

# Aya Mamdouh Ahmed Elhabashy

22101087

VPC Creation:

Name: aiu-learning-platform-vpc

IPv4 CIDR: 10.0.0.0/16

Tier	Subnets (2 AZs)	Purpose
Public	10.0.1.0/24, 10.0.2.0/24	Internet-facing ALB, NAT
Private	10.0.10.0/24, 10.0.11.0/24	Container hosts
Data	10.0.20.0/24, 10.0.21.0/24	RDS
Kafka	10.0.30.0/24, 10.0.31.0/24	Kafka brokers & Zookeeper

## Subnets:

The screenshot shows the AWS VPC Subnets page with 14 subnets listed. The subnets are categorized by tier: Public, Private, Data, and Kafka. Each subnet has a unique ID, state (Available), VPC (vpc-05d98ee95b8642cf7), and various IP ranges and CIDRs. The Kafka tier includes two subnets for Kafka brokers and Zookeeper.

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR	IPv6 CIDR
subnet-031c430a44erd5b8c5	Available	vpc-05d98ee95b8642cf7	Off	172.31.48.0/20	-	-
subnet-077aa94afaca3d52	Available	vpc-05d98ee95b8642cf7	Off	172.31.32.0/20	-	-
subnet-077aa94afaca3d52	Available	vpc-05d98ee95b8642cf7	Off	172.31.64.0/20	-	-
subnet-044527f8a0d81f5b1	Available	vpc-05d98ee95b8642cf7	Off	172.31.80.0/20	-	-
subnet-086610309f94803b84	Available	vpc-05d98ee95b8642cf7	Off	172.31.0.0/20	-	-
subnet-0a0d521f040eae7	Available	vpc-05d98ee95b8642cf7	Off	172.31.16.0/20	-	-
subnet-0eb42bb1ca58fb52	Available	vpc-01b8a082ccf989590   aiu-l...	Off	10.0.1.0/24	-	-
public-subnet-1	Available	vpc-0dd494cc0366b37	Available	10.0.2.0/24	-	-
public-subnet-2	Available	vpc-01b8a082ccf989590   aiu-l...	Off	10.0.10.0/24	-	-
private-subnet-1	Available	vpc-08b31307ed63adde5	Available	10.0.11.0/24	-	-
private-subnet-2	Available	vpc-09a7651b8eab09	Available	10.0.20.0/24	-	-
data-subnet-1	Available	vpc-07a913fbff8082d09	Available	10.0.21.0/24	-	-
data-subnet-2	Available	vpc-09e468a93rfe4fc2	Available	10.0.30.0/24	-	-
kafka-subnet-1	Available	vpc-0987f9a3389df9d13	Available	10.0.31.0/24	-	-
kafka-subnet-2	Available	vpc-0b1e0f184e98fdddad	Available	vpc-01b8a082ccf989590   aiu-l...	-	-

## Internet Gateway:

**Internet gateways (1/2) Info**

Name	Internet gateway ID	State	VPC ID	Owner
igw-02fcfaa750be5f54084	Attached	vpc-05d98ec95b8642cf7	74469944198	
aliu-igw	Attached	vpc-01b8a082ccf989590   aiu-learning-...	74469944198	

**igw-082a6b2aaa6befbbe / aliu-igw**

**Details**

## NAT gateways:

**NAT gateways (1/3) Info**

Name	NAT gateway ID	Connectivity...	State	State message	Availability...	Route table ID	Primary public I...	Primary private I...
nat-gateway-2	nat-0be1e81d5b7287928	Public	Available	-	Zonal	-	18.213.129.6	10.0.2.92
nat-gateway-2	nat-04b95b49a9e1fda28	Public	Deleted	-	Zonal	-	98.95.151.45	10.0.2.8
nat-gateway-1	nat-0d03ab33f3a95e95e	Public	Available	-	Zonal	-	35.153.249.55	10.0.1.97

**nat-04b95b49a9e1fda28 / nat-gateway-2**

**Details**

## Security Groups:

The screenshot shows the AWS VPC Security Groups console. On the left, there's a navigation sidebar with options like VPC dashboard, Virtual private cloud, Security, and PrivateLink and Lattice. The main area is titled "Security Groups (7) Info" and contains a table with columns: Name, Security group ID, Security group name, VPC ID, Description, and Owner. The table lists seven security groups: default (VPC), public-alb, container, rds, default (PostgreSQL), zookeeper, and kafka. Each row includes a "Actions" button and a "Create security group" button at the top right.

Name	Security group ID	Security group name	VPC ID	Description	Owner
-	sg-0098b045e7e948fe0	default	vpc-05d98ec95b8642cf7	default VPC security group	744699944198
-	sg-0caafe530cc2cc06	public-alb	vpc-01b8a082cf989590	Security group for Application Load Bal...	744699944198
-	sg-02547dabf12c2ee93	container	vpc-01b8a082cf989590	Security group for container hosts	744699944198
-	sg-075dd43430977b6	rds	vpc-01b8a082cf989590	PostgreSQL SG	744699944198
-	sg-053f7ef6731dba9df	default	vpc-01b8a082cf989590	default VPC security group	744699944198
-	sg-02dc46b5ae836f42	zookeeper	vpc-01b8a082cf989590	Zookeeper SG	744699944198
-	sg-08df5f561b52e28e9	kafka	vpc-01b8a082cf989590	Kafka brokers SG	744699944198

For Phase2 i worked on **AWS Apache Kafka**

I created the Kafka & Zookeeper

```
ec2-user@ip-10-0-1-61 ~]$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
52f35f51808b confluentinc/cp-kafka:7.4.2 "/etc/confluent/docker..." 12 minutes ago Up 12 minutes 0.0.0.0:9092->9092/tcp, :::9092->9092/tcp kafka
b024e44c1964 zookeeper:3.6.3 "/docker-entrypoint..." 12 minutes ago Up 12 minutes 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, :::2181->2181/tcp, 8080/tcp zookeeper
[ec2-user@ip-10-0-1-61 ~]$
```

Using this:

For kafka :

```
sudo docker run -d --name kafka \
--network kafka-net \
-e KAFKA_BROKER_ID=1 \
-e KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181 \
-e KAFKA_LISTENERS=PLAINTEXT://0.0.0.0:9092 \
-e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://10.0.1.61:9092 \
-e KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1 \
-p 9092:9092 \
confluentinc/cp-kafka:7.4.2
```

For zookeeper:

```
sudo docker run -d --name zookeeper \
--network kafka-net \
-p 2181:2181 \
```

```
-v /home/ec2-user/zookeeper-data:/data \
-e ZOO_MY_ID=1 \
-e ZOO_PORT=2181 \
-e ZOO_TICK_TIME=2000 \
-e ZOO_INIT_LIMIT=5 \
-e ZOO_SYNC_LIMIT=2 \
zookeeper:3.6.3
```

And for creating the topics:

```
TOPICS=(  
document.uploaded  
document.processed  
notes.generated  
quiz.requested  
quiz.generated  
audio.transcription.requested  
audio.transcription.completed  
audio.generation.requested  
audio.generation.completed  
chat.message  
)
```

```
for topic in "${TOPICS[@]}"; do
  sudo docker exec -it kafka /usr/bin/kafka-topics \
    --create \
    --bootstrap-server 10.0.1.61:9092 \
    --replication-factor 1 \
    --partitions 3 \
    --topic "$topic"
done
```

I did it on kafka broker1 but suddenly kafka & zookepeer stopped working so i used kafka broker 2 and started it all again

I created the producer & consumer using this:

And finally the producer sends a message to the consumer

```
# Produce a message
sudo docker exec -it kafka /usr/bin/kafka-console-producer \
--bootstrap-server 10.0.1.61:9092 \
--topic document.uploaded
>Test document upload
# Press Enter to send
```

```
# Consume the message
sudo docker exec -it kafka /usr/bin/kafka-console-consumer \
--bootstrap-server 10.0.1.61:9092 \
--topic document.uploaded \
--from-beginning
```

The screenshot shows a terminal window within the AWS CloudShell interface. The user has run a command to consume messages from a Kafka topic named 'document.uploaded'. The output shows several messages being printed to the screen, including 'Hello', 'Hello aiu learning', and 'Hello aiu learning'. At the bottom of the terminal, it displays the instance ID 'i-0d972bac211e2a677 (Kafka-broker-2)' and its public and private IP addresses.

```
--bootstrap-server 10.0.1.61:9092 \
--topic document.uploaded
# Test document upload
# Press Enter to send
> Hello
>>>bash: document: command not found
+ [ec2-user@ip-10-0-1-61 ~]$ [ec2-user@ip-10-0-1-61 ~]$ ^C[ec2-user@ip-10-0-1-61 ~]$ [ec2-user@ip-10-0-1-61 ~]$ # Consume the message
sudo docker exec -it kafka /usr/bin/kafka-console-consumer \
--bootstrap-server 10.0.1.61:9092 \
--topic document.uploaded \
--from-beginning

Hello
Hello
Hello aiu learning
Hello aiu learning

i-0d972bac211e2a677 (Kafka-broker-2)
PublicIPs: 44.203.52.168 PrivateIPs: 10.0.1.61

CloudShell Feedback Console Mobile App
```

I created the infrastructure as code “IAC”

**WHY?**

## Infrastructure as Code Implementation

In this phase, AWS CloudFormation was used to implement the infrastructure using Infrastructure as Code (IaC). This approach allows the entire AWS environment to be defined declaratively in a YAML template, ensuring consistency, repeatability, and automation across deployments.

The infrastructure was designed in a modular manner, starting with a Virtual Private Cloud (VPC) that serves as an isolated network boundary for all application resources. Public and private subnets were created across multiple Availability Zones to support high availability and fault tolerance. Public subnets host internet-facing components

such as the Application Load Balancer, while private subnets are reserved for backend services like ECS tasks and databases, improving security.

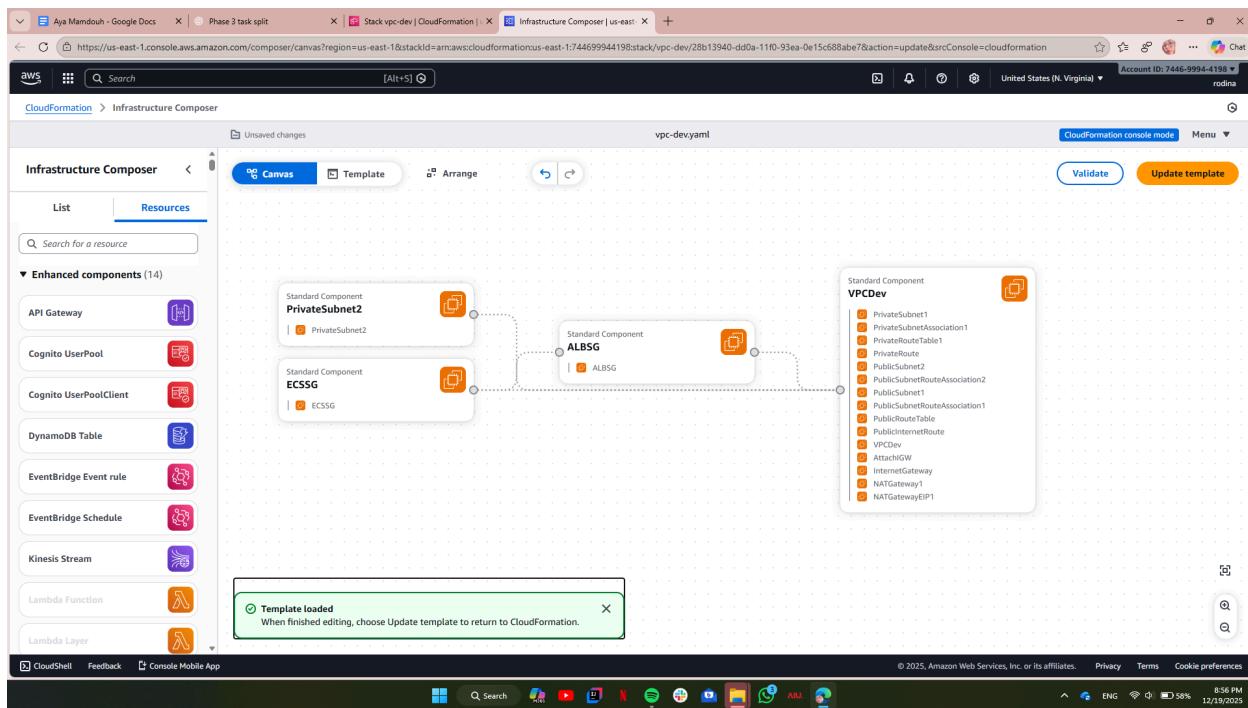
To enable outbound internet access for private resources without exposing them directly, a NAT Gateway was deployed in the public subnet and connected to private route tables. This design follows AWS best practices for secure networking.

Environment separation was achieved using CloudFormation parameters, allowing the same template to be reused for development, staging, and production environments. Resource naming and tagging dynamically adapt based on the selected environment, reducing duplication and simplifying management.

Security was enforced through the use of dedicated security groups. The Application Load Balancer security group allows HTTP and HTTPS traffic from the internet, while the ECS security group only permits traffic originating from the load balancer, implementing the principle of least privilege.

Finally, essential infrastructure identifiers such as VPC ID, subnet IDs, and security group IDs were exposed using CloudFormation outputs. These outputs enable seamless integration with CI/CD pipelines, allowing automated deployments to reference the infrastructure reliably.

## That was the canvas:



It said completed Elhamdullah

**Stacks (1)**

**vpc-dev**

**Overview**

**Status**: UPDATE\_COMPLETE

**Description**: Modular VPC infrastructure with public and private subnets, NAT gateways, security groups, and environment separation for dev, staging, and prod.

**Latest operations**

Operation 1

## And that the resources:

Logical ID	Physical ID	Type	Status	Module
NATGateway2	nat-0298bd8dce67e6137	AWS::EC2::NatGateway	CREATE_COMPLETE	-
NATGatewayEIP1	98.95.91.7	AWS::EC2::EIP	CREATE_COMPLETE	-
NATGatewayEIP2	98.86.206.231	AWS::EC2::EIP	CREATE_COMPLETE	-
PrivateRoute	rtb-0c1637e017d1705ed 0.0.0.0/0	AWS::EC2::Route	CREATE_COMPLETE	-
PrivateRouteTable1	rtb-0c1637e017d1705ed	AWS::EC2::RouteTable	CREATE_COMPLETE	-
PrivateSubnet1	subnet-04629f544c4ea1fb	AWS::EC2::Subnet	CREATE_COMPLETE	-
PrivateSubnet2	subnet-0da4595a92d029c42	AWS::EC2::Subnet	CREATE_COMPLETE	-
PrivateSubnetAssociation1	rtbassoc-021f3559822d7c47	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE	-
PublicInternetRoute	rtb-0bcf3cc95e223fdc 0.0.0.0/0	AWS::EC2::Route	CREATE_COMPLETE	-
PublicRouteTable	rtb-0bcf3cc95e223fdc	AWS::EC2::RouteTable	UPDATE_COMPLETE	-
PublicSubnet1	subnet-0c64d4619676bacd	AWS::EC2::Subnet	UPDATE_COMPLETE	-
PublicSubnet2	subnet-0846aecd17411653c	AWS::EC2::Subnet	UPDATE_COMPLETE	-
PublicSubnetRouteAssociation1	rtbassoc-0184625846c3b1473	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE	-
PublicSubnetRouteAssociation2	rtbassoc-08316c4d97e684d59	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE	-
VPCDev	vpc-0aab577e051a51953	AWS::EC2::VPC	CREATE_COMPLETE	-