
6. Configure Security Group:
   1. Edit in the Networking Section: Click on "Edit" in the networking section to configure security group settings.
   2. Create a New Security Group:
      1. Security Group Name: Provide a name for the security group (e.g., tech-one-public-sg).
      2. Description: Provide a description for clarity.
      3. Add Rules:
         1. SSH (Port 22): Allow SSH access from your IP. Set the source to "My IP" to restrict SSH access only to your current IP address.
         2. HTTP (Port 80): Allow HTTP traffic from anywhere. Set the source to "My IP" for public web traffic.
         3. HTTPS (Port 443): Allow HTTPS traffic from anywhere. Set the source to "My IP" for secure web traffic.
         4. Finally do the same with Port 3000, we will need this for the AWS JuiceShop
      4. Ensure the SSH Rule is Locked Down: Double-check that the SSH rule is restricted to "My IP" to ensure only your IP can access the instance via SSH.
7. **Review and Launch:** 
   1. Review all the configuration settings.
   2. Click "Launch Instance".

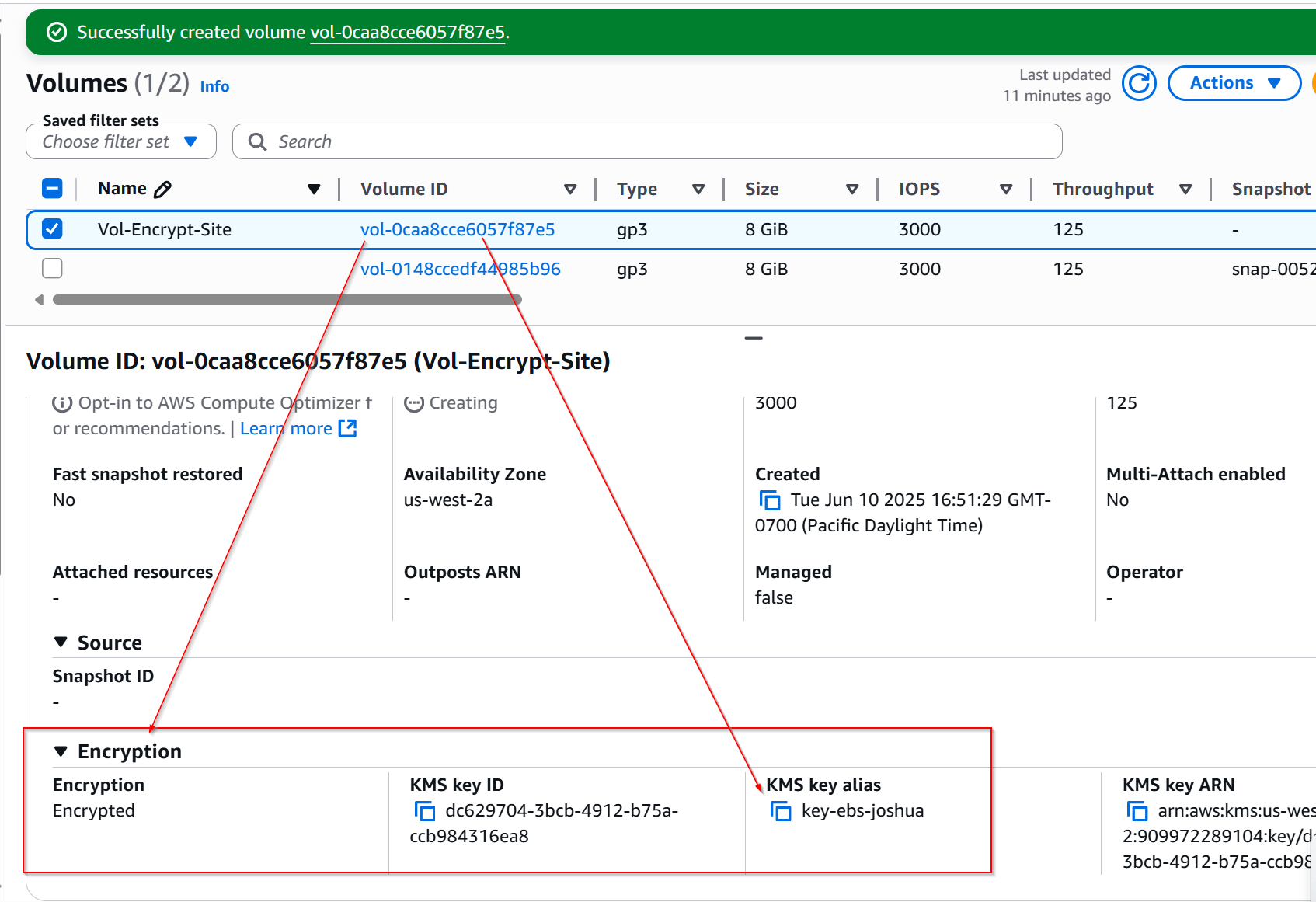
**Configure EBS**

Before we launch our second instance, we want to create a custom EBS volume to be used here. We are going to encrypt it with KMS. The overall goal is to attach it to our private EC2 instance and use it for notes when trying the challenges on the OWASP Juice Shop app.

**Step 1: Create an AWS KMS Key**

1. Navigate to KMS Dashboard: In the AWS Management Console, type "KMS" in the search bar and select "Key Management Service (KMS)" from the results.
2. Click on "Customer Managed Keys": On the KMS Dashboard, click on "Customer managed keys" in the left-hand menu.
3. Create a Key: Click the "Create key" button.
4. Configure Key Settings:
   1. Key Type: Choose "Symmetric" (the most common type for encryption).
   2. Key Usage: Choose "Encrypt and decrypt".
   3. Aliases: Enter an alias for your key (techone-ebsencryption-key).
   4. Description: Optionally, provide a description for the key.
5. Configure Key Administrative Permissions: Specify the IAM users or roles that can administer the key.
6. Configure Key Usage Permissions: Specify the IAM users or roles that can use the key to encrypt and decrypt data. Yours and the session manager role.
7. Review and Create: Review the configuration and click "Finish" to create the key.

**Step 2: Create an EBS Volume**

1. Navigate to EC2 Dashboard: In the AWS Management Console, type "EC2" in the search bar and select "EC2" from the results.
2. Click on "Volumes": In the left-hand menu, click on "Volumes" under the "Elastic Block Store" section. Create Volume: Click the "Create volume" button.
3. Configure Volume Settings:
   1. Volume Type: Choose the type of volume you need (e.g., gp3 for General Purpose SSD).
   2. Size (GiB): Specify the size of the volume - 8GB is fine.
   3. Availability Zone: Select the same Availability Zone as your EC2 instance to attach the volume later.
   4. Encryption: Enable encryption and select the KMS key you created earlier from the dropdown menu.
   5. Snapshot ID: Leave on default
4. Create Volume: Click the "Create volume" button.

You should now have a fresh EBS Volume, encrypted with your own KMS key. If you fancy going down a rabbit hole here, I recommend you investigate encrypted EBS snapshots and specifically how EBS encryption works when the snapshot is encrypted.

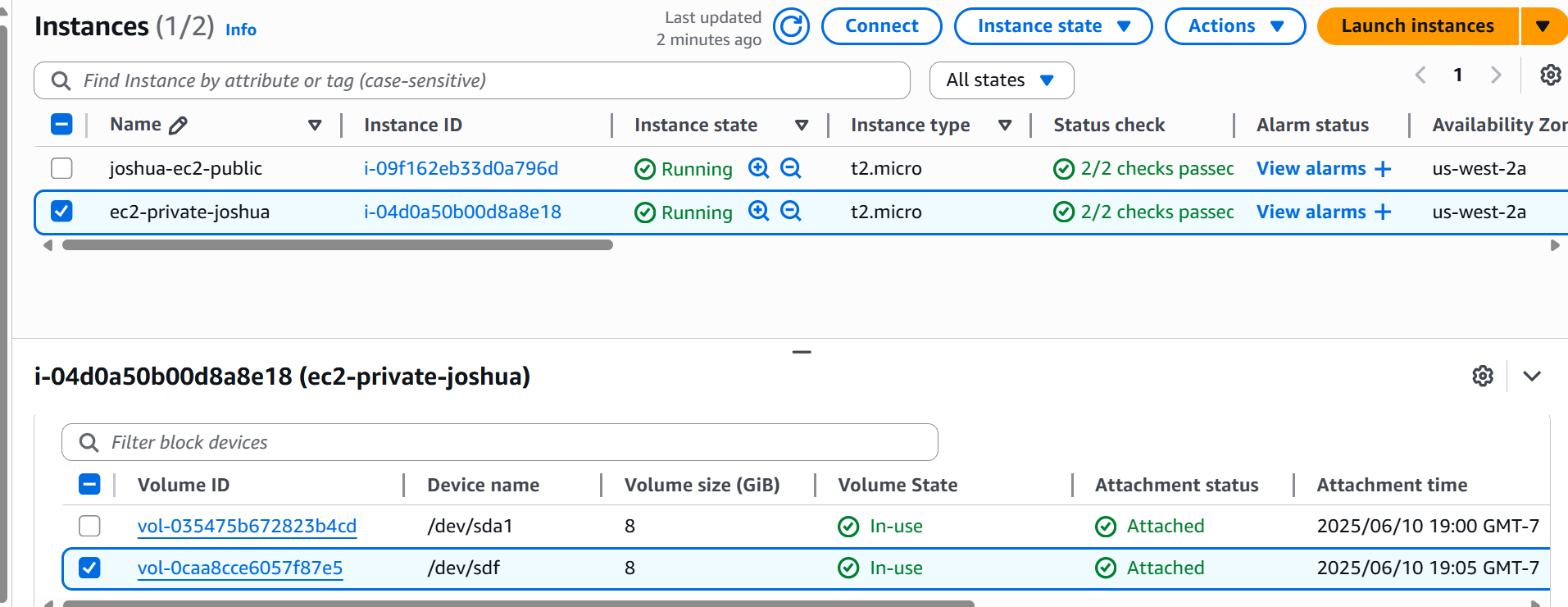
**Private EC2 Creation**

**Step 1:** Launch EC2 Instance in Private Subnet

1. Navigate to EC2 Dashboard:
   1. In the AWS Management Console, type "EC2" in the search bar and select "EC2" from the results.
2. Click on "Launch Instance":
3. Configure Instance Details:
   1. Name: Set the name to tech-one-ec2-private.
   2. AMI: Choose "Ubuntu Server 22.04 LTS"
   3. Instance Type: Select t2.micro.
   4. Key Pair: Select the key pair you created earlier (e.g., myec2-keypair) to enable SSH access.
4. Network Settings:
   1. VPC: Select your VPC (10.0.0.0/16).
   2. Subnet: Select your private subnet (10.0.2.0/24).
   3. Auto-assign Public IP: Disable this option since the instance is in a private subnet.
   4. IAM Role: Leave this field as "None" unless you have specific IAM requirements for this instance. You could maybe use an SSM role to have a managed session connection - fantastic for logging and not leaving SSH keys around but technically more challenging to configure.
5. Configure Storage: Leave the default storage settings.
   1. Leave the default storage settings
6. Configure Security Group:
   1. Create a New Security Group:
      1. Security Group Name: Provide a name for the security group (e.g., tech-one-private-sg).
      2. Description: Provide a description for clarity.
      3. Add Rules:
         1. SSH (Port 22): Allow SSH access from within your VPC or from a bastion host if you have one set up.
         2. Set the source to the CIDR block of your VPC (10.0.0.0/16) This enables SSH access only from within your VPC
         3. Launch the instance

**Step 2: Attach the EBS Volume to an EC2 Instance**

1. Select the Volume: In the "Volumes" list, select the volume you just created.
2. Attach Volume: Click on "Actions" and select "Attach volume".
3. Instance: Choose the EC2 instance to which you want to attach the volume to.
4. Device: Specify a device name (/dev/sdd). Note: Specifying /dev/sdd as the device name is typical, but depending on the Linux distribution, the device might appear differently (, /dev/nvme1n1 on newer instance types).
5. Attach: Click "Attach volume" to attach the volume to the instance.

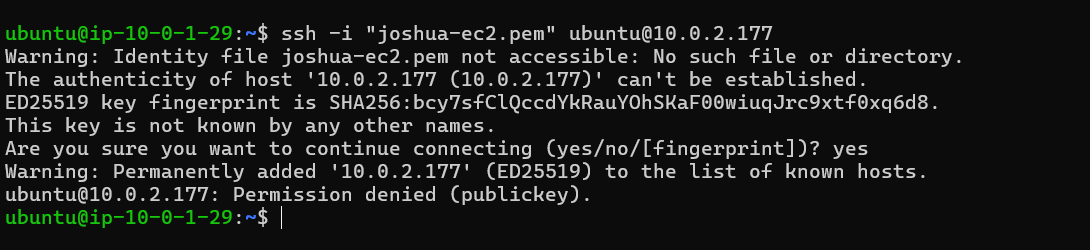
**First, ensure we can connect to both instances from our local machine:**

Try connecting to public instance with:

**ssh -i "tech-one-ec2.pem" ubuntu@<yourip>.**

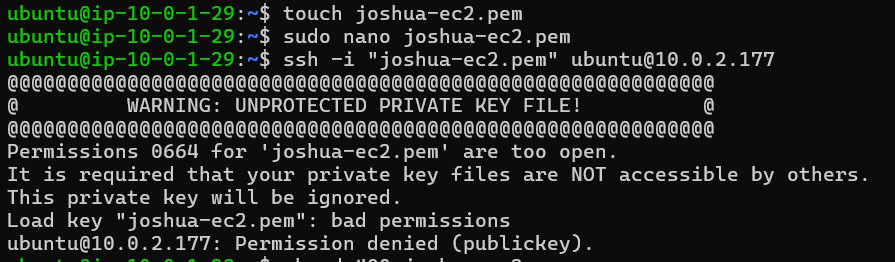
Once connected, we can connect to our private instance. Use the same command but change the IP to the private instance:

**ssh -i "tech-one-ec2.pem" ubuntu@<yourip>**

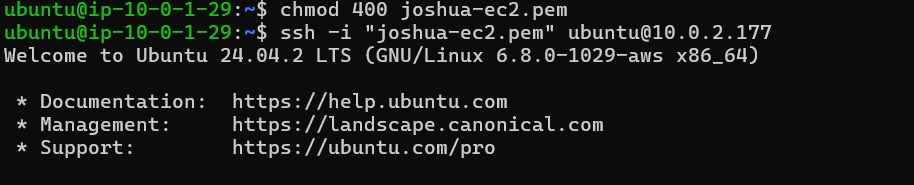


We received and error message stating permission denied due to public key.

Copy the value of your pem file from your local machine to your clipboard and on the public instance, create a new text file called tech-one-ec2.pem and paste in the value.

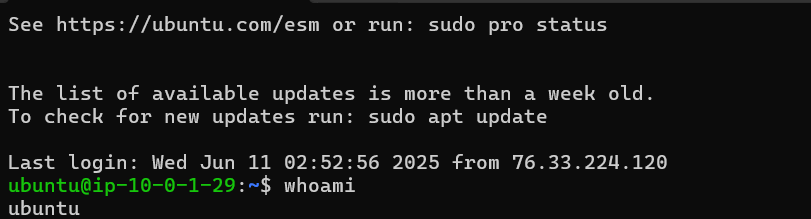


Oops! I forgot to update the pem file permissions to 400.

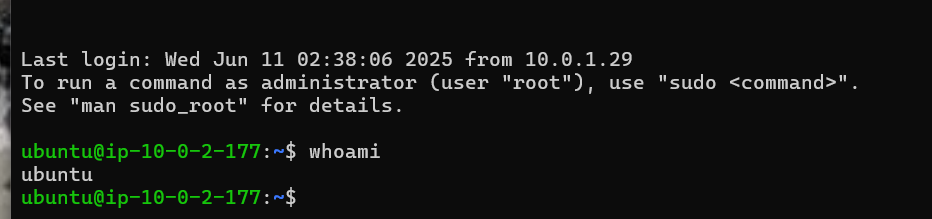


**SUCCESS!!!**

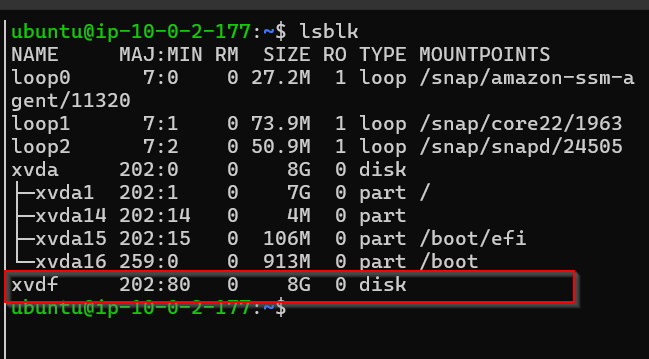
**Public Instance:**

****

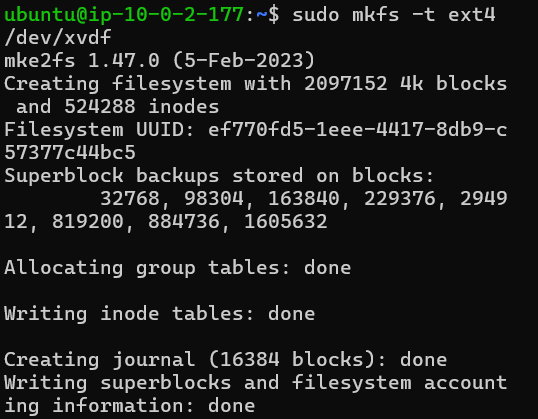
**Private Instance:**

****

Remember, we want to take notes for the Juice Shop Web app on the private instance encrypted volume.

**Mount your EBS volume on Ubuntu **

Check for the Attached Volume: **Command:** lsblk

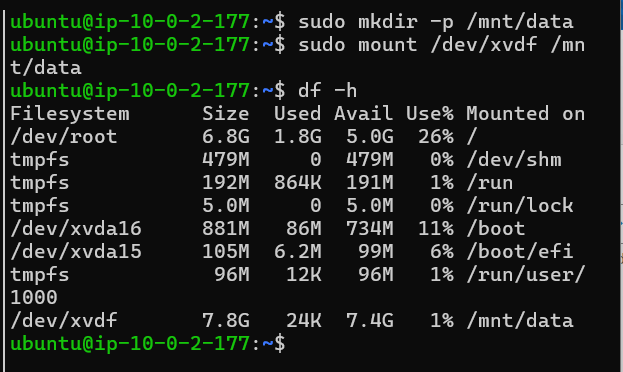
**Create a Filesystem:** The device name /dev/xvdf might be different depending on the instance type and the number of devices attached. 

**Command:** sudo mkfs -t ext4 /dev/xvdf

**Create a Mount Point:**

**Command:** sudo mkdir -p /mnt/data

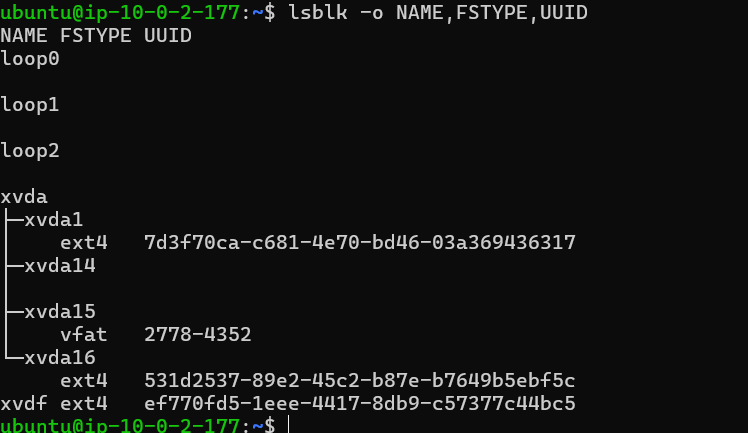
**Mount the Volume:**

**Command:** sudo mount /dev/xvdf /mnt/data 

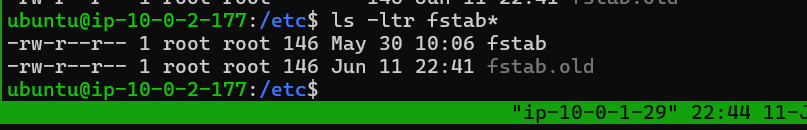
**Verify the Mount:** **Command:** df -h

Setup automatic mounting here as, we you plan on turning instance off and on.

**Command:** lsblk -o NAME,FSTYPE,UUID

(Note the UUID of the partition you want to automount)

**UUID:** ef770fd5-1eee-4417-8db9-c57377c44bc5

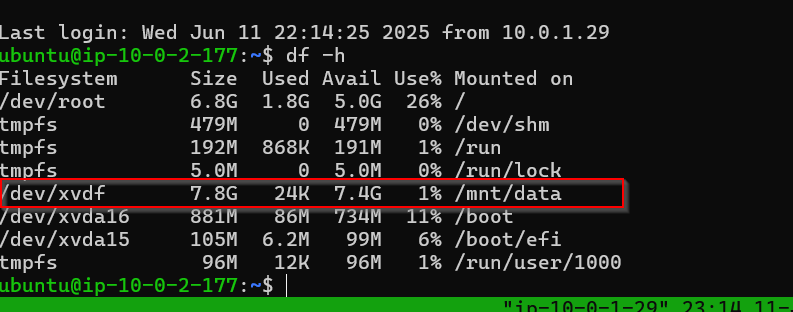
Create a backup of the fstab file just in case something unwanted happens.

**Command:** sudo cp /etc/fstab /etc/fstab.old

**Command to open file:** sudo nano fstab

Copy below and add it to the end of the file.

**Copy** UUID=ef770fd5-1eee-4417-8db9-c57377c44bc5 /mnt/data ext4 defaults 0 2



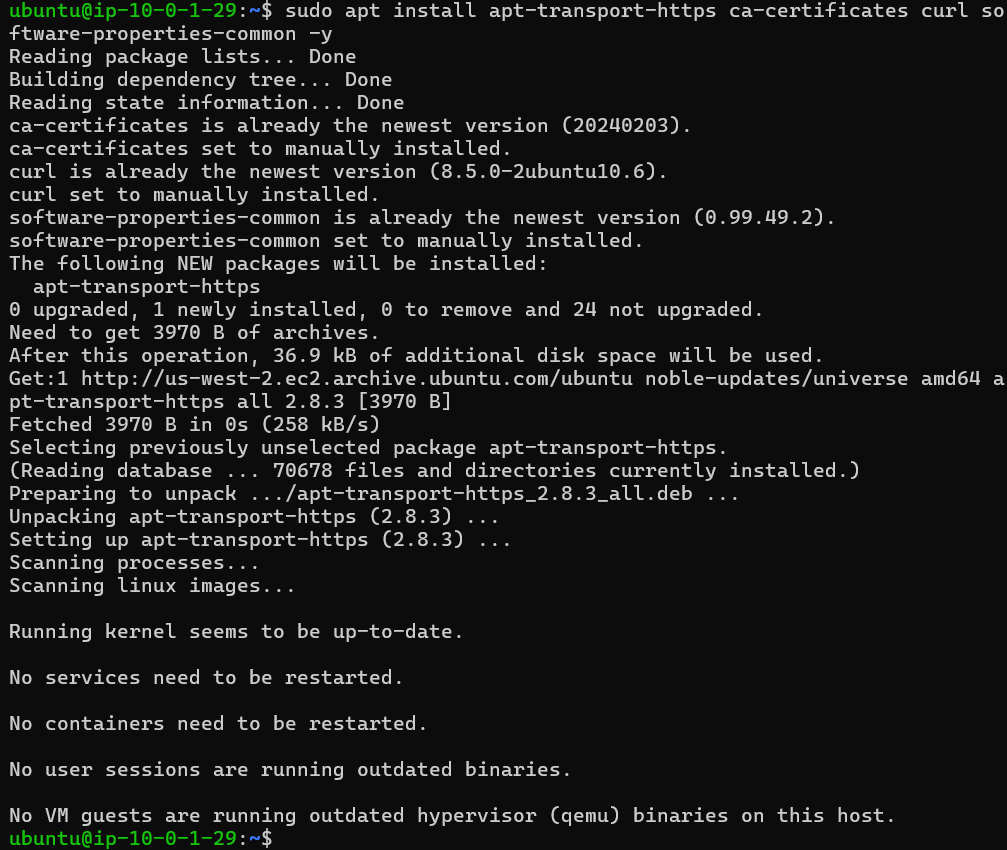
As we can see here. File system auto mounted after reboot.

**Installing Docker**

We are going to be installing Docker here. To install Docker, we will do the following on our **public instance**, remember we are only using the private instance for secret note-taking about the JuiceShop app challenges.

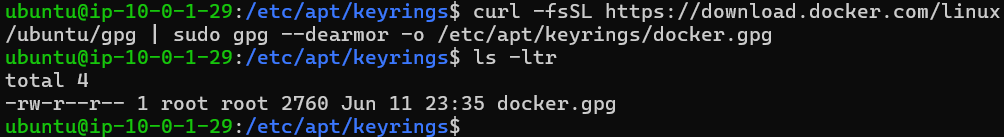
**Update Package Information:**

**Command:** sudo apt update

Install Packages: 

**Command:**

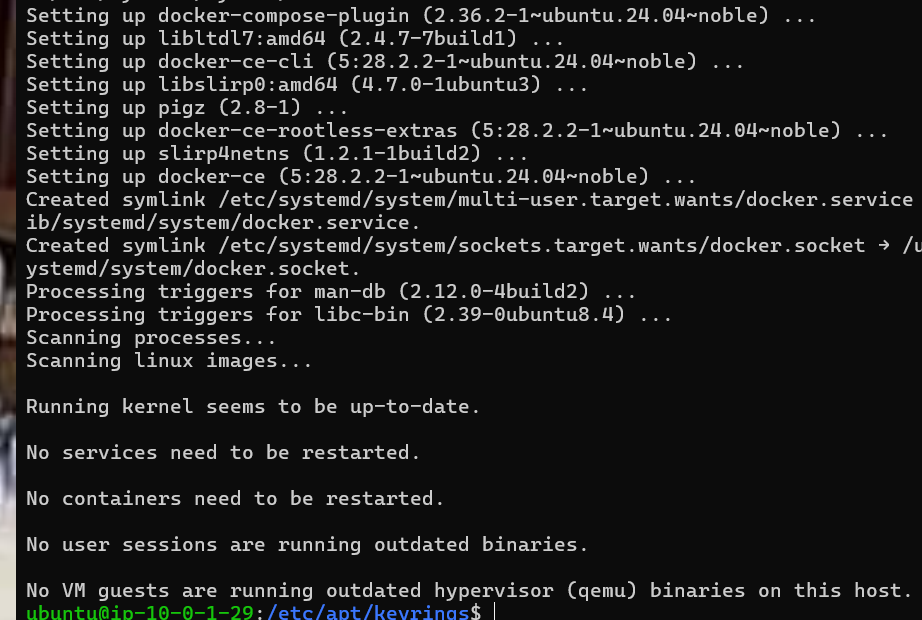
sudo apt install apt-transport-https ca-certificates curl software-properties-common -y

**Download Docker’s Official GPG Key:** 

**Command:** curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

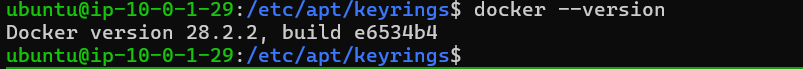
**Add the Docker repository using the keyring location:**

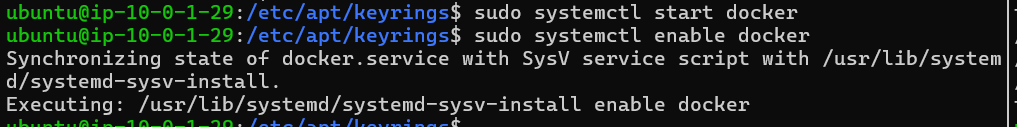
**Command:** echo "deb [signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

**Install Docker:**

**Command:** sudo apt update

**Command:** sudo apt install docker-ce -y



**Start and Enable Docker Service: **

**Command:** sudo systemctl start docker

**Command:** sudo systemctl enable docker

Add Current User to Docker Group: This allows running Docker commands without sudo. Log out and back in for the changes to take effect.

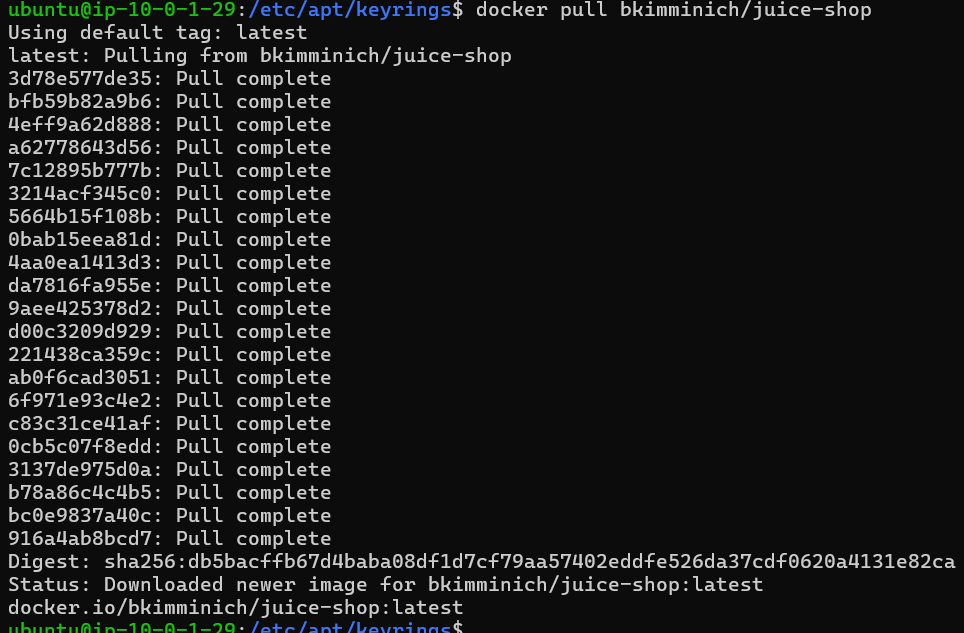
**Command:** sudo usermod -aG docker ${USER}

**Run Juice Shop with Docker**

**What exactly is the OWASP Juice Shop?**

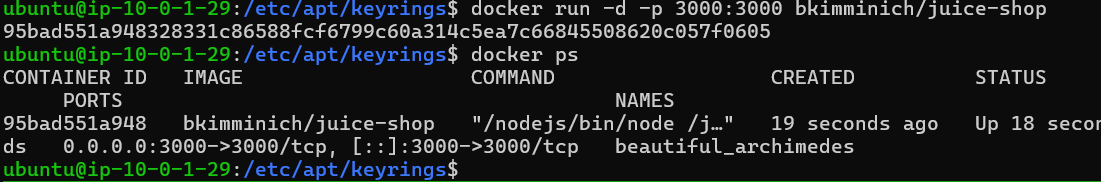
The OWASP Juice Shop is a deliberately insecure web application developed by the Open Web Application Security Project (OWASP) designed for security testing, training, and demonstration purposes. Juice Shop includes a challenge-based learning model where users complete various security challenges of varying difficulty levels. As you find and exploit vulnerabilities, you earn points and unlock additional challenges, making the learning process engaging and interactive.

**Pull the Juice Shop Docker Image:**



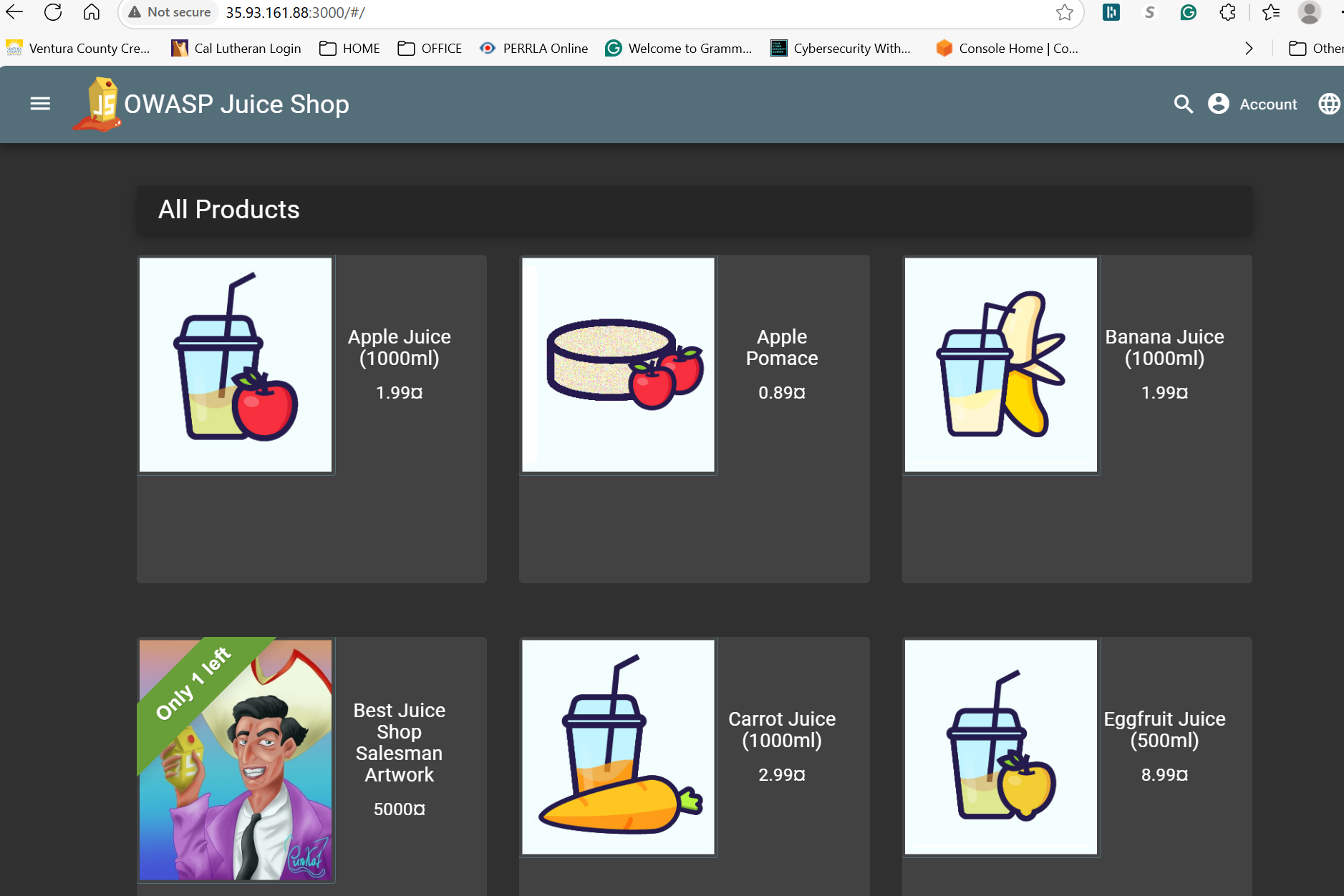
**Command:** docker pull bkimminich/juice-shop

**Run the Docker Container:**

**Command:** docker run -d -p 3000:3000 bkimminich/juice-shop

You should see bkimminich/juice-shop listed as running.

**Connect to Juice Shop**



**Access Juice Shop in Your Browser:**

Open a web browser. Go to http://<YOURIP>:3000

🛠️ \*\*Takeaways\*\*

✔️ Strengthened my understanding of \*\*VPC architecture, IAM roles, security groups, and cloud encryption\*\*.

✔️ Explored \*\*real-world attack surfaces\*\* with a \*\*deliberately vulnerable web app\*\*, emphasizing defensive cloud security practices.

✔️ Reinforced \*\*hands-on troubleshooting\*\*, ensuring all instances interact securely within the \*\*AWS cloud environment\*\*.

💡 \*\*Next Steps\*\*

Planning to expand this project by automating deployments with Terraform and integrating security scans to detect vulnerabilities dynamically.

References

Pearce, W. (n.d.). Tech one twenty. assets.stanwith.me. Retrieved May 18, 2025, from <https://assets.stanwith.me/live/msc/26232161/qs6wz/tech%20one%20twenty.pdf>