

Deploy a High-Availability Web App using CloudFormation

PROJECT SPECIFICATION:

Project Introduction

As your final project, you'll be faced with a real scenario.

Creating this project will give you the hands-on experience you need to confidently talk about infrastructure as code. So, for that reason, we have chosen a realistic scenario where you deploy an application (Apache Web Server) and you also pick up code (JavaScript and HTML) from S3 Storage and deploy it in the appropriate folder on the web server.

There will be two parts to this project:

You'll first develop a diagram that you can present as part of your portfolio and as a visual aid to understand the CloudFormation script. The second part is to interpret the instructions as well as your own diagram and create a matching CloudFormation script.

Problem

Your company is creating an Instagram clone called Udagram. Developers pushed the latest version of their code in a zip file located in a public S3 Bucket.

You have been tasked with deploying the application, along with the necessary supporting software into its matching infrastructure.

This needs to be done in an automated fashion so that the infrastructure can be discarded as soon as the testing team finishes their tests and gathers their results.

Project Requirements

Server spec

You'll need to create a Launch Configuration for your application servers in order to deploy four servers, two located in each of your private subnets. The launch configuration will be used by an auto-scaling group.

You'll need two vCPUs and at least 4GB of RAM. The Operating System to be used is Ubuntu 18. So, choose an Instance size and Machine Image (AMI) that best fits this spec. Be sure to allocate at least 10GB of disk space so that you don't run into issues.

Security Groups and Roles

Since you will be downloading the application archive from an S3 Bucket, you'll need to create an IAM Role that allows your instances to use the S3 Service.

Udagram communicates on the default HTTP Port: 80, so your servers will need this inbound port open since you will use it with the Load Balancer and the Load Balancer Health Check. As for outbound, the servers will need unrestricted internet access to be able to download and update its software.

The load balancer should allow all public traffic (0.0.0.0/0) on port 80 inbound, which is the default HTTP port. Outbound, it will only be using port 80 to reach the internal servers.

The application needs to be deployed into private subnets with a Load Balancer located in a public subnet.

One of the output exports of the CloudFormation script should be the public URL of the LoadBalancer.

Bonus points if you add `http://` in front of the load balancer DNS Name in the output, for convenience.

Other Considerations

You can deploy your servers with an SSH Key into Public subnets while you are creating the script. This helps with troubleshooting. Once done, move them to your private subnets and remove the SSH Key from your Launch Configuration.

It also helps to test directly, without the load balancer. Once you are confident that your server is behaving correctly, increase the instance count and add the load balancer to your script.

While your instances are in public subnets, you'll also need the SSH port open (port 22) for your access, in case you need to troubleshoot your instances.

Log information for UserData scripts is located in this file: `cloud-init-output.log` under the folder: `/var/log`.

You should be able to destroy the entire infrastructure and build it back up without any manual steps required, other than running the CloudFormation script.

The provided UserData script should help you install all the required dependencies. Bear in mind that this process takes several minutes to complete. Also, the application takes a few seconds to load. This information is crucial for the settings of your load balancer health check.

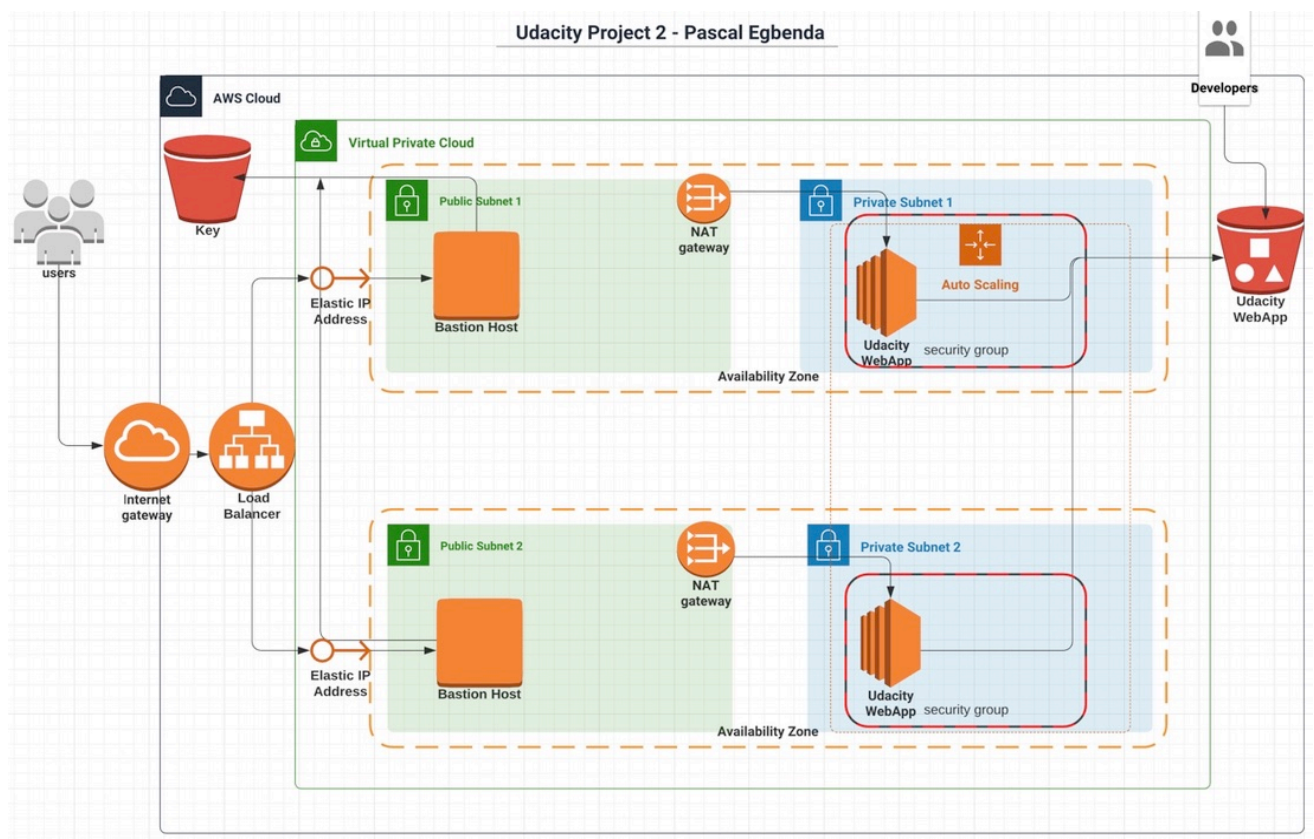
It's up to you to decide which values should be parameters and which you will hard-code in your script.

See the provided supporting code for help and more clues.

If you want to go the extra mile, set up a bastion host to allow you to SSH into your private subnet servers. This bastion host would be on a Public Subnet with port 22 open only to your home IP address, and it would need to have the private key that you use to access the other servers.

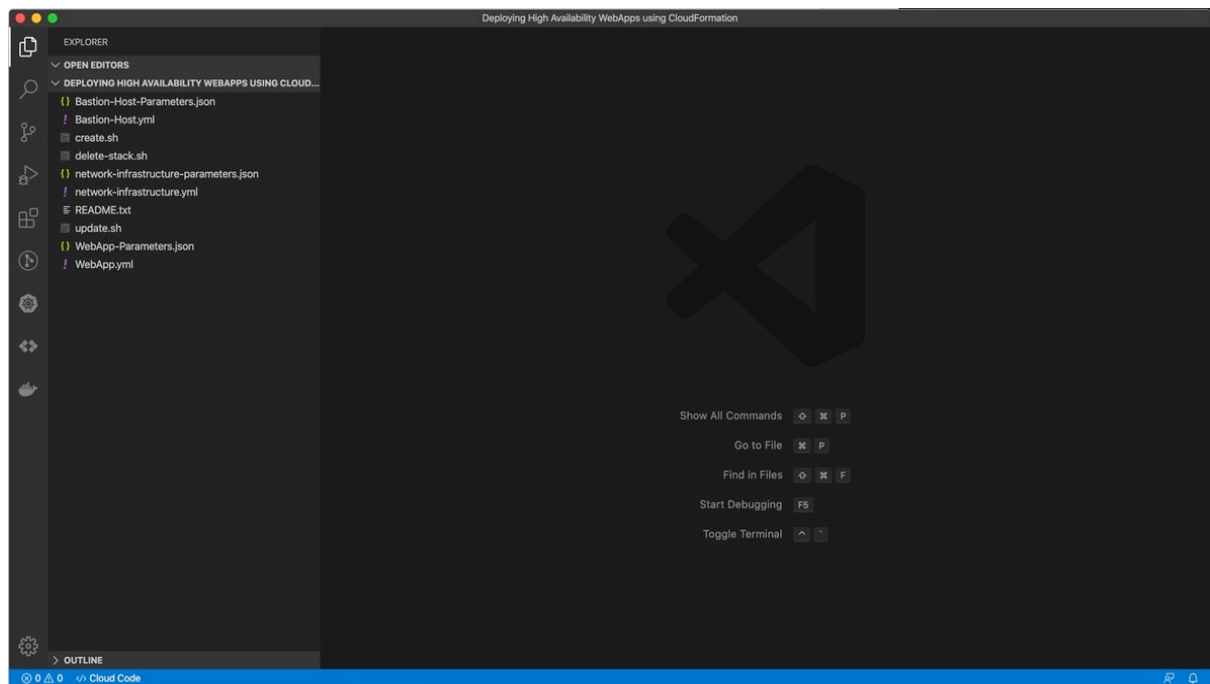
Last thing: Remember to delete your CloudFormation stack when you're done to avoid recurring charges!

Solution Diagram



CloudFormation Templates

The CloudFormation templates for the projects were developed in VSCode as shown below:



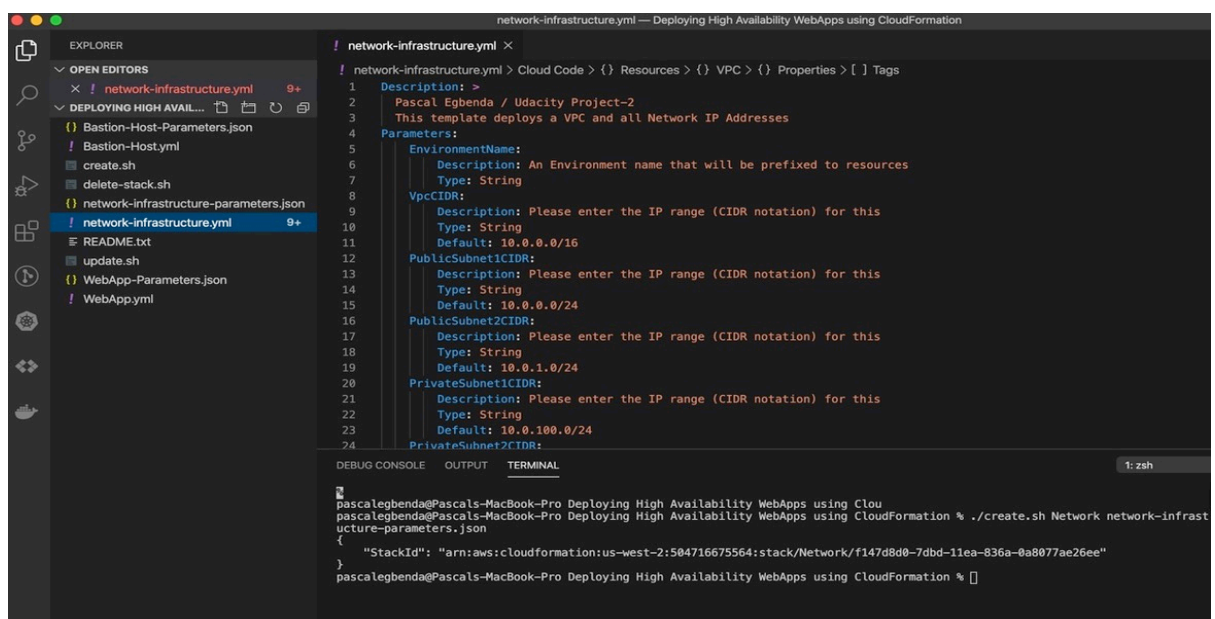
Network Stack

This template deploys a VPC and all Network IP Addresses:

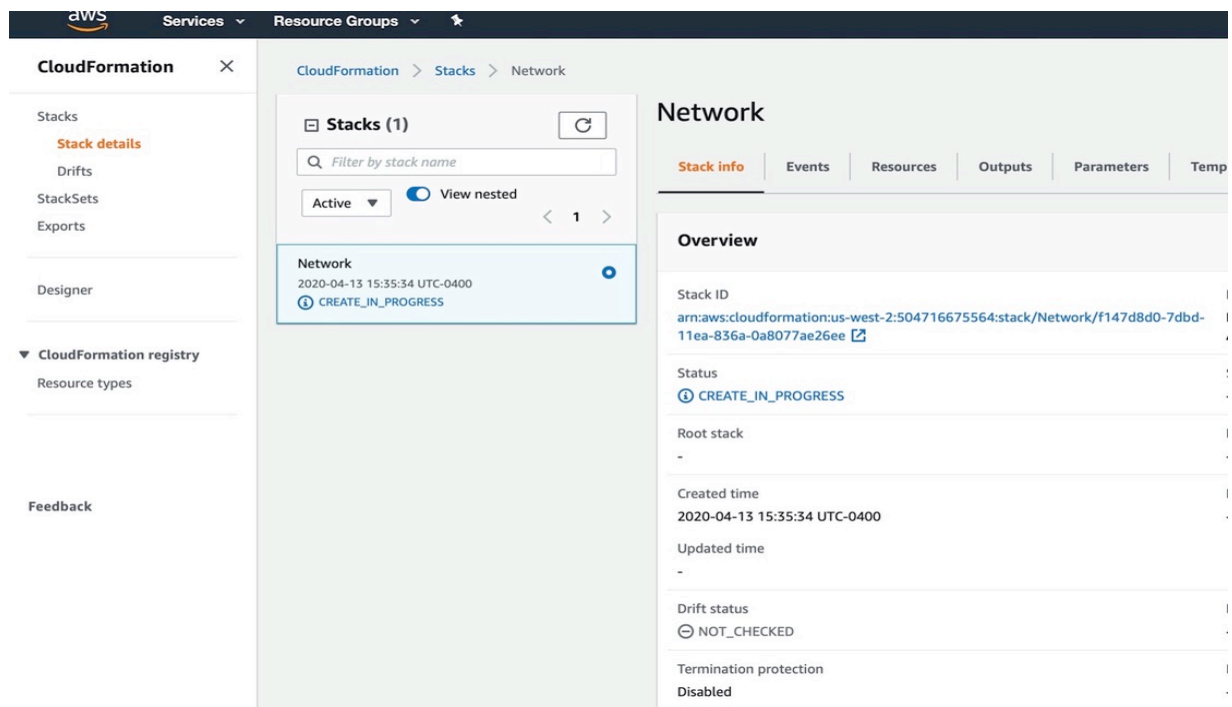
1. This template created the Network Stack:

Command:

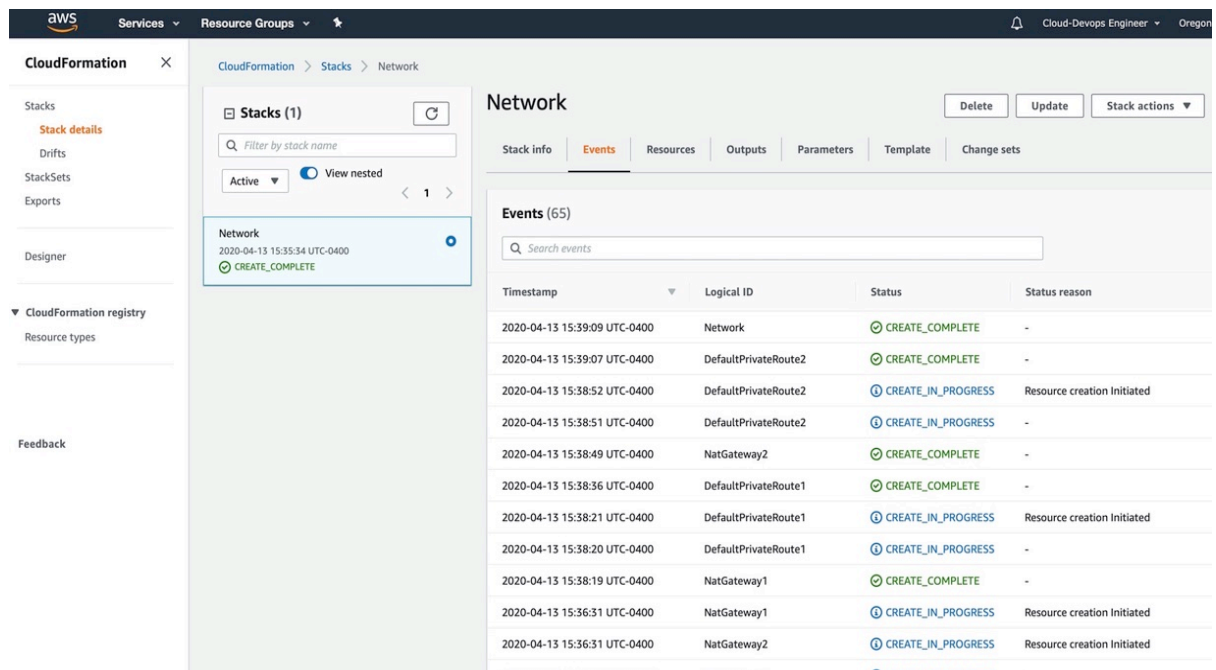
./create.sh Network network-infrastructure.yml network-infrastructure-parameters.json



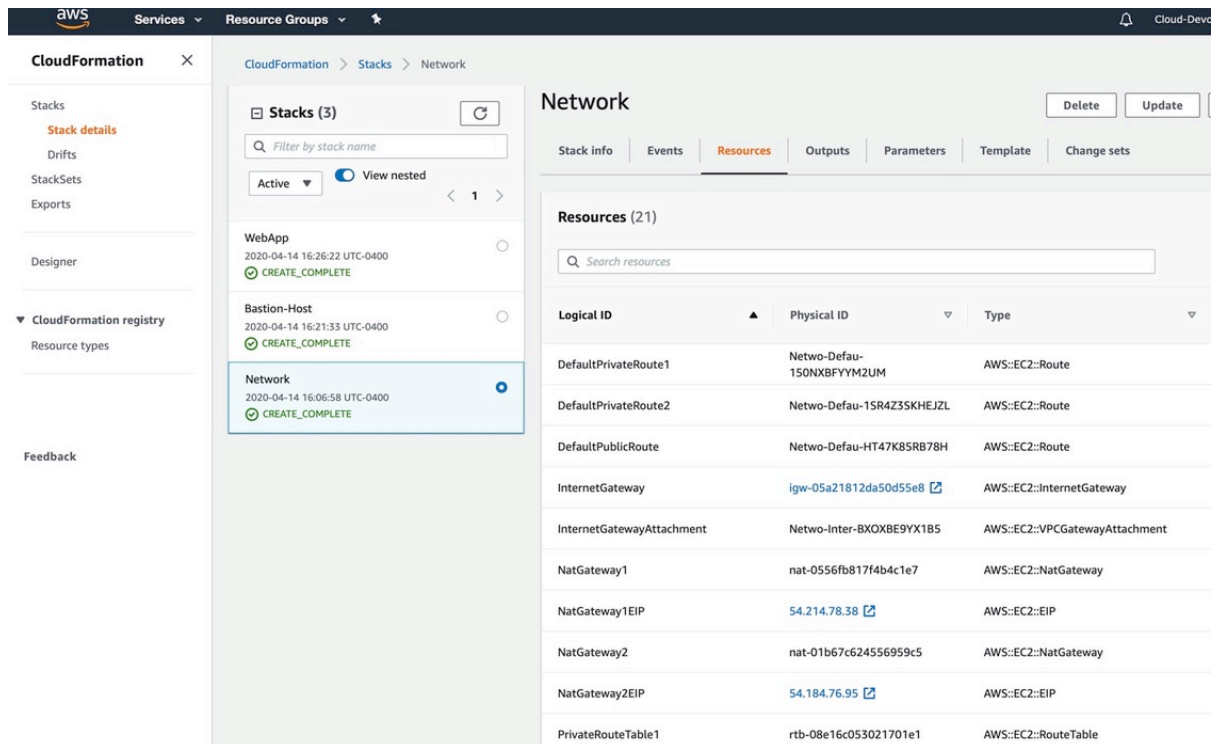
2. From the console we see as below:



3. The completed created Network stack is shown here:



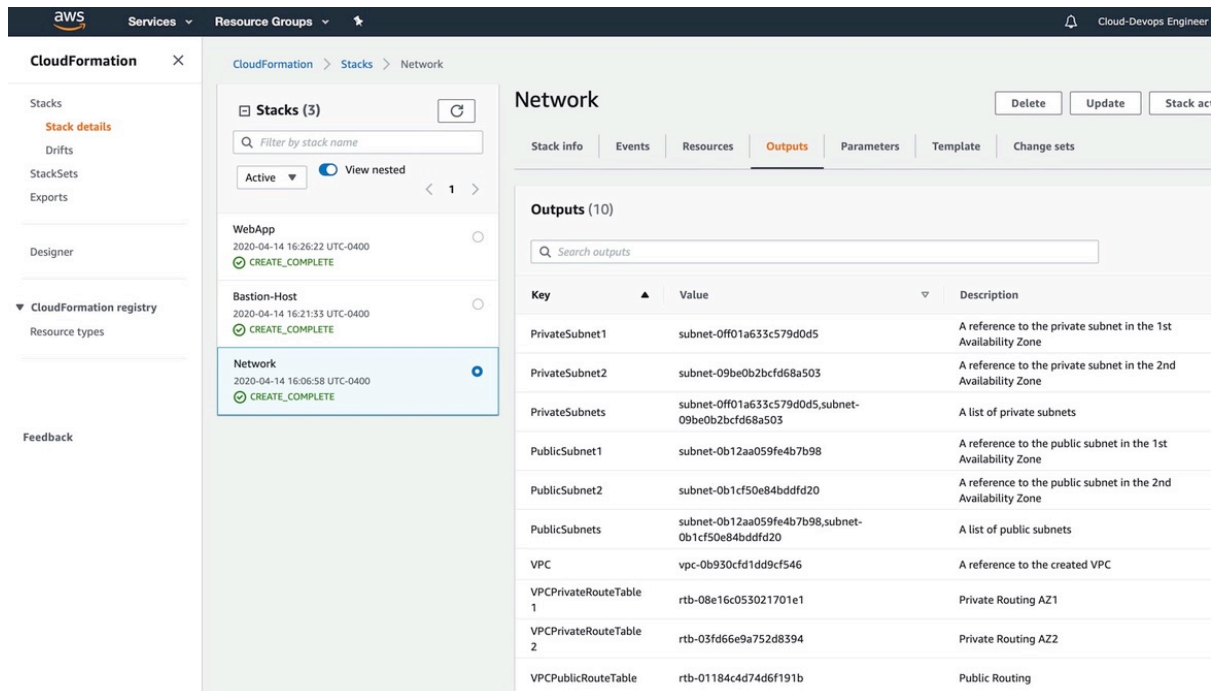
4. The Resources created from the Network Stack:



The screenshot displays the AWS CloudFormation console. On the left, the 'Stacks' section is expanded, showing three stacks: 'WebApp', 'Bastion-Host', and 'Network'. The 'Network' stack is selected. The main panel shows the 'Resources' tab for the 'Network' stack, displaying a list of 21 resources created by the stack.

Logical ID	Physical ID	Type
DefaultPrivateRoute1	Netwo-Defau-150NXBFYYM2UM	AWS::EC2::Route
DefaultPrivateRoute2	Netwo-Defau-15R4Z3SKHEJZL	AWS::EC2::Route
DefaultPublicRoute	Netwo-Defau-HT47K8SRB78H	AWS::EC2::Route
InternetGateway	igw-05a21812da50d55e8	AWS::EC2::InternetGateway
InternetGatewayAttachment	Netwo-Inter-BXOXBE9YX1B5	AWS::EC2::VPCGatewayAttachment
NatGateway1	nat-0556fb817f4b4c1e7	AWS::EC2::NatGateway
NatGateway1EIP	54.214.78.38	AWS::EC2::EIP
NatGateway2	nat-01b67c624556959c5	AWS::EC2::NatGateway
NatGateway2EIP	54.184.76.95	AWS::EC2::EIP
PrivateRouteTable1	rtb-08e16c053021701e1	AWS::EC2::RouteTable

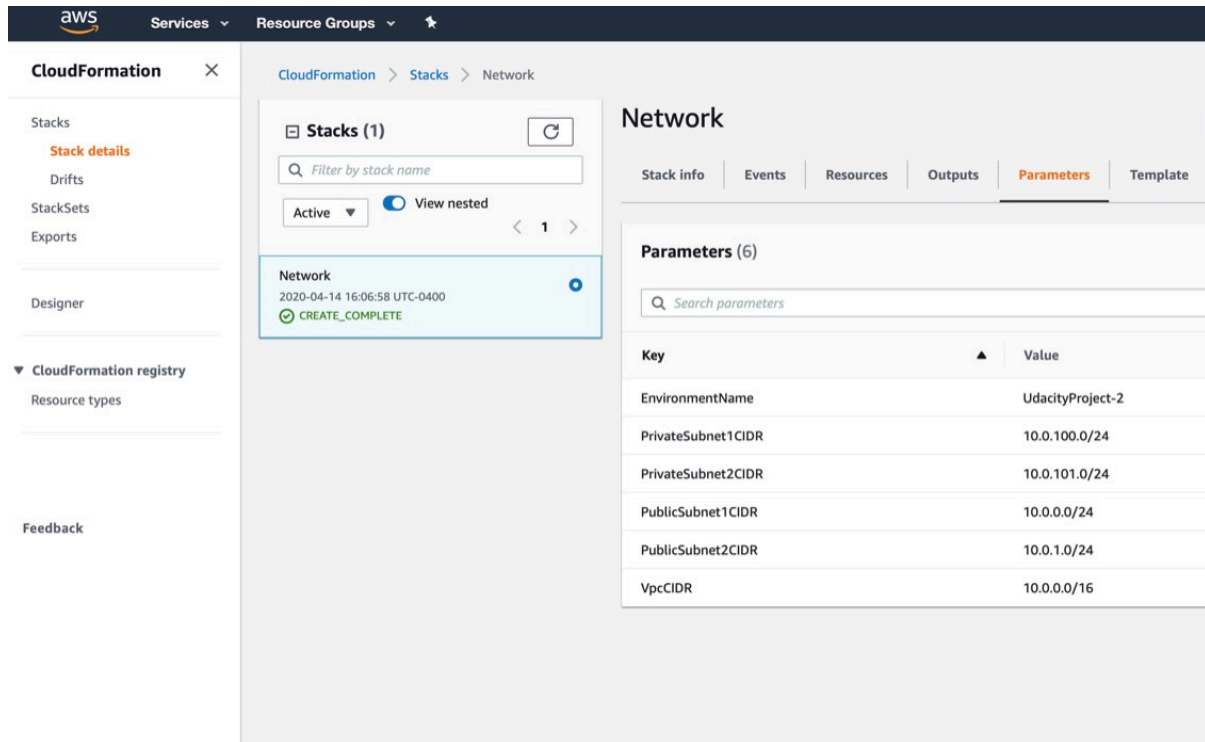
5. The Output of the Network Stack is shown below:



The screenshot displays the AWS CloudFormation console. On the left, the 'Stacks' section is expanded, showing three stacks: 'WebApp', 'Bastion-Host', and 'Network'. The 'Network' stack is selected. The main panel shows the 'Outputs' tab for the 'Network' stack, displaying a list of 10 outputs generated by the stack.

Key	Value	Description
PrivateSubnet1	subnet-0ff01a633c579d0d5	A reference to the private subnet in the 1st Availability Zone
PrivateSubnet2	subnet-09be0b2bcfd68a503	A reference to the private subnet in the 2nd Availability Zone
PrivateSubnets	subnet-0ff01a633c579d0d5,subnet-09be0b2bcfd68a503	A list of private subnets
PublicSubnet1	subnet-0b12aa059fe4b7b98	A reference to the public subnet in the 1st Availability Zone
PublicSubnet2	subnet-0b1cf50e84bddfd20	A reference to the public subnet in the 2nd Availability Zone
PublicSubnets	subnet-0b12aa059fe4b7b98,subnet-0b1cf50e84bddfd20	A list of public subnets
VPC	vpc-0b930cfd1dd9cf546	A reference to the created VPC
VPCPrivateRouteTable 1	rtb-08e16c053021701e1	Private Routing AZ1
VPCPrivateRouteTable 2	rtb-03fd66e9a752d8394	Private Routing AZ2
VPCPublicRouteTable	rtb-01184c4d74d6f191b	Public Routing

6. The Parameters configured with the Network Stack is shown below:

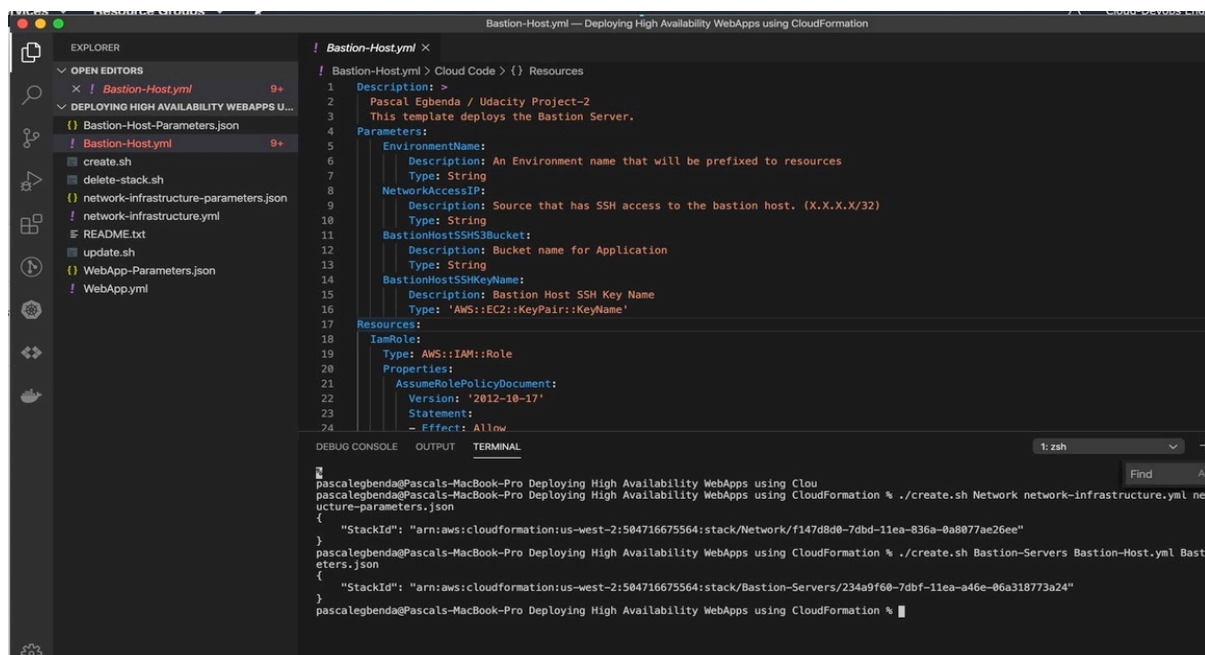


Bastion-Servers Stack

This Bastion-Host template deployed the Bastion-Servers Stack
Command:

./create.sh Bastion-Servers Bastion-Host.yml Bastion-Host-parameters.json

1. The Bastion-Servers stack below was created from the above command:



2. From console I got this:

The screenshot shows the AWS CloudFormation console. On the left, the 'CloudFormation' sidebar is visible with options like 'Stacks', 'Stack details', 'StackSets', 'Exports', 'Designer', 'CloudFormation registry', and 'Feedback'. The main area displays the 'Bastion-Servers' stack. The 'Stacks (2)' list shows two stacks: 'Bastion-Servers' (2020-04-13 15:44:07 UTC-0400) with status 'CREATE_IN_PROGRESS' and 'Network' (2020-04-13 15:35:34 UTC-0400) with status 'CREATE_COMPLETE'. The right pane shows the 'Overview' for the 'Bastion-Servers' stack, including its ID, status, root stack, created time, updated time, drift status, and termination protection.

Bastion-Servers Overview

- Stack ID: `arn:aws:cloudformation:us-west-2:504716675564:stack/Bastion-Servers/234a9f60-7dbf-11ea-a46e-06a318773a24`
- Status: **CREATE_IN_PROGRESS**
- Root stack: -
- Created time: 2020-04-13 15:44:07 UTC-0400
- Updated time: -
- Drift status: **NOT_CHECKED**
- Termination protection: Disabled

3. The completed Bastion-Servers Stack is shown below:

The screenshot shows the AWS CloudFormation console with the 'Bastion-Servers' stack now in a 'completed' state. The 'Stacks (2)' list shows both 'Bastion-Servers' and 'Network' with status 'CREATE_COMPLETE'. The right pane shows the 'Events' tab for the 'Bastion-Servers' stack, displaying a list of 32 events. The events table shows the progression of resource creation, including EIP associations and EC2 instances.

Bastion-Servers Events (32)

Timestamp	Logical ID	Status	Status reason
2020-04-13 15:47:08 UTC-0400	Bastion-Servers	CREATE_COMPLETE	-
2020-04-13 15:47:06 UTC-0400	EIPAssociation2	CREATE_COMPLETE	-
2020-04-13 15:46:59 UTC-0400	EIPAssociation1	CREATE_COMPLETE	-
2020-04-13 15:46:51 UTC-0400	EIPAssociation2	CREATE_IN_PROGRESS	Resource creation Initiated
2020-04-13 15:46:50 UTC-0400	EIPAssociation2	CREATE_IN_PROGRESS	-
2020-04-13 15:46:48 UTC-0400	BastionEC2Instance2	CREATE_COMPLETE	-
2020-04-13 15:46:43 UTC-0400	EIPAssociation1	CREATE_IN_PROGRESS	Resource creation Initiated
2020-04-13 15:46:42 UTC-0400	EIPAssociation1	CREATE_IN_PROGRESS	-
2020-04-13 15:46:40 UTC-0400	BastionEC2Instance1	CREATE_COMPLETE	-
2020-04-13 15:46:33 UTC-0400	BastionEC2Instance1	CREATE_IN_PROGRESS	Resource creation Initiated
2020-04-13 15:46:33 UTC-0400	BastionEC2Instance2	CREATE_IN_PROGRESS	Resource creation Initiated
2020-04-13 15:46:32 UTC-0400	BastionEC2Instance1	CREATE_IN_PROGRESS	-
2020-04-13 15:46:32 UTC-0400	BastionEC2Instance2	CREATE_IN_PROGRESS	-

4. The Bastion-Servers Stack Resources are shown below:

The screenshot shows the AWS CloudFormation console for the 'Bastion-Servers' stack. The left sidebar displays the 'Stacks (2)' list, showing 'Bastion-Servers' and 'Network' both in a 'CREATE_COMPLETE' state. The main panel shows the 'Resources (10)' tab, listing various AWS resources created by the stack.

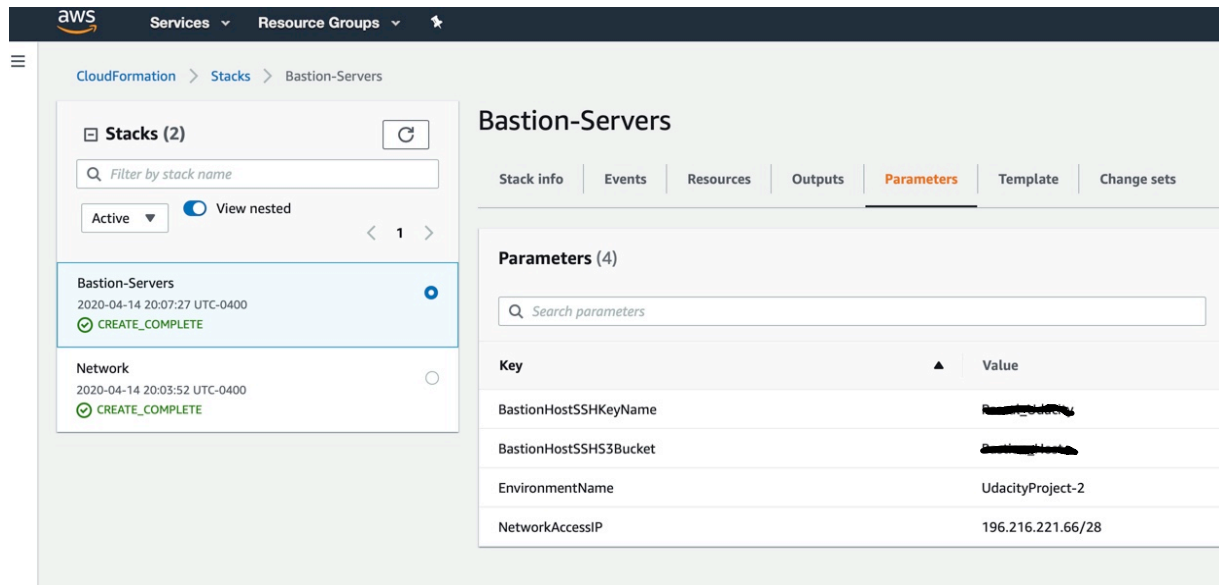
Logical ID	Physical ID	Type	Status	Status reason
BastionEC2Instance1	i-09dd6d7baaf414de4	AWS::EC2::Instance	CREATE_COMPLETE	-
BastionEC2Instance2	i-0135b7bfce986b51	AWS::EC2::Instance	CREATE_COMPLETE	-
BastionSecurityGroup	sg-06a89f3e8c8c25e3	AWS::EC2::SecurityGroup	CREATE_COMPLETE	-
EIP1	54.218.181.43	AWS::EC2::EIP	CREATE_COMPLETE	-
EIP2	44.231.180.231	AWS::EC2::EIP	CREATE_COMPLETE	-
EIPAssociation1	eipassoc-03ab4185583f06171	AWS::EC2::EIPAssociation	CREATE_COMPLETE	-
EIPAssociation2	eipassoc-086944ac9d38d9f69	AWS::EC2::EIPAssociation	CREATE_COMPLETE	-
IamInstanceProfile	Bastion-Servers-IamInstanceProfile-BVQNCZHYISSZ	AWS::IAM::InstanceProfile	CREATE_COMPLETE	-
IamPolicies	Basti-IamP-17SO5JJBQRIPJ	AWS::IAM::Policy	CREATE_COMPLETE	-
IamRole	Bastion-Servers-IamRole-1QXNIDVXF6JR	AWS::IAM::Role	CREATE_COMPLETE	-

5. The Output of the Bastion-Servers Stack is shown below:

The screenshot shows the AWS CloudFormation console for the 'Bastion-Servers' stack, specifically the 'Outputs' tab. The left sidebar shows the 'Stacks (2)' list. The main panel displays the 'Outputs (3)' section, listing three outputs: BastionHost1CIDR, BastionHost2CIDR, and BastionHostSSHKeyName.

Key	Value	Description
BastionHost1CIDR	10.0.0.53/32	A reference to the Bastion Host CIDR
BastionHost2CIDR	10.0.1.170/32	A reference to the Bastion Host CIDR
BastionHostSSHKeyName	Redacted	A reference to the Bastion Host SSH Key Name

6. The Parameter from the Bastion-Host Stack is shown below:

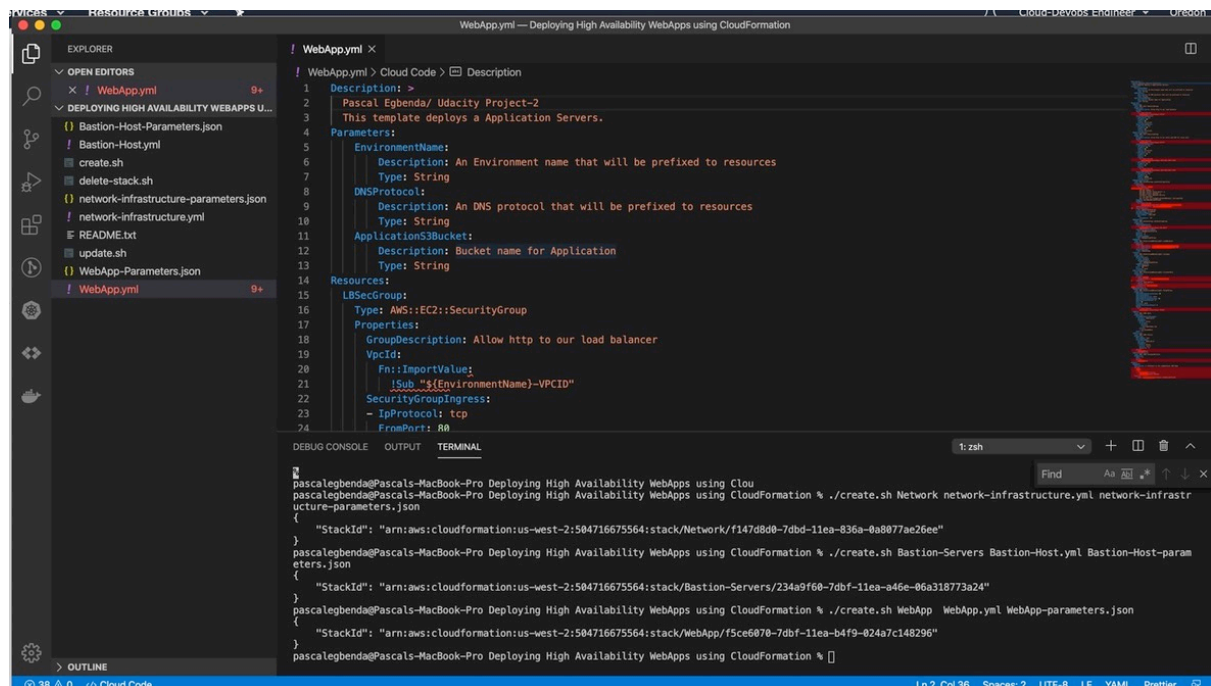


WebApp Stack

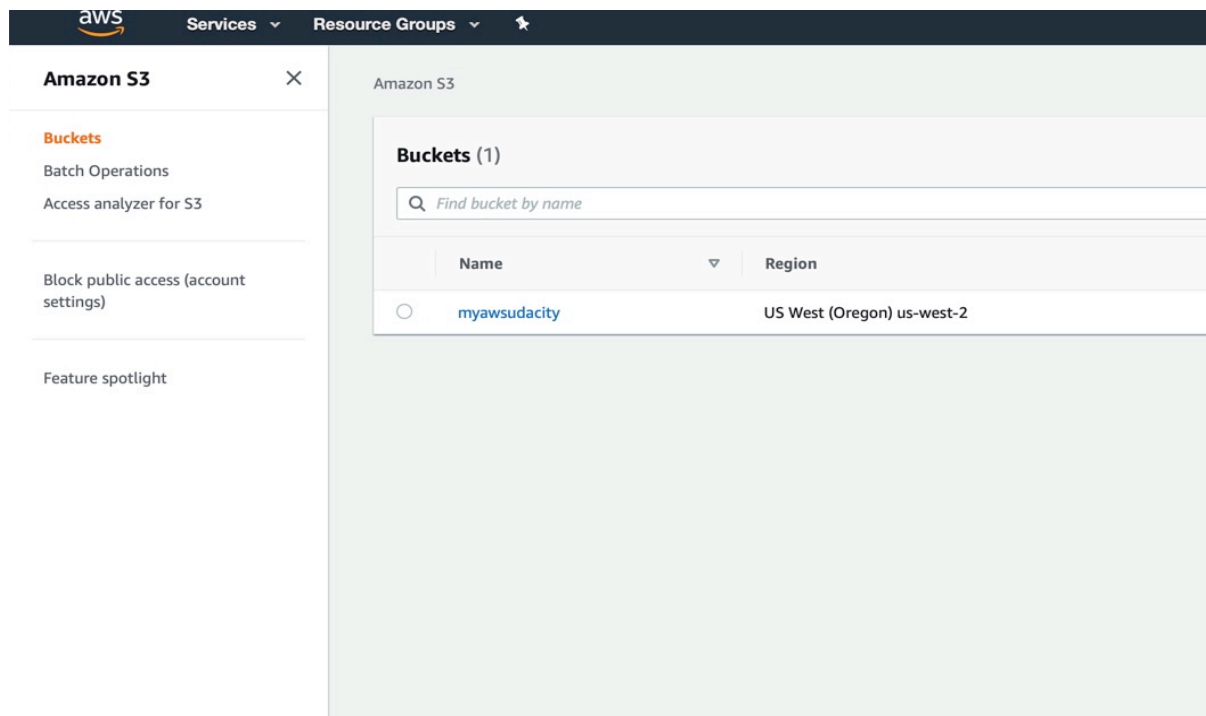
The WebApp template created the Application servers, Load Balancers etc
Command:

```
./create.sh WebApp WebApp.yml WebApp-parameters.json
```

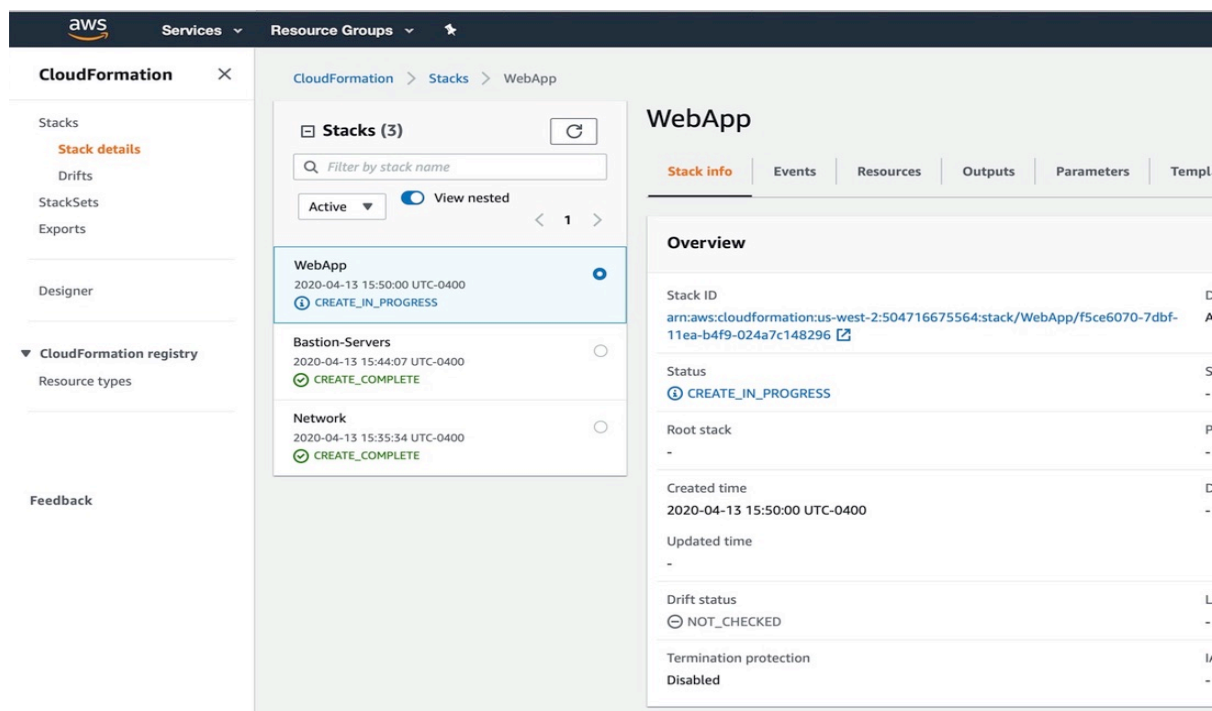
1. The WebApp Stack template was created:



2. S3 bucket created was and the sample application was sent in there:



3. The console view is shown below:



4. The completed WebApp Stack created is shown below:

The screenshot shows the AWS CloudFormation console for the 'WebApp' stack. The stack is in the 'CREATE_COMPLETE' state. The left sidebar shows the 'Stack details' tab selected. The main content area shows the 'Parameters' tab with three parameters:

Key	Value
ApplicationS3Bucket	myawsudacity
DNSProtocol	http://
EnvironmentName	UdacityProject-2

5. The WebApp Stack Resources are shown below:

The screenshot shows the AWS CloudFormation console for the 'WebApp' stack. The stack is in the 'CREATE_COMPLETE' state. The left sidebar shows the 'Stack details' tab selected. The main content area shows the 'Resources' tab with 11 resources listed:

Logical ID	Physical ID
ALBListenerRule	arn:aws:elasticloadbalancing:us-west-2:504716675564:listener-rule/app/WebApp-1LUPRGL3PNGDD/7a22b573b1ea0f35/6313bb68236db795/3d66515618ae0b5a
iamInstanceProfile	WebApp-iamInstanceProfile-NNQ6CCX75GY0
iamPolicies	WebApp-iamP-7D9SBT6KRHRH
iamRole	WebApp-iamRole-1DPX1EY63S37A
LBSecGroup	sg-0f8c48b3b4a4393b6
Listener	arn:aws:elasticloadbalancing:us-west-2:504716675564:listener/app/WebApp-1LUPRGL3PNGDD/7a22b573b1ea0f35/6313bb68236db795
WebAppGroup	WebApp-WebAppGroup-19Z7ED0KNS09Z

6. The Autoscaling Group created from the Stack is shown below:

The screenshot shows the AWS Management Console interface for an Auto Scaling Group. The left sidebar contains navigation links for various AWS services. The main content area displays the details for the Auto Scaling Group 'WebApp-WebAppGroup-19Z7ED0KNS09Z'. A notification banner at the top indicates that the older console is being replaced by the new EC2 Auto Scaling console. Below the notification, there is a 'Create Auto Scaling group' button and a table listing the instances. The table has columns for Name, Launch Configuration, Instances, Desired, Min, Max, Availability Zones, DefaultCooldown, and HealthCheckGracePeriod. The table shows one instance named 'WebApp-Web...' with a Launch Configuration of 'WebApp-WebAppLaun...', 4 instances, a desired capacity of 4, a minimum of 4, a maximum of 5, and availability zones of 'us-west-2a, us-west-2b'. Below the table, the 'Details' tab is selected, showing the 'Launch Configuration' as 'WebApp-WebAppLaunchConfig-1H2U4KT120J42' and 'Availability Zone(s)' as 'us-west-2a, us-west-2b'. Other details include 'Subnet(s)', 'Classic Load Balancers', 'Target Groups', 'Health Check Type', 'Health Check Grace Period', 'Instance Protection', and 'Termination Policies'.

Name	Launch Configuration	Instances	Desired	Min	Max	Availability Zones	DefaultCooldown	HealthCheckGracePeriod
WebApp-Web...	WebApp-WebAppLaun...	4	4	4	5	us-west-2a, us-west-2b	300	0

Auto Scaling Group: WebApp-WebAppGroup-19Z7ED0KNS09Z

Details | Activity History | Scaling Policies | Instances | Monitoring | Notifications | Tags | Scheduled Actions | Lifecycle Hooks

Launch Configuration: WebApp-WebAppLaunchConfig-1H2U4KT120J42

Availability Zone(s): us-west-2a, us-west-2b

Subnet(s): subnet-0eb57745ffe877a84, subnet-05c4c88a22f2db418

Classic Load Balancers:

Target Groups: WebApp-WebApp-1XOCMU9VPOPUU

Health Check Type: EC2

Health Check Grace Period: 0

Instance Protection:

Termination Policies: Default

7. The four instances created in the two Availability Zones are shown below:

The screenshot shows the AWS Management Console interface for EC2 instances. The left sidebar contains navigation links for various AWS services. The main content area displays a table of instances. The table has columns for Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, Alarm Status, Public DNS (IPv4), and IPv4 Public IP. The table shows four instances, all in a 'running' state. The first three instances are t3.medium instances in us-west-2a and us-west-2b availability zones. The fourth instance is a t3.small instance in us-west-2a availability zone. The table also shows the Public DNS (IPv4) and IPv4 Public IP for each instance.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
	i-031a7de314ae53f38	t3.medium	us-west-2a	running	2/2 checks ...	None		
	i-03bbce9f7c2df2747	t3.medium	us-west-2b	running	2/2 checks ...	None		
	i-05a9d7ea8bd0104...	t3.medium	us-west-2b	running	2/2 checks ...	None		
	i-0fde9f5480170b143	t3.medium	us-west-2a	running	2/2 checks ...	None		
	i-05ce720ca2518fecd	t3.small	us-west-2b	running	2/2 checks ...	None	ec2-34-223-133-109.us...	34.223.133.109
	i-08f5d554a5ecc58e3	t3.small	us-west-2a	running	2/2 checks ...	None	ec2-54-70-49-191.us-w...	54.70.49.191

8. The Configuration/User data used to deploy the Application is shown:

The screenshot shows the AWS Management Console interface. A modal window titled 'User data' is open, displaying a shell script for installing and starting an Apache2 service. The script is as follows:

```
#!/bin/bash
apt-get update -y
apt-get install unzip awscli -y
apt-get install apache2 -y
systemctl start apache2.service
cd /var/www/html
aws s3 sync s3://myawsudacity /var/www/html
```

The background shows the 'Details' tab for the 'WebApp-WebAppLaunchConfig-1H2U4KT1' launch configuration. The details include:

- AMI ID: ami-0d1cd67c2b5fca19
- IAM Instance Profile: WebApp-IamInstanceProfile-NNQ6CCK75GY0
- Key Name: Pascal_Udacity
- EBS Optimized: false
- Spot Price: (empty)
- RAM Disk ID: (empty)
- User data: [View user data](#)
- Instance Type: t3.medium
- Kernel ID: (empty)
- Monitoring: true
- Security Groups: sg-0e4961f59b2003249
- Creation Time: Mon Apr 13 15:52:25 GMT-400 2020
- Block Devices: /dev/sdk
- IP Address Type: Only assign a public IP address to instances launched in the default VPC and subnet. (default)

9. The Load Balancer created is shown below:

The screenshot shows the AWS Management Console interface. The 'Basic Configuration' tab is selected for the 'WebApp-WebApp-1LUPRGL3PNGDD' load balancer. The details include:

- Name: WebApp-WebApp-1LUPRGL3PNGDD
- ARN: arn:aws:elasticloadbalancing:us-west-2:504716675564:loadbalancer/app/WebApp-WebApp-1LUPRGL3PNGDD/7a22b573b1ea0f35
- DNS name: WebApp-WebApp-1LUPRGL3PNGDD-844628642.us-west-2.elb.amazonaws.com (A Record)
- State: active
- Type: application
- Scheme: internet-facing
- IP address type: ipv4
- VPC: vpc-0b62f25813fe53017
- Availability Zones: subnet-04eb4430cebeb51fe - us-west-2a
- IPV4 address: Assigned by AWS

10. The Application Load Balancer listening on port 80 is shown below:

The screenshot shows the AWS Management Console interface for an Application Load Balancer. The left sidebar contains navigation links for EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES, IMAGES, ELASTIC BLOCK STORE, and NETWORK & SECURITY. The main content area displays the 'WebAp-WebAp-1LUPRGL3PNGDD' load balancer. The 'Listeners' tab is selected, showing a single listener for HTTP on port 80. The listener rules are configured to forward requests to the target group 'WebAp-WebAp-1XOCMU9VPOPUU'.

Name	DNS name	State	VPC ID	Availability Zones	Type	Created At
WebAp-WebAp-1LUPRGL3...	WebAp-WebAp-1LUPRGL3...	active	vpc-0b62f25813fe53017	us-west-2a, us-west-2b	application	April 13, 2020 at 3:50:12 F

Load balancer: WebAp-WebAp-1LUPRGL3PNGDD

Description | **Listeners** | Monitoring | Integrated services | Tags

A listener checks for connection requests using its configured protocol and port, and the load balancer uses the listener rules to route requests to targets. You can add, remove, or update listeners and listener rules.

[Add listener](#) [Edit](#) [Delete](#)

Listener ID	Security policy	SSL Certificate	Rules
<input type="checkbox"/> HTTP : 80 arn...6313bb68236db795 +	N/A	N/A	Default: forwarding to WebAp-WebAp-1XOCMU9VPOPUU View/edit rules

11. Below is the Target Group created showing the instances in two different Availability Zones:

The screenshot shows the AWS Management Console interface for a Target Group. The left sidebar contains navigation links for EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES, IMAGES, ELASTIC BLOCK STORE, and NETWORK & SECURITY. The main content area displays the 'WebAp-WebAp-1XOCMU9VPOPUU' target group. The 'Targets' tab is selected, showing a list of registered targets. The targets are distributed across two Availability Zones: us-west-2a and us-west-2b.

Name	Port	Protocol	Target type	Load Balanc	VPC ID	Monitoring
WebAp-WebAp-1XOCMU9V...	80	HTTP	instance	WebAp-We...	vpc-0b62f25813fe53017	<input checked="" type="checkbox"/>

Target group: WebAp-WebAp-1XOCMU9VPOPUU

Description | **Targets** | Health checks | Monitoring | Tags

The load balancer starts routing requests to a newly registered target as soon as the registration process completes and the target passes the initial health checks. If demand on your targets increases, you can register more targets. If demand on your targets decreases, you can deregister targets.

[Edit](#)

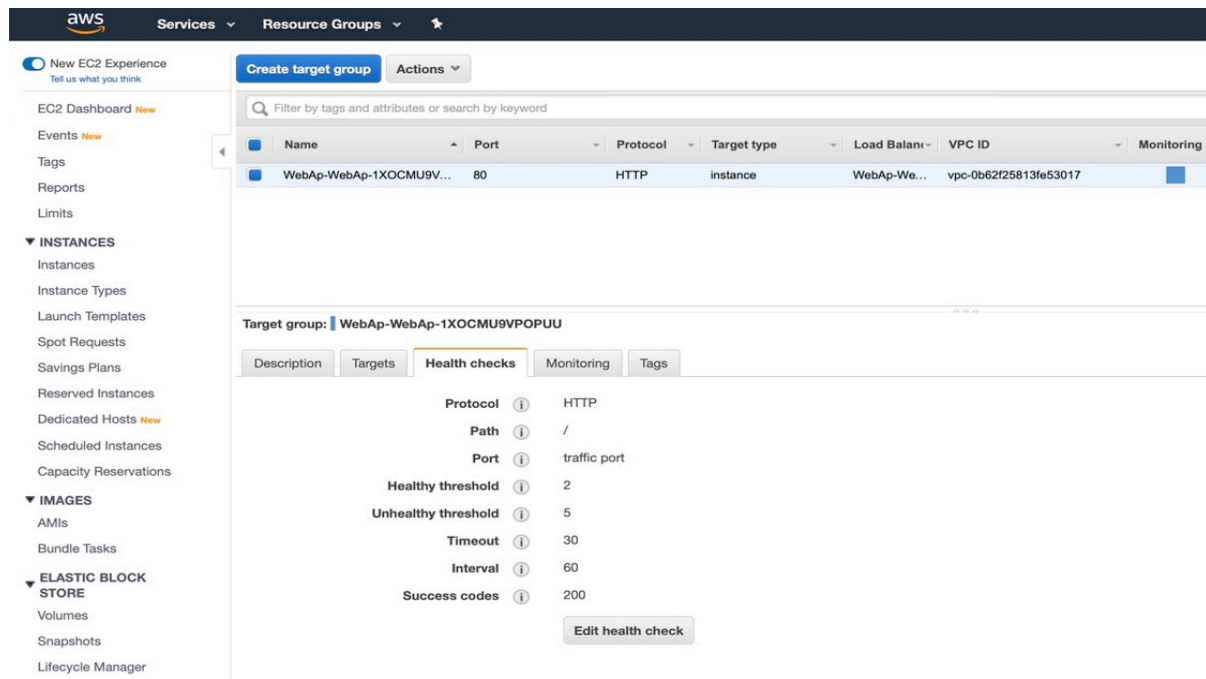
Registered targets

Instance ID	Name	Port	Availability Zone	Status	Description
i-050e0e0e0e0e0e0e0		80	us-west-2b	healthy	This target is currently passing target group's health checks.
i-07ef20a7e38d0ca9		80	us-west-2b	healthy	This target is currently passing target group's health checks.
i-033992af933d09ca7		80	us-west-2a	healthy	This target is currently passing target group's health checks.
i-0a6b172c4f04bede7		80	us-west-2a	healthy	This target is currently passing target group's health checks.

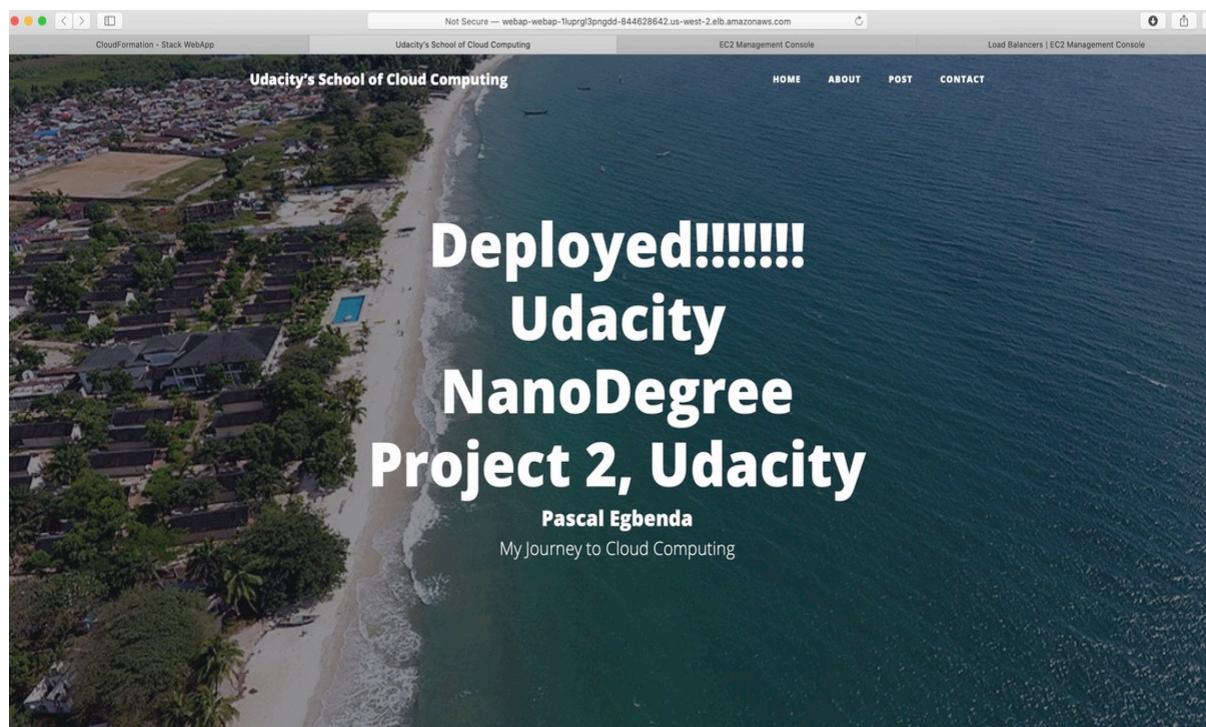
Availability Zones

Availability Zone	Target count	Healthy?
us-west-2a	2	Yes
us-west-2b	2	Yes

12. The Health checks created from the WebApp Stack is shown below:



13. The output from the WebApp Stack is shown below:



14. Other templates created to help automate the process are:

- *create.sh*
- *update.sh*
- *delete-stack.sh*