TechHealth AWS Migration Report

TechHealth Inc., a healthcare technology company that built their AWS infrastructure manually through the AWS Console 5 years ago -- has a patient portal web application that has now been modernized and migrated to Infrastructure as Code. This report covers the migration process including, the new infrastructure diagram, IaC snippets, security configurations, and proof of appropriately configured connections working properly in the AWS console.

Infrastructure Diagram

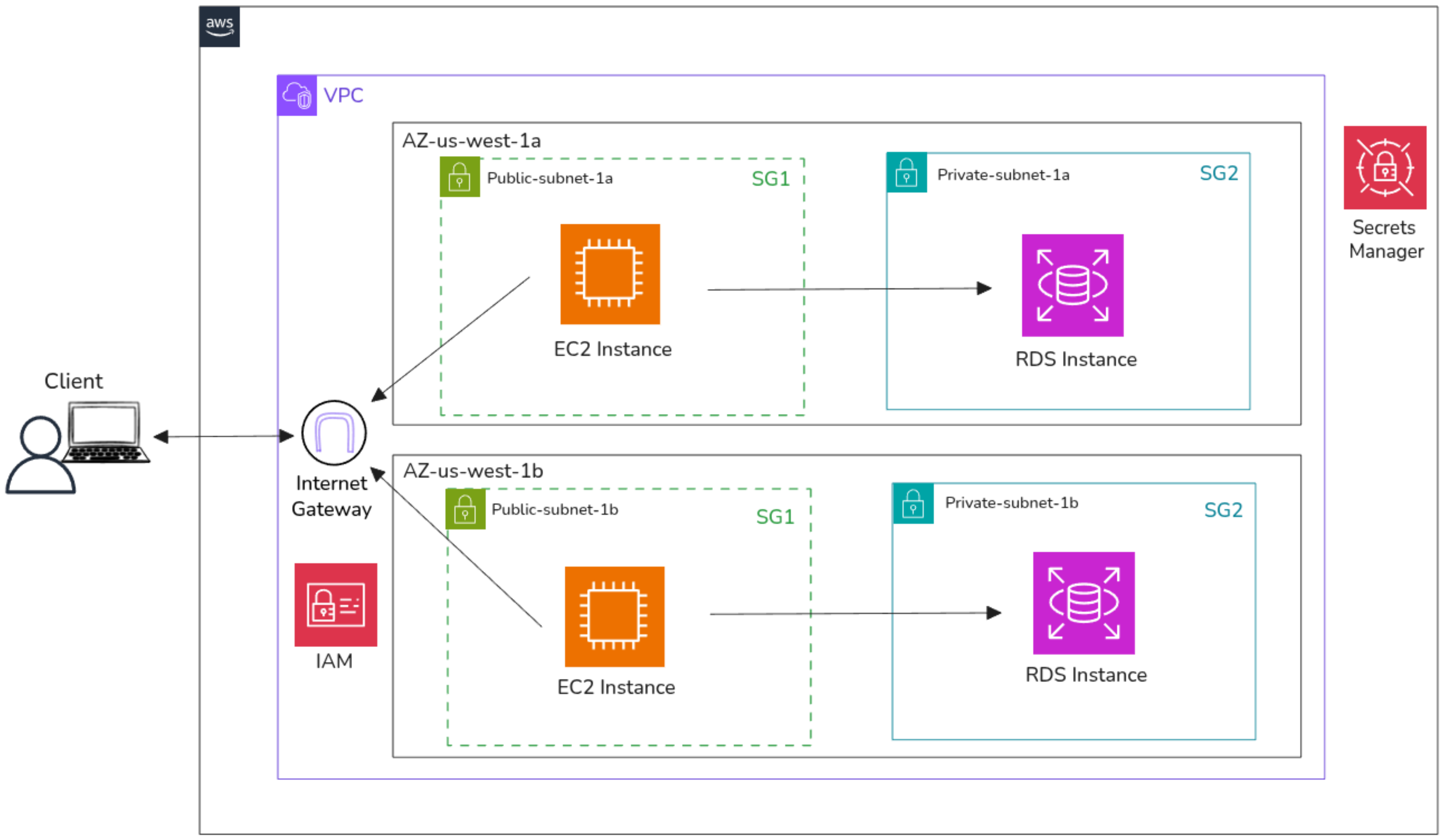


Diagram Description:

VPC Network Foundation, our secure facility perimeter. It's built with AWS CDK using TypeScript, providing a clear separation of environments. The VPC handles**:**

- Public and private subnet segregation
- Multi-Availability Zone redundancy
- Traffic flow control
- Network isolation for sensitive data

- Secure communication pathways

EC2 Application Layer, this acts as our application server that:

- Hosts the patient portal web application
- Processes user requests
- Connects securely to database resources
- Runs in public subnets with controlled access
- Implements defense-in-depth security measures

RDS Database Layer, our secure data vault that:

- Stores sensitive patient information
- Operates exclusively in private subnets
- Provides data persistence and reliability
- Ensures consistent performance
- Implements encryption and access controls

Security Groups Control, our intelligent security system that:

- Implements least-privilege access controls
- Protects EC2 instances from unauthorized access
- Shields database resources from direct public exposure
- Permits only necessary communication paths
- Creates a defense-in-depth strategy

IAM Authorization Layer, our comprehensive permission system that:

- Maintains role-based access control
- Ensures principle of least privilege
- Manages service-to-service authorizations
- Controls resource creation and modification
- Documents access patterns through code

Traffic Flow:

1. *Client* app (in browser) issues HTTP request to web server.

2. The *internet gateway* enables public traffic to flow through a defined network path within established route tables.

3. Public traffic stops at the *EC2 security group*.

4. A secure connection is made from the *EC2 instance* to the RDS.

5. The *RDS,* deployed in a private isolated subnet, allows traffic inbound from the *EC2 security group* ONLY.

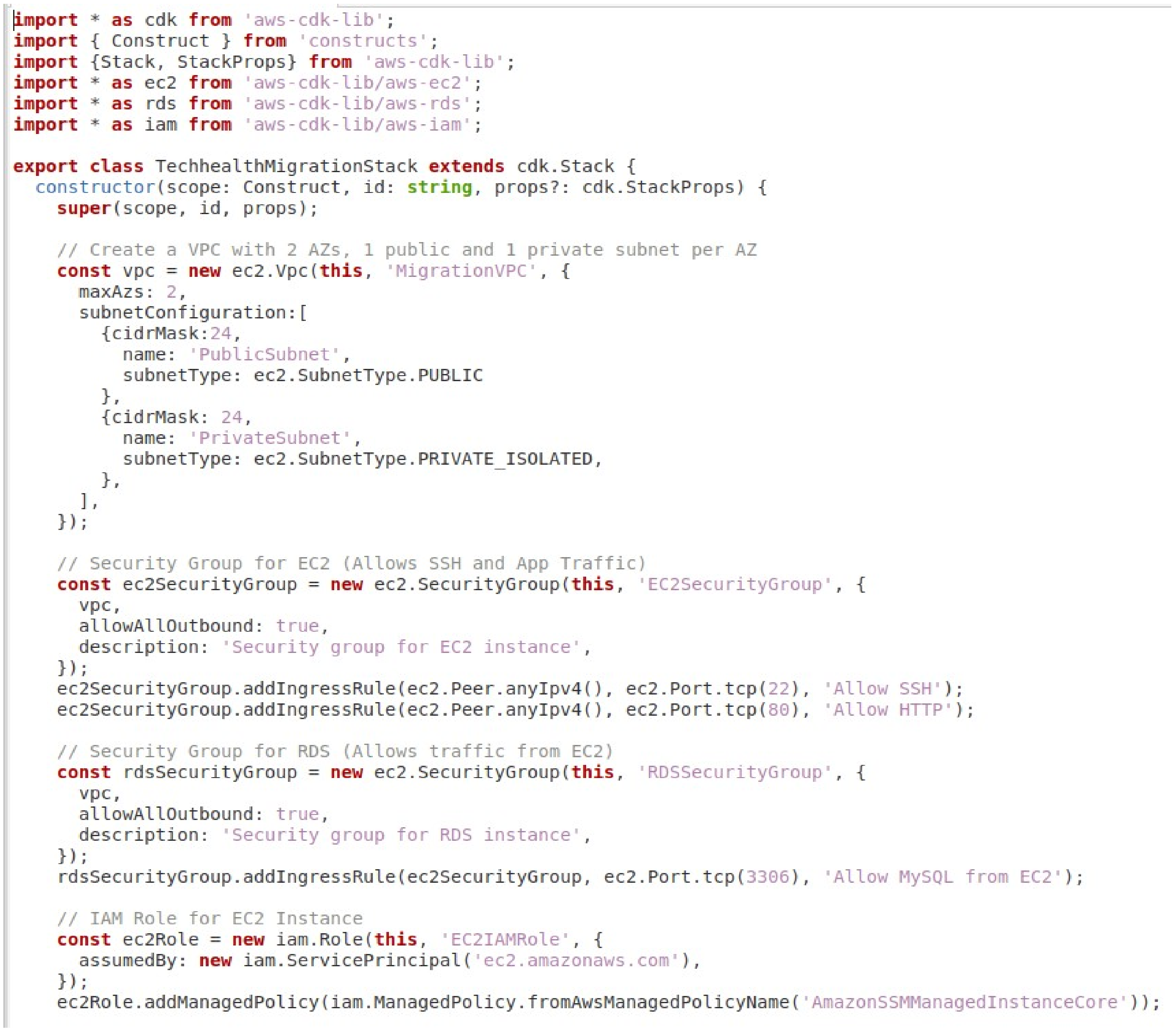
6. This connection is authenticated through use of the AWS *Secrets Manager.*

7. This processed is mirrored in a separate AZ for fault tolerance and high availability.

Cost Saving Measures:

- **IaC adoption –** reduces downtime and misconfigurations, speeds up deployment, and is reusable.
- **Optimized VPC design –** properly configured subnets ensure only the necessary components are publicly accessible which reduces risk and potential data breach costs. Using a multi-AZ setup improves reliability and prevents costly downtime.
- **RDS –** is a managed service which offloads backups, patching, and replication – reducing admin overhead.
- **Security Hardening –** least-privilege IAM policies and tight security group rules reduce risk of breach which helps to avoid reputational and or regulatory penalties. Controlled access to the EC2 and RDS minimizes lateral movement in case of compromise.
- **Proof of correctly working configurations –** in the console reduces rework and manual testing cycles.

aws-cdk TypeScript to implement the VPC, security groups and IAM config depicted above:



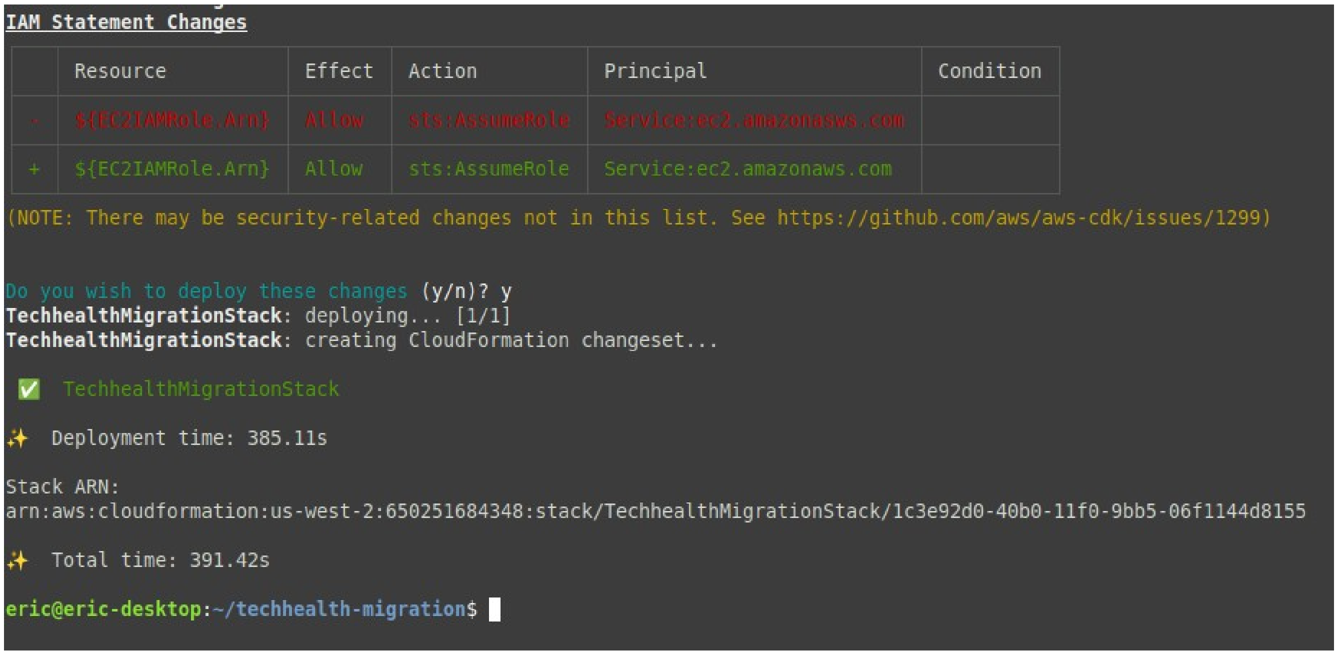
```typescript
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import {Stack, StackProps} from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as rds from 'aws-cdk-lib/aws-rds';
import * as iam from 'aws-cdk-lib/aws-iam';

export class TechhealthMigrationStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Create a VPC with 2 AZs, 1 public and 1 private subnet per AZ
    const vpc = new ec2.Vpc(this, 'MigrationVPC', {
      maxAzs: 2,
      subnetConfiguration:[
        {cidrMask:24,
          name: 'PublicSubnet',
          subnetType: ec2.SubnetType.PUBLIC
        },
        {cidrMask: 24,
          name: 'PrivateSubnet',
          subnetType: ec2.SubnetType.PRIVATE_ISOLATED,
        },
      ],
    });

    // Security Group for EC2 (Allows SSH and App Traffic)
    const ec2SecurityGroup = new ec2.SecurityGroup(this, 'EC2SecurityGroup', {
      vpc,
      allowAllOutbound: true,
      description: 'Security group for EC2 instance',
    });
    ec2SecurityGroup.addIngressRule(ec2.Peer.anyIpv4(), ec2.Port.tcp(22), 'Allow SSH');
    ec2SecurityGroup.addIngressRule(ec2.Peer.anyIpv4(), ec2.Port.tcp(80), 'Allow HTTP');

    // Security Group for RDS (Allows traffic from EC2)
    const rdsSecurityGroup = new ec2.SecurityGroup(this, 'RDSSecurityGroup', {
      vpc,
      allowAllOutbound: true,
      description: 'Security group for RDS instance',
    });
    rdsSecurityGroup.addIngressRule(ec2SecurityGroup, ec2.Port.tcp(3306), 'Allow MySQL from EC2');

    // IAM Role for EC2 Instance
    const ec2Role = new iam.Role(this, 'EC2IAMRole', {
      assumedBy: new iam.ServicePrincipal('ec2.amazonaws.com'),
    });
    ec2Role.addManagedPolicy(iam.ManagedPolicy.fromAwsManagedPolicyName('AmazonSSMManagedInstanceCore'));
```

aws-cdk TypeScript for the EC2 and RDS instances:

```
    // Launch EC2 Instance in Public Subnet
    const ec2Instance = new ec2.Instance(this, 'MigrationEC2', {
      vpc,
      vpcSubnets: {subnetType: ec2.SubnetType.PUBLIC},
      instanceType: ec2.InstanceType.of(ec2.InstanceClass.T3, ec2.InstanceSize.MICRO),
      machineImage: ec2.MachineImage.latestAmazonLinux2(),
      securityGroup: ec2SecurityGroup,
      role: ec2Role,
    });

    // Create RDS Instance in Private Subnet
    new rds.DatabaseInstance(this, 'MigrationRDS', {
      engine: rds.DatabaseInstanceEngine.mysql({version: rds.MysqlEngineVersion.VER_8_0}),
      vpc,
      vpcSubnets: {subnetType: ec2.SubnetType.PRIVATE_ISOLATED},
      securityGroups: [rdsSecurityGroup],
      instanceType: ec2.InstanceType.of(ec2.InstanceClass.BURSTABLE3, ec2.InstanceSize.MICRO),
      allocatedStorage: 20,
      removalPolicy: cdk.RemovalPolicy.DESTROY, // WARNING:Deletes DB on stack removal
    })
  }
}
```

Evidence of Successful Testing:

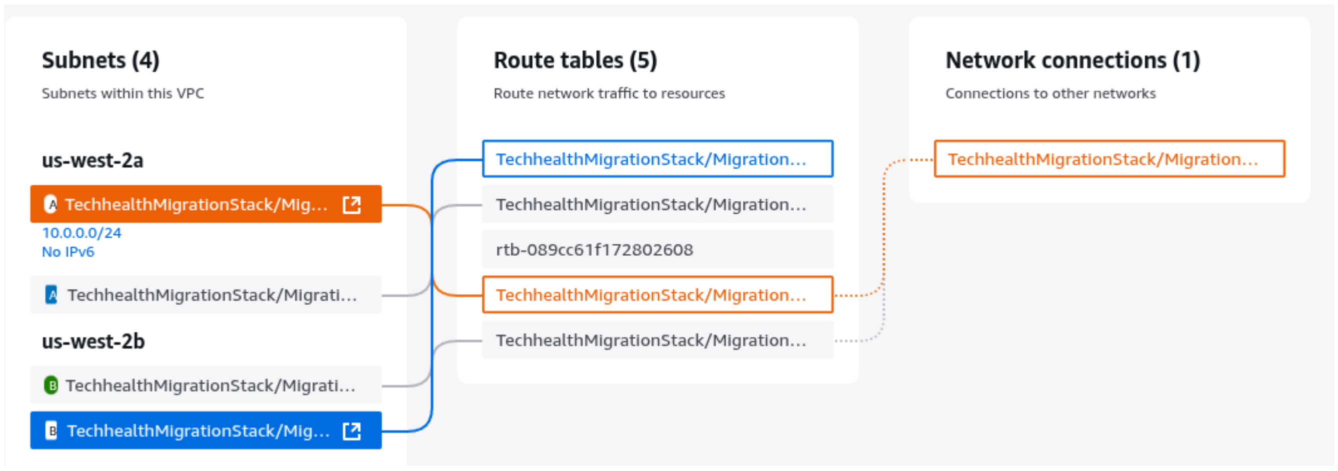
Successful deployment



```
IAM Statement Changes
┌───┬─────────────────────┬────────┬────────────────┬──────────────────────────────┬───────────┐
│   │ Resource            │ Effect │ Action         │ Principal                    │ Condition │
├───┼─────────────────────┼────────┼────────────────┼──────────────────────────────┼───────────┤
│ - │ ${EC2IAMRole.Arn}   │ Allow  │ sts:AssumeRole │ Service:ec2.amazonasws.com   │           │
├───┼─────────────────────┼────────┼────────────────┼──────────────────────────────┼───────────┤
│ + │ ${EC2IAMRole.Arn}   │ Allow  │ sts:AssumeRole │ Service:ec2.amazonaws.com    │           │
└───┴─────────────────────┴────────┴────────────────┴──────────────────────────────┴───────────┘
(NOTE: There may be security-related changes not in this list. See https://github.com/aws/aws-cdk/issues/1299)


Do you wish to deploy these changes (y/n)? y
TechhealthMigrationStack: deploying... [1/1]
TechhealthMigrationStack: creating CloudFormation changeset...

 ✅  TechhealthMigrationStack

✨  Deployment time: 385.11s

Stack ARN:
arn:aws:cloudformation:us-west-2:650251684348:stack/TechhealthMigrationStack/1c3e92d0-40b0-11f0-9bb5-06f1144d8155

✨  Total time: 391.42s

eric@eric-desktop:~/techhealth-migration$ 
```

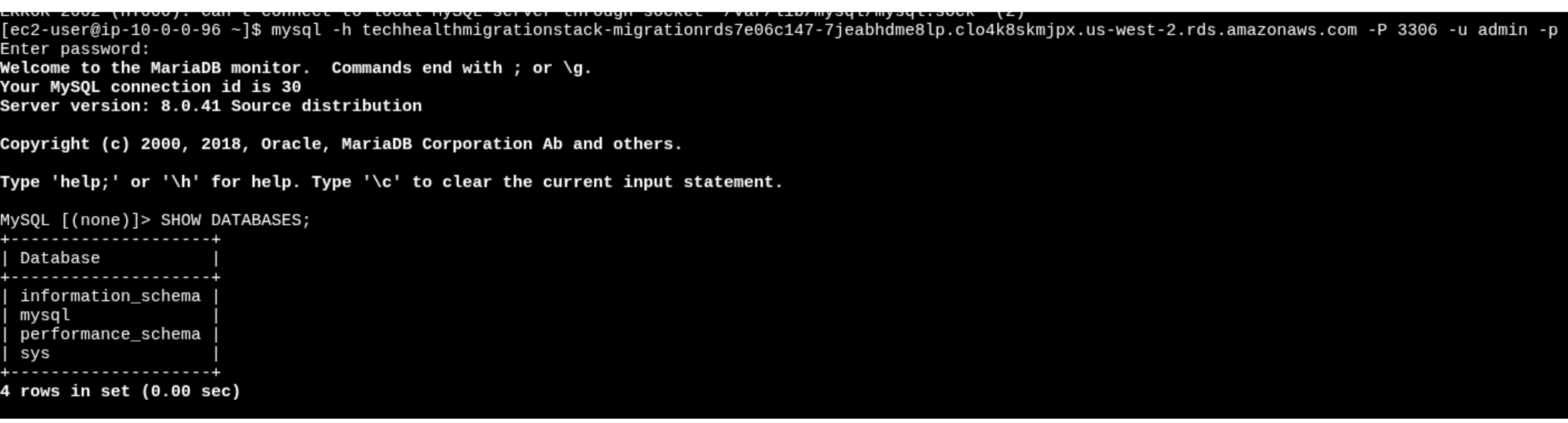
The screenshot above displays successful deployment of the cdk code shown earlier. As depicted in the screenshot, I initially had a typo in the IAM role for the EC2 which caused a rollback on the first attempt but it launched successfully after correcting that error.

Visual of traffic flow in the VPC


Subnets (4)
Subnets within this VPC
us-west-2a
TechhealthMigrationStack/Mig...
10.0.0.0/24
No IPv6
TechhealthMigrationStack/Migrati...
us-west-2b
TechhealthMigrationStack/Migrati...
TechhealthMigrationStack/Mig...
Route tables (5)
Route network traffic to resources
TechhealthMigrationStack/Migration...
TechhealthMigrationStack/Migration...
rtb-089cc61f172802608
TechhealthMigrationStack/Migration...
TechhealthMigrationStack/Migration...
Network connections (1)
Connections to other networks
TechhealthMigrationStack/Migration...

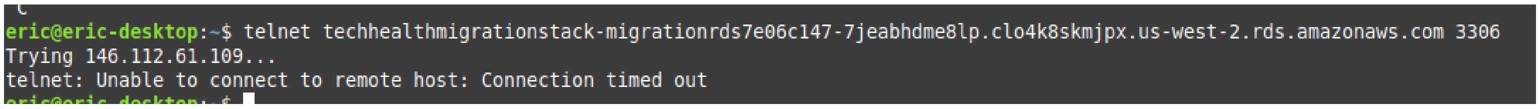
Highlighted in orange is a public subnet which leads to the internet gateway. Highlighted in blue is a private subnet which is isolated to the private subnet within the VPC unless a route or security group is explicitly specified to allow traffic.

Connection established between EC2 and RDS instances


```
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/var/lib/mysql/mysql.sock' (2)
[ec2-user@ip-10-0-0-96 ~]$ mysql -h techhealthmigrationstack-migrationrds7e06c147-7jeabhdme8lp.clo4k8skmjpx.us-west-2.rds.amazonaws.com -P 3306 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 30
Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

Above I used the command: mysql -h <hostname> -P 3306 -u admin -p ---- to log into database running on the RDS instance. I initially stumbled here because I didn't explicitly setup credentials for the database so I had to retrieve the username and password from the secrets manager on AWS.

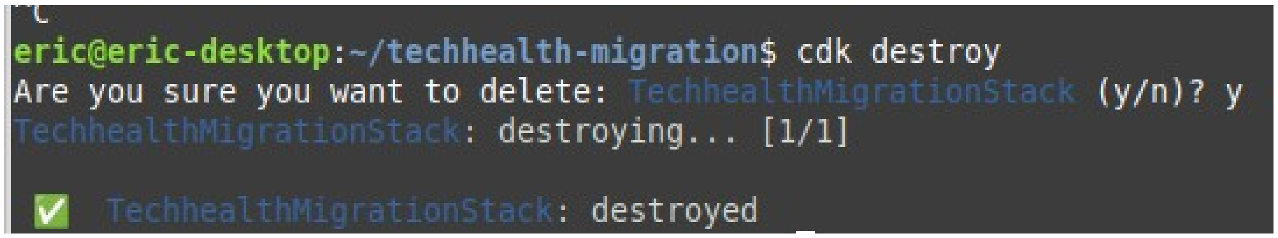
Unable to connect to database from outside resource



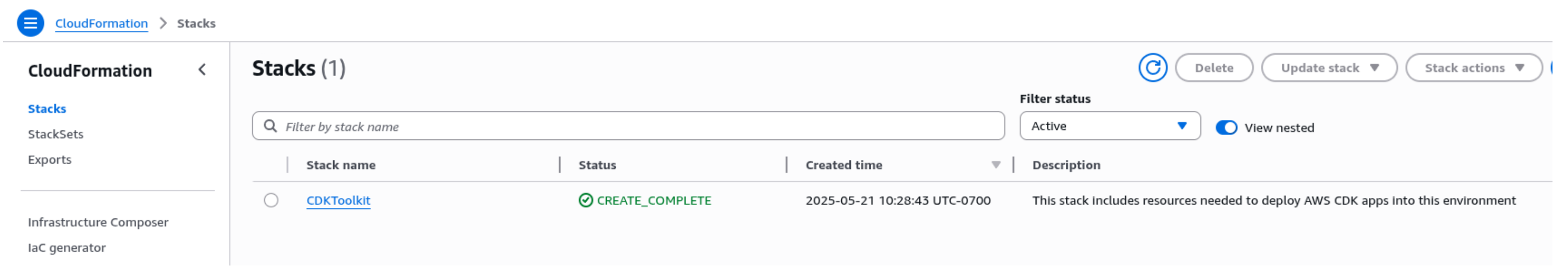
```
eric@eric-desktop:~$ telnet techhealthmigrationstack-migrationrds7e06c147-7jeabhdme8lp.clo4k8skmjpx.us-west-2.rds.amazonaws.com 3306
Trying 146.112.61.109...
telnet: Unable to connect to remote host: Connection timed out
```

To quickly test the security configurations, I attempted to connect to the RDS instance using my local machine. The connection timed-out indicating the security groups were appropriately configured.

Wrap up



```
eric@eric-desktop:~/techhealth-migration$ cdk destroy
Are you sure you want to delete: TechhealthMigrationStack (y/n)? y
TechhealthMigrationStack: destroying... [1/1]

 ✅  TechhealthMigrationStack: destroyed
```



Stacks (1)

Filter status: Active

Stack name	Status	Created time	Description
CDKToolkit	CREATE_COMPLETE	2025-05-21 10:28:43 UTC-0700	This stack includes resources needed to deploy AWS CDK apps into this environment

Abracadabra or more precisely “cdk destroy” and just like that our entire stack disappears. Pictured above is the notification that the stack has been destroyed in the command-line and then a check that it is not saved anywhere in CloudFormation on the AWS console.

Lesson Learned:

Description:

TechHealth’s original AWS infrastructure was manually created through the AWS console, leading to inconsistencies across environments, difficulty in auditing, and challenging is reproducing resources.

Impact:

- Inconsistent configurations across development, staging, and production
- Lack of traceability for changes and permissions

- Increased operational overhead during troubleshooting and onboarding

Resolution:

Migrating to aws-cdk using TypeScript enabled a fully automated, repeatable, and version-controlled infrastructure. This change supports better governance, scalability, and long-term maintainability. Further changes and architecture/access patterns can be documented as part of the IaC codebase.