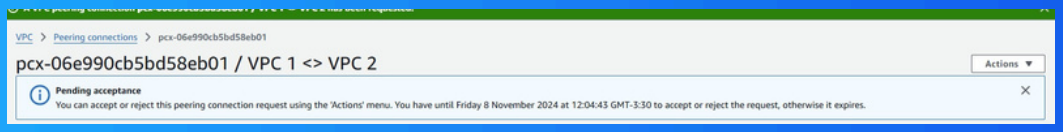
NextWork.org

VPC Peering

Kanika Mathur
github.com/KanikaGenesis




Kanika Mathur
github.com/KanikaGenesis

In the first part of my project...

Step 1 Set up my VPC

In this step, I am using the VPC resource map/launch wizard to create Two VPCs and their components in just minutes.

Step 2 Create a Peering Connection

In this step, I am setting up a VPC Peering Connection,a VPC component designed to connect two VPCs directly.

Step 3 Update Route Tables

I am editing the route table for VPC 1 andVPC 2 so that traffic can be directed to the Peering connection set up.

Step 4 Launch EC2 Instances

I am launching an EC2 instance in each of the VPCs (VPC 1 AND VPC 2), so that I can directly connect with my instances later and test my VPC peering connection.

Kanika Mathur
github.com/KanikaGenesis

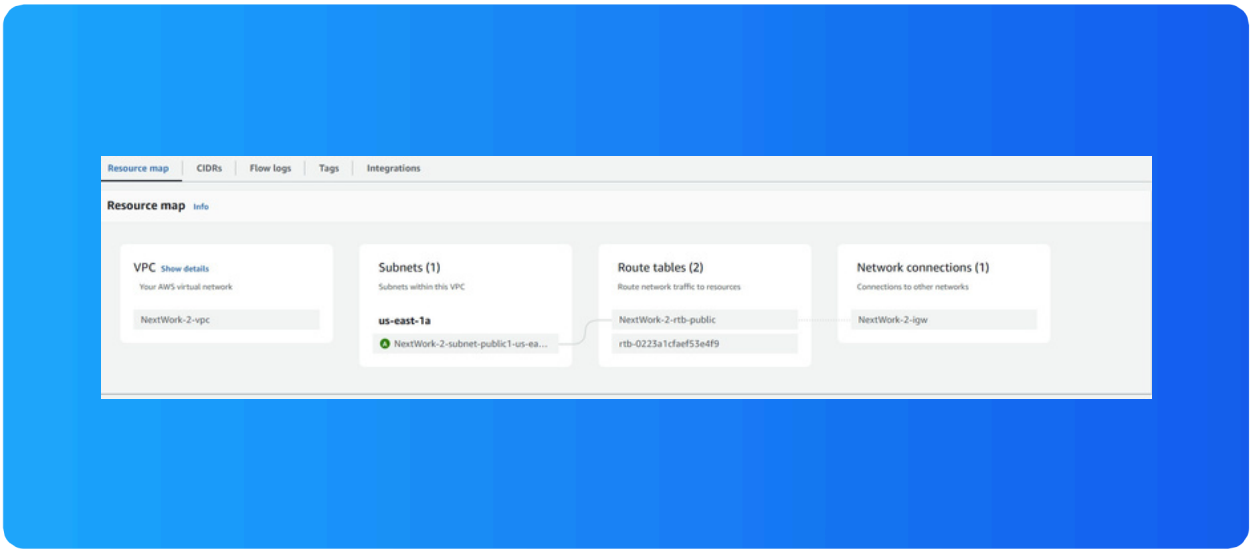
Multi-VPC Architecture

I started my project by launching two VPC - they have unique CIDR blocks and they each have one public subnet.

The CIDR blocks for VPCs 1 and 2 are 10.1.0.0/16 and 10.2.0.0/16 respectively. They have to be unique because once you set up a VPC peering connection, the route table needs unique addresses for correct routing across VPC.

I also launched 2 EC2 instances

I didn't set up key pairs for these EC2 instances because I am using EC2 Instance Connect to directly connect with my EC2 instance later in this project, which handles key pair creation and management for me.




Kanika Mathur
github.com/KanikaGenesis

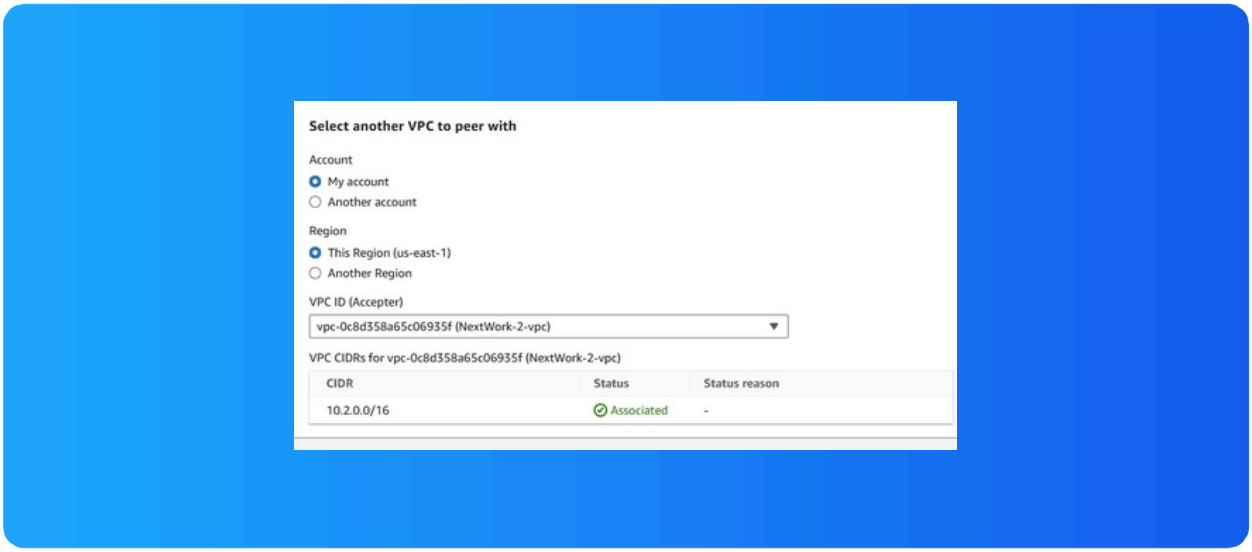
NextWork.org

VPC Peering

A VPC peering connection is a direct connection between two VPCs!

A peering connection lets VPCs and their resources route traffic between them using their private IP addresses.

The Requester is the VPC that initiates a peering connection. They will be sending the other VPC an invitation. The Accepter is the VPC that receives a peering connection request! The Accepter can either accept or decline the invitation.



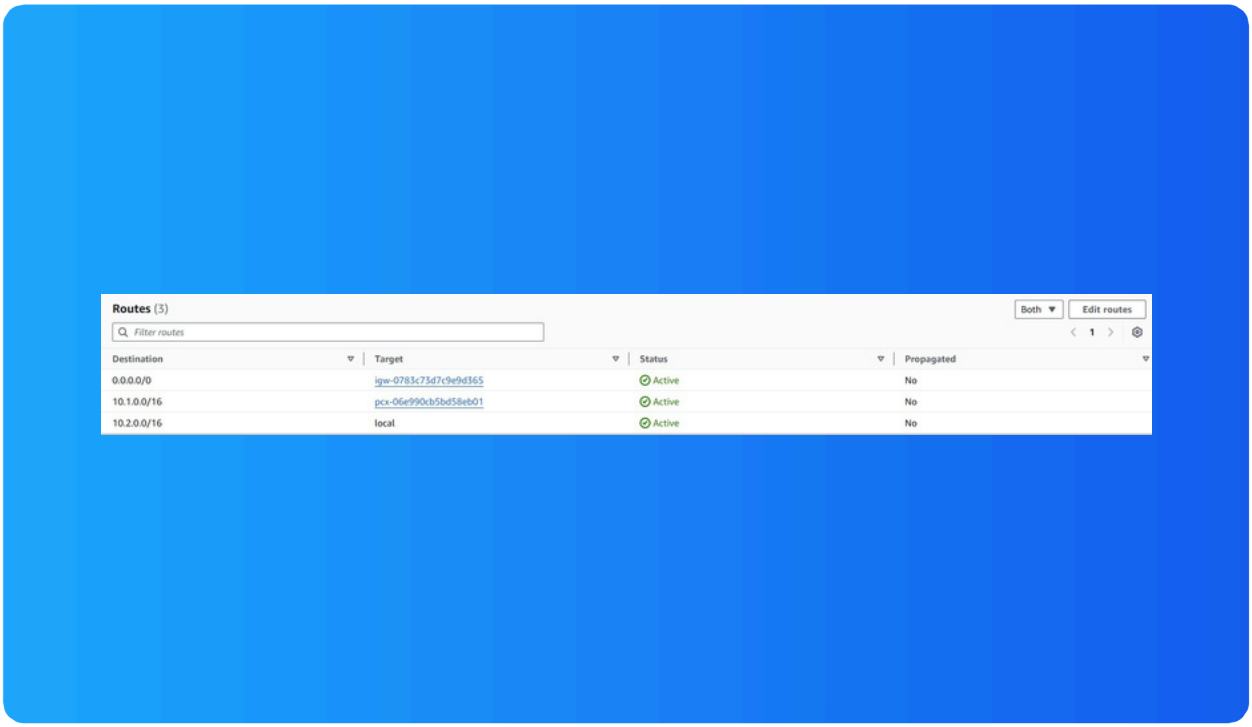
Kanika Mathur
github.com/KanikaGenesis

NextWork.org

Updating route tables

The default route table doesn't have a route using the peering connection yet; this needs to be set up so that resources can be directed to the peering connection when trying to reach the other VPC.

My VPCs' new routes have a destination of the other VPCs CIDR Block. The routes' target was the Peering Connection.




Kanika Mathur
github.com/KanikaGenesis

In the second part of my project...

Step 5 Use EC2 Instance Connect

I am using EC2 Instance Connect to connect directly with my first EC2 instance. I need to do this because I will need to use my EC2 instance for connectivity (to validate our VPC peering) test later in this project.

Step 6 Connect to EC2 Instance 1

I am reattempting my connection to Instance - NextWork VPC 1 and resolving another error preventing me from using EC2 Instance Connect to directly connect with the instance.

Step 7 Test VPC Peering

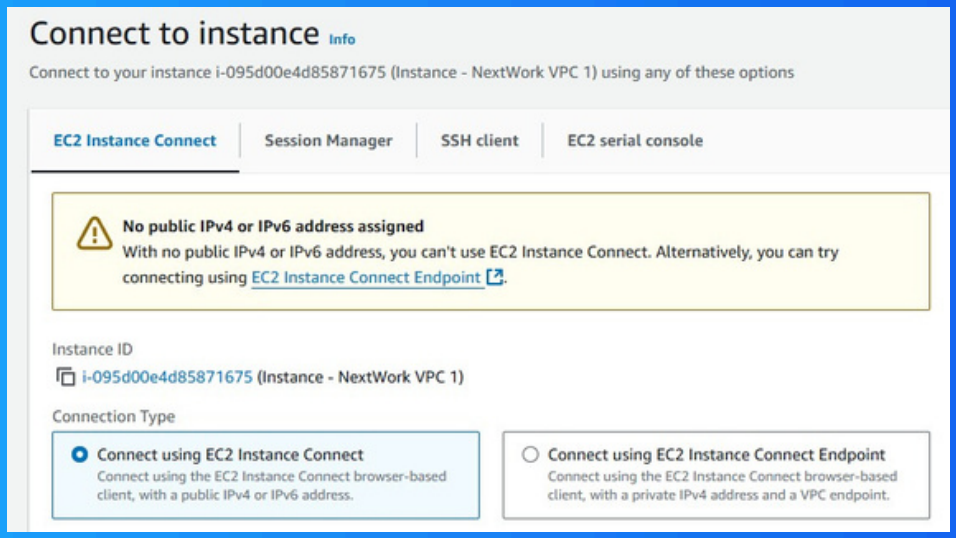
Iam going to use the instance - NextWork VPC 1 to attempt a direct connection with instance - NextWork VPC 2 so that I can validate that my peering connection is set up properly.

Kanika Mathur
github.com/KanikaGenesis

Troubleshooting Instance Connect

Next, I used EC2 Instance Connect to directly connect with Instance - NextWork VPC 1 just by the AWS management console.

I was stopped from using EC2 Instance Connect because my instance did not have a public IPv4 address. An EC2 instance must have a public IPv4 address and be in a public subnet.




Kanika Mathur
github.com/KanikaGenesis

Elastic IP addresses

To resolve this error, I set up Elastic IP addresses. Elastic IP addresses are public IPv4 addresses that we can request for our AWS Account and then delegate to specific resources.

Associating an Elastic IP address resolved the error because it gives my EC2 instance a public IP address, fulfilling all the requirements for Instance Connect to work.

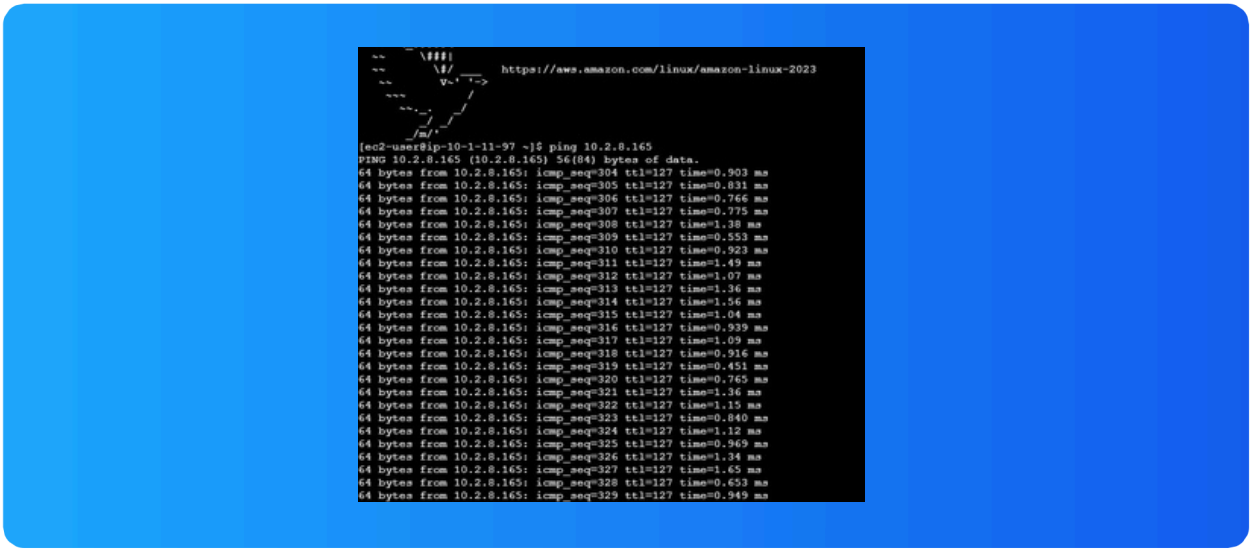
Kanika Mathur
github.com/KanikaGenesis

Troubleshooting ping issues

To test VPC peering, I ran the command ping 10.2.xxx.xxx. i.e. the private IPv4 address of the other EC2 instance in VPC 2.

The ping test would not get ANY replies from the other EC2 instance if the peering connection did not successfully connect out 2 VPCs. Getting ping replies = peering connection was set up properly.

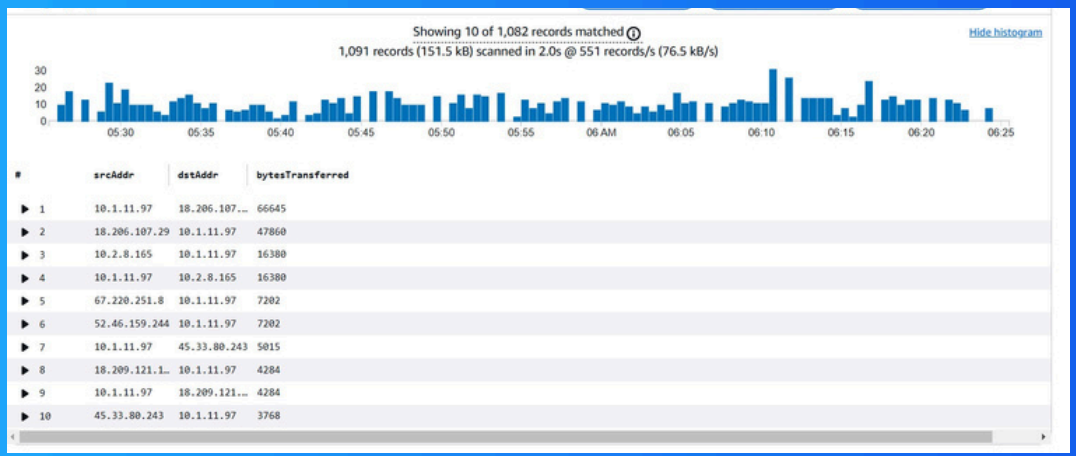
I had to update my second EC2 instance's security group because it was not letting in ICMP traffic - which is the traffic type of a ping message. I added a new rule that allows ICMP traffic coming in from any resource in VPC 2.



NextWork.org

VPC Monitoring with Flow Logs

Kanika Mathur
github.com/KanikaGenesis



#	srcAddr	dstAddr	bytesTransferred
1	10.1.11.97	18.206.107.…	66645
2	18.206.107.29	10.1.11.97	47860
3	10.2.8.165	10.1.11.97	16380
4	10.1.11.97	10.2.8.165	16380
5	67.220.251.8	10.1.11.97	7202
6	52.46.159.244	10.1.11.97	7202
7	10.1.11.97	45.33.80.243	5015
8	18.209.121.1…	10.1.11.97	4284
9	10.1.11.97	18.209.121.…	4284
10	45.33.80.243	10.1.11.97	3768


Kanika Mathur
github.com/KanikaGenesis

In the first part of my project...

Step 1 Set up VPCs

In this step, we will set up two VPCs from scratch in minutes. Networking monitoring can still be done with just a single VPC, but it's great to have the extra challenge and tackle VPC peering in this project, too!

Step 2 Launch EC2 instances

I am launching an EC2 instance in each of the VPCs (VPC 1 AND VPC 2). Doing this is important to set up the remainder of my project, as my EC2 instances would generate traffic that VPC Flow Logs will monitor.

Step 3 Set up Logs

In this step I am setting up VPC flow logs to start monitoring network traffic. I am also setting up a storage space for my flow logs.

Step 4 Set IAM permissions for Logs

In this step, I will give VPC Flow Logs permission to create logs and upload them into my log groups in CloudWatch.

Kanika Mathur
github.com/KanikaGenesis

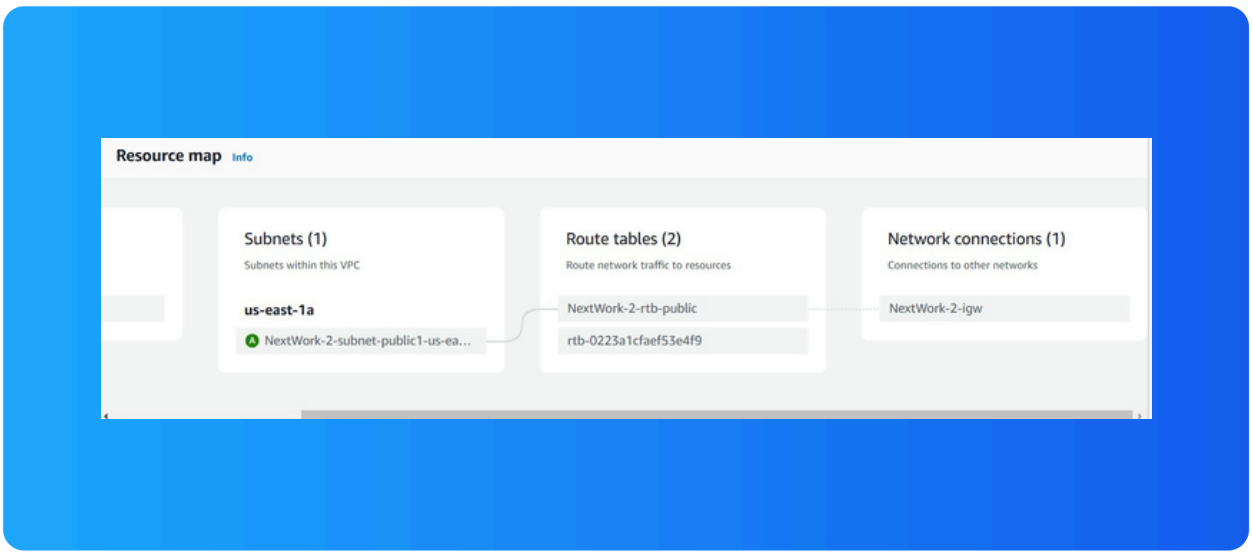
Multi-VPC Architecture

I started my project by launching two VPC - they have unique CIDR blocks and they each have one public subnet.

The CIDR blocks for VPCs 1 and 2 are 10.1.0.0/16 and 10.2.0.0/16 respectively. They have to be unique because once you set up a VPC peering connection, the route table needs unique addresses for correct routing across VPC.

I also launched EC2 instances in each subnet

My EC2 instances security groups allow SSH and ICMP types of traffic. This is because EC2 Instance Connect will need to access my EC2 Instance using SSH-type traffic and because we need to allow ICMP-type traffic for connectivity test later.

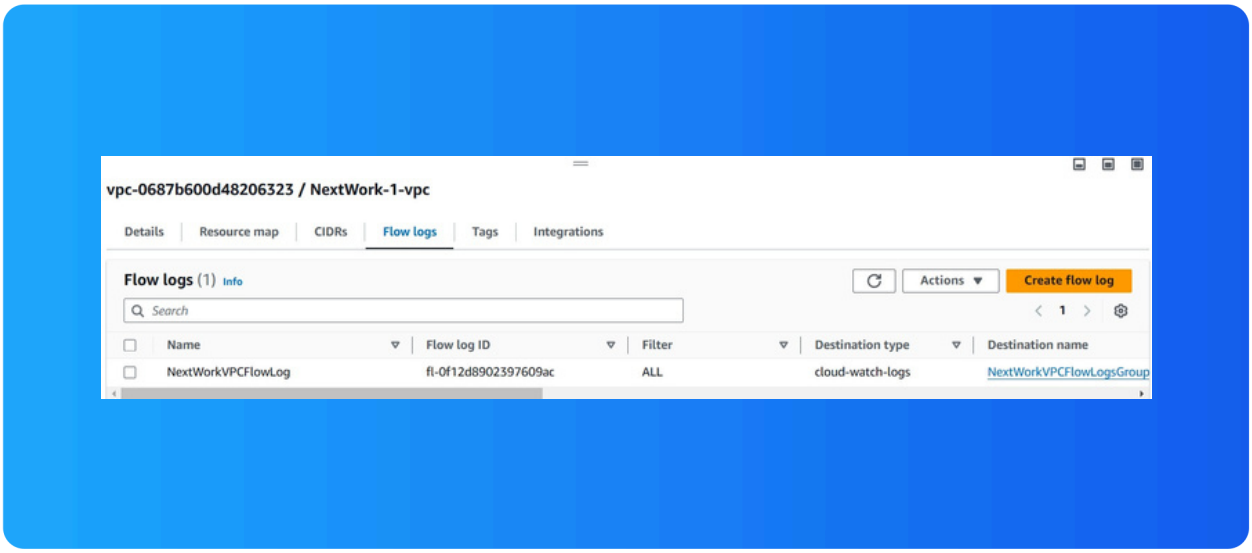


Kanika Mathur
github.com/KanikaGenesis

Logs

Logs are like diary entries of our computer system - they are detailed records of any kind of activity related to the traffic/ resource/ AWS service that the log is tracking.

Log groups are groups of logs, i.e., logs that belong to the project/ application/resource are often in the log group together.



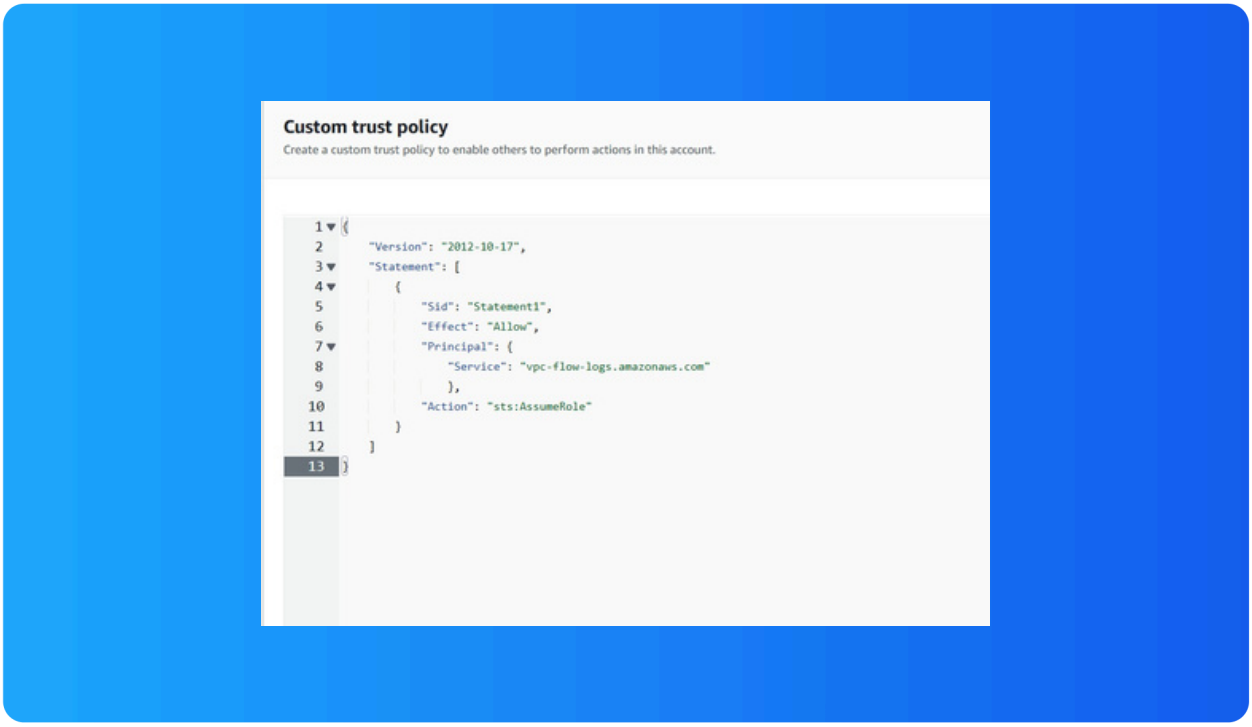

Kanika Mathur
github.com/KanikaGenesis

IAM Policy and Roles

I created an IAM policy so that I can define the rule that allows policy holders e.g. my VPC Flow Log service the ability to create log streams and upload them into CloudWatch.

I also created an IAM role because services like VPC Flow Logs have to be associated with a role instead of JSON! Creating an IAM role will be necessary to give my VPC Flow Logs the access they need to record and upload logs.

A custom trust policy is a specific type of policy used for designing who or what is allowed access to the IAM role.



In the second part of my project...

Step 5 Ping testing and troubleshooting

In this step, we will generate network traffic. This becomes important when we are communicating about cloudworks/ cloud networking/ cloud engineering.

Step 6 Set up a peering connection

In this step, we are setting up a peering connection so that VPCs 1 and 2 can talk directly with each other.

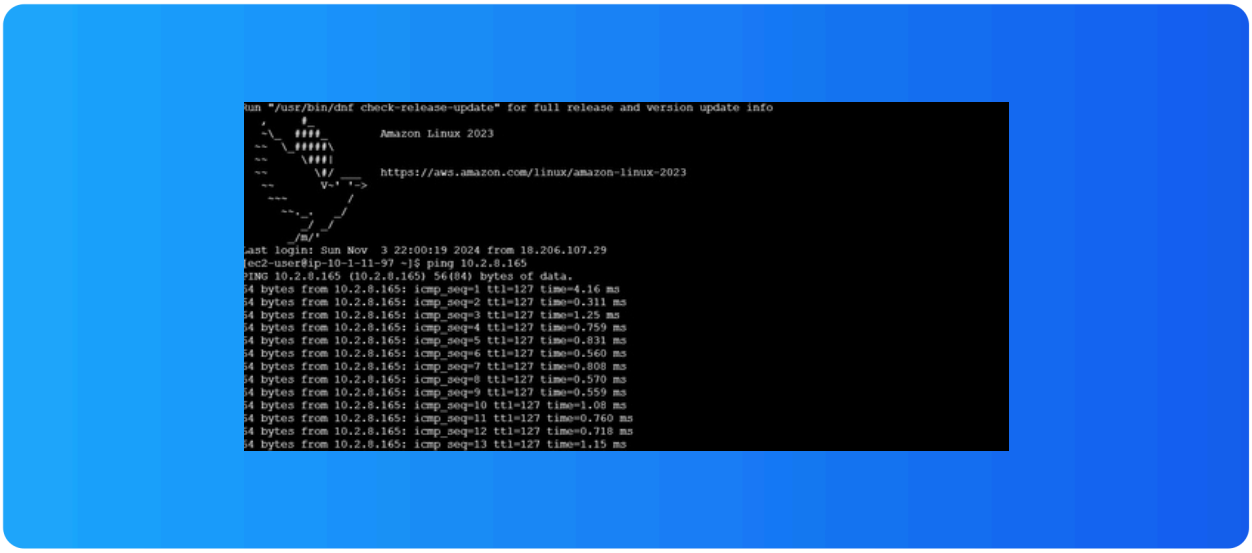
Step 7 Analyze flow logs

In this step, I will track the network data collected on my VPCs and analyze it to extract insights.

Kanika Mathur
github.com/KanikaGenesis

Connectivity troubleshooting

My first ping test between my EC2 instances had no replies, which means ICMP traffic could be blocked by security group/ network ACLs, or maybe the traffic is being routed to the wrong path.



I could receive ping replies if I ran the ping test using the other instance's public IP address, which means my second instance is allowing ICMP traffic.

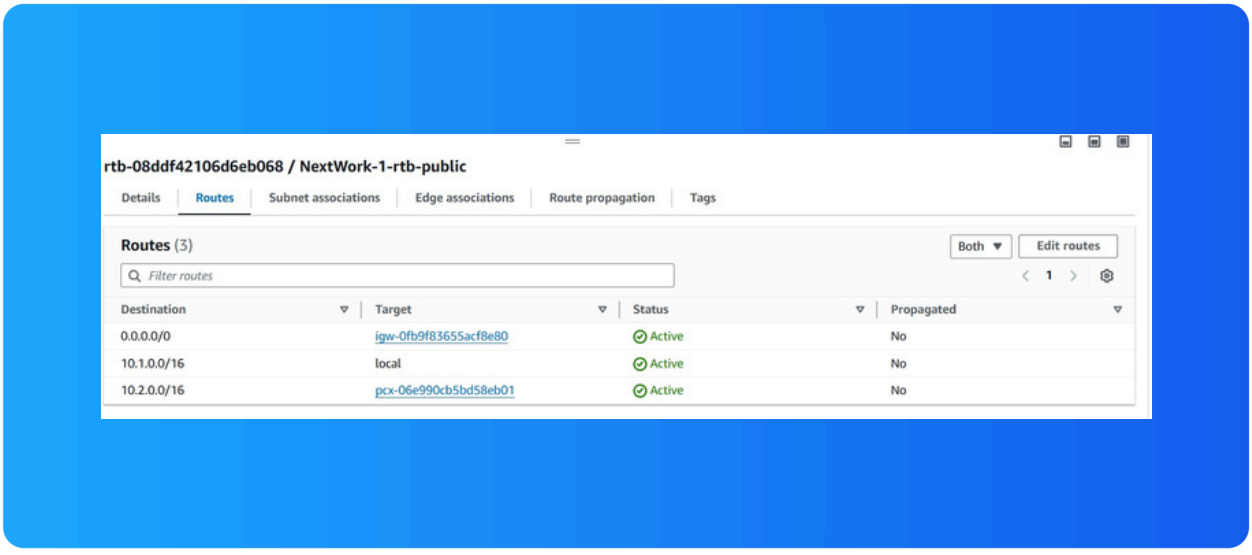
Kanika Mathur
github.com/KanikaGenesis

Connectivity troubleshooting

Looking at VPC 1's route table, I identified that the ping test with Instance 2's private address failed because I do not have a route in my VPC's route table that directs traffic from one VPC to another.

To solve this, I set up a peering connection between my VPCs

I also updated both VPCs' route tables so that traffic from one VPC heading to the other VPCs' private IPv4 can be directed to go through the peering connection instead of the public internet.

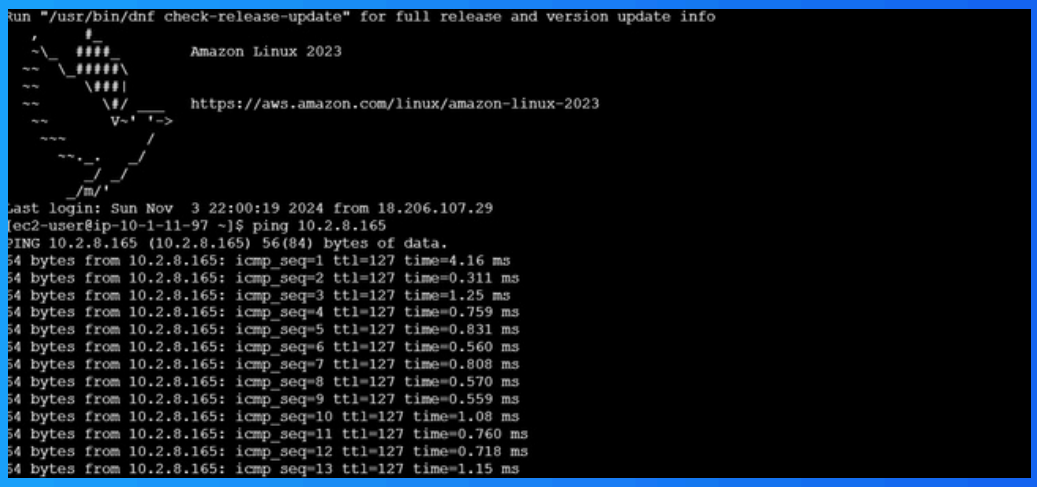


Kanika Mathur
github.com/KanikaGenesis

NextWork.org

Connectivity troubleshooting

I received ping replies from Instance 2's private IP address! This means setting up the peering connection and then the route table to solve the connectivity error of our VPC traffic not being able to navigate from one VPC to another.



Kanika Mathur
github.com/KanikaGenesis

Analyzing flow logs

Flow logs tell us about the source and destination of the network traffic, the amount of data being transferred, and whether the traffic was accepted or rejected.

For example, the flow log I've captured tells us that traffic went from 18.206.107.29 to 10.1.11.97. We can also extract that the traffic was accepted by my VPC's security groups and network ACLs.

Kanika Mathur
github.com/KanikaGenesis

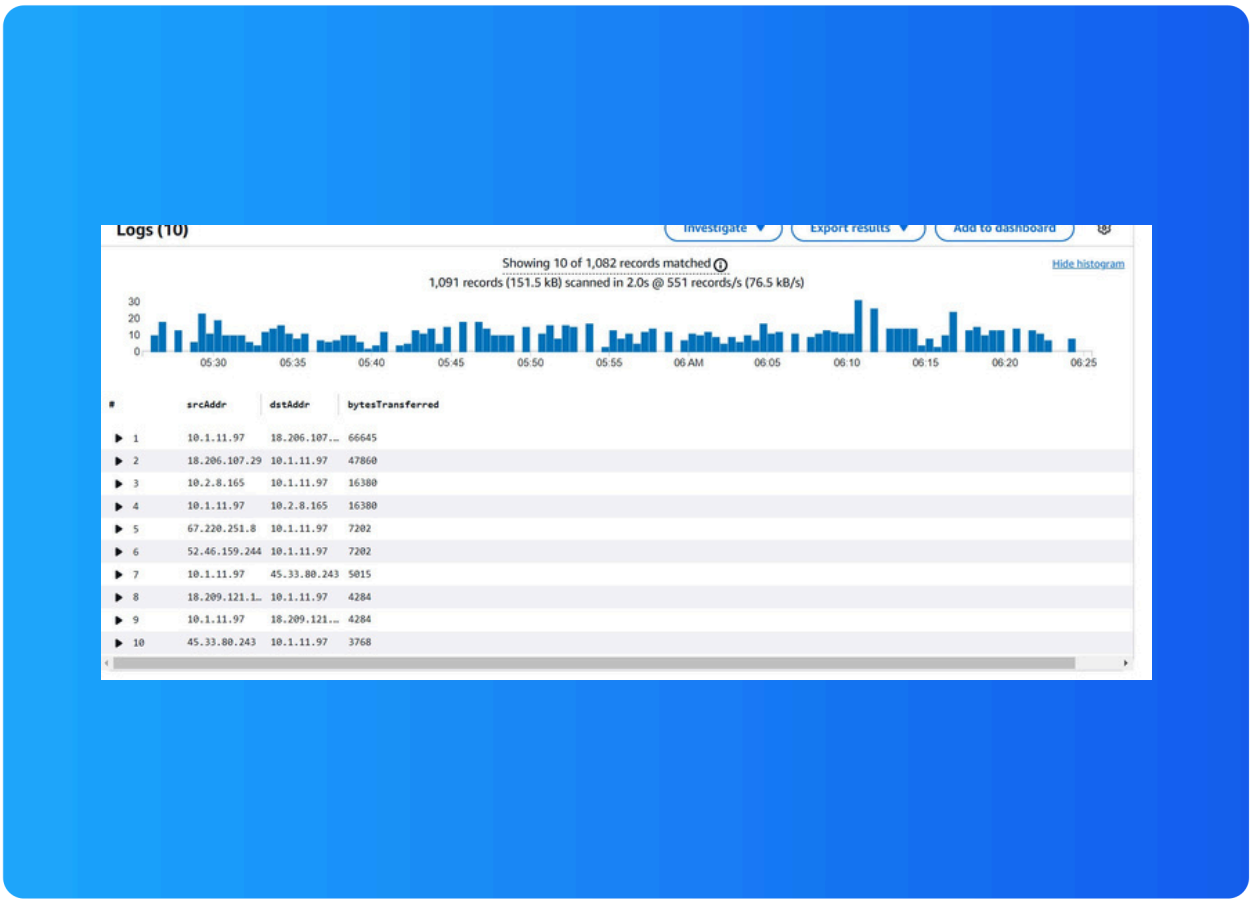
Logs Insights

Logs Insights is a special tool within Amazon CloudWatch that helps us analyze logs and create visual graphs and charts through queries.

I ran the query "Return the top 10 byte transfers by source and destination IP addresses",

```
stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
  sort bytesTransferred desc
  limit 10
```

which analyzes the flow logs collected on EC2 Instance 1 and returns the top 10 pairs of IP addresses based on the amount of data transferred between them.



#	srcAddr	dstAddr	bytesTransferred
1	10.1.11.97	18.206.107.…	66645
2	18.206.107.29	10.1.11.97	47860
3	10.2.8.165	10.1.11.97	16380
4	10.1.11.97	10.2.8.165	16380
5	67.220.251.8	10.1.11.97	7202
6	52.46.159.244	10.1.11.97	7202
7	10.1.11.97	45.33.80.243	5015
8	18.209.121.1…	10.1.11.97	4284
9	10.1.11.97	18.209.121.…	4284
10	45.33.80.243	10.1.11.97	3768

NextWork.org

Access S3 from a VPC

Kanika Mathur
github.com/KanikaGenesis


Kanika Mathur
github.com/KanikaGenesis

In the first part of my project...

Step 1 Architecture set up

In this step, I launch a VPC with a public subnet and an EC2 instance inside that public subnet. I am doing this because the EC2 instance will need a VPC when it launches. Also, I will use that EC2 instance to access the S3 bucket directly from VPC.

Step 2 Connect to my EC2 instance

In this step, I directly access an EC2 instance using EC2 Instance Connect.

Step 3 Set up access keys

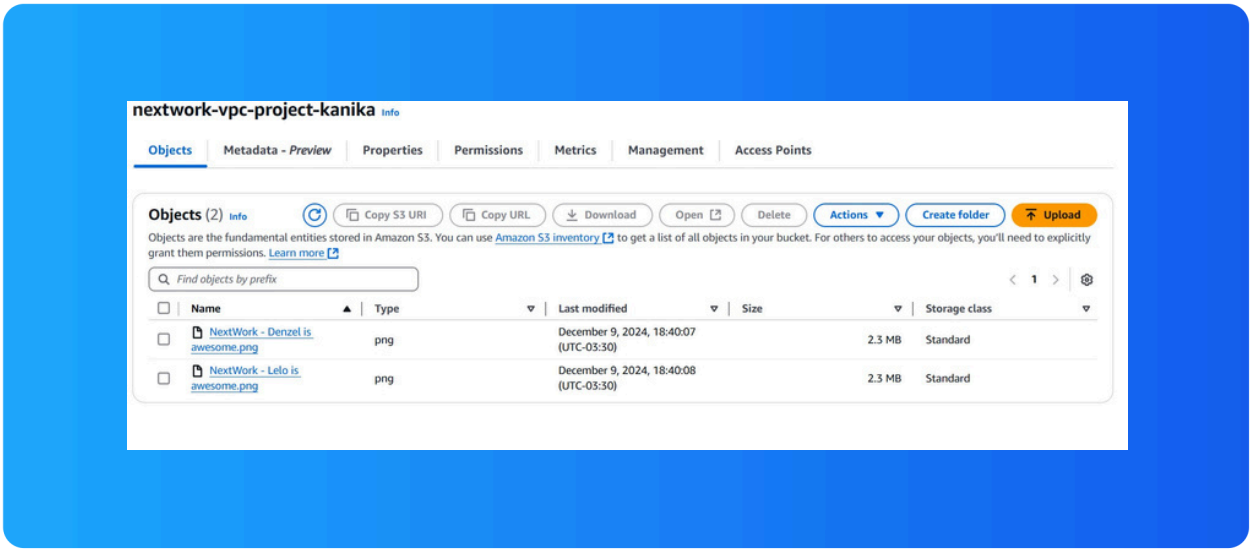
In this step, I create access keys so that my EC2 instance can access my AWS environment, specifically the ability to interact with the S3 bucket.

Kanika Mathur
github.com/KanikaGenesis

Architecture set up

I started my project by launching a VPC with a public subnet and an EC2 instance inside the public subnet.

I also set up an S3 bucket with two files inside.



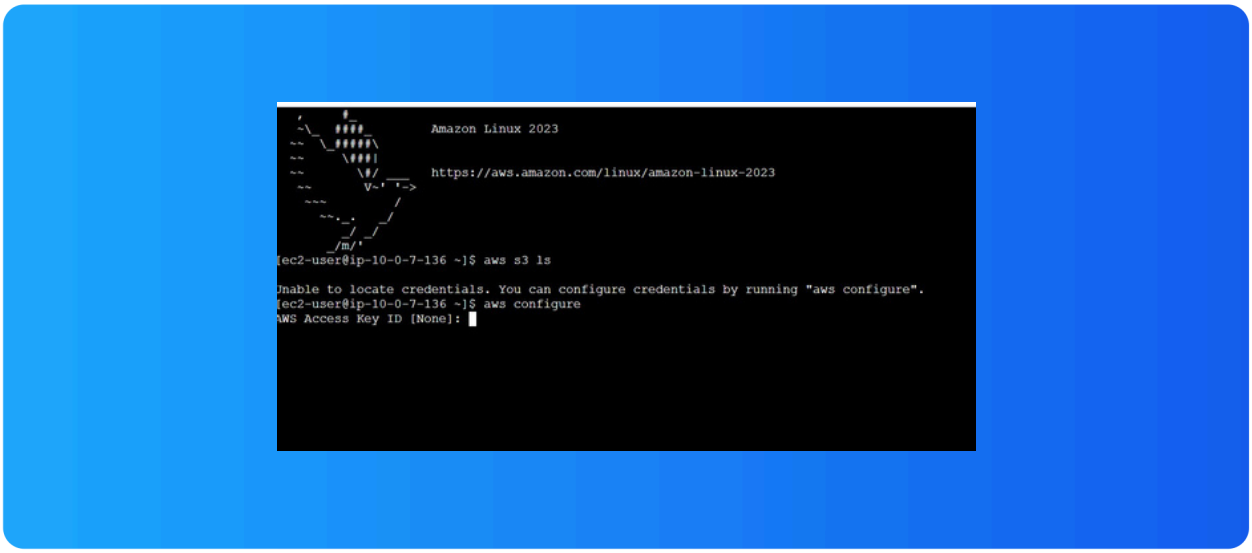
Kanika Mathur
github.com/KanikaGenesis

Running CLI commands

AWS CLI is software we can download to a local computer's terminal so that we can access our AWS account and take different actions without needing to use the AWS Management Console. I have access to AWS CLI because it's pre-installed in EC2 instance.

The first command I ran was "aws s3 ls." This command is used to list all s3 buckets inside the AWS account (that the EC2 instance/ application) has access to.

The second command I ran was "aws configure." This command is used to set up my EC2 instance's credentials in order to access my AWS environment.




Kanika Mathur
github.com/KanikaGenesis

Access keys

Credentials

To set up my EC2 instance to interact with my AWS environment, I configured an Access Key ID, Secret Access Key, default region and default output format.

Access keys are credentials that my EC2 instance and other applications need to access my AWS environment and interact with its resources/ services.

Secret Access keys are like passwords for my access keys/credentials. My EC2 instance/ other applications would need secret access keys as authentication and "log in" to my AWS environment.

Best practice

Although I'm using access keys in this project, a best practice alternative is to use IAM Roles with the permissions attached. This is a more secure way to grant access to an EC2 instance, as it is much easier to attach and detach IAM policies.


Kanika Mathur
github.com/KanikaGenesis

In the second part of my project...

Step 4 Set up an S3 bucket

In this step, I launch an Amazon S3 bucket with two files inside. This S3 bucket will be accessed by my EC2 instance later in this project, so we can test whether my access key has successfully given AWS access to my EC2 instance.

Step 5 Connecting to my S3 bucket

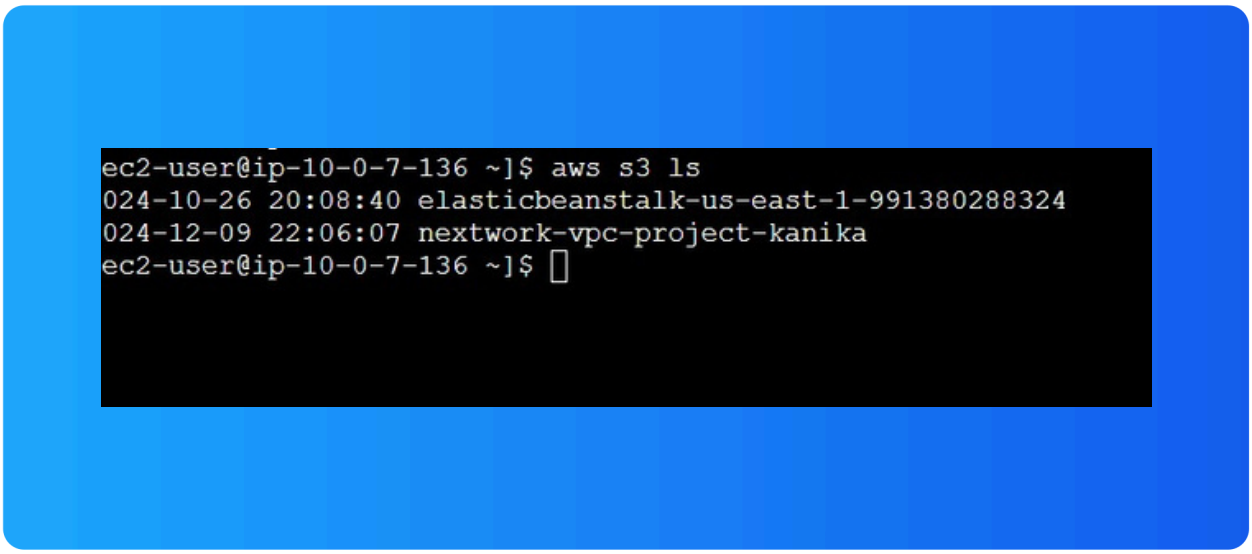
In this step, I am using the AWS CLI command to try controlling/managing my S3 bucket. This means I am interacting with my S3 bucket through my EC2 instance/ VPC instead of the AWS Management Console.


Kanika Mathur
github.com/KanikaGenesis

Connecting to my S3 bucket

The first command I ran was "aws s3 ls." This command is used to list all s3 buckets inside the AWS account (that the EC2 instance/ application) has access to.

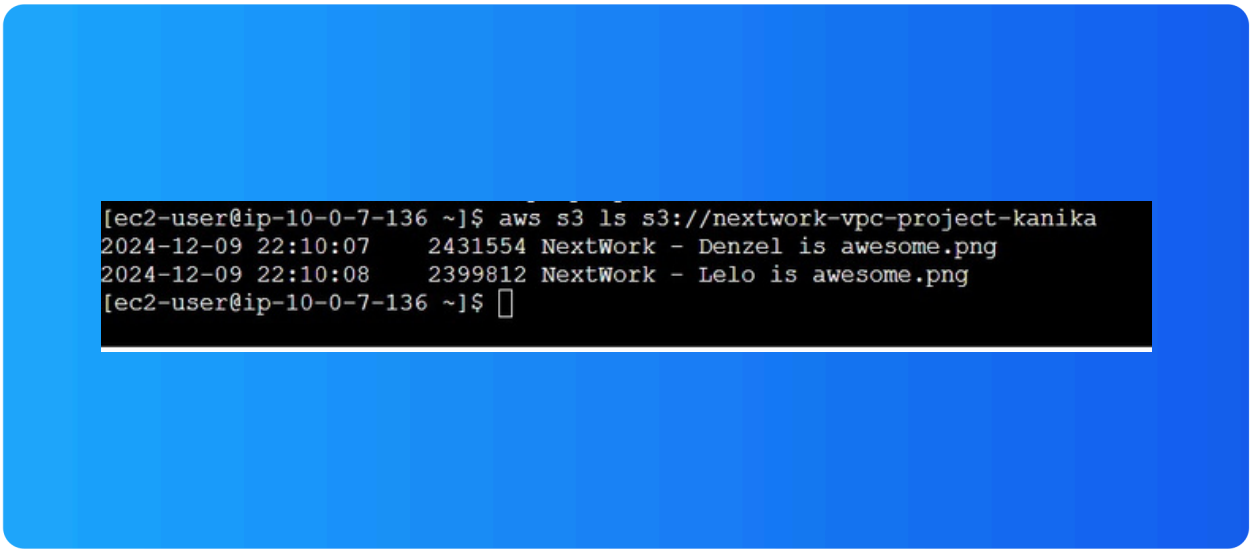
When I ran the command "aws s3 ls" again, the terminal responded with a list of my S3 buckets. This indicated that my access keys work, i.e., my EC2 instance has access to my S3 bucket.



Kanika Mathur
github.com/KanikaGenesis

Connecting to my S3 bucket

Another CLI command I ran was "aws s3 ls s3://nextwork-vpc-project-kanika" which returned a list of objects inside my S3 bucket.



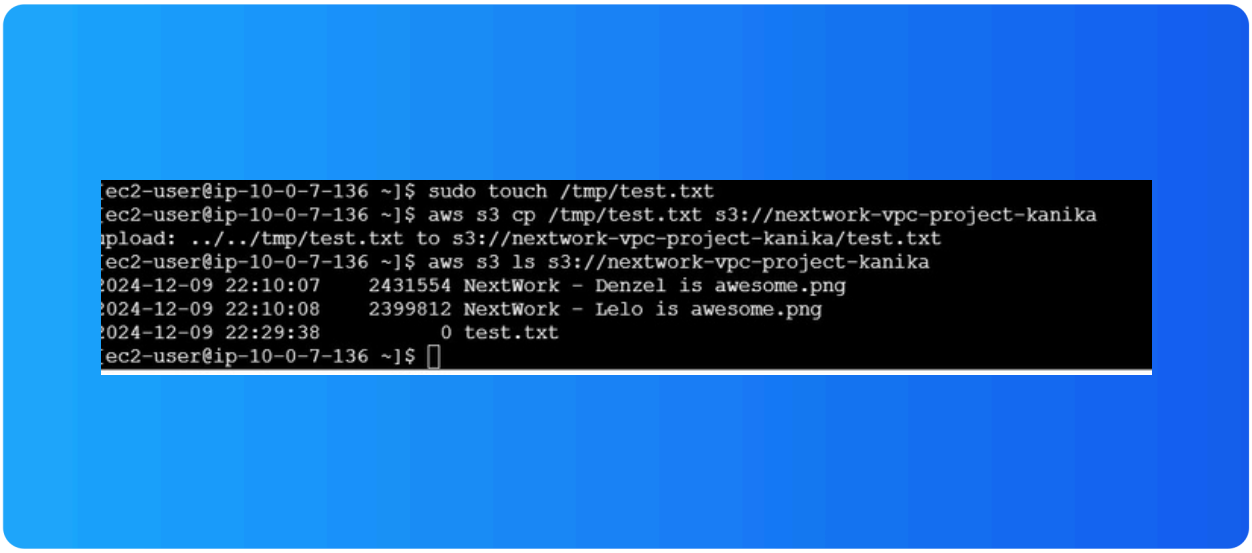

Kanika Mathur
github.com/KanikaGenesis

Uploading objects to S3

To upload a new file to my bucket, I first ran the command "sudo touch /tmp/test.txt." This command creates a blank file called test.txt in my EC2 instance's local directory.

The second command I ran was "aws s3 cp /tmp/test.txt s3://nextwork-vpc-project-yourname." This command will will "copy" i,e. upload the blank file created into my S3 bucket.

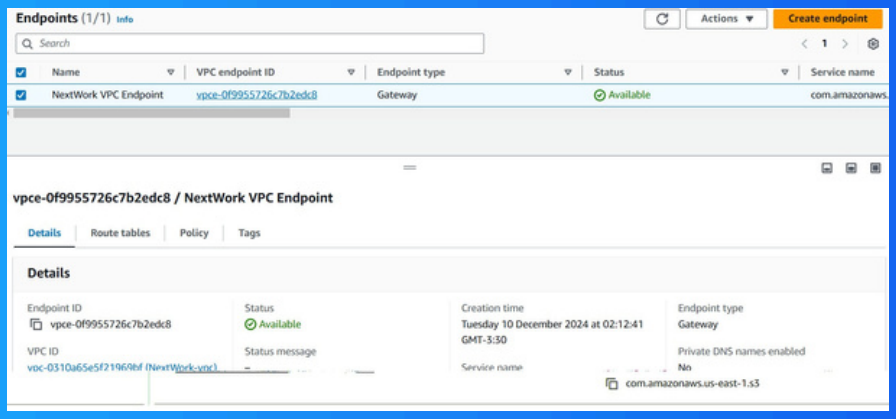
The third command I ran was "aws s3 ls s3://nextwork-vpc-project-yourname," which returned a list of objects in my S3 bucket, including test.txt. This validated that using my EC2 instance through AWS CLI commands can access other AWS services (S3).



NextWork.org

VPC Endpoints

Kanika Mathur
github.com/KanikaGenesis




Kanika Mathur
github.com/KanikaGenesis

In the first part of my project...

Step 1 Architecture set up

In this step, I am setting up the project's foundation by launching a VPC, EC2 instance, and S3 bucket so that I can set up an endpoint architecture and test that architecture in the last step of this project.

Step 2 Connect to EC2 instance

In this step, I connect directly to my EC2 instance using EC2 instance connect. Connecting to the instance will help me access S3 and run commands later in this project.

Step 3 Set up access keys

In this step, I will set up an access key so that my EC2 instance can access my AWS environment. Access keys are almost like login details for instances/applications to interact with our AWS services.

Step 4 Interact with S3 bucket

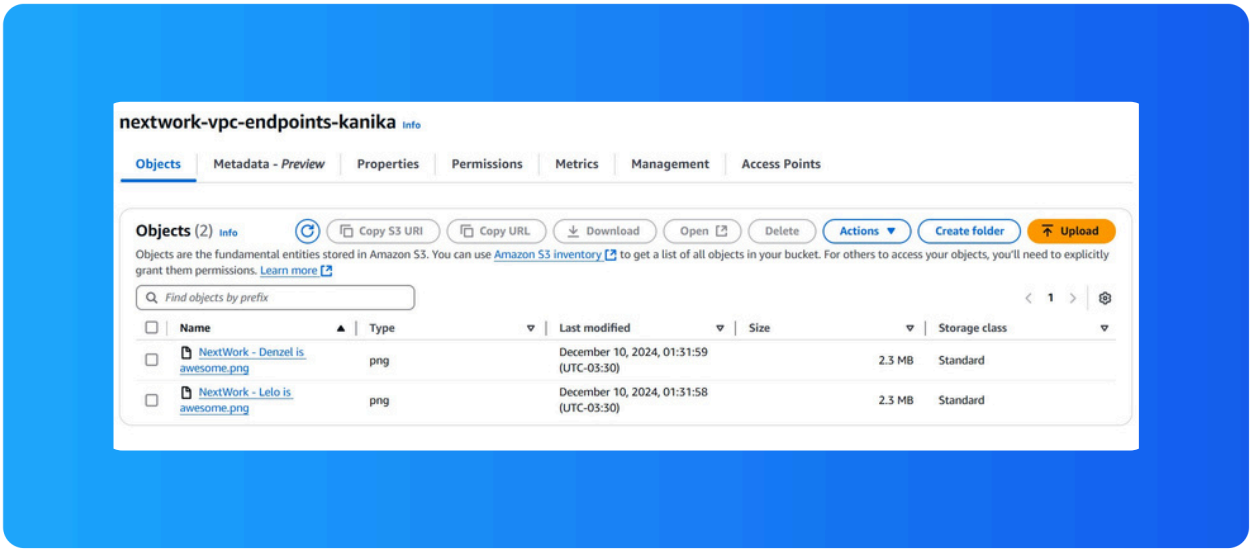
In this step, I am applying my access key credentials to my EC2 instance and then I am using AWS CLI and my EC2 instance to access Amazon S3.

Kanika Mathur
github.com/KanikaGenesis

Architecture set up

I started my project by launching three key services - a VPC, an EC2 instance and a S3 bucket.

I also set up an S3 bucket with two files in my bucket.




Kanika Mathur
github.com/KanikaGenesis

Access keys

Credentials

To set up my EC2 instance to interact with my AWS environment, I configured AWS Access Key ID, AWS Secret Access Key, Default region name and Default output format.

Access keys are credentials that an EC2 instance, other server, or application would need to access my AWS environment, e.g., to create resources, read what's inside my AWS account, etc.

Secret access keys are like passwords in the context of access keys/ credentials for our EC2 instance to get access to our AWS services/ environment.

Best practice

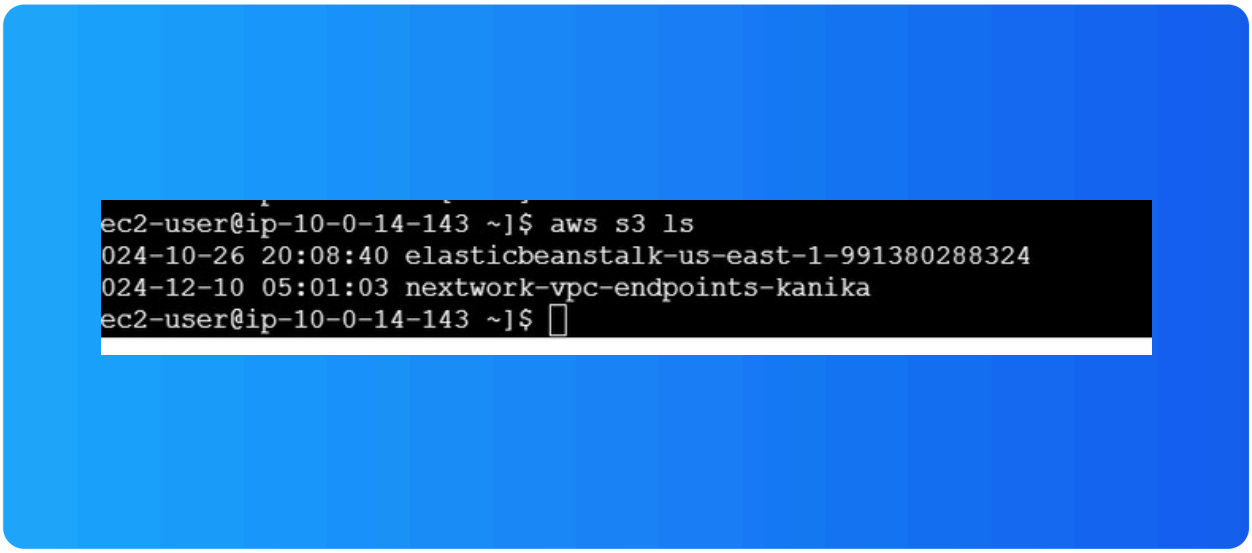
Although I'm using access keys in this project, a best practice alternative is to use IAM admin roles instead! This means the necessary permissions will be attached to an IAM role, which will then be associated with the relevant resources.

Kanika Mathur
github.com/KanikaGenesis

Connecting to my S3 bucket

The command I ran was "aws s3 ls." This command is used to list all the buckets in my AWS account.

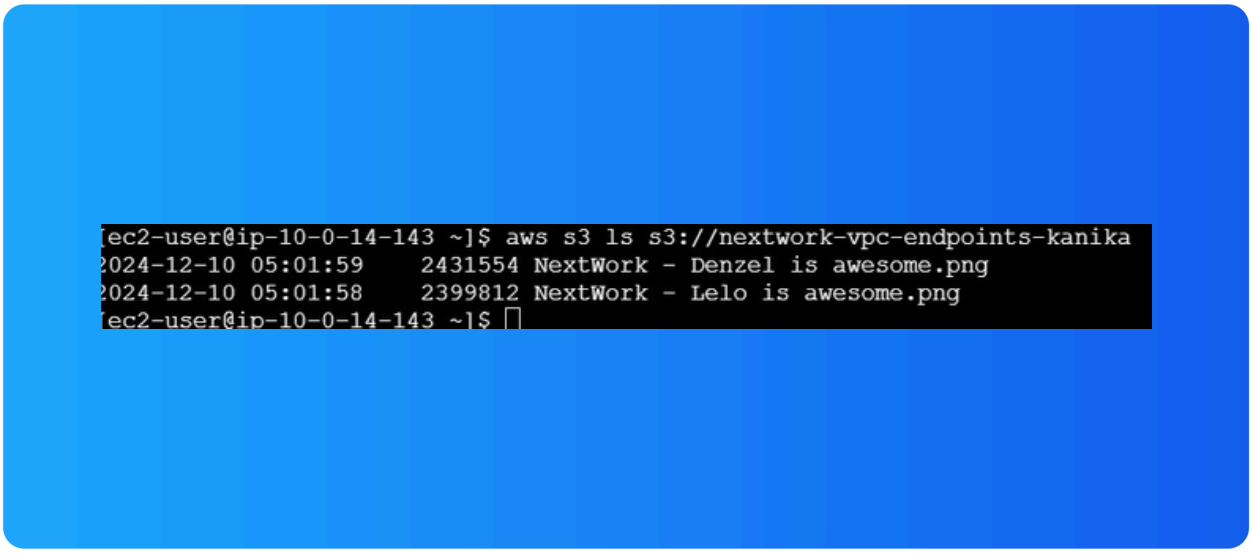
The terminal responded with a list of my account's S3 buckets. This indicated that the access keys I set up correctly gave my EC2 Instance access to my AWS account and environment.



Kanika Mathur
github.com/KanikaGenesis

Connecting to my S3 bucket

I also tested the command "aws s3 ls s3://nextwork-vpc-endpoints-kanika" which returned a list of all of the objects inside my S3 bucket.



Kanika Mathur
github.com/KanikaGenesis

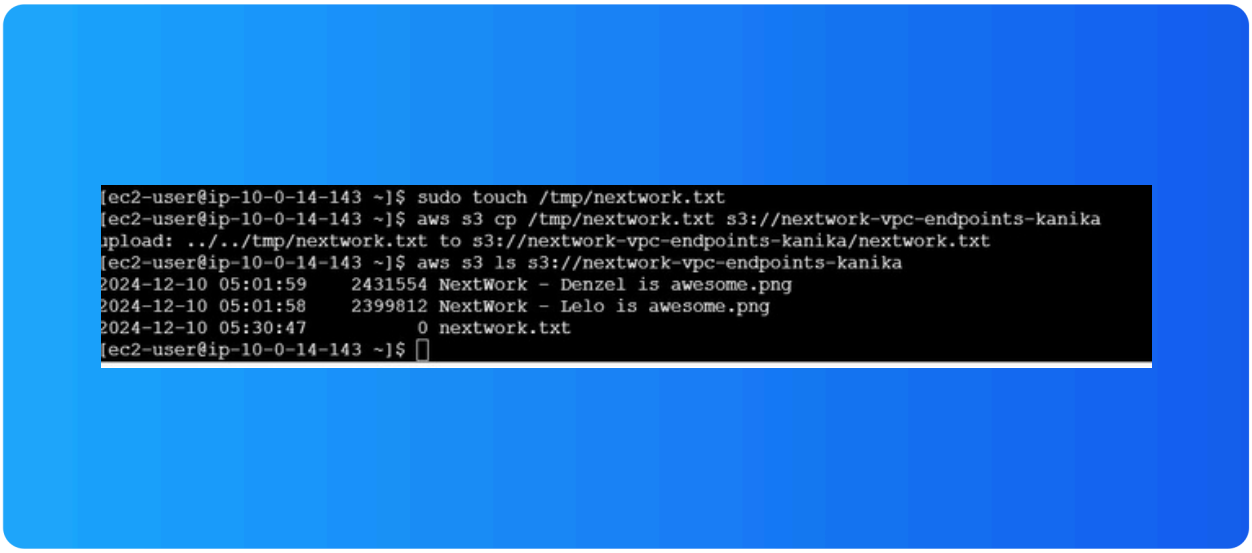
NextWork.org

Uploading objects to S3

To upload a new file to my bucket, I first ran the command "sudo touch/tmp/nextwork.txt." This command creates a blank file called nextwork.txt in my EC2 instance's local directory.

The second command I ran was "aws s3 cp /tmp/nextwork.txt s3://nextwork-vpc-project-kanika." This command will will "copy" i,e. upload the blank file created into my S3 bucket.

The third command I ran was "aws s3 ls s3://nextwork-vpc-project-kanika," which returned a list of objects in my S3 bucket, including test.txt. This validated that using my EC2 instance through AWS CLI commands can access other AWS services (S3)




Kanika Mathur
github.com/KanikaGenesis

In the second part of my project...

Step 5 Set up a Gateway

In this step, I am setting up a VPC endpoint so that communication between my VPC and other services (especially S3) is direct and secure.

Step 6 Bucket policies

In this step, I am testing my endpoint connection by blocking all traffic to my S3 bucket except for traffic coming from my endpoint.

Step 7 Update route tables

In this step, I am testing the endpoint connection between my bucket and the EC2 instance.

Step 8 Validate endpoint conection

In this step, I am going to validate my VPC endpoint setup one more time. I am also going to use endpoint policies to restrict my EC2 instance's access to my AWS environment.

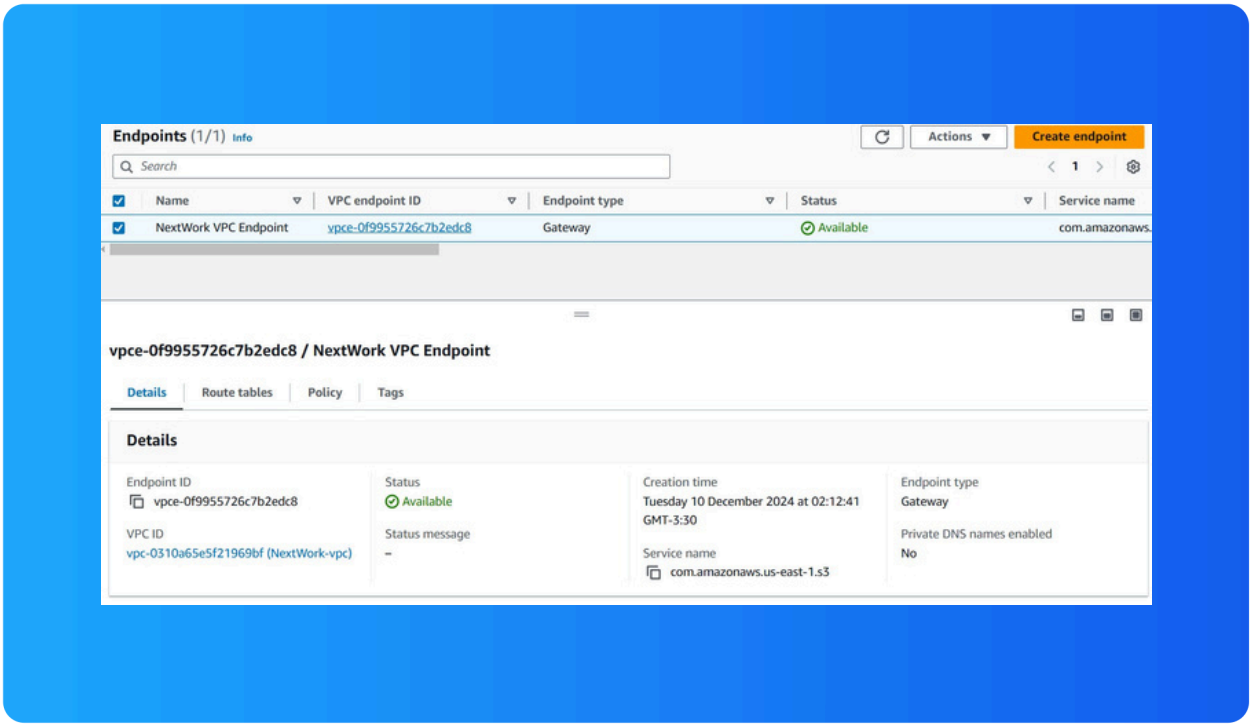
Kanika Mathur
github.com/KanikaGenesis

Setting up a Gateway

I set up an S3 Gateway, a type of endpoint specifically designed for Amazon S3. Gateways work by updating the route table of the associated subnet so that S3-bound traffic goes through the gateway instead of the internet.

What are endpoints?

An endpoint is a VPC component that allows my VPC to have a direct connection to AWS services so that traffic doesn't have to go through the public internet.

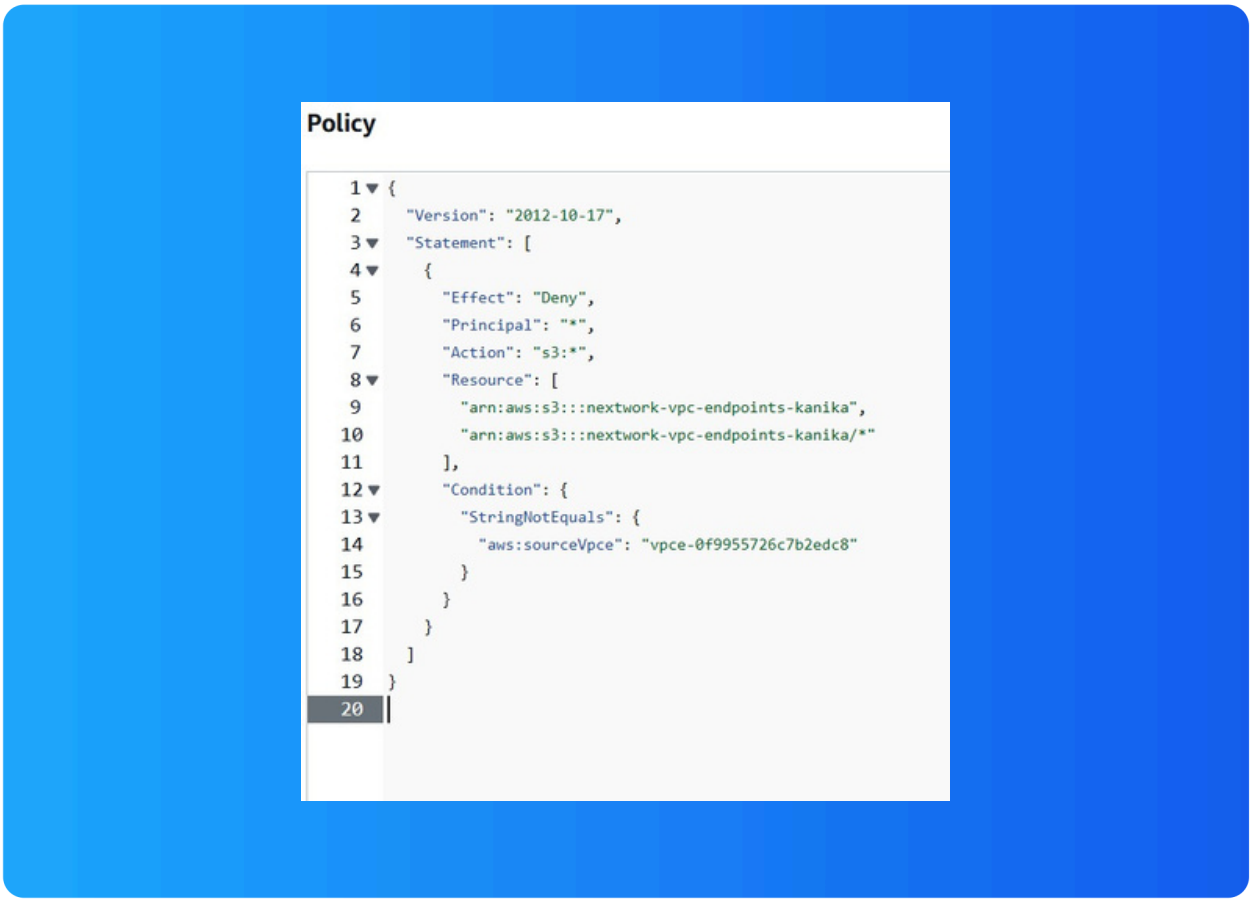


Kanika Mathur
github.com/KanikaGenesis

Bucket policies

A bucket policy is a type of policy that has granular control over who has access to an S3 bucket, and what are the actions that they can perform.

My bucket policy will deny traffic from ALL sources - except for the traffic coming from my VPC endpoint.

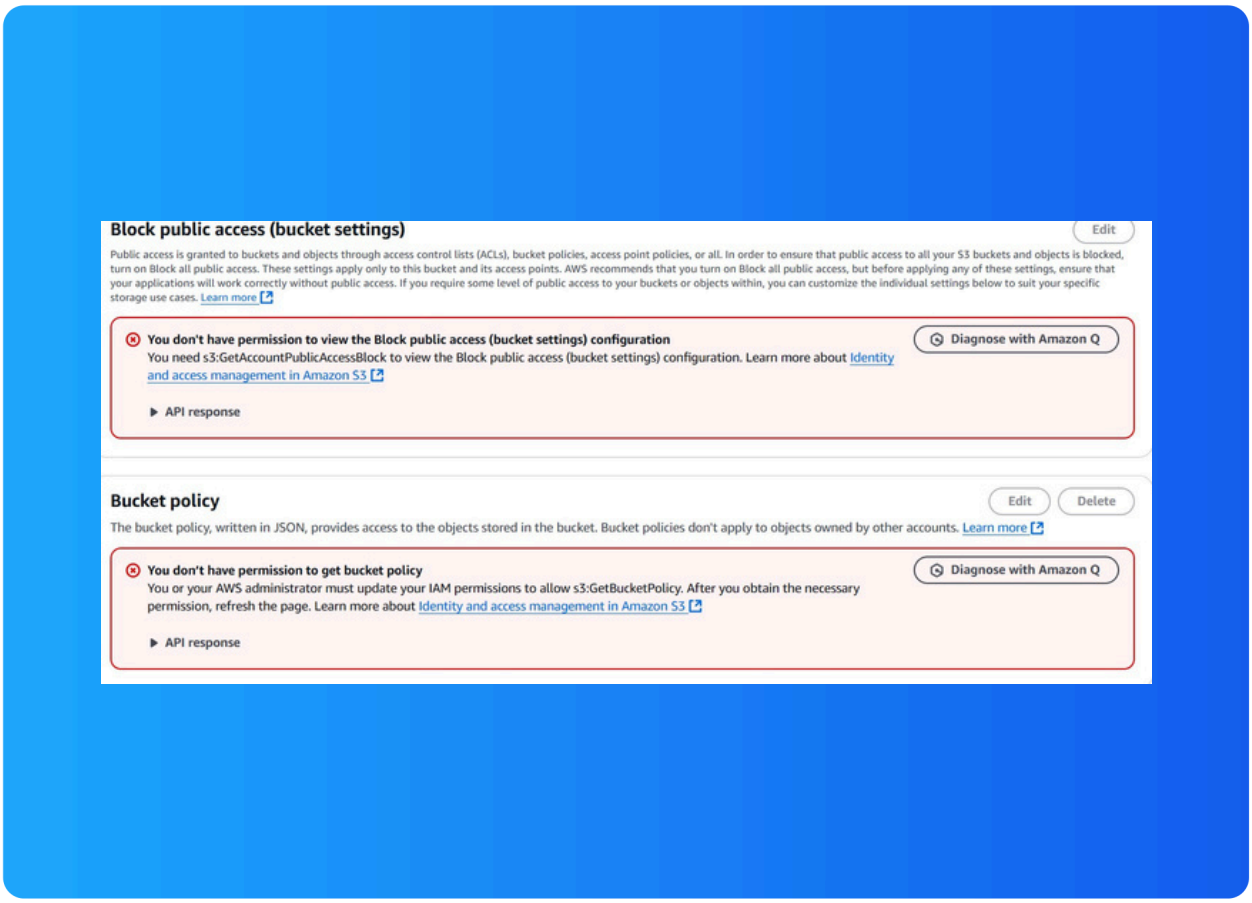


Kanika Mathur
github.com/KanikaGenesis

Bucket policies

Right after saving my bucket policy, my S3 bucket page showed 'denied access' warnings. This was because my bucket policy is denying all traffic that is not coming from my endpoint, i.e. the policy denies traffic from the AWS management console!

I also had to update my route table because my route table, by default, doesn't provide a route for traffic in my public subnet to the VPC endpoint.

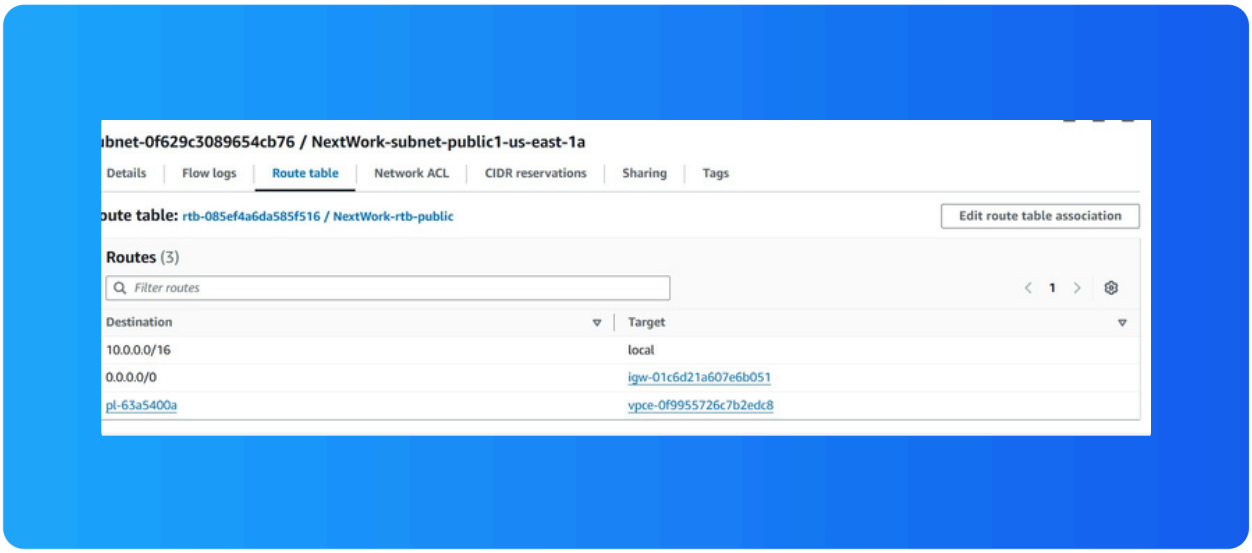


Kanika Mathur
github.com/KanikaGenesis

Route table updates

To update my route table, I visited the Endpoints page in my VPC console, and I modified the route table from there to associate my VPC's public subnet.

After updating my public subnet's route table, my EC2 instance could connect to my S3 bucket. Access was no longer denied!




Kanika Mathur
github.com/KanikaGenesis

Endpoint policies

An endpoint policy is a type of policy designed for specifying the range of resources and actions permitted by the endpoint.

I updated my endpoint's policy by changing the effect from "Allow" to "Deny." I could see the effect of this right away because my EC2 instance was denied access to S3 when I tried to run another S3 command.

Everyone should be in a job they love.

Check out nextwork.org for more projects