Network Security: Add Private Subnets with NAT Gateway

We'll cover the following

- Objective
- Steps
- Add private subnets and NAT gateway
- Switching our ASG to use private subnets

Objective#

Make our instances inaccessible from the internet.

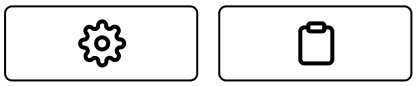
Steps#

- Add private subnets with a NAT gateway.
- Switch our ASGs to use the private subnets.

Add private subnets and NAT gateway

Now, we're going to add new security groups for our private subnets that allows ports 22 and 8443 only.

allows ports 22 and 8443 only.



```
PrivateSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: !Ref VPC
    GroupDescription:
      !Sub 'Internal Security group for ${AWS::StackName}'
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 8443
        ToPort: 8443
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
```

stage.yml

Next, we have to change the `SecurityGroupIds` property inside the `InstanceLaunchTemplate` resource, so that it refers to `PrivateSecurityGroup.GroupId` instead of `SecurityGroup.GroupId`. In this way, new instances automatically become part of our new private security group.

```
SecurityGroupIds:
  - !GetAtt PrivateSecurityGroup.GroupId
```

stage.yml

Next, we add a new subnet per availability zone with `MapPublicIpOnLaunch` set to `false`, and a CIDR range that doesn't overlap with any of our other subnets.

```
PrivateSubnetAZ1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [ 0, !GetAZs '' ]
    CidrBlock: 10.0.128.0/18
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
      - Key: AZ
        Value: !Select [ 0, !GetAZs '' ]

PrivateSubnetAZ2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [ 1, !GetAZs '' ]
    CidrBlock: 10.0.192.0/18
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
      - Key: AZ
        Value: !Select [ 1, !GetAZs '' ]
```

stage.yml

Now we must create an Elastic IP address for each NAT gateway.

```
EIPAZ1:
  Type: AWS::EC2::EIP
  DependsOn: InternetGatewayAttachment
  Properties:
    Domain: vpc

EIPAZ2:
  Type: AWS::EC2::EIP
  DependsOn: InternetGatewayAttachment
  Properties:
    Domain: vpc
```

stage.yml

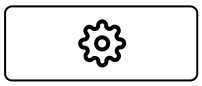
Next, let’s add the NAT gateways.

```
NATGatewayAZ1:
  Type: AWS::EC2::NatGateway
  Properties:
    AllocationId: !GetAtt EIPAZ1.AllocationId
    SubnetId: !Ref SubnetAZ1
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
      - Key: AZ
        Value: !Select [ 0, !GetAZs '' ]

NATGatewayAZ2:
  Type: AWS::EC2::NatGateway
  Properties:
    AllocationId: !GetAtt EIPAZ2.AllocationId
    SubnetId: !Ref SubnetAZ2
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
      - Key: AZ
        Value: !Select [ 1, !GetAZs '' ]
```

stage.yml

Now let’s add route tables to map outgoing internet traffic to the NAT gateways.



```
PrivateSubnetRouteTableAZ1:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
      - Key: AZ
        Value: !Select [ 0, !GetAZs '' ]

PrivateSubnetRouteTableAZ2:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
      - Key: AZ
        Value: !Select [ 1, !GetAZs '' ]

PrivateRouteAZ1:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PrivateSubnetRouteTableAZ1
    DestinationCidrBlock: 0.0.0.0/0
    NatGatewayId: !Ref NATGatewayAZ1

PrivateRouteAZ2:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PrivateSubnetRouteTableAZ2
    DestinationCidrBlock: 0.0.0.0/0
    NatGatewayId: !Ref NATGatewayAZ2

PrivateSubnetRouteTableAssociationAZ1:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: !Ref PrivateSubnetRouteTableAZ1
    SubnetId: !Ref PrivateSubnetAZ1

PrivateSubnetRouteTableAssociationAZ2:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: !Ref PrivateSubnetRouteTableAZ2
    SubnetId: !Ref PrivateSubnetAZ2
```

stage.yml

Switching our ASG to use

private subnets#

Finally, we have to switch the ASG to launch new instances in the private subnets rather than the public. The instances in the public subnets won't be terminated until the new ones in the private subnets are launched.

Let's change the `VPCZoneIdentifier` in `ScalingGroup` to refer to `PrivateSubnetAZ1` and `PrivateSubnetAZ2` instead of `SubnetAZ1` and `SubnetAZ2`.

```
VPCZoneIdentifier:
  - !Ref PrivateSubnetAZ1
  - !Ref PrivateSubnetAZ2
```

stage.yml

```
./deploy-infra.sh


=========== Deploying setup.yml ===========

Waiting for changeset to be created..

No changes to deploy. Stack awsbootstrap-setup is up to date


=========== Packaging main.yml ===========


=========== Deploying main.yml ===========

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - awsbootstrap
[
    "https://prod.the-good-parts.com",
    "https://staging.the-good-parts.com"
]
```

terminal

After the new instances have been launched in the new private subnets, and the old ones have been terminated, we can verify that our application

is still reachable through the load balancer endpoints.

```
for run in {1..20}; do curl -s https://staging.the-good-parts.com; done | sort |
10 Hello HTTPS World from ip-10-0-187-72.ec2.internal in awsbootstrap-Staging-10
10 Hello HTTPS World from ip-10-0-222-16.ec2.internal in awsbootstrap-Staging-10
```

terminal

```
for run in {1..20}; do curl -s https://prod.the-good-parts.com; done | sort un
10 Hello HTTPS World from ip-10-0-128-220.ec2.internal in awsbootstrap-Prod-1PT6
10 Hello HTTPS World from ip-10-0-248-112.ec2.internal in awsbootstrap-Prod-1PT6
```

terminal

And now is a good time to push all our changes to GitHub.

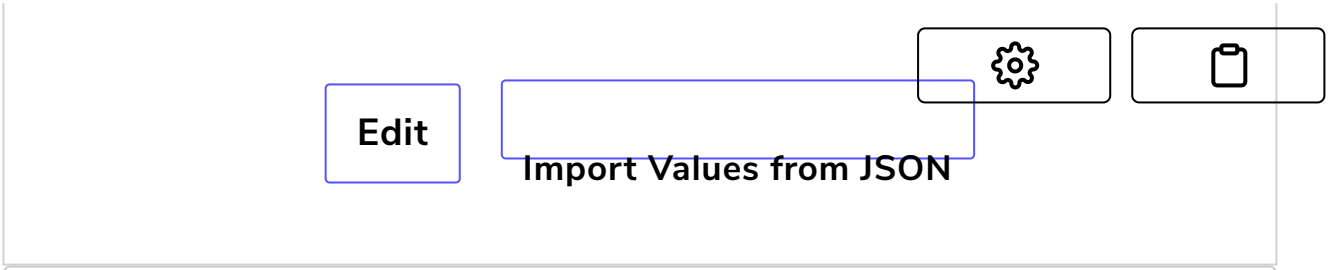
```
git add stage.yml
git commit -m "Move instances into private subnets"
git push
```

terminal

> **Note:** All the code has been already added and we are pushing it on our repository as well.

This code requires the following API keys to execute:

username	Not Specified...
AWS_ACCESS_KE...	Not Specified...
AWS_SECRET_AC...	Not Specified...
AWS_REGION	us-east-1
Github_Token	Not Specified...




```
{
  "name": "aws-bootstrap",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node ./node_modules/pm2/bin/pm2 start ./server.js --name hello_aws
    "stop": "node ./node_modules/pm2/bin/pm2 stop hello_aws",
    "build": "echo 'Building...'"
  },
  "dependencies": {
    "pm2": "^4.2.0"
  }
}
```

In the next lesson, we will only allow HTTPS port in the public subnets.

← Back

Next →

Network Security: Set up SSM for SS...

Network Security: Enabling HTTPS po...

Mark as Completed

Report an Issue