# AWS CloudFormation Infrastructure Automation

**JULEANNY NAVAS**
AWS Cloud Professional

## Introduction

In this project, I automated the deployment and management of a multi-layer AWS infrastructure using **AWS CloudFormation**. The project was designed to demonstrate a full lifecycle approach to I**nfrastructure as Code (IaC)**, starting with the provisioning of a Virtual Private Cloud (**VPC**) networking layer and expanding to include an application layer with an **EC2** instance and a **Security Group**.

By leveraging CloudFormation templates, I defined and deployed both the networking and application stacks, **updated stack configurations** to reflect changing requirements (e.g., modifying Security Group settings), and **implementing deletion policies** to preserve critical resources like Amazon EBS snapshots. I also used **Infrastructure Composer** (formerly CloudFormation Designer) to visually explore and validate the template structure.

This hands-on experience emphasizes the value of automating infrastructure deployment, improving reproducibility, minimizing human error, and aligning with modern DevOps and cloud architecture best practices.

---

## Project structure

**Section 1: Deploy Networking Layer Stack**

Use AWS CloudFormation to deploy a foundational networking stack that includes:

- Virtual Private Cloud (VPC)
- Internet Gateway (IGW)
- Route Table and Public Subnet

**Section 2: Deploy Application Layer Stack**

Deploy a second CloudFormation stack that references the networking layer and provisions:

- Amazon EC2 instance and Security Group associated with the EC2 instance

**Section 3: Stack Update - Modify Security Group rules**

Update the application-layer stack to apply changes to the security group settings, demonstrating real-time infrastructure updates using CloudFormation.

**Section 4: Exploring Templates with Infrastructure Composer**

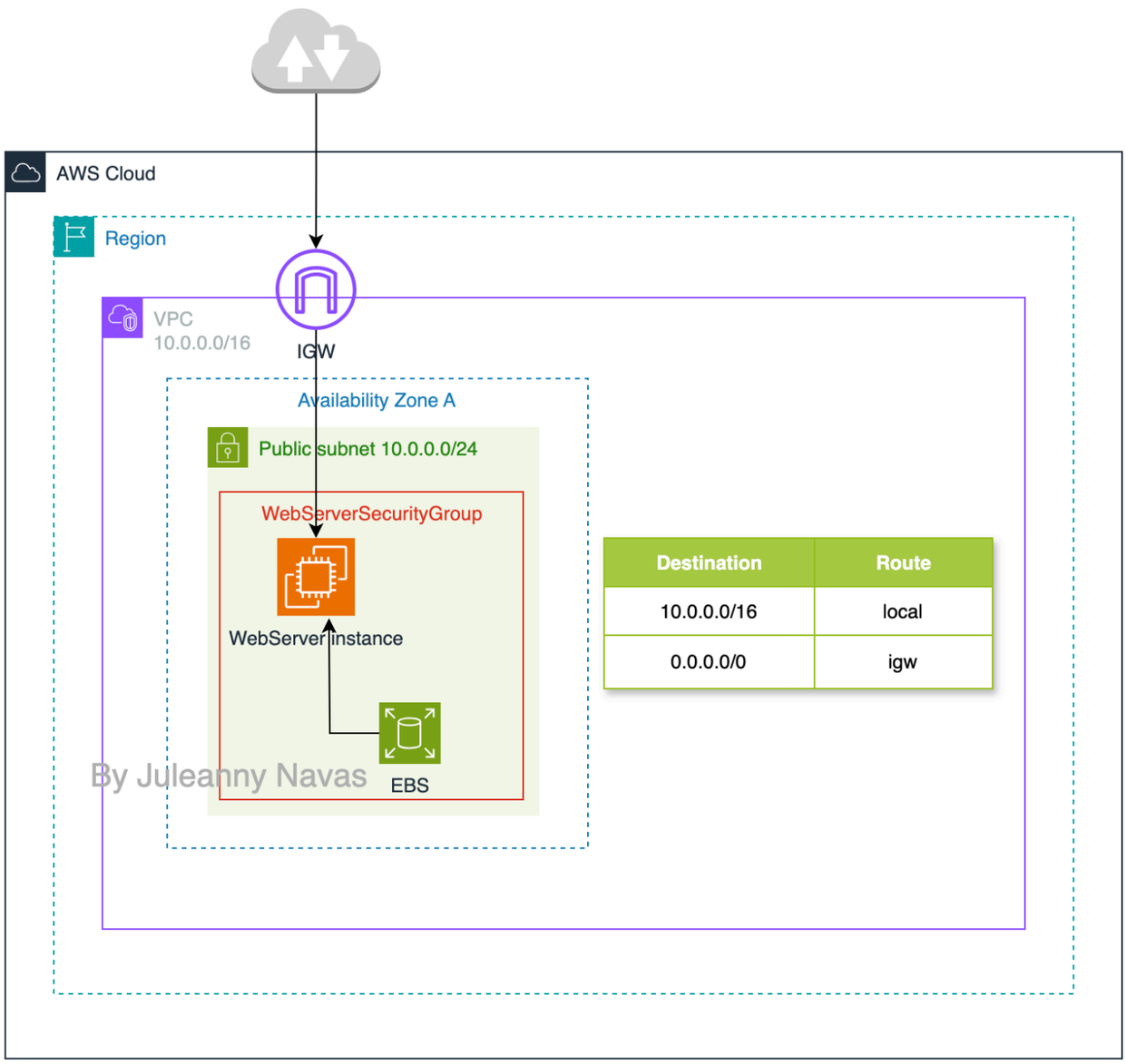
**Section 5: Stack Deletion and Snapshot retention via Deletion Policy**

- Delete the application stack while preserving critical data by applying a Deletion Policy. Confirm that an Amazon EBS volume snapshot is created automatically as part of the cleanup process.

# Tools & Services used

- **AWS CloudFormation** – Defined and deployed networking and application infrastructure as code, enabling consistent and repeatable provisioning of AWS resources.
- **Amazon VPC** – Created a custom virtual network layer to securely isolate and control cloud resources.
- **AWS Internet Gateway, Route Tables, and Subnets** – Enabled external connectivity and routing within the VPC.
- **AWS Security Groups** – Defined and updated firewall rules to manage traffic to and from the EC2 instance.
- **Amazon EC2** – Provisioned a virtual machine to serve as the application server.
- **Amazon Elastic Block Store (Amazon EBS)** – Attached persistent block storage to the EC2 instance; included deletion policy for snapshot creation.
- **AWS Application Composer** – Used to visually explore and understand CloudFormation templates (formerly CloudFormation Designer).
- **YAML** – Employed as the format for writing human-readable CloudFormation templates.
- **Architecture Diagrams** – Created infrastructure diagrams using app.diagrams.net to visualize stack components and relationships.

# Architecture Overview



# Step-by-Step Implementation

## Section 1: Deploy Networking Layer Stack

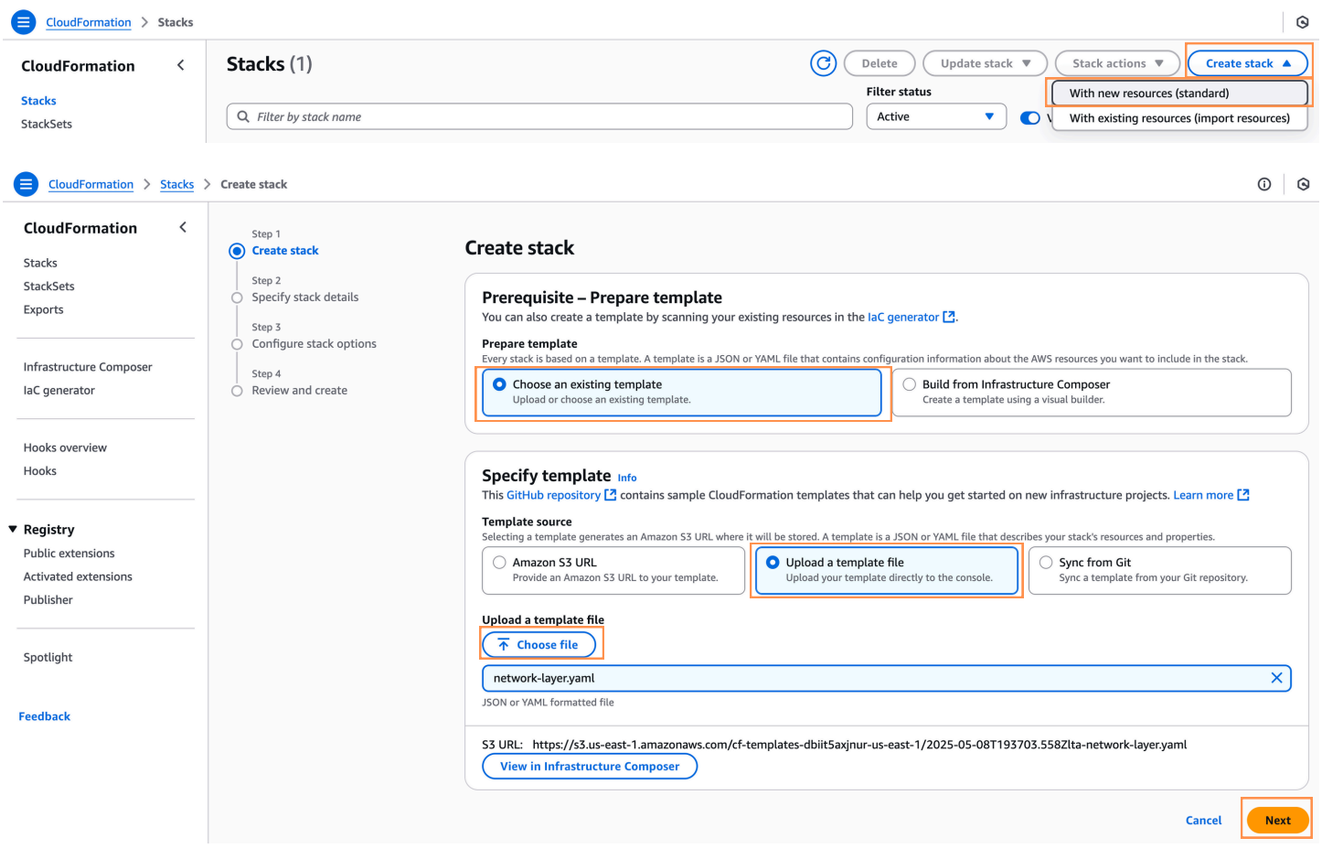
I deployed a foundational networking layer using AWS CloudFormation to establish a custom virtual private cloud (**VPC**) with a **public subnet.** This setup enables full control over IP address ranges, subnets, routing, and security.

### Key Highlights:

- **Custom VPC**: Defined with the IPv4 CIDR block 10.0.0.0/16, providing up to 65,536 private IP addresses, ideal for future scalability.
- **Public Subnet**: Created to host internet-accessible resources.
- **CloudFormation Template**: Used network-layer.yaml (see attached .yaml file in repository) to provision all components declaratively.

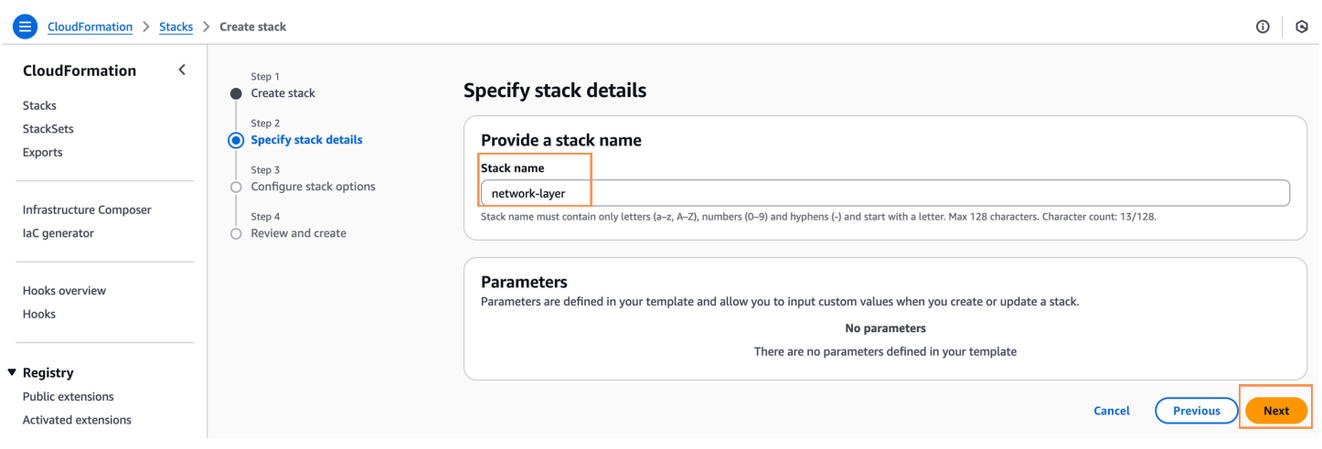
### Deployment process:

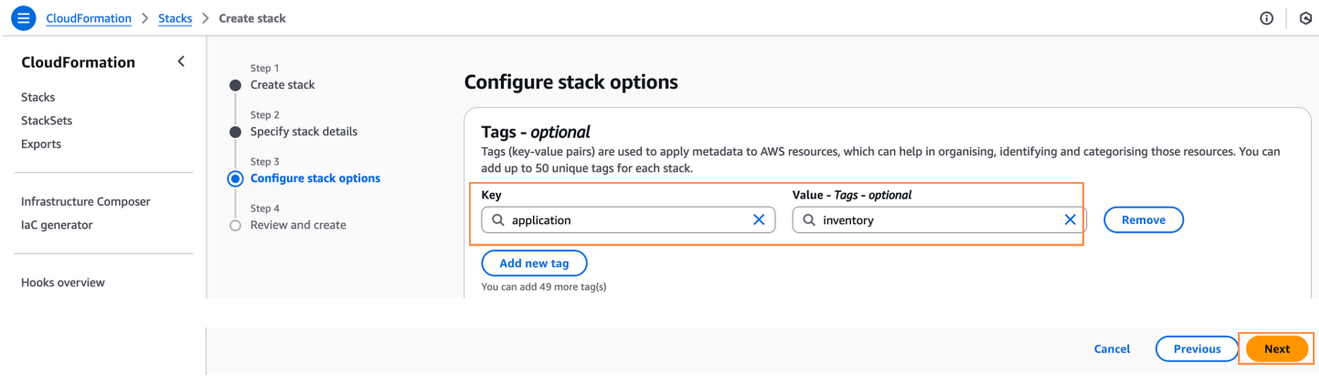
1. **Prepare the template:**
   - Uploaded the *network-layer.yaml* file using the **AWS CloudFormation Console**.

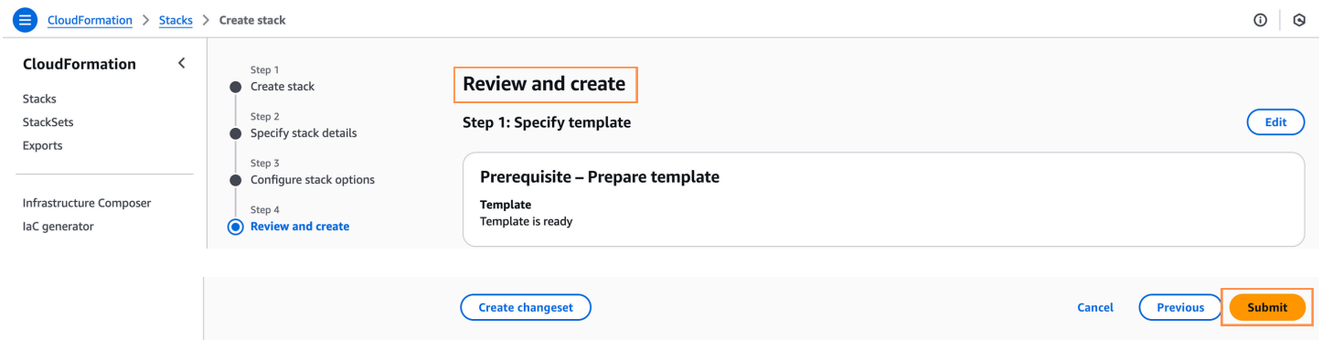




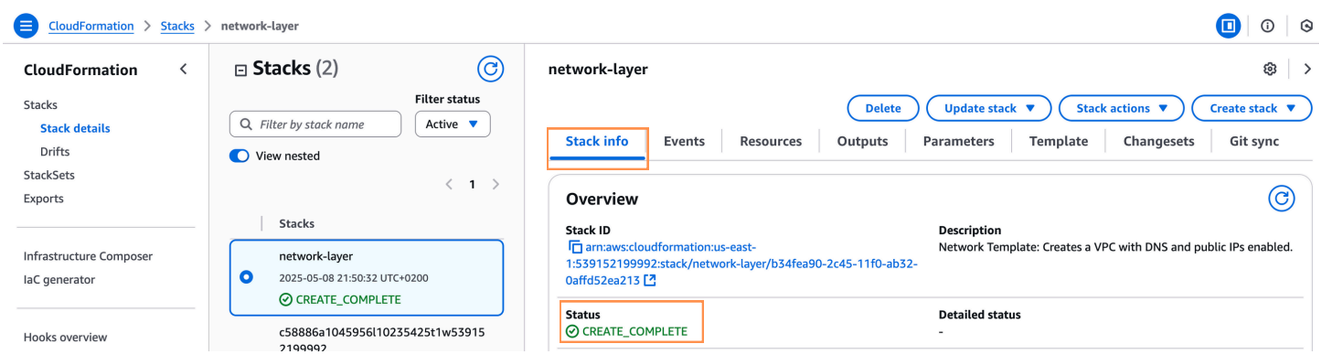




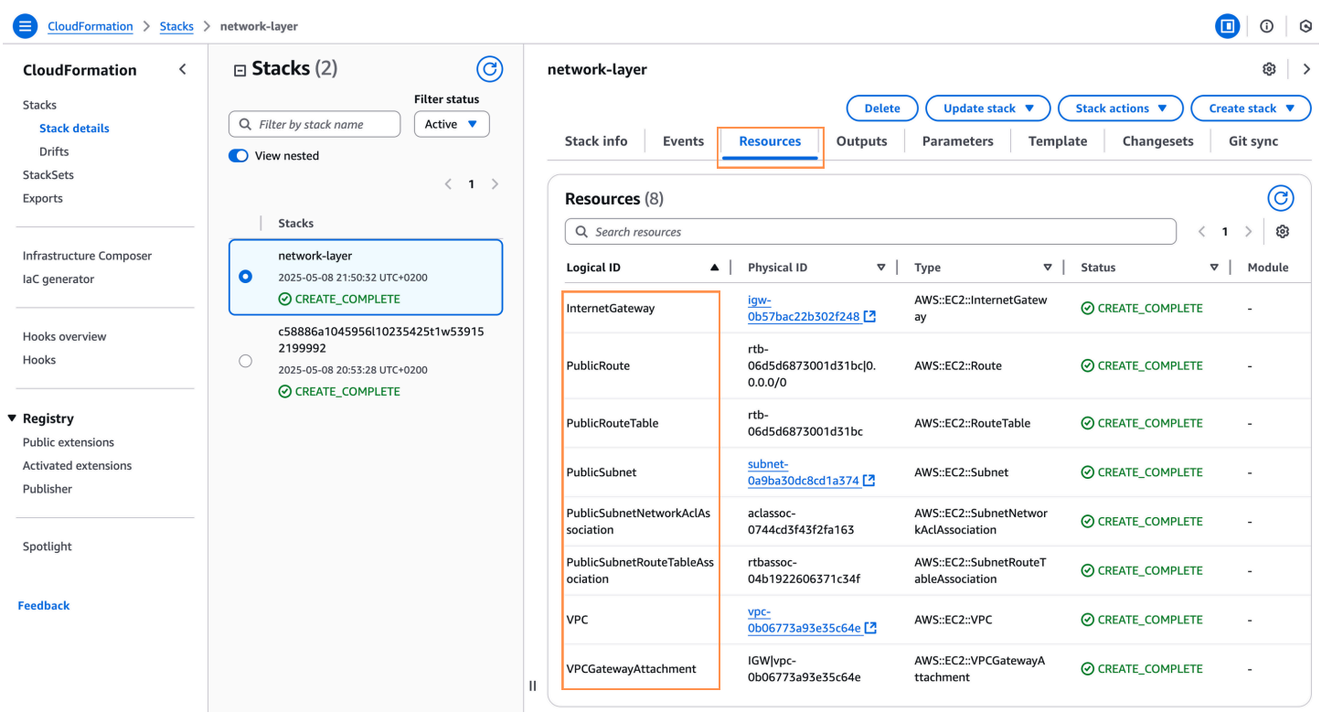


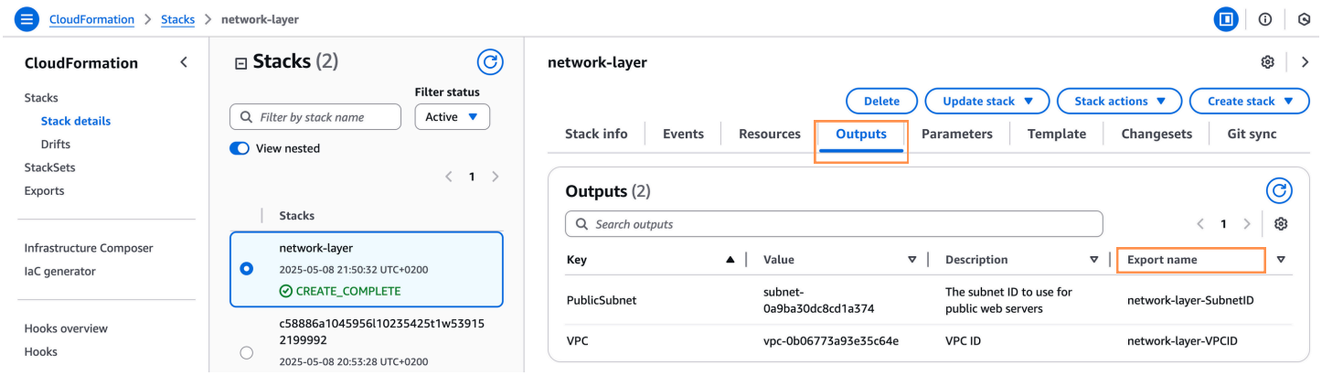

- Choose the **Stack info** tab. Wait for the **Status** to change to **CREATE_COMPLETE**. Refresh if necessary.



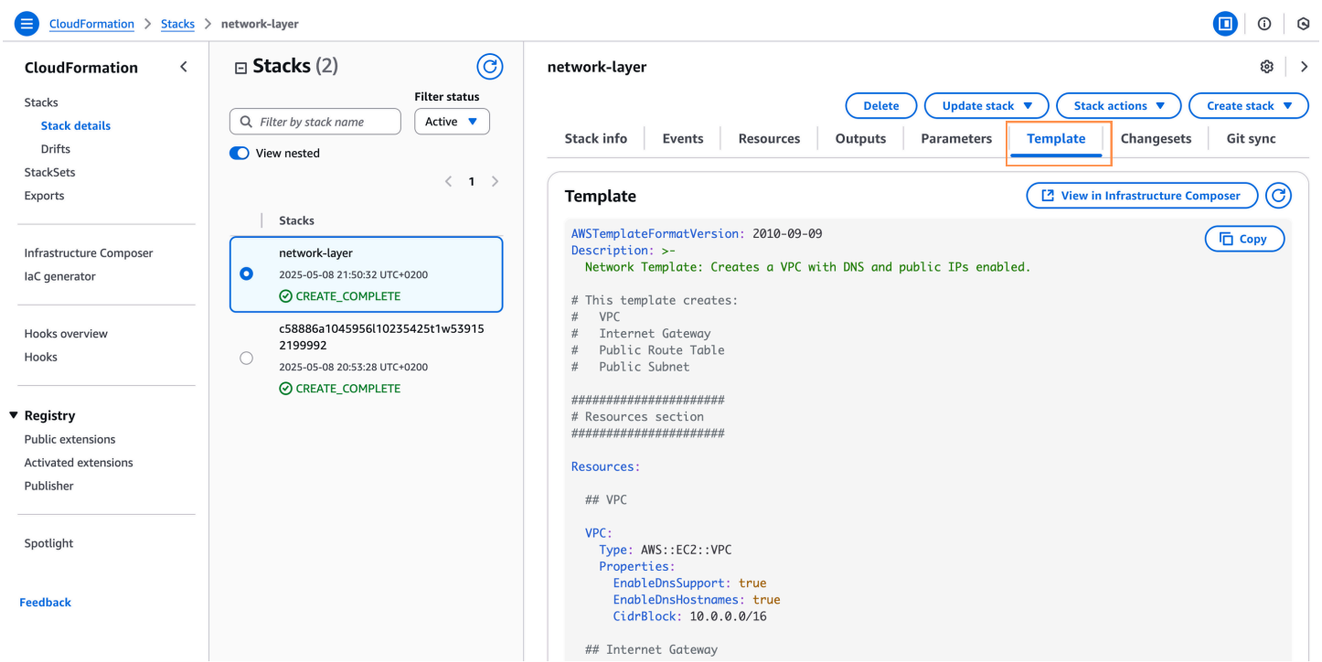

- Choose the **Resources** tab to see the list of the resources that were created by the template



- Choose the **Outputs** tab to see the VPC and Subnet IDs created by the stack. These values were exported with names to enable referencing from other CloudFormation stacks (Export name), allowing the application-layer to reuse this networking infrastructure in the next phase.




- Choose the **Template** tab to see the template that was used to create the stack.




---

## Section 2: Deploy Application Layer Stack

Launched a second CloudFormation stack (*application-layer*) to provision an **EC2** instance and **Security Group** (see attached *application-layer.yaml* file in repository). The stack dynamically imported the VPC and Subnet IDs from the netwotk-layer stack outputs using *Fn::ImportValue*, enabling resource creation within the existing networking layer.

Once deployed, the stack exposed a URL via the Outputs tab, confirming that the application server was successfully launched in the specified public subnet.

- The template takes the subnet ID from the *netwotk-layer* stack and uses it in the *application-layer* stack to launch the EC2 instance into the public subnet, which was created by the first stack.

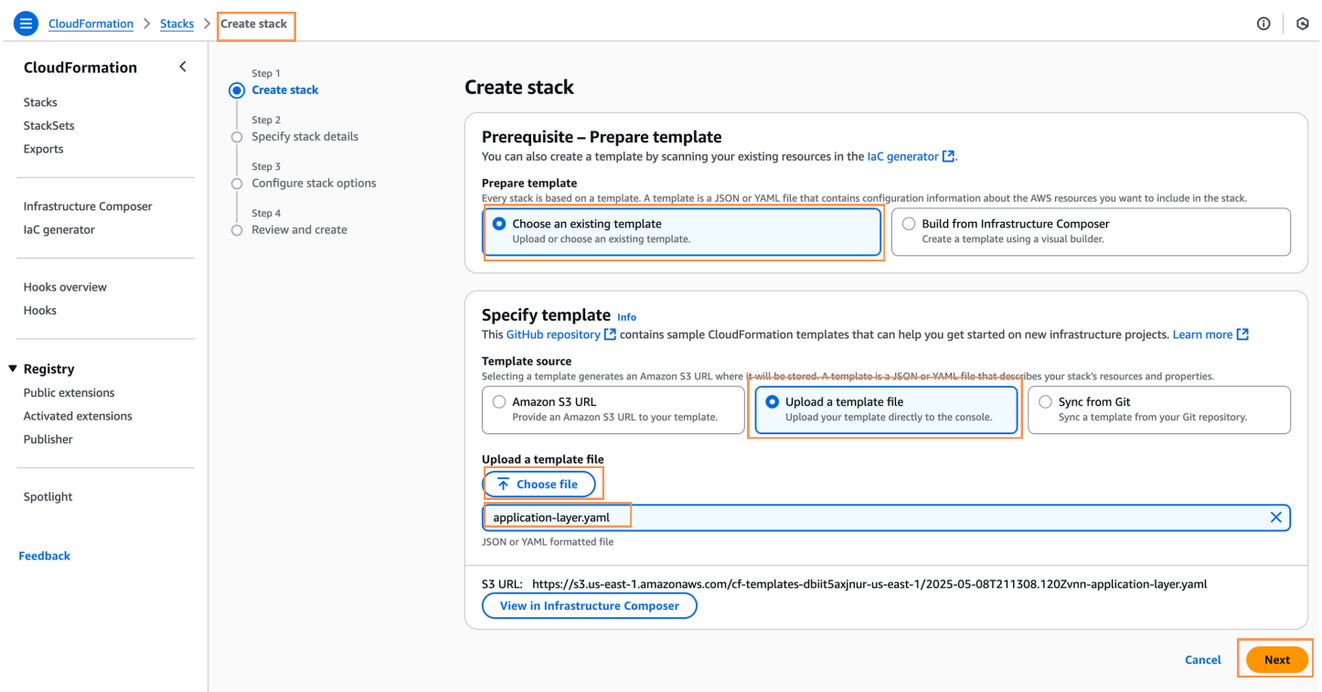
```
SubnetId:
  Fn::ImportValue:
    !Sub ${NetworkStackName}-SubnetID
```

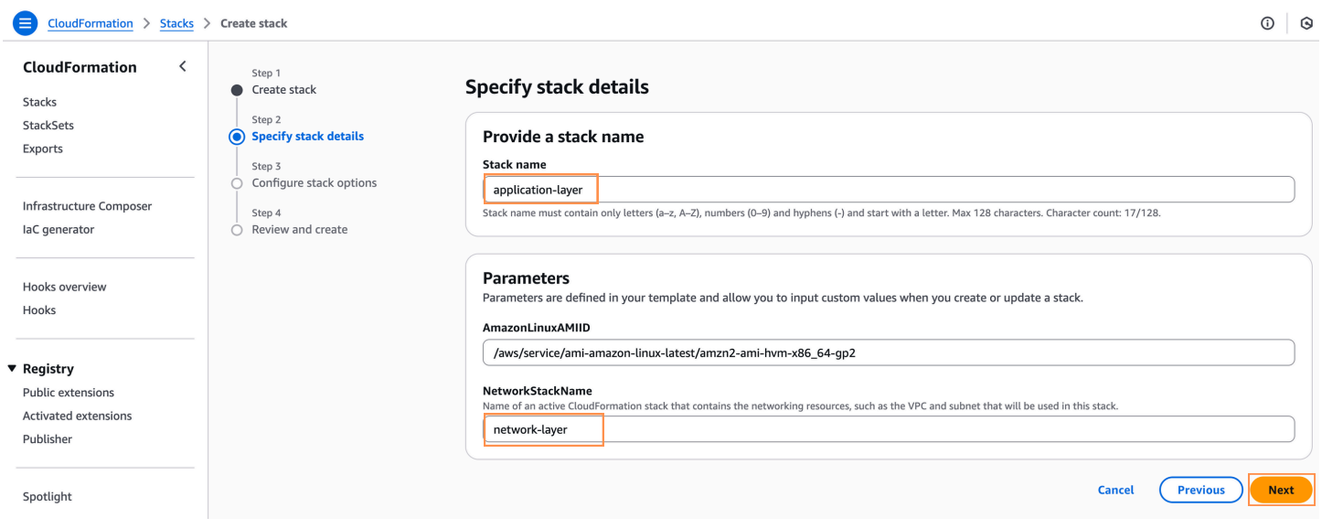
- In the WebServerSecurityGroup resource, the *application-layer* stack references values from the *network-layer* stack by importing the *network-layer-VPCID* from its Outputs. This ensures the Security Group is created within the same VPC that was provisioned by the first stack.

```yaml
WebServerSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Enable HTTP ingress
    VpcId:
      Fn::ImportValue:
        !Sub ${NetworkStackName}-VPCID
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0
    Tags:
      - Key: Name
        Value: Web Server Security Group
```

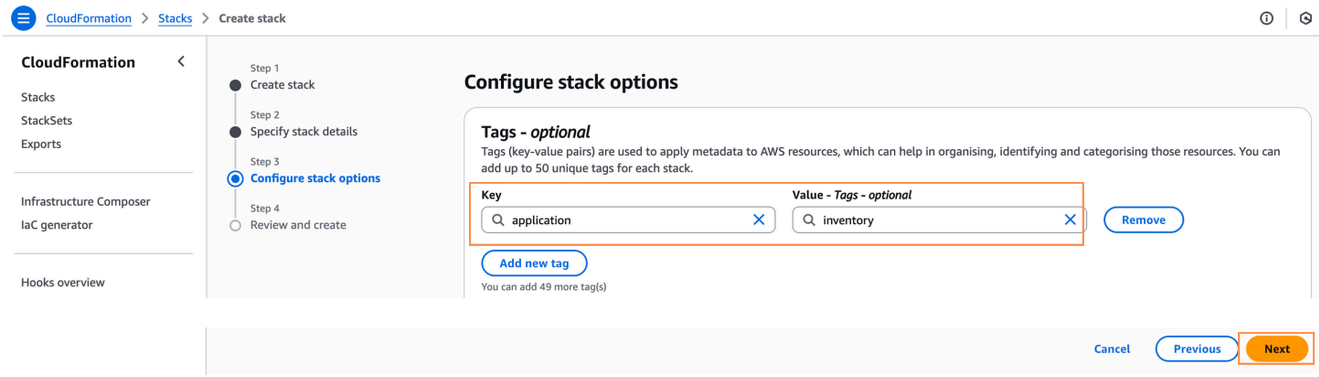
## Deployment process:

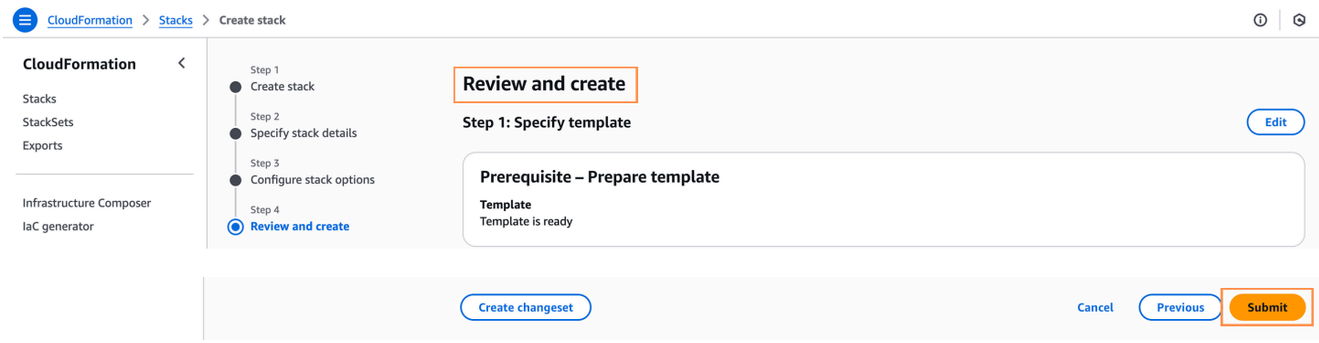
- Uploaded the *application-layer.yaml* file using the **AWS CloudFormation Console**.



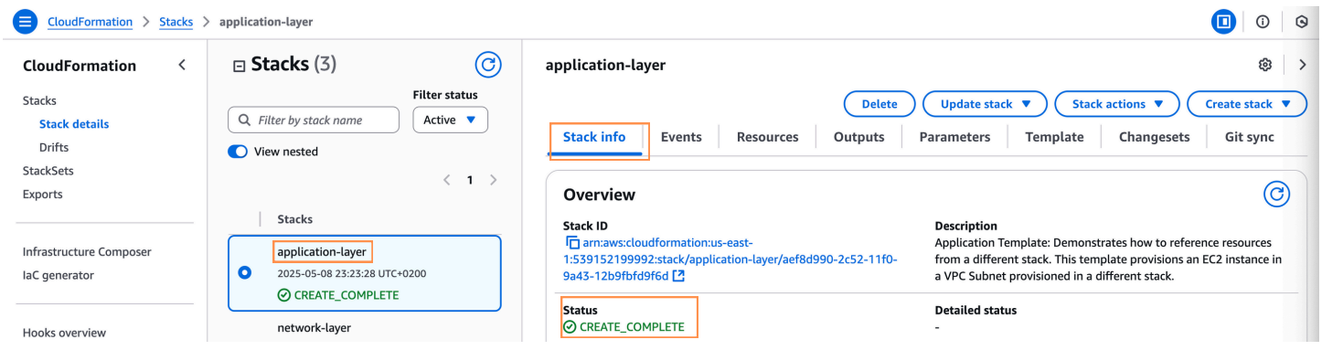




- The *Network Stack Name* parameter tells the template the name of the first stack that you created (*network-layer*), so it can retrieve values from the *Outputs*.

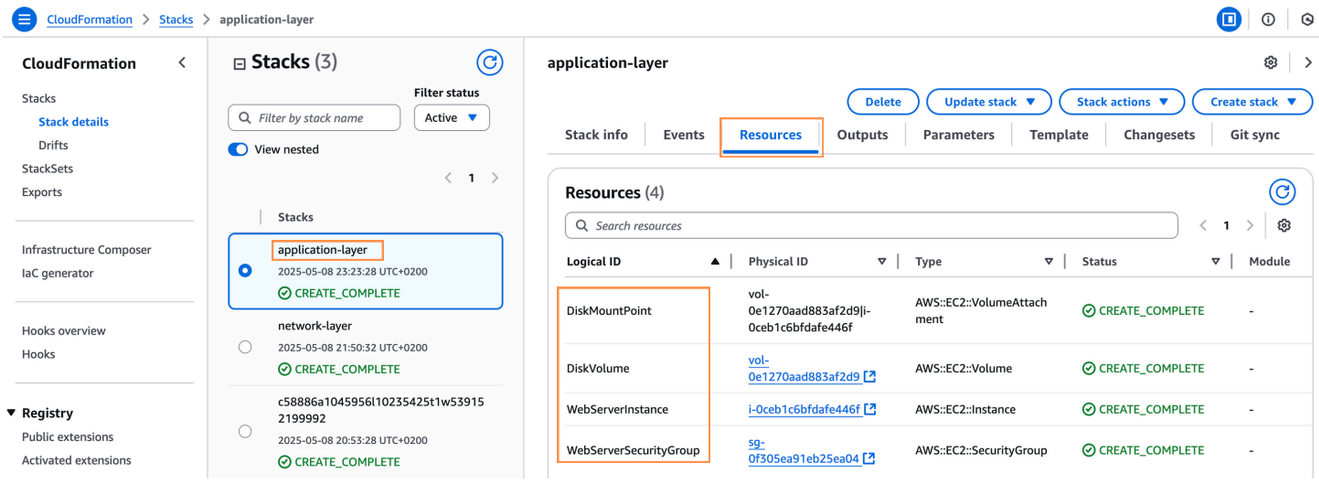


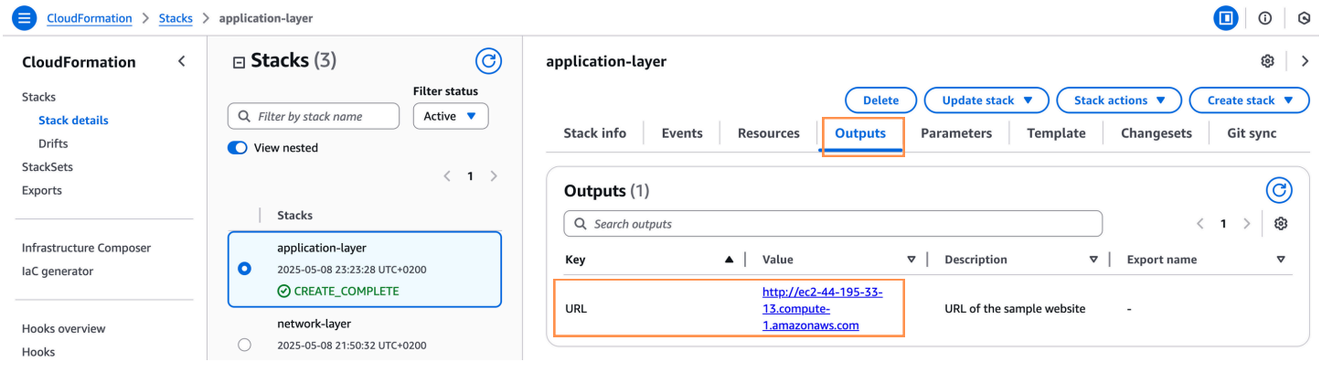

- Choose the **Stack info** tab. Wait for the **Status** to change to **CREATE_COMPLETE**. Refresh if necessary.




- Choose the **Resources** tab to see the list of the resources that were created by the template



- Choose the **Outputs** tab, copy the **URL** provided, open a new browser tab, and paste the URL to access the application. This confirms that the web server, deployed by the new CloudFormation stack, is running successfully.





**Congratulations, you have successfully launched the AWS CloudFormation sample.**

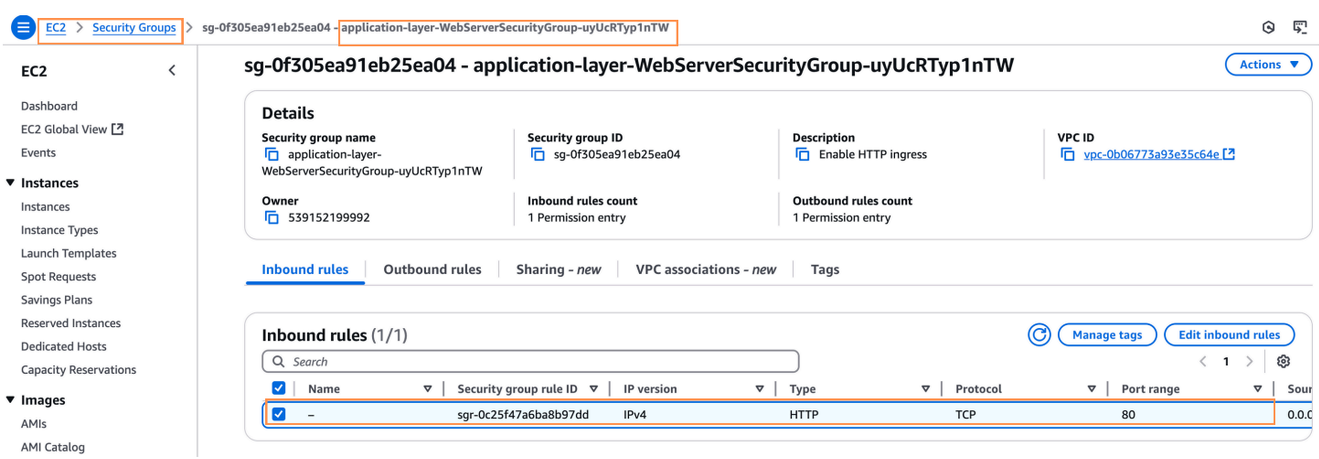
---

## Section 3: Stack update - Modify Security Group inbound rules

In this section, I updated the existing *application-layer* CloudFormation stack to modify the Security Group by adding an inbound rule for SSH (port 22). Using an updated template (see attached *application-layer2.yaml* file in repository), I replaced the existing stack template via the AWS CloudFormation console. The update applied a non-disruptive change to the WebServerSecurityGroup, demonstrating how CloudFormation can be used to manage infrastructure changes reliably and with version control best practices.

### Deployment process:

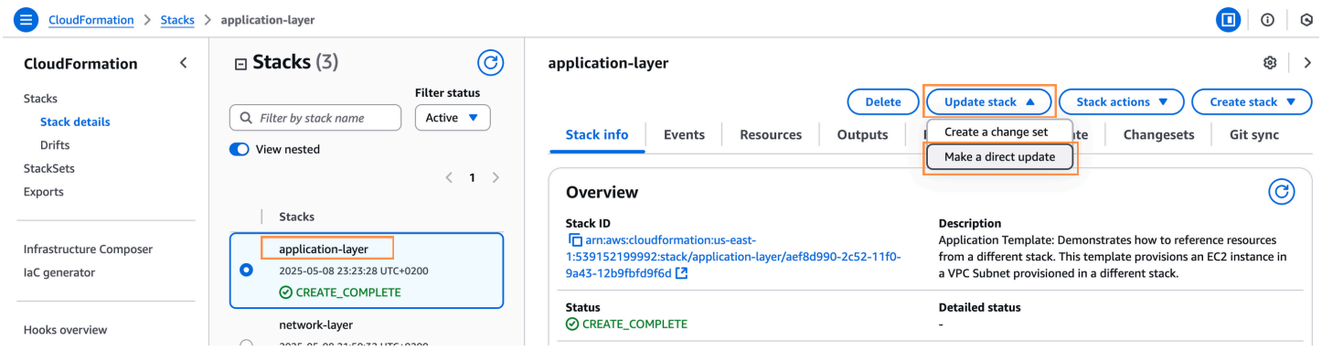
- Verify the current **inbound rules** of the EC2 **application-layer-WebServerSecurityGroup**

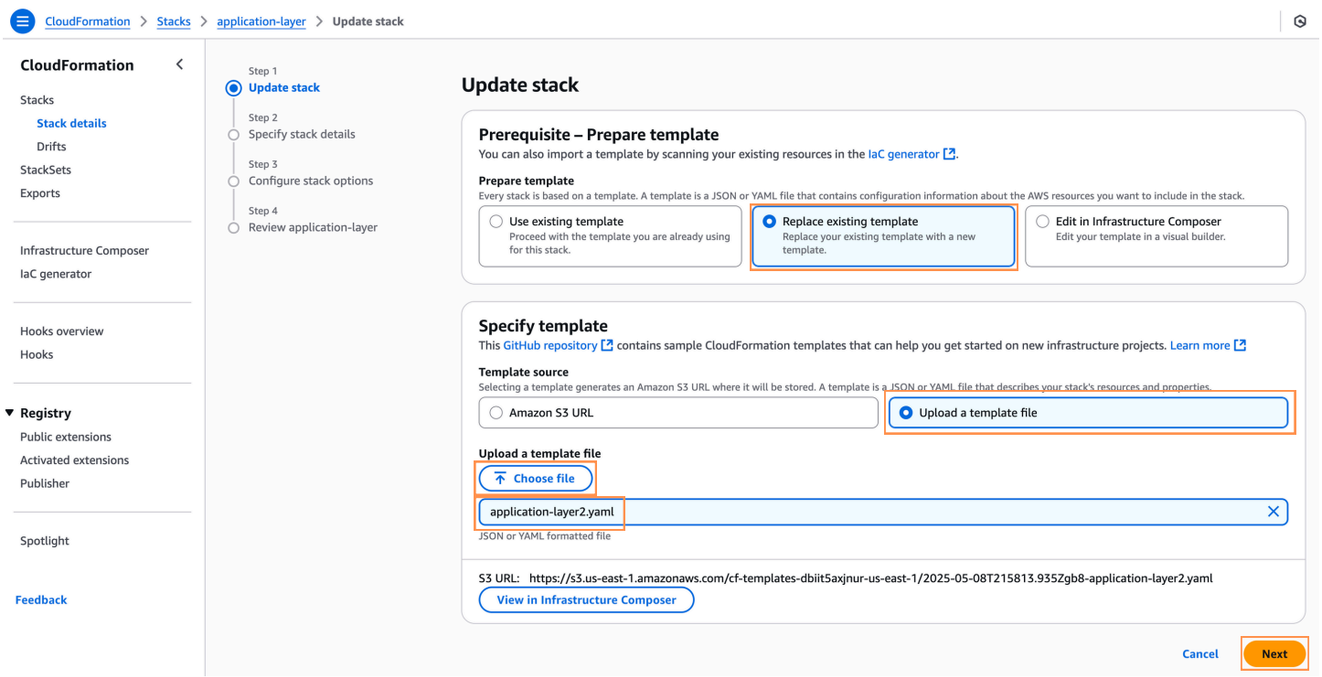


- Uploaded the *network-layer2.yaml* file using the **AWS CloudFormation Console**. This template has an additional configuration to permit inbound Secure Shell (SSH) traffic on port 22.

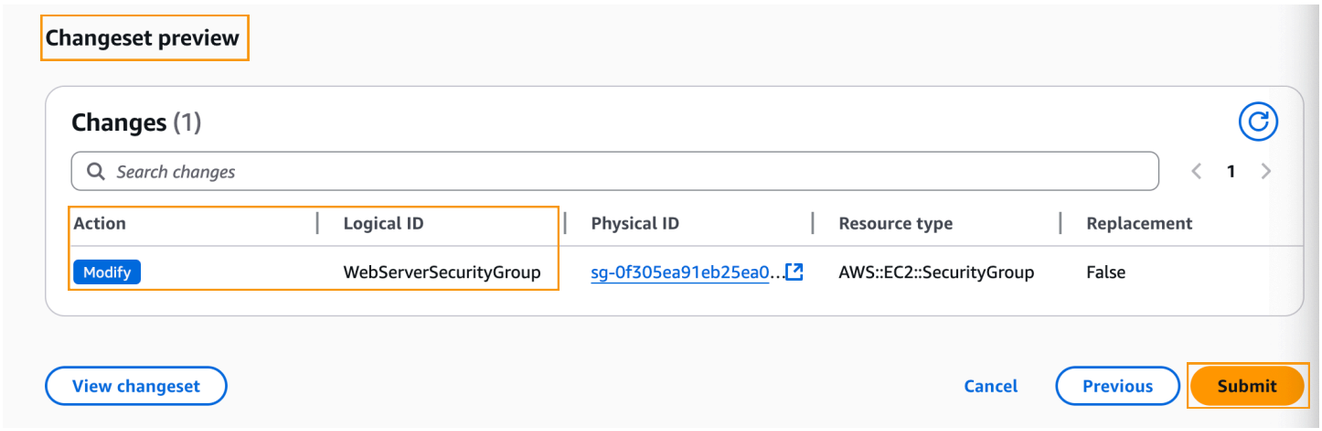
```yaml
  WebServerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable HTTP ingress
      VpcId:
        Fn::ImportValue:
          !Sub ${NetworkStackName}-VPCID
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
      Tags:
        - Key: Name
          Value: Web Server Security Group
```

- Using **AWS CloudFormation Console** update the *application-layer* stack

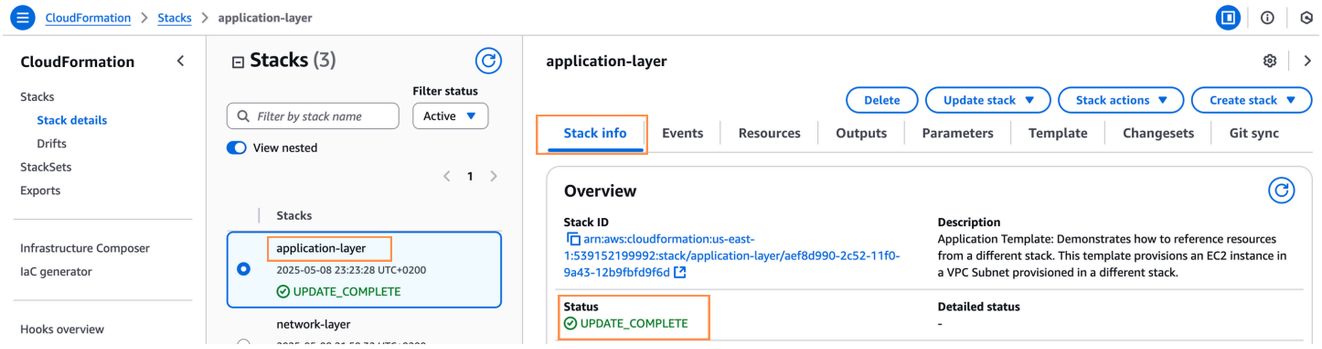


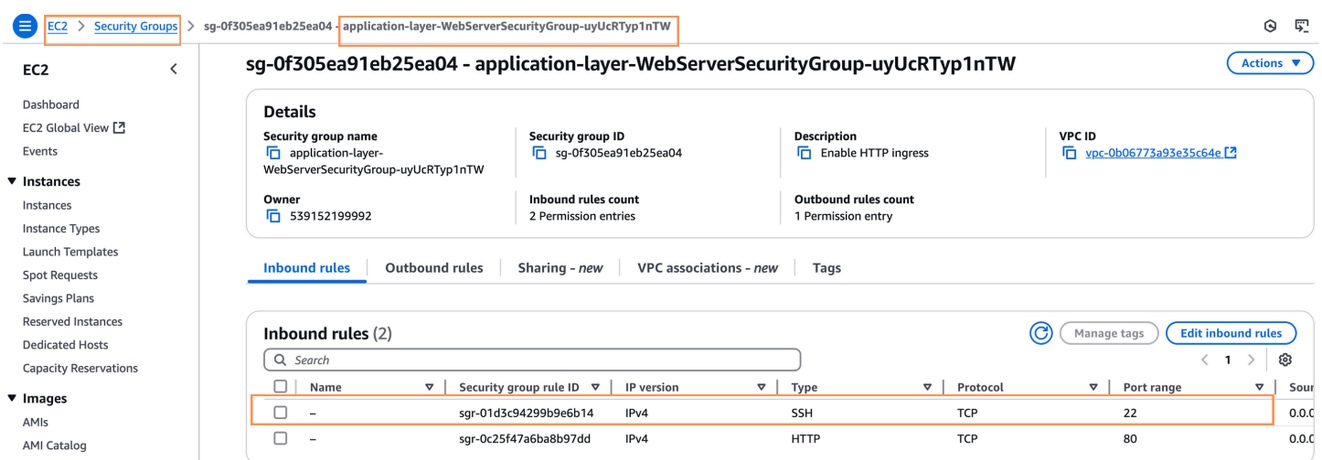
- Proceed through the next screens to reach the **Review-application-layer** page. At the bottom, in the **Change set preview** section, AWS CloudFormation outlines the specific resources that will be updated.




- In the **Stack info** tab. Wait for the **Status** to change to **UPDATE_COMPLETE**. Refresh if necessary.



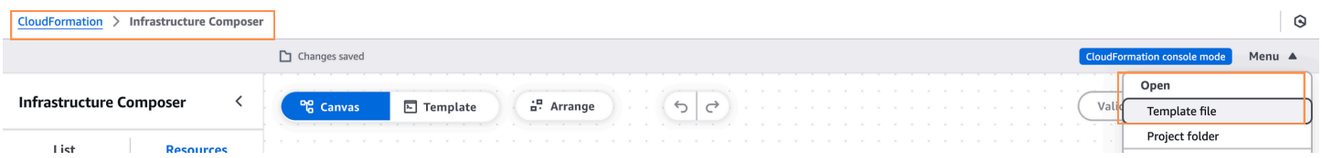

- Go back to the EC2 **application-layer-WebServerSecurityGroup** and verify the inbound rules, now there is an additional rule allowing SSH access on TCP port 22. This step highlights how infrastructure changes can be applied in a consistent and auditable manner. By storing CloudFormation templates in a version-controlled repository like AWS CodeCommit, you can track modifications, maintain version history, and ensure repeatable deployments.

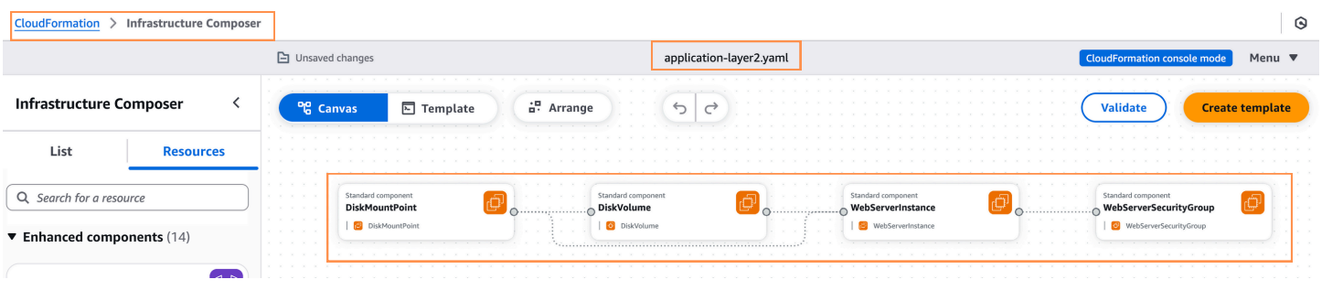


## Section 4: Exploring Templates with Infrastructure Composer

I used Infrastructure Composer (formerly AWS CloudFormation Designer), a visual tool for designing and editing CloudFormation templates. By loading the *application-layer2.yaml* I could visually inspect the relationships between resources, modify configurations through the integrated YAML/JSON editor, and experiment with adding or connecting resources using the drag-and-drop interface. This provides hands-on experience with visualizing and refining IaC.

- Open the local file *application-layer2.yaml* and review the graphical representation of the template.



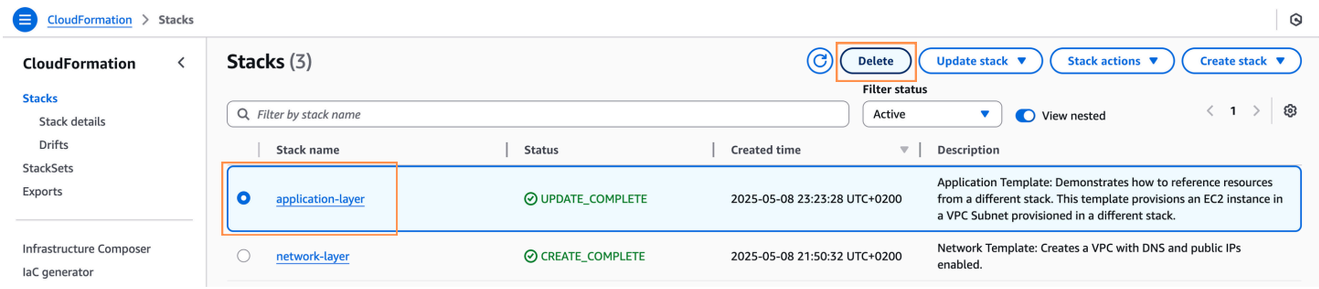

## Section 5: Stack deletion and Snapshot retention via Deletion Policy

I deleted the *application-layer* stack using AWS CloudFormation. I configured a Deletion Policy on the EBS volume to automatically create a snapshot before removal. This ensures critical data is preserved even after the stack is deleted. Upon deletion, the EBS snapshot was successfully created and verified in the EC2 console, while the *network-layer* stack remained intact, demonstrating how independent stacks can be managed separately by different teams.

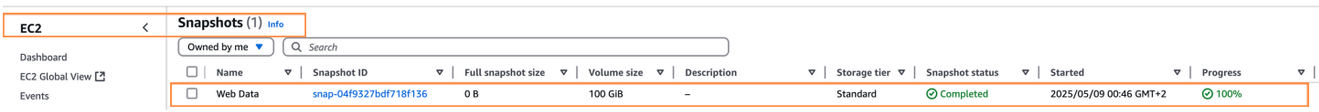
- The *application-layer* stack was set up to automatically create a snapshot of an Amazon Elastic Block Store (EBS) volume before it gets deleted. This behavior is defined in the CloudFormation template with the following configuration:

```
87    DiskVolume:
88      Type: AWS::EC2::Volume
89      Properties:
90        Size: 100
91        AvailabilityZone: !GetAtt WebServerInstance.AvailabilityZone
92        Tags:
93          - Key: Name
94            Value: Web Data
95      DeletionPolicy: Snapshot
```

- Delete the *application-layer* stack.




- Verifying the Snapshot was created.




# Conclusions & Lessons Learned

**Infrastructure as Code Enhances Repeatability and Scalability**
Using AWS CloudFormation to deploy infrastructure demonstrated the power of defining environments as code. By separating the networking and application layers into distinct stacks, I followed best practices for modular design and scalable infrastructure.

**Stack Referencing Enables Cross-Layer Integration**
The application-layer stack imported values from the network-layer stack to deploy dependent resources like the EC2 instance and security group. This illustrated how stack outputs and import values enable decoupled stacks to work together seamlessly.

**CloudFormation Supports Safe and Controlled Updates**
Modifying the security group to allow SSH access through a stack update reinforced how CloudFormation manages infrastructure changes without disrupting existing resources. The Change Set preview offered clear visibility into what would be updated.

**Automation with Safeguards: Deletion Policies Preserve Critical Data**
By adding a DeletionPolicy to the EBS volume, the stack ensured a snapshot was created before deleting the resource. This demonstrated how CloudFormation can automate not just deployment but also safe teardown, preserving important data automatically.

## Final Thoughts

This project provided hands-on experience in automating AWS infrastructure using CloudFormation. It reinforced key DevOps principles such as Infrastructure as Code (IaC), modular stack design, secure network setup, and safe resource updates. Leveraging tools like **Infrastructure Composer** for template visualization also helped in understanding resource relationships at a glance.

Overall, the project laid a strong foundation for managing cloud environments in a structured, maintainable, and secure way. It prepares me for more advanced concepts such as CI/CD integration.