



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 **Infrastructure 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

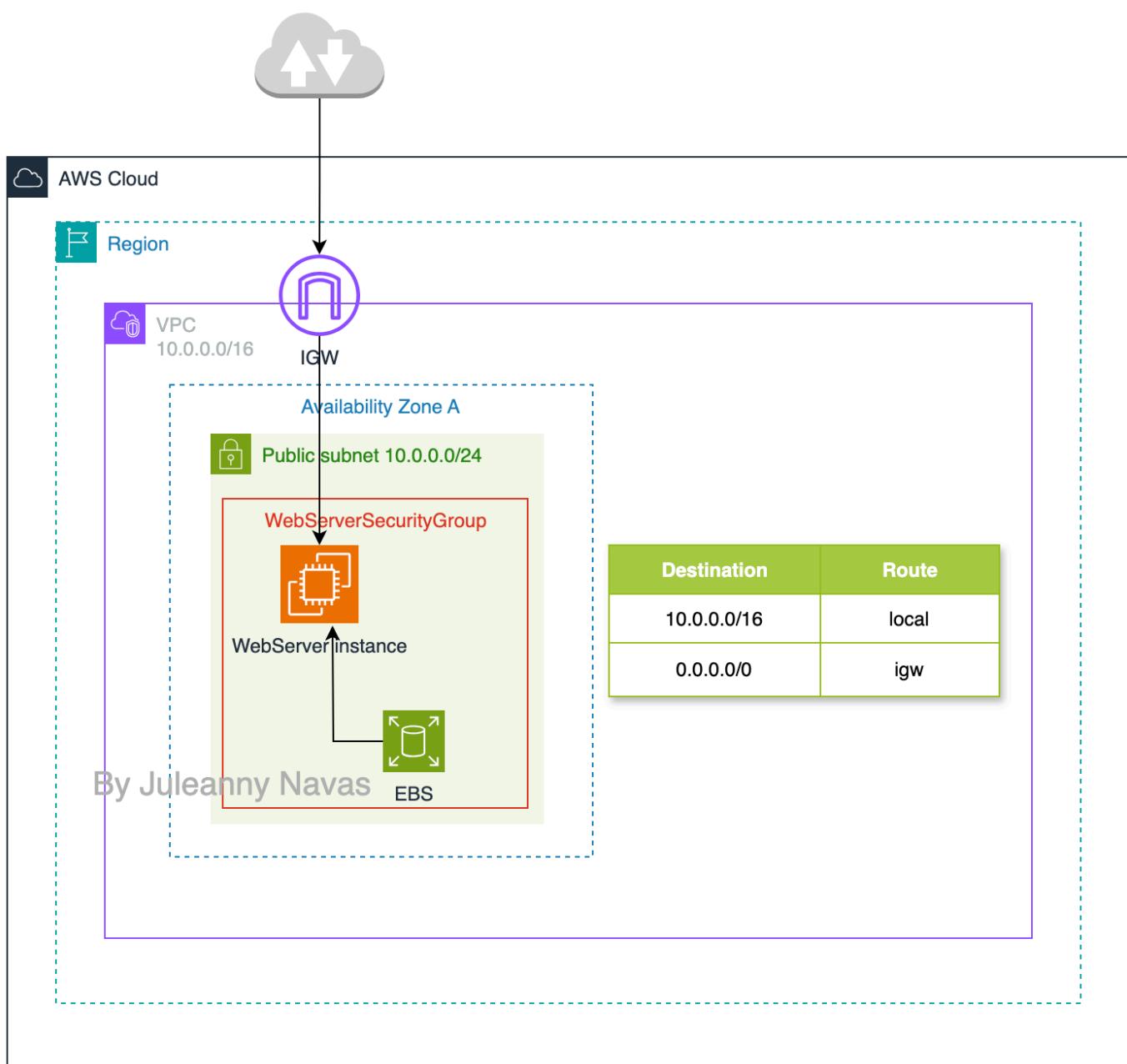
Section 5: Stack Deletion and Snapshot retention via Deletion Policy

- 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**.

The screenshot shows the AWS CloudFormation console with the 'Create stack' wizard open. On the left, the navigation sidebar includes 'CloudFormation', 'Stacks', 'StackSets', 'Exports', 'Infrastructure Composer', 'IaC generator', 'Hooks overview', 'Hooks', 'Registry', 'Public extensions', 'Activated extensions', 'Publisher', 'Spotlight', and 'Feedback'. The main area is titled 'Create stack' under 'Step 1: Prerequisite – Prepare template'. It contains a 'Prerequisite – Prepare template' section with a note about creating a template by scanning existing resources or using the IaC generator. Below this is a 'Prepare template' section where 'Choose an existing template' is selected. Other options include 'Build from Infrastructure Composer' and 'Sync from Git'. A 'Specify template' section follows, showing 'Upload a template file' with a 'Choose file' button and a text input field containing 'network-layer.yaml'. A note states this is a JSON or YAML formatted file. At the bottom, an S3 URL is provided: <https://s3.us-east-1.amazonaws.com/cf-templates-dbiit5axjnur-us-east-1/2025-05-08T193703.558Zlta-network-layer.yaml>, with a 'View in Infrastructure Composer' link. Navigation buttons at the bottom right include 'Cancel', 'Next', and a 'Previous' button.

The screenshot shows the 'Specify stack details' step of the 'Create stack' wizard. The left sidebar remains the same. The main area is titled 'Specify stack details' under 'Step 2'. It has two sections: 'Provide a stack name' and 'Parameters'. In 'Provide a stack name', the 'Stack name' field is filled with 'network-layer'. A note specifies that stack names must contain letters (a-z, A-Z), numbers (0-9), and hyphens (-) and start with a letter. Character count is 13/128. The 'Parameters' section notes that parameters are defined in the template and allows input of custom values. It currently shows 'No parameters' and 'There are no parameters defined in your template'. Navigation buttons at the bottom right include 'Cancel', 'Previous', and a 'Next' button.

CloudFormation > Stacks > Create stack

CloudFormation

- Stacks
- StackSets
- Exports
- Infrastructure Composer
- IaC generator
- Hooks overview

Step 1 Create stack
Step 2 Specify stack details
Step 3 Configure stack options
Step 4 Review and create

Configure stack options

Tags - optional
Tags (key-value pairs) are used to apply metadata to AWS resources, which can help in organising, identifying and categorising those resources. You can add up to 50 unique tags for each stack.

Key	Value - Tags - optional
application	inventory

Add new tag

You can add 49 more tag(s)

Cancel Previous Next

CloudFormation > Stacks > Create stack

CloudFormation

- Stacks
- StackSets
- Exports
- Infrastructure Composer
- IaC generator
- Hooks overview

Step 1 Create stack
Step 2 Specify stack details
Step 3 Configure stack options
Step 4 Review and create

Review and create

Step 1: Specify template

Prerequisite – Prepare template

Template
Template is ready

Create changeset

Cancel Previous Submit

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

CloudFormation > Stacks > network-layer

CloudFormation

- Stacks
- Stack details**
- Drifts
- StackSets
- Exports
- Infrastructure Composer
- IaC generator
- Hooks overview

Stacks (2)

Filter status Active

View nested

Stacks

network-layer
2025-05-08 21:50:32 UTC+0200
CREATE_COMPLETE

c58886a104595610235425t1w53915
2199992

network-layer

Stack info Events Resources Outputs Parameters Template Changesets Git sync

Overview

Stack ID arn:aws:cloudformation:us-east-1:539152199992:stack/network-layer/b34fea90-2c45-11f0-ab32-0affd52ea213
Description Network Template: Creates a VPC with DNS and public IPs enabled.

Status CREATE_COMPLETE

Detailed status -

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

CloudFormation > Stacks > network-layer

CloudFormation

- Stacks
- Stack details**
- Drifts
- StackSets
- Exports
- Infrastructure Composer
- IaC generator
- Hooks overview
- Hooks
- Registry
- Public extensions
- Activated extensions
- Publisher
- Spotlight
- Feedback

Stacks (2)

Filter status Active

View nested

Stacks

network-layer
2025-05-08 21:50:32 UTC+0200
CREATE_COMPLETE

c58886a104595610235425t1w53915
2199992

network-layer

Stack info Events Resources Outputs Parameters Template Changesets Git sync

Resources (8)

Logical ID	Physical ID	Type	Status
InternetGateway	igw-0b57bac22b302f248	AWS::EC2::InternetGateway	CREATE_COMPLETE
PublicRoute	rtb-06d5d6873001d31bcj0.0.0/0	AWS::EC2::Route	CREATE_COMPLETE
PublicRouteTable	rtb-06d5d6873001d31bc	AWS::EC2::RouteTable	CREATE_COMPLETE
PublicSubnet	subnet-095ba30dc8cd1a374	AWS::EC2::Subnet	CREATE_COMPLETE
PublicSubnetNetworkAclAssociation	aclassoc-0744cd3f43f2fa163	AWS::EC2::SubnetNetworkAclAssociation	CREATE_COMPLETE
PublicSubnetRouteTableAssociation	rtbassoc-04b1922606371c34f	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE
VPC	vpc-0b06773a93e35c64e	AWS::EC2::VPC	CREATE_COMPLETE
VPCGatewayAttachment	IGWvpc-0b06773a93e35c64e	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE

- 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.

Key	Value	Description	Export name
PublicSubnet	subnet-0a9ba30dc8cd1a374	The subnet ID to use for public web servers	network-layer-SubnetID
VPC	vpc-0b06773a93e35c64e	VPC ID	network-layer-VPCID

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

```

AWSTemplateFormatVersion: 2010-09-09
Description: >-
  Network Template: Creates a VPC with DNS and public IPs enabled.

# This template creates:
#   VPC
#   Internet Gateway
#   Public Route Table
#   Public Subnet

#####
# Resources section
#####

Resources:

## VPC

VPC:
  Type: AWS::EC2::VPC
  Properties:
    EnableDnsSupport: true
    EnableDnsHostnames: true
    CidrBlock: 10.0.0.0/16

## Internet Gateway
  
```

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 network-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 *network-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.

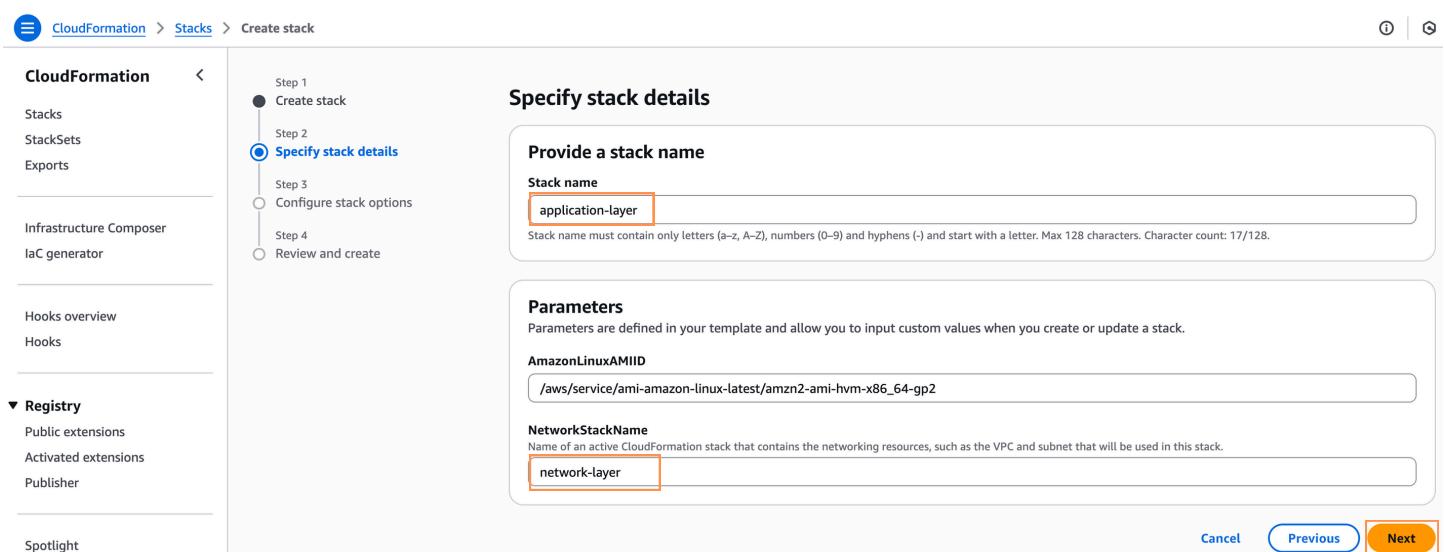
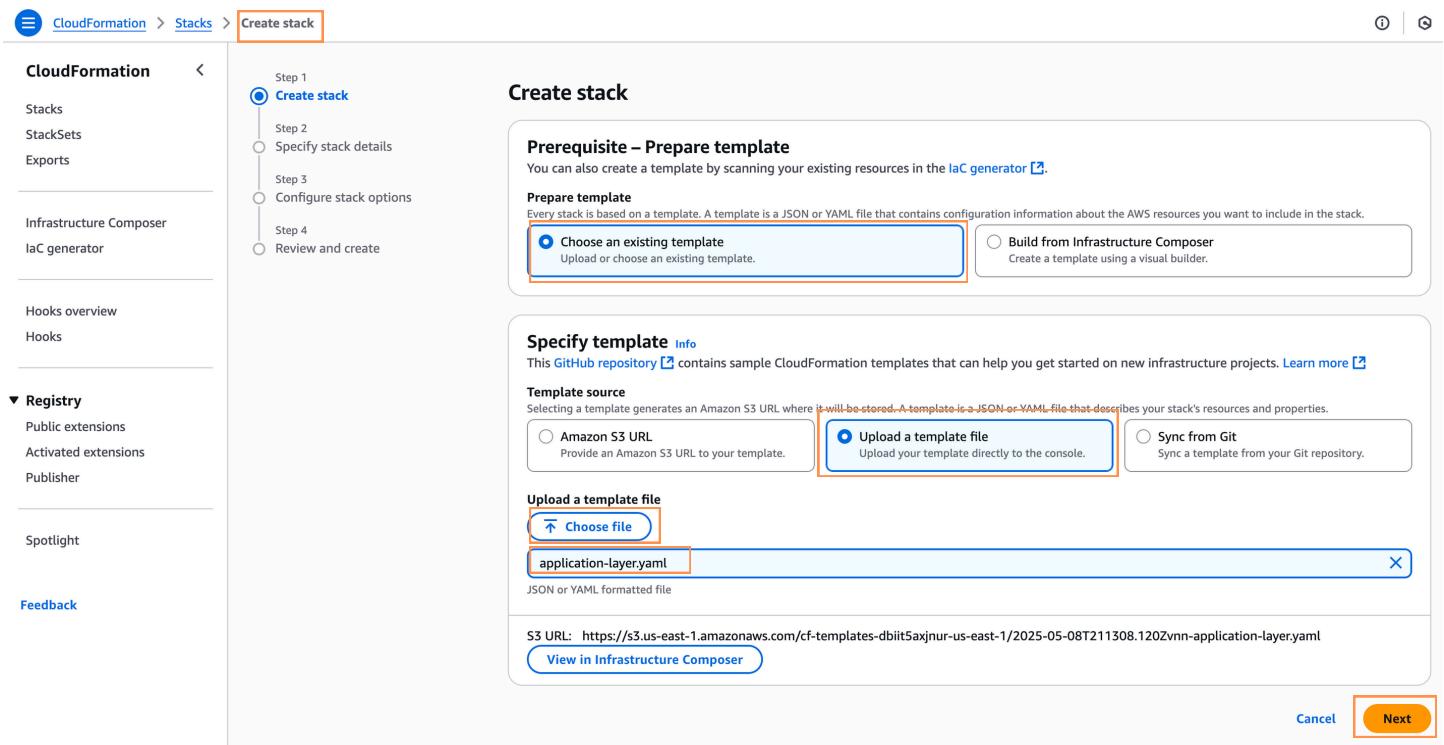
```

104  WebServerSecurityGroup:
105    Type: AWS::EC2::SecurityGroup
106    Properties:
107      GroupDescription: Enable HTTP ingress
108      VpcId:
109        Fn::ImportValue:
110          !Sub ${NetworkStackName}-VPCID
111      SecurityGroupIngress:
112        - IpProtocol: tcp
113          FromPort: 80
114          ToPort: 80
115          CidrIp: 0.0.0.0/0
116      Tags:
117        - Key: Name
118          Value: Web Server Security Group
119

```

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*.

CloudFormation > Stacks > Create stack

CloudFormation

- Stacks
- StackSets
- Exports
- Infrastructure Composer
- IaC generator
- Hooks overview

Step 1 Create stack
Step 2 Specify stack details
Step 3 Configure stack options
Step 4 Review and create

Configure stack options

Tags - optional
Tags (key-value pairs) are used to apply metadata to AWS resources, which can help in organising, identifying and categorising those resources. You can add up to 50 unique tags for each stack.

Key	Value - Tags - optional
application	inventory

Add new tag
You can add 49 more tag(s)

Cancel Previous Next

CloudFormation > Stacks > Create stack

CloudFormation

- Stacks
- StackSets
- Exports
- Infrastructure Composer
- IaC generator
- Hooks overview

Step 1 Create stack
Step 2 Specify stack details
Step 3 Configure stack options
Step 4 Review and create

Review and create

Step 1: Specify template

Prerequisite – Prepare template

Template
Template is ready

Create changeset
Cancel Previous Submit

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

CloudFormation > Stacks > application-layer

CloudFormation

- Stacks
- Stack details**
- Drifts
- StackSets
- Exports
- Infrastructure Composer
- IaC generator
- Hooks overview

Stacks (3)

Stack ID	Created	Status
arnaws:cloudformation:us-east-1:539152199992:stack/application-layer/aef8d990-2c52-11f0-9443-12b9fbfd9f6d	2025-05-08 23:23:28 UTC+0200	CREATE_COMPLETE
c58886a104595610235425t1w539152199992	2025-05-08 20:53:28 UTC+0200	CREATE_COMPLETE
network-layer		

application-layer

Stack info Events Resources Outputs Parameters Template Changesets Git sync

Overview

Stack ID
arnaws:cloudformation:us-east-1:539152199992:stack/application-layer/aef8d990-2c52-11f0-9443-12b9fbfd9f6d

Description
Application Template: Demonstrates how to reference resources from a different stack. This template provisions an EC2 instance in a VPC Subnet provisioned in a different stack.

Status
CREATE_COMPLETE

Detailed status-

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

CloudFormation > Stacks > application-layer

CloudFormation

- Stacks
- Stack details**
- Drifts
- StackSets
- Exports
- Infrastructure Composer
- IaC generator
- Hooks overview
- Hooks
- Registry
- Public extensions
- Activated extensions

Stacks (3)

Stack ID	Created	Status
arnaws:cloudformation:us-east-1:539152199992:stack/application-layer/aef8d990-2c52-11f0-9443-12b9fbfd9f6d	2025-05-08 23:23:28 UTC+0200	CREATE_COMPLETE
c58886a104595610235425t1w539152199992	2025-05-08 20:53:28 UTC+0200	CREATE_COMPLETE
network-layer		

application-layer

Stack info Events **Resources** Outputs Parameters Template Changesets Git sync

Resources (4)

Logical ID	Physical ID	Type	Status	Module
DiskMountPoint	vol-0e1270aad883af2d9j-0ceb1c6bfdfa446f	AWS::EC2::VolumeAttachment	CREATE_COMPLETE	-
DiskVolume	vol-0e1270aad883af2d9j	AWS::EC2::Volume	CREATE_COMPLETE	-
WebServerInstance	i-0ceb1c6bfdfa446f	AWS::EC2::Instance	CREATE_COMPLETE	-
WebServerSecurityGroup	sg-0f305ea91eb25ea04	AWS::EC2::SecurityGroup	CREATE_COMPLETE	-

- 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.

The screenshot shows the AWS CloudFormation console with the 'application-layer' stack selected. The 'Outputs' tab is active, displaying one output: 'URL' with the value 'http://ec2-44-195-33-13.compute-1.amazonaws.com'. A red box highlights this row. The 'Stack info', 'Events', 'Resources', 'Parameters', 'Template', 'Changesets', and 'Git sync' tabs are also visible at the top.

The screenshot shows a web browser window with the URL 'http://ec2-44-195-33-13.compute-1.amazonaws.com' in the address bar. The page content includes a heading 'Introducing AWS CloudFormation' and a diagram illustrating various AWS services like Amazon EC2, Amazon S3, and Amazon CloudWatch.

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**

The screenshot shows the AWS EC2 Security Groups console for the security group 'sg-0f305ea91eb25ea04'. The 'Inbound rules' tab is selected, showing one rule: 'sgr-0c25f47a6ba8b97dd' (Protocol: TCP, Port range: 80, Source: 0.0.0.0/0). A red box highlights this row. Other tabs include 'Outbound rules', 'Sharing - new', 'VPC associations - new', and 'Tags'.

- 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.

```

104  | WebServerSecurityGroup:
105  |   Type: AWS::EC2::SecurityGroup
106  |   Properties:
107  |     GroupDescription: Enable HTTP ingress
108  |     VpcId:
109  |       Fn::ImportValue:
110  |         !Sub ${NetworkStackName}-VPCID
111  |     SecurityGroupIngress:
112  |       - IpProtocol: tcp
113  |         FromPort: 80
114  |         ToPort: 80
115  |         CidrIp: 0.0.0.0/0
116  |       - IpProtocol: tcp
117  |         FromPort: 22
118  |         ToPort: 22
119  |         CidrIp: 0.0.0.0/0
120  |     Tags:
121  |       - Key: Name
122  |         Value: Web Server Security Group
123

```

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

The screenshot shows the AWS CloudFormation console interface. On the left, there's a navigation sidebar with options like 'CloudFormation', 'Stacks', 'Drifts', 'StackSets', 'Exports', 'Infrastructure Composer', 'IaC generator', and 'Hooks overview'. The main area shows a list of stacks under 'Stacks (3)'. One stack, 'application-layer', is selected and highlighted with a blue box. Its status is 'CREATE_COMPLETE'. On the right, the 'application-layer' stack details page is displayed. At the top, there are buttons for 'Delete', 'Update stack', 'Stack actions', 'Create stack', 'Create a change set', 'Make a direct update' (which is also highlighted with a red box), and 'Changesets' and 'Git sync'. Below this, the 'Overview' section shows the 'Stack ID' (arn:aws:cloudformation:us-east-1:53915219992:stack/application-layer/aef8d990-2c52-11f0-9a43-12b9fbfd9f6d) and the 'Status' (CREATE_COMPLETE). A 'Description' field contains a note about referencing resources from a different stack.

This screenshot shows the 'Update stack' wizard in the AWS CloudFormation console, specifically Step 1: 'Prerequisite – Prepare template'. It includes a sidebar with steps: 'Step 1 Update stack', 'Step 2 Specify stack details', 'Step 3 Configure stack options', and 'Step 4 Review application-layer'. The main content area has sections for 'Prerequisite – Prepare template' (with a note about importing templates via IaC generator) and 'Specify template'. Under 'Specify template', it says 'This GitHub repository' contains sample CloudFormation templates. It shows 'Template source' options: 'Amazon S3 URL' (unchecked) and 'Upload a template file' (checked, highlighted with a red box). Below this is a file input field containing 'application-layer2.yaml'. At the bottom, it shows the S3 URL: 'https://s3.us-east-1.amazonaws.com/cf-templates-dbiit5axjnur-us-east-1/2025-05-08T215813.9352gb8-application-layer2.yaml' and a 'View in Infrastructure Composer' button. Navigation buttons 'Cancel' and 'Next' are at the bottom right.

- 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.

Changes (1)

Action	Logical ID	Physical ID	Resource type	Replacement
Modify	WebServerSecurityGroup	sg-Of305ea91eb25ea0...	AWS::EC2::SecurityGroup	False

View changeset **Cancel** **Previous** **Submit**

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

CloudFormation < **Stacks (3)** **application-layer**

Stack ID: arn:aws:cloudformation:us-east-1:53915219992:stack/application-layer/aef8d990-2c52-11f0-9a43-12b9fbfd9f6d **Description:** Application Template: Demonstrates how to reference resources from a different stack. This template provisions an EC2 instance in a VPC Subnet provisioned in a different stack.

Status: **UPDATE_COMPLETE** **Detailed status:** -

- 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.

sg-Of305ea91eb25ea04 - application-layer-WebServerSecurityGroup-uyUcRTyp1nTW

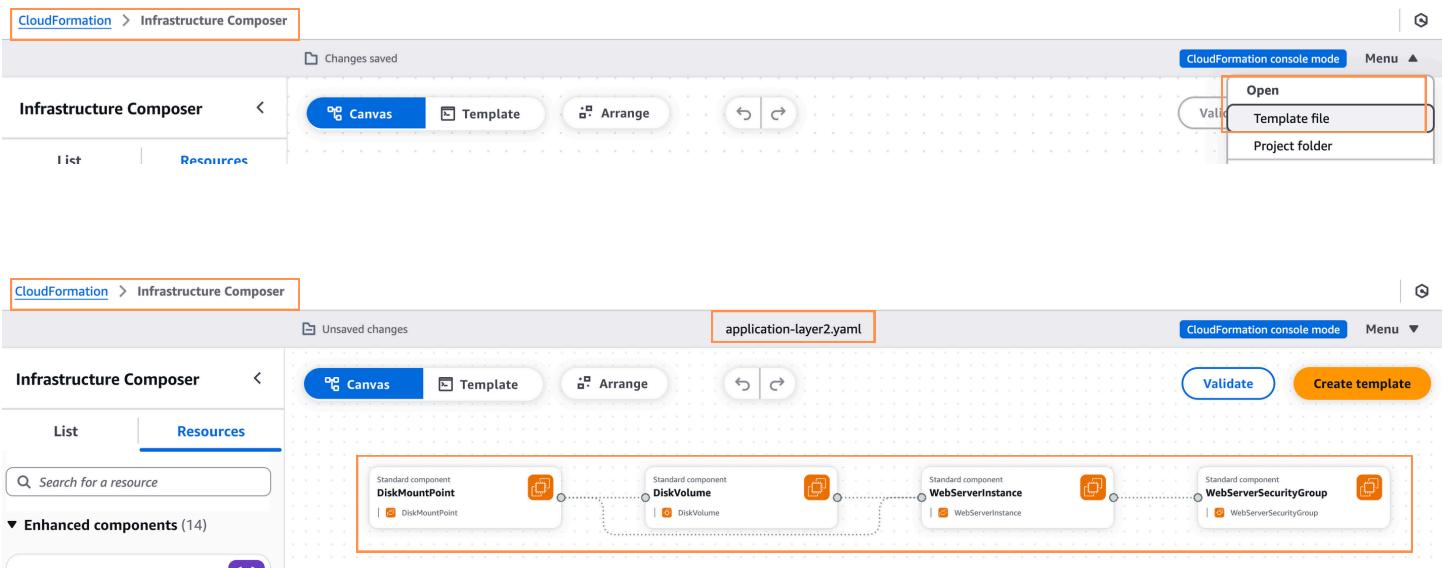
Inbound rules (2)

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
-	sgr-01d5c94299b9e6b14	IPv4	SSH	TCP	22	0.0.0.0/0
-	sgr-0c25f47a6ba8b97dd	IPv4	HTTP	TCP	80	0.0.0.0/0

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.

The screenshot shows the AWS CloudFormation Stacks interface. On the left, there's a sidebar with 'CloudFormation' selected under 'Stacks'. The main area is titled 'Stacks (3)' and lists two stacks: 'application-layer' and 'network-layer'. The 'application-layer' stack is highlighted with a blue border. It has a status of 'UPDATE_COMPLETE', was created on '2025-05-08 23:23:28 UTC+0200', and has a detailed description: 'Application Template: Demonstrates how to reference resources from a different stack. This template provisions an EC2 instance in a VPC Subnet provisioned in a different stack.' The 'network-layer' stack has a status of 'CREATE_COMPLETE', was created on '2025-05-08 21:50:32 UTC+0200', and its description is 'Network Template: Creates a VPC with DNS and public IPs enabled.' At the top right, there are buttons for 'Delete', 'Update stack', 'Stack actions', and 'Create stack'.

- Verifying the Snapshot was created.

The screenshot shows the AWS EC2 Snapshots interface. On the left, there's a sidebar with 'EC2' selected. The main area is titled 'Snapshots (1) Info' and shows a single snapshot entry. The snapshot is named 'snap-04f9327bdf718f136', has a size of '0 B', and is associated with a volume of '100 GiB'. It is listed as 'Standard' storage tier, 'Completed' status, and started at '2025/05/09 00:46 GMT+2' with a progress of '100%'. A search bar and filter options are also visible at the top.

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.