**Routing policies (Route 53 (DNS))**

* In **AWS**, specifically in **Amazon Route 53**, **routing policies** are rules that control how **DNS queries** are answered. In simple terms, routing policies **determine where user traffic goes** when someone accesses your domain or application.
* Routing policies determine how traffic is distributed across your AWS resources (e.g., EC2 instances, S3 buckets, or Lambda functions). AWS offers different policies for **Route 53 (DNS)** and **Load Balancers**, each optimized for specific use cases.
* A **routing policy** defines **how Route 53 responds** to a DNS request (like a user typing your domain into a browser). Based on the policy, Route 53 decides **which resource (server, IP, load balancer, etc.) to send the user to.**
* It is not an exact Load Balancer it just act as a Load Balancer, it route/distribute the traffic based on the routing policies/methods.

**Types of Routing Policies in AWS Route 53**

| **Policy Type** | **Description** | **Use Case** |
| --- | --- | --- |
| **Simple** | Routes to one resource (no logic). | Static websites, single server. |
| **Weighted** | Splits traffic based on weights. | Load balancing, A/B testing. |
| **Latency-based** | Routes to the resource with the lowest latency for the user. | Fast global response. |
| **Failover** | Sends traffic to a backup if the main resource fails. | High availability. |
| **Geolocation** | Routes users based on country/region. | Localized content or compliance. |
| **Geoproximity** | Routes based on user and resource location, with optional biasing. | Traffic control by geography. |
| **Multi-value answer** | Returns multiple IPs with health checks (like round-robin DNS). | Basic load balancing + redundancy. |

While Route 53 routing policies are **powerful for traffic control**, they don’t directly inspect or handle traffic like **Elastic Load Balancer (ELB)**. Instead, they direct clients to the right resource at the **DNS level**.

The main Route 53 routing policies are:

1. Failover🡺primary and secondary region.
2. Latency
3. Weightage
4. Geo-Location

### 1. ****Failover Routing Policy****

**Purpose**: Improves **availability** by routing traffic to a **secondary (backup)** server if the **primary** fails.

**Example**:

* You host your website on a **primary server in Mumbai**.
* You have a **backup server in Delhi**.
* If Route 53 detects that the Mumbai server is **down** (via health check), it **automatically switches** users to the Delhi server.

### 2. ****Latency-Based Routing Policy****

**Purpose**: Improves **performance** by routing users to the server with the **lowest latency** (fastest response time).

**Example**:

* You have servers in **Tokyo** and **Frankfurt**.
* A user in **Germany** gets routed to **Frankfurt**.
* A user in **Japan** gets routed to **Tokyo**.

### 3. ****Weighted Routing Policy****

**Purpose**: Distributes traffic based on **assigned weights**, useful for **load balancing** or **A/B testing**.

**Example**:

* Two servers: one in **Bangalore** (weight 80), one in **Chennai** (weight 20).
* 80% of the traffic goes to Bangalore, 20% to Chennai.

### 4. ****Geolocation Routing Policy****

**Purpose**: Routes users based on their **geographic location**, useful for **local content** or compliance.

**Example**:

* Users from the **US** are routed to a server in **Oregon**.
* Users from **India** are routed to a server in **Mumbai**.
* Other regions go to a **default** server.

So that the Users get content in their **local language** or **currency** which is configured in the specific region servers.

Here we can assign a specific region server to the specific users based on the content which is present in it.

Let’s work with Routing policies practically:

Process:

1. Create two EC2 instances in different regions.
2. In this instances install the Nginx webservers and edit the Heading content of it.
3. First configure the Health check in Rout 53.
4. Now create the hosted zones (primary & secondary) in which the instances IP’s are configured.
5. Now by using Record name, brows in any browser you will get the specific Nginx web page.
6. Then by stopping any of the instance again brows, and check where the traffic is going and hit.

Step1: Create the two instances (Ohio-server & NVirginia-server).

For that first we have to create the VPC, within it subnet, Internet gateway, Route table, SG and then EC2 instance.

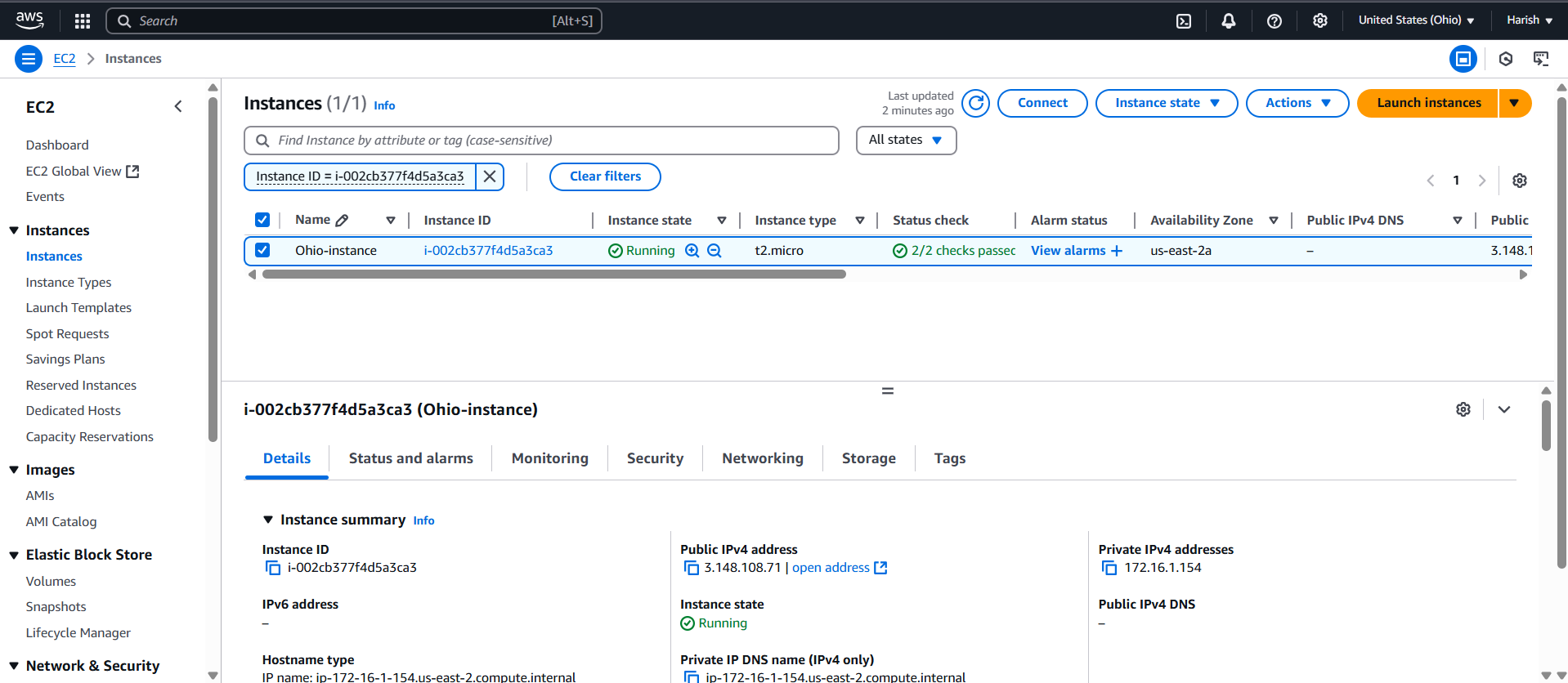


Fig: Instance in the Ohio region (Ohio-instance).

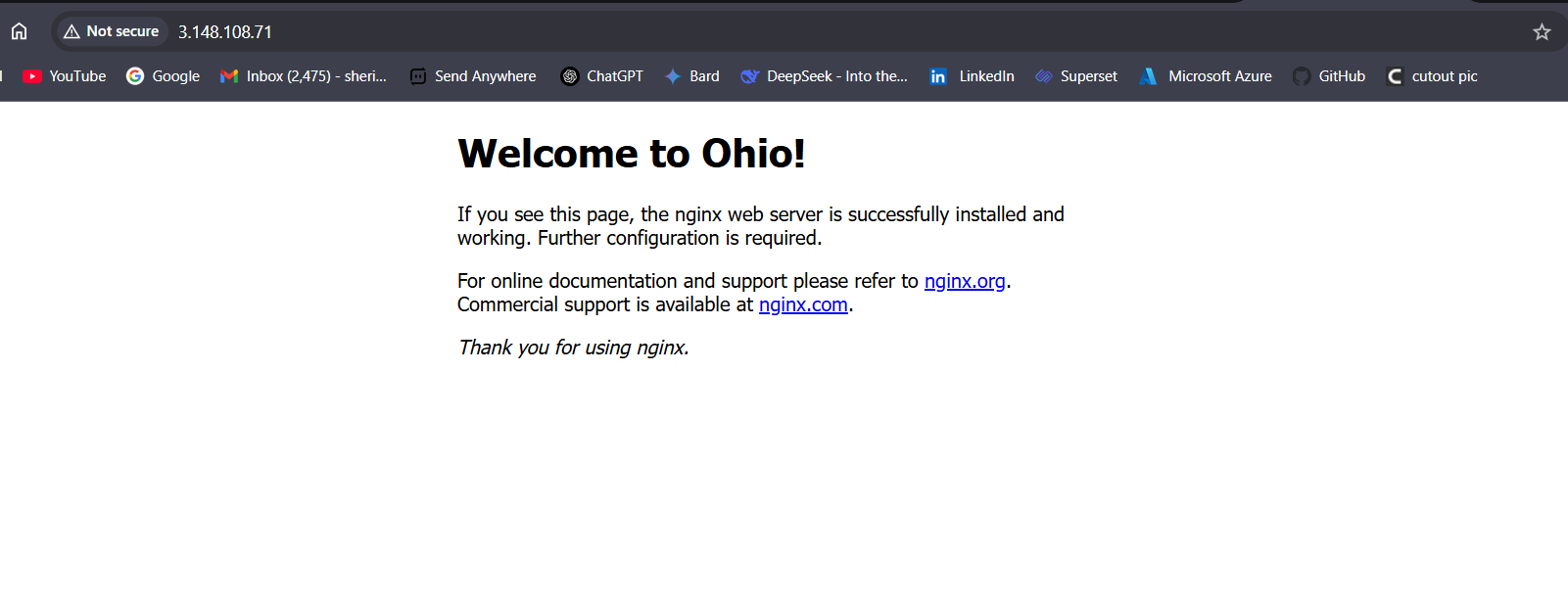


Fig: Nginx web server in the instance (Ohio-instance).

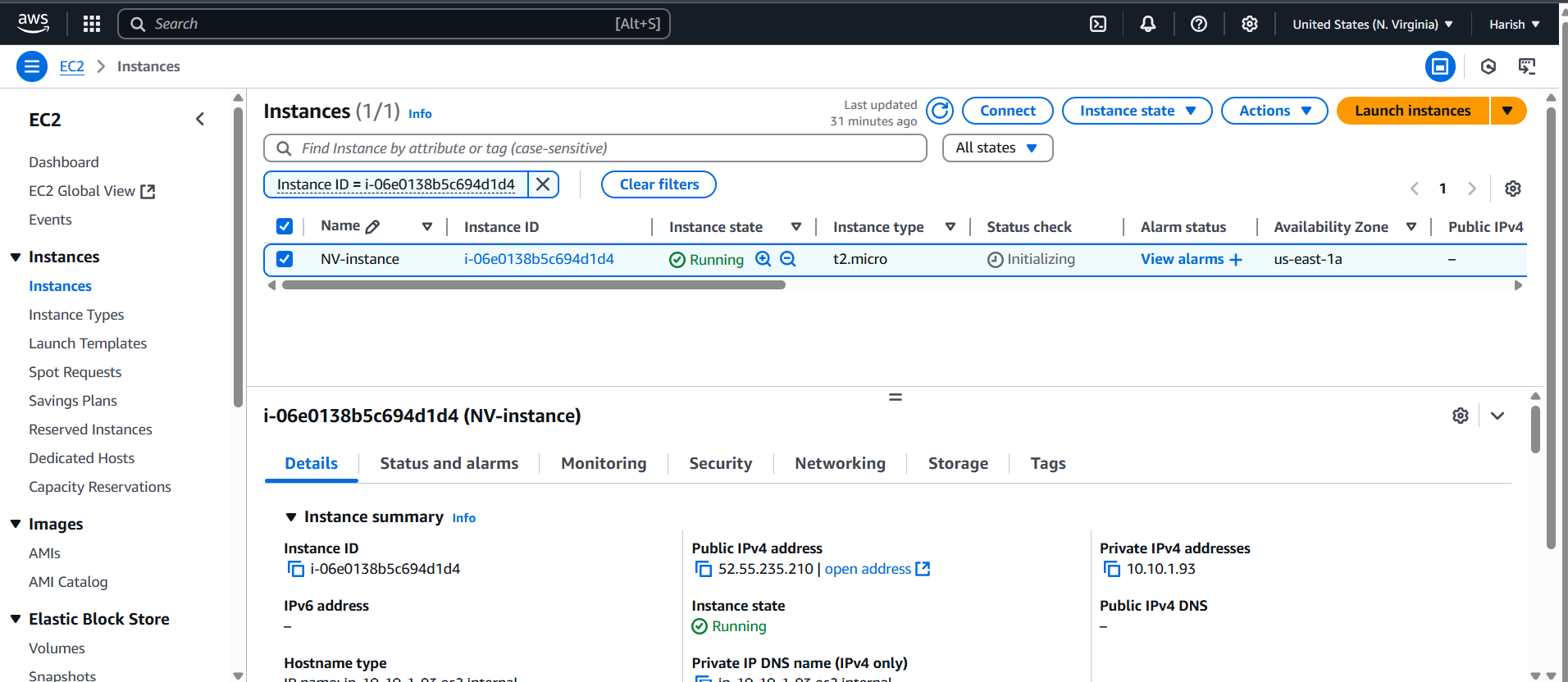
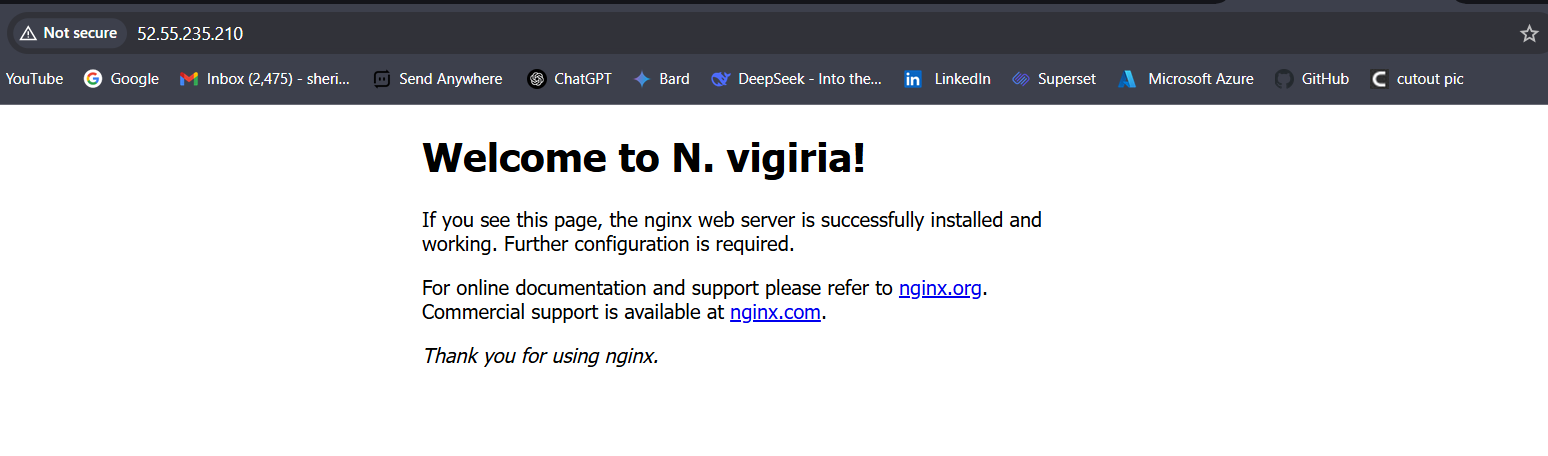


Fig: Instance in the N. Virginia (NV-instance).

 Fig: Nginx web server in the instance (NV-instance).

**Step2:** Configure the Route 53 routing policies.

In order to configure the routing policies first we have to configure the Health checks for both the instances.

And next we have to configure the Hosted zones, here we can able to choose which policies/method is used to route our traffic.

**Configuring & create Health check:**

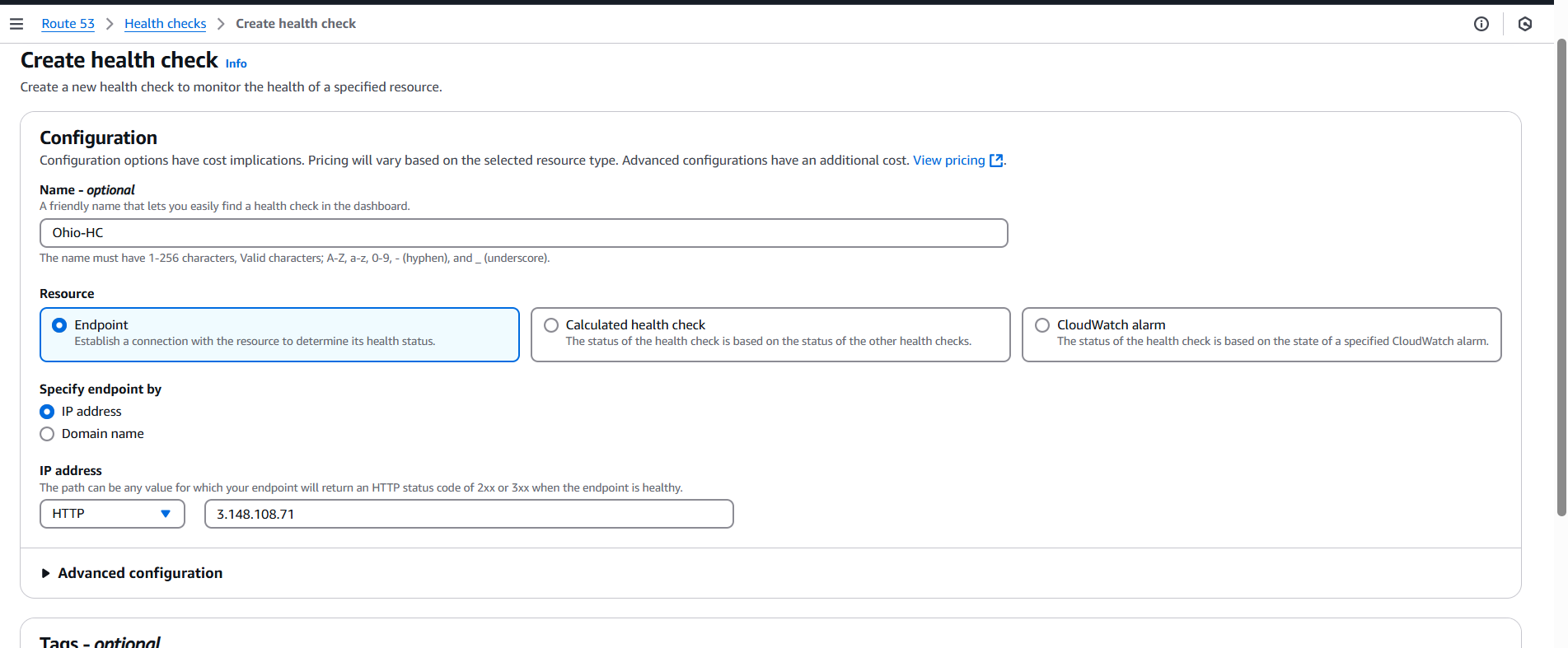


Fig: Configuring of health check for instance (Ohio-instance).

Public IP: 3.148.108.71. (Ohio-instance).

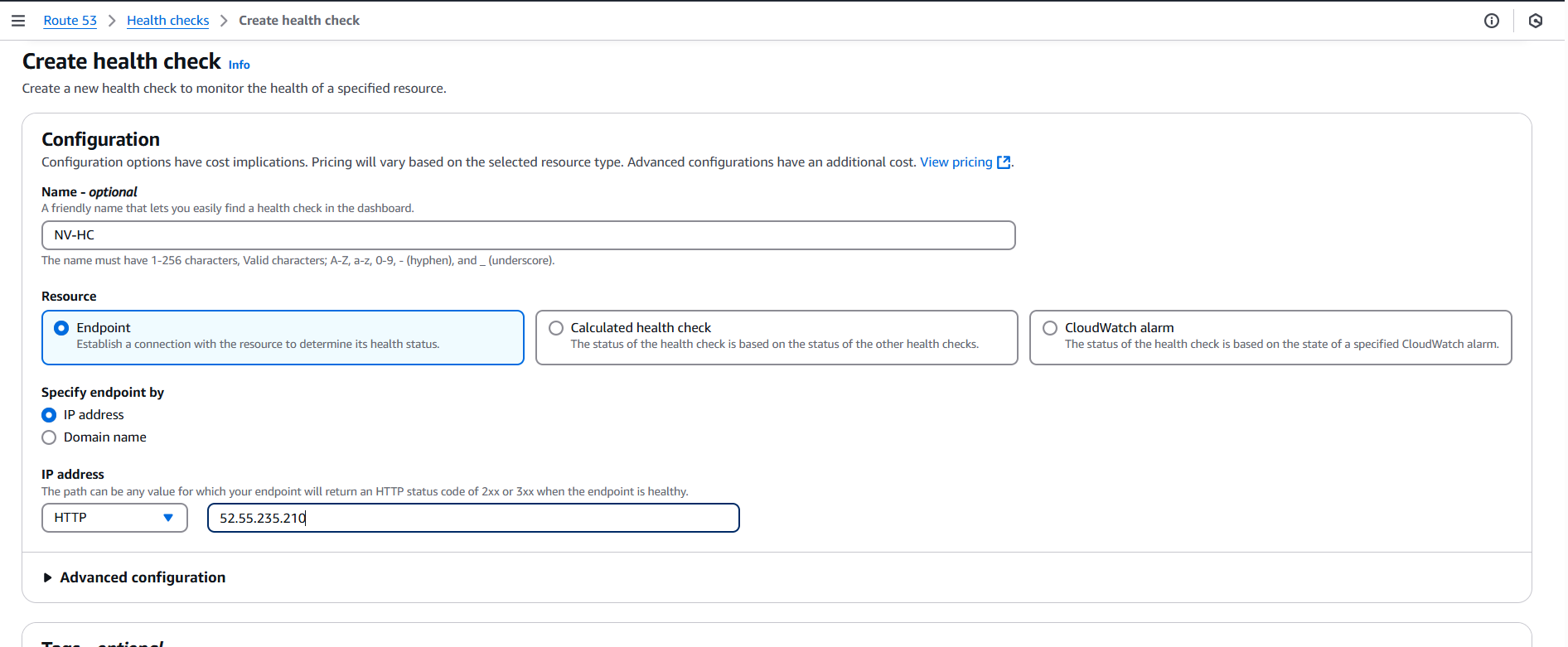


Fig: Configuring of health check for instance (NV-instance). [public IP: 52.55.235.210]

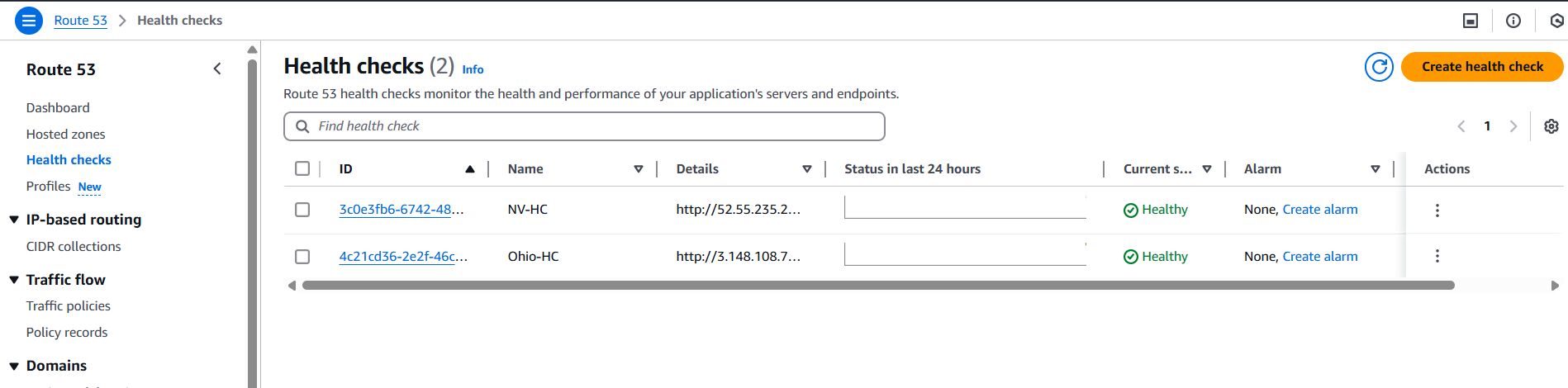


Fig: Two Health checks for two instances are created successfully.

**Configure & create Hosted Zones:**

First we have to create the Hosted zone with our specific domain name, then within the Hosted zone create a Record. The record may be failover, weight, Geo-location, Latency and IP-based.

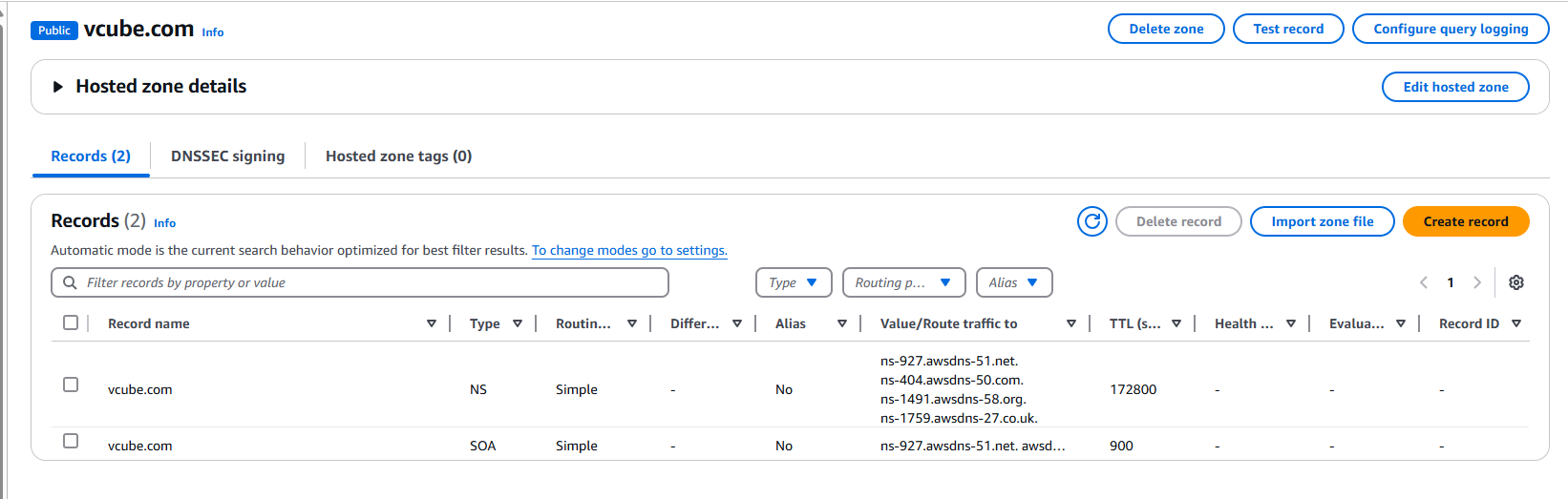


Fig: Create a record.

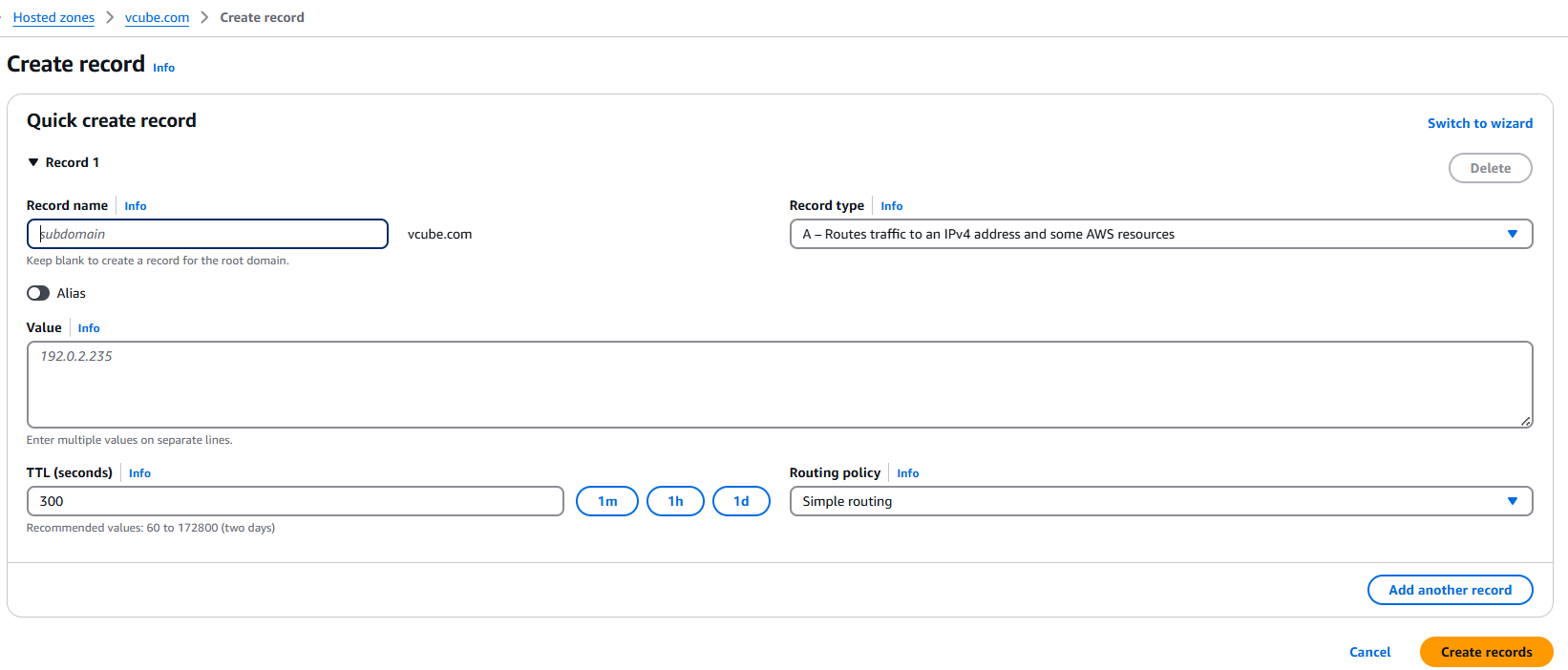
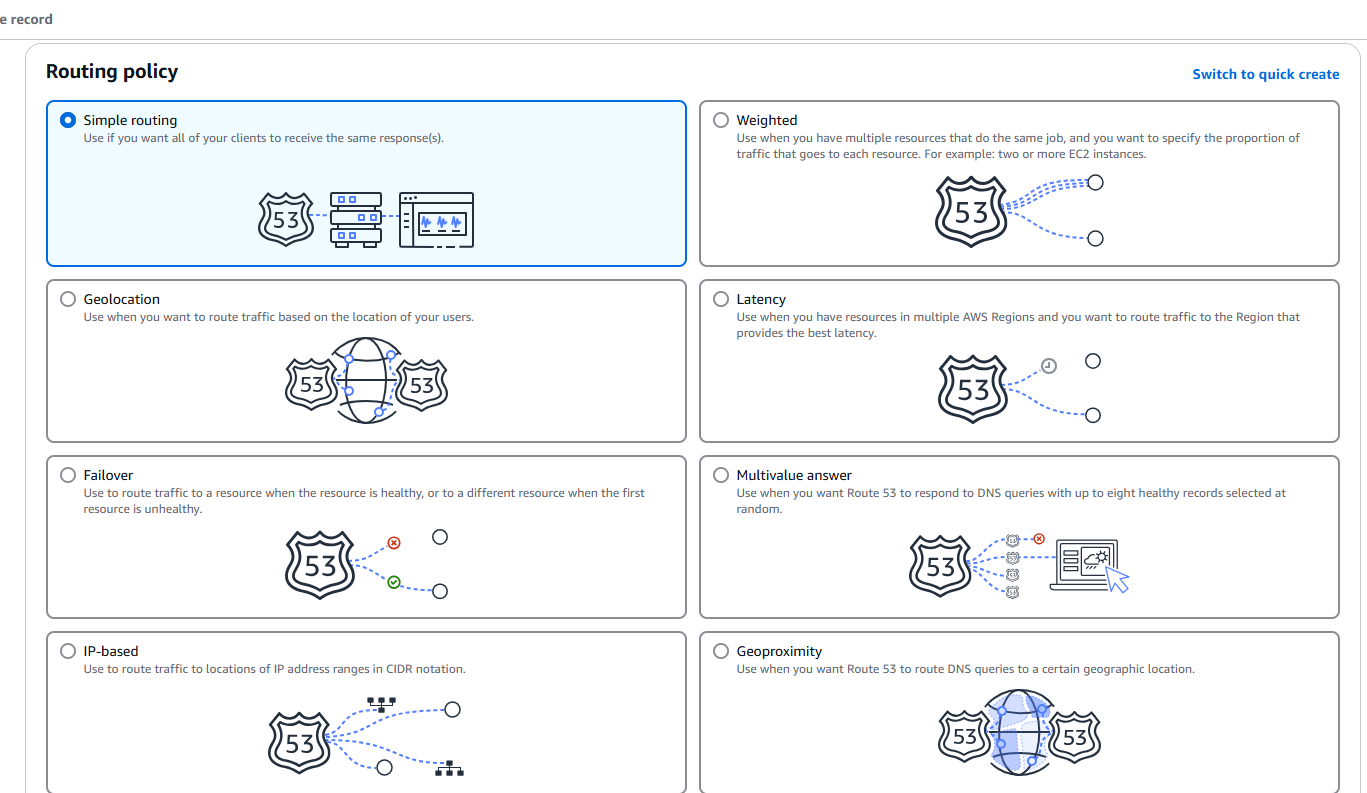


Fig: Change the GUI view of creating record.



**Fig:** Configuring failover (In Wizard GUI view).

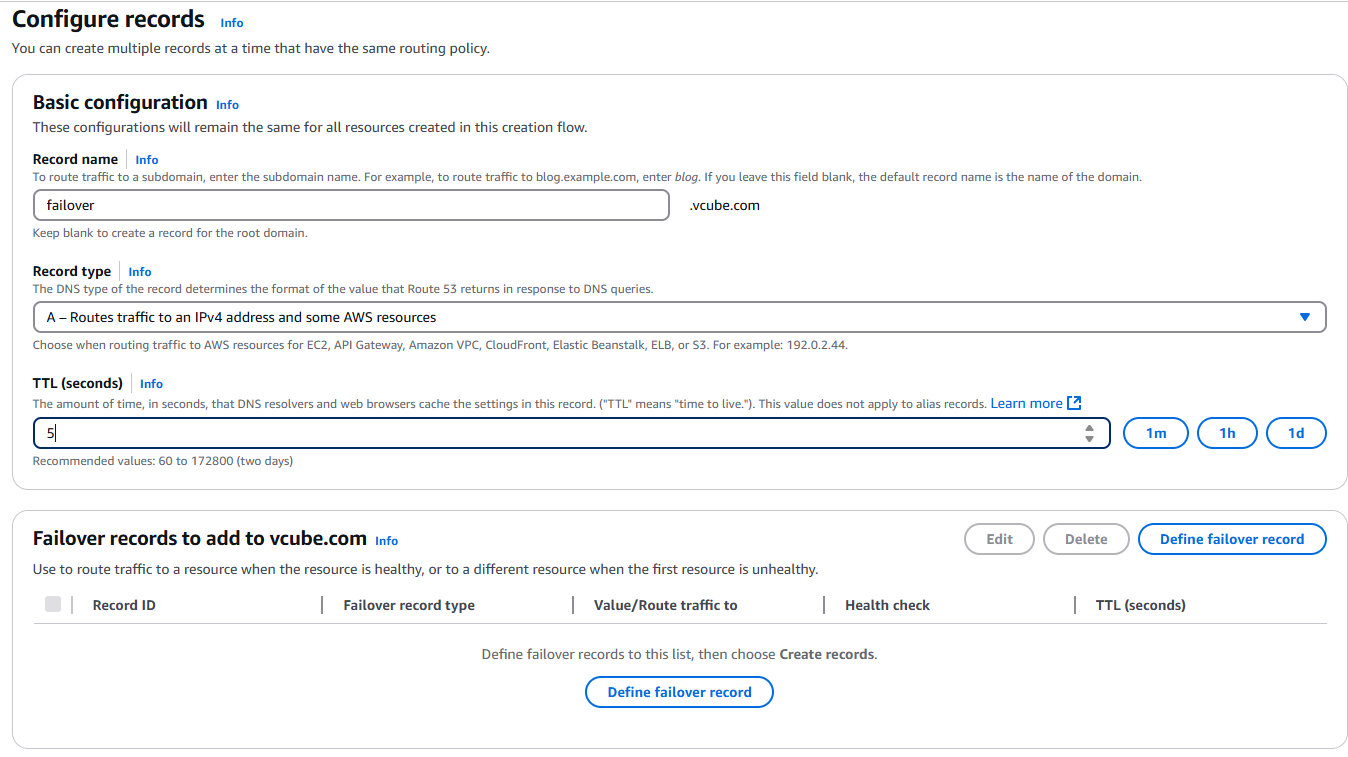
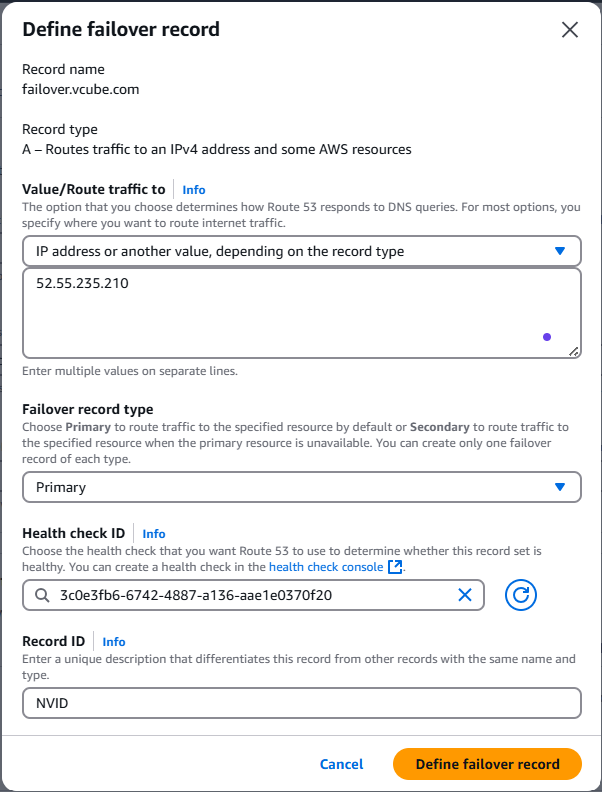


Fig: Configuring Failover basics.



Public IP: 52.55.235.210 (NV-instance)

Health check NV-HC (N. Virginia)

Fig: Adding of NV-instance as a primary web server.

Now create one more failover for the secondary server.

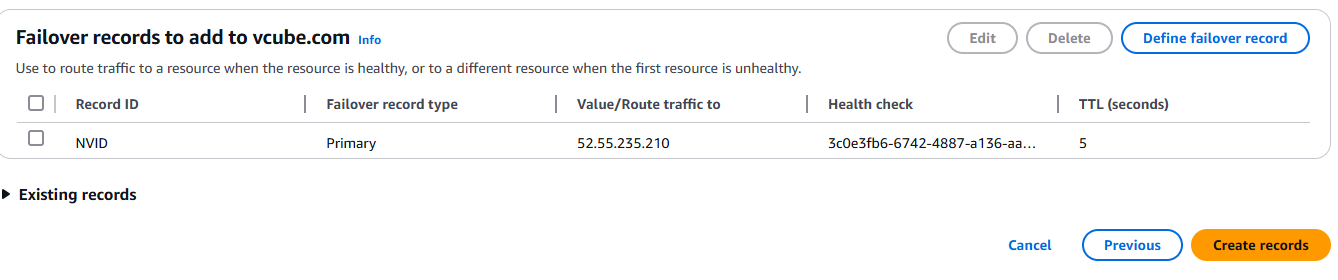
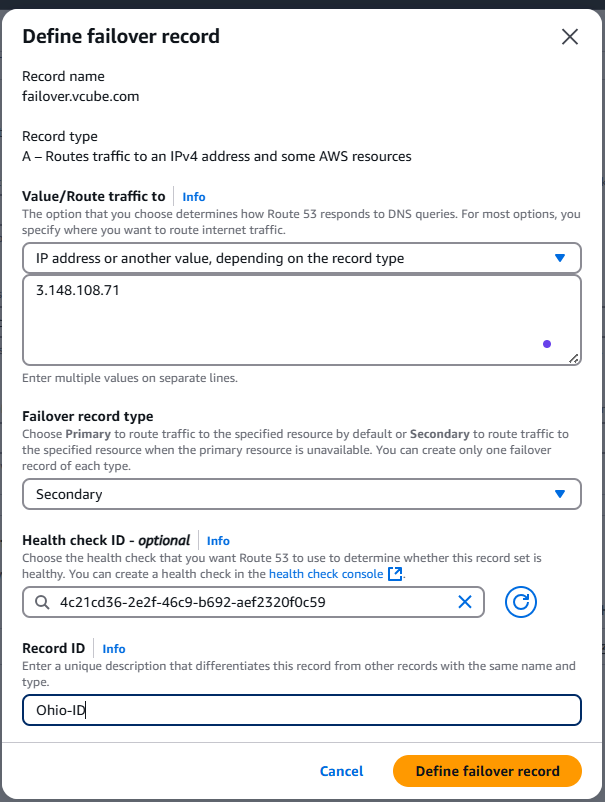


Fig: Define failover record.



Public IP: 3.148.108.71 (Ohio-instance).

Health check of Ohio-instance

Fig: Configuring of failover for Ohio-instance as a secondary server.

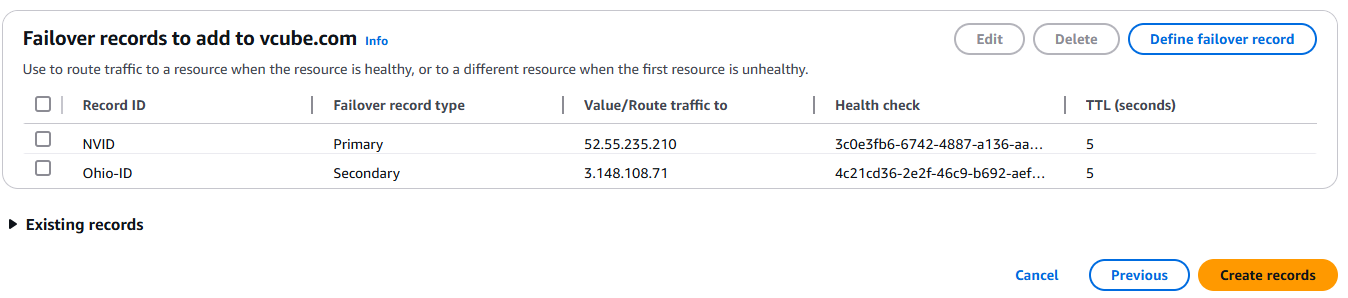


Fig: Both the failovers at N. Virginia and Ohio region are created successfully.

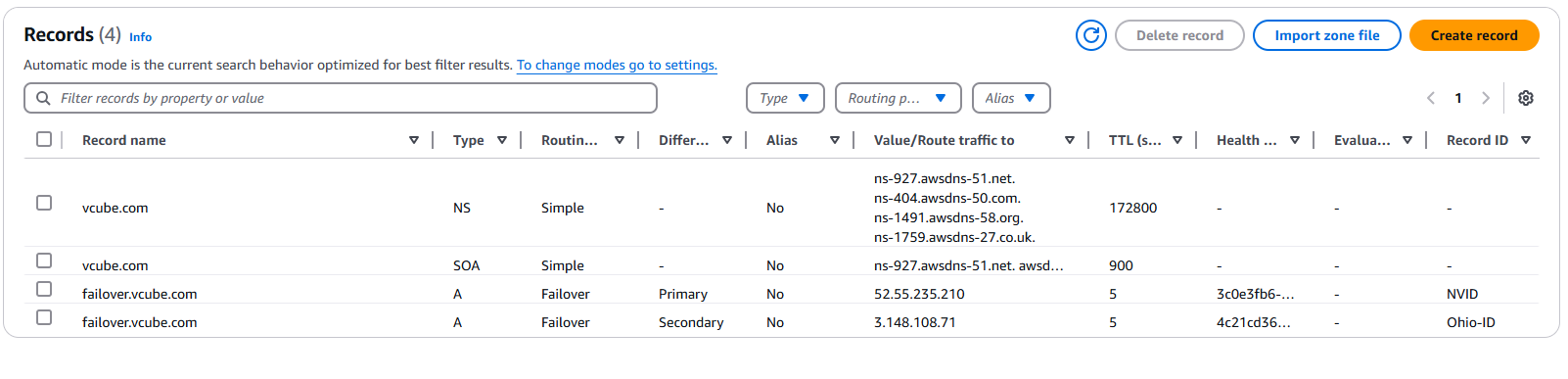


Fig: Record for the failovers (primary & secondary) are created successfully.

Now copy the record name and browse in any browser, and check whether it is working or not.

**Note:** We cannot browse our Nginx web browser unless its four name servers get updated in the Godaddy.