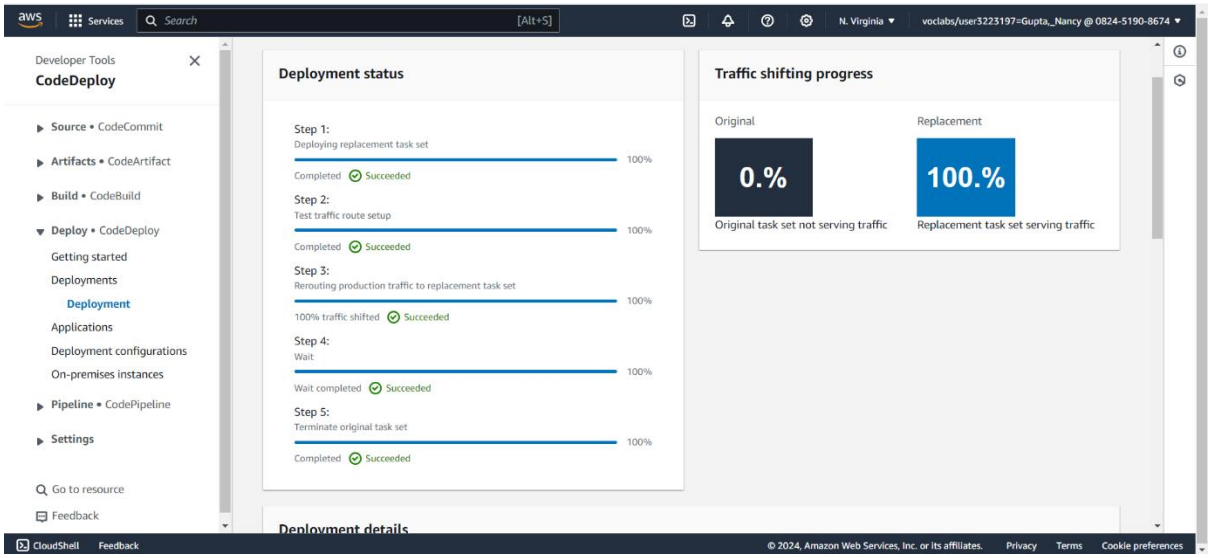
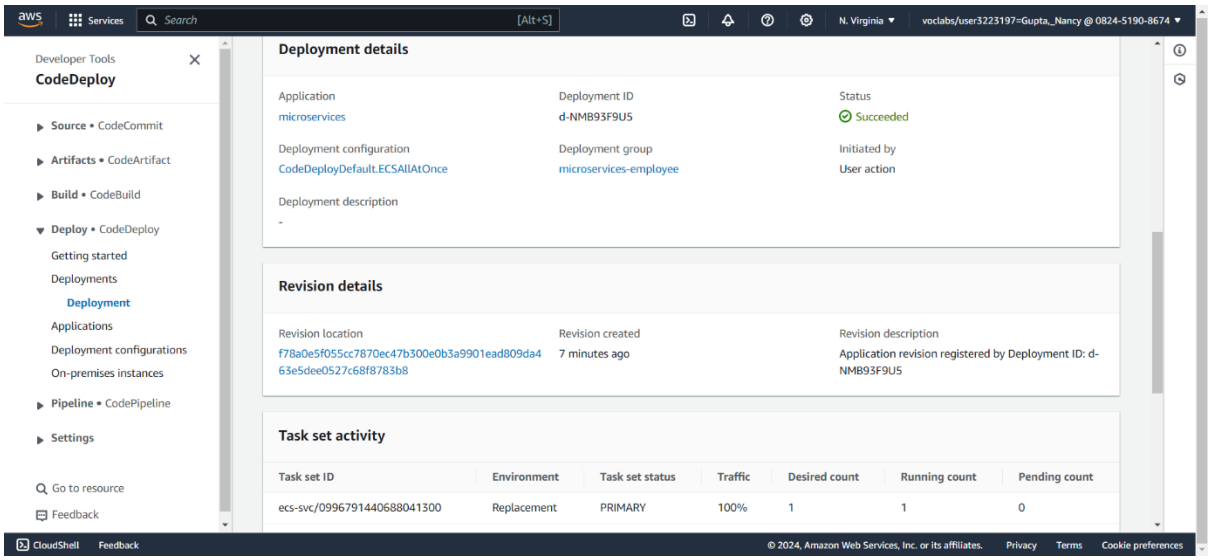
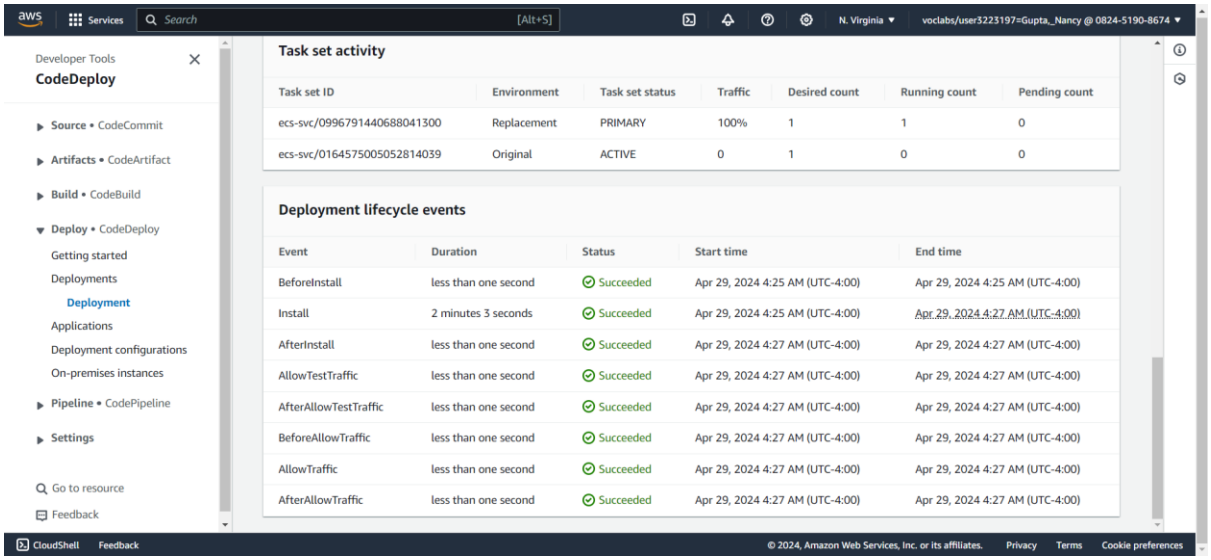
4. Return to the CodeDeploy page to confirm that all five steps of the deployment succeeded and the replacement task set is now serving traffic.

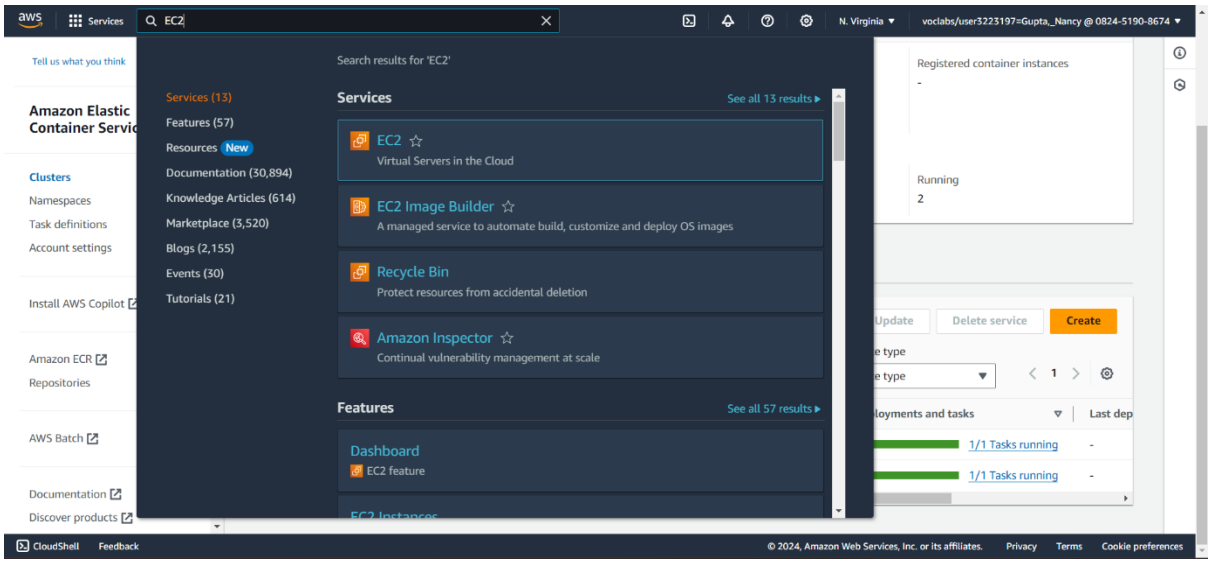






Congratulations! You successfully deployed the employee microservice to Amazon ECS on Fargate by using a CI/CD pipeline.

Task 8.6: Observe how CodeDeploy modified the load balancer listener rules

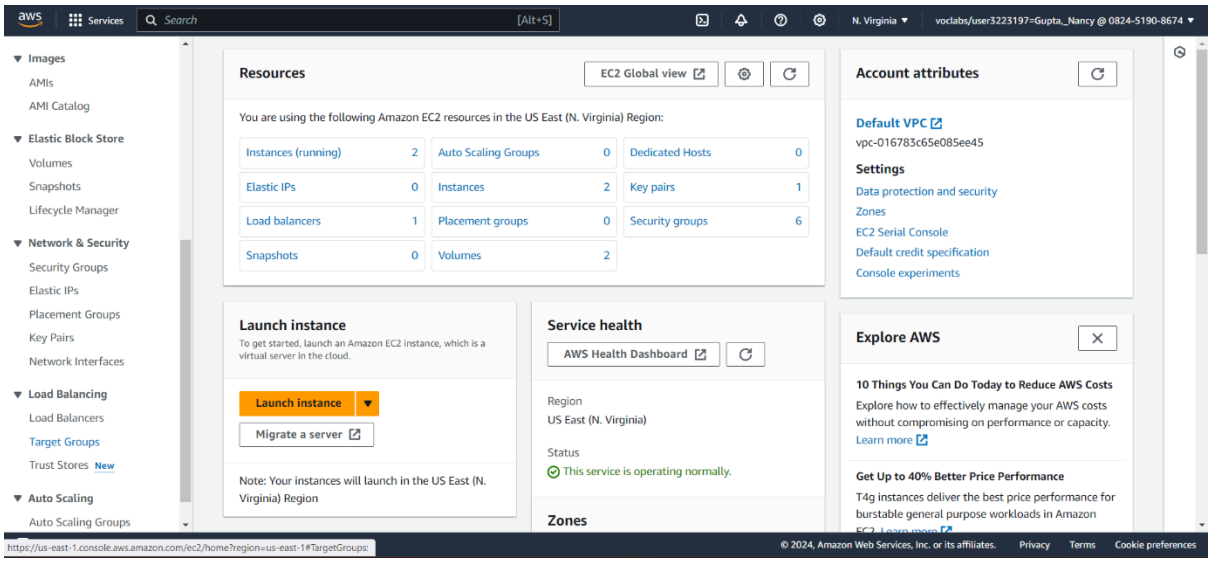


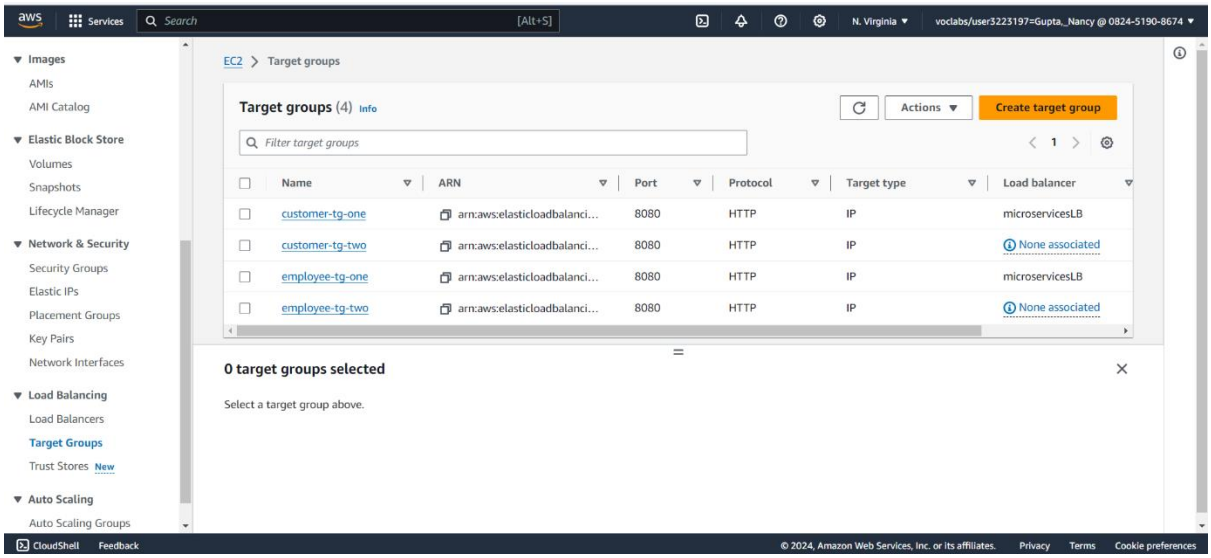
1. Observe the load balancer and target group settings.
    - In the Amazon EC2 console, choose **Target Groups**.

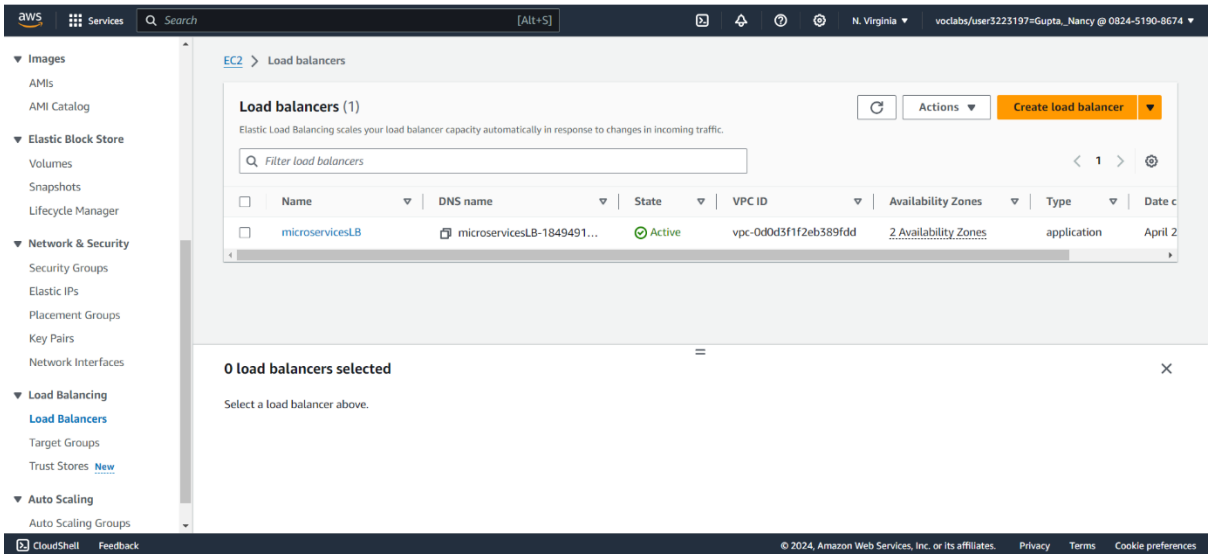
Note: If you already had the page open, refresh it.

Notice that the *customer-tg-two* target group is no longer associated with the load balancer. This is because CodeDeploy is managing the load balancer listener rules.

Note: If you are repeating this step, the target groups that are currently attached and unattached might be different.

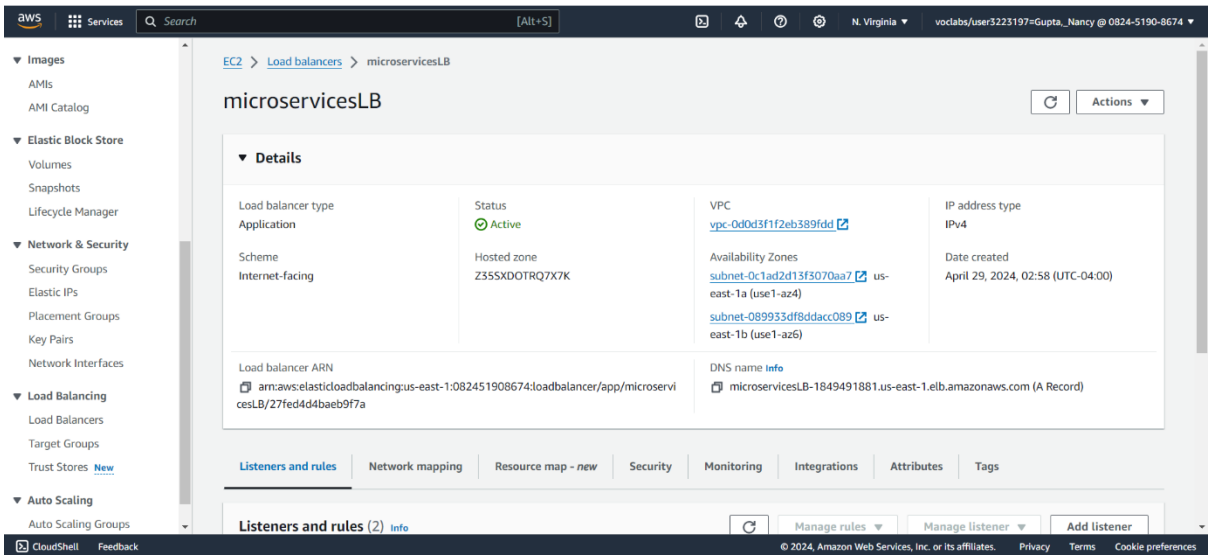


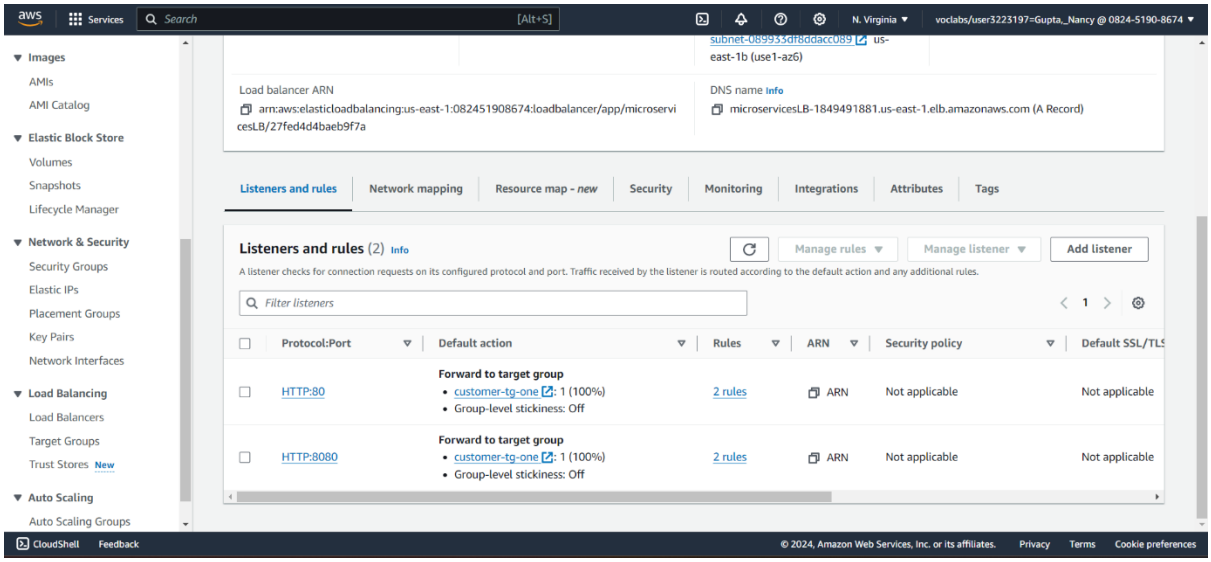


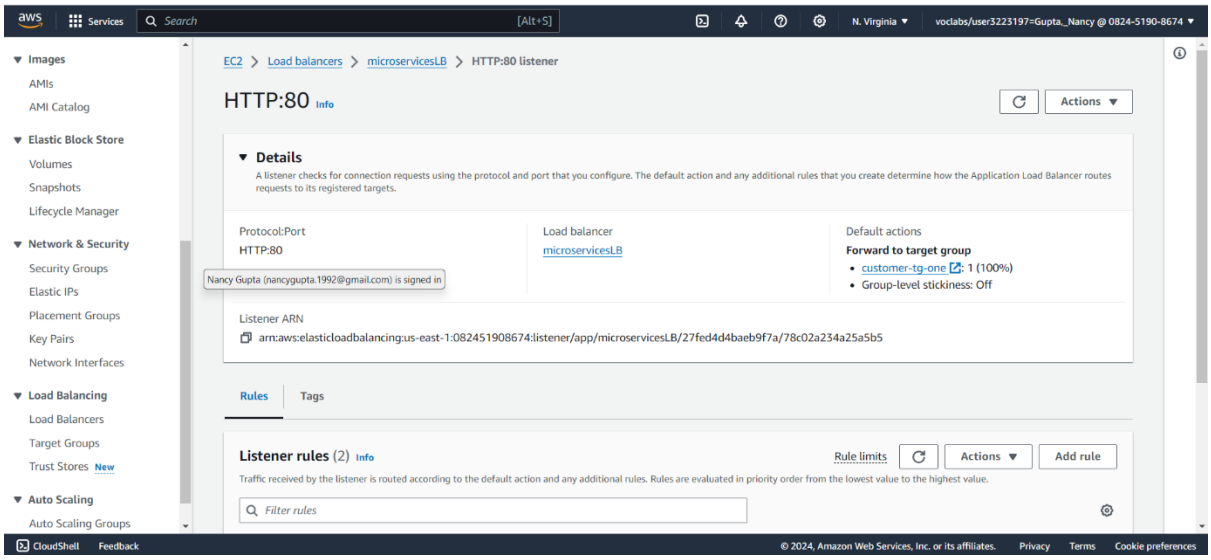


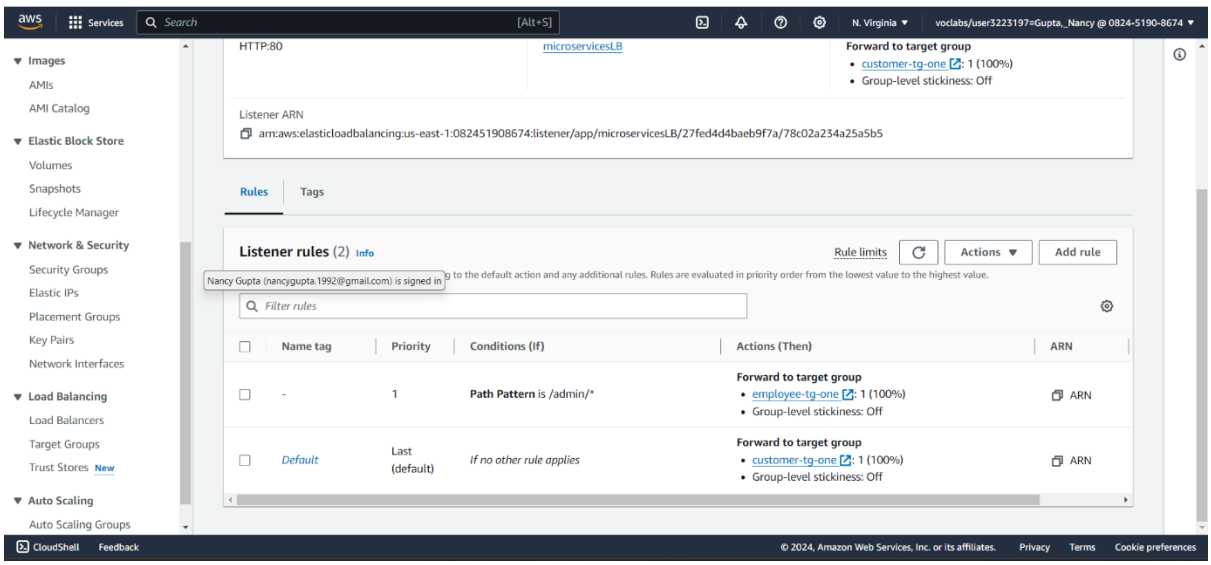
- Observe the HTTP:80 listener rules.

The default rule has changed here. For the default "If no other rule applies" rule, the "forward to target group" previously pointed to *customer-tg-two*, but now it points to *customer-tg-one*.



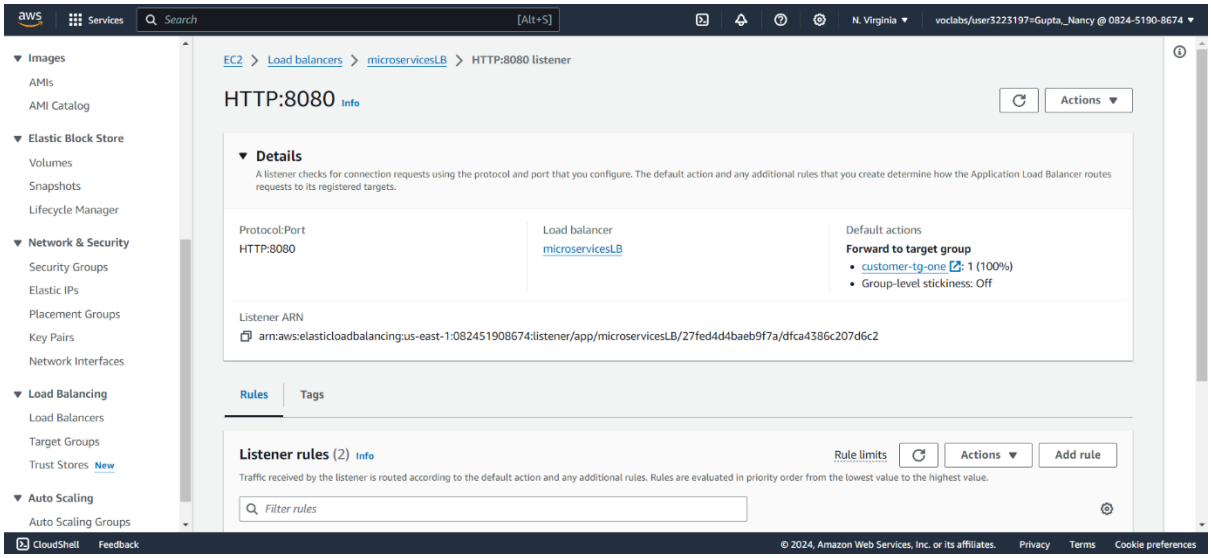


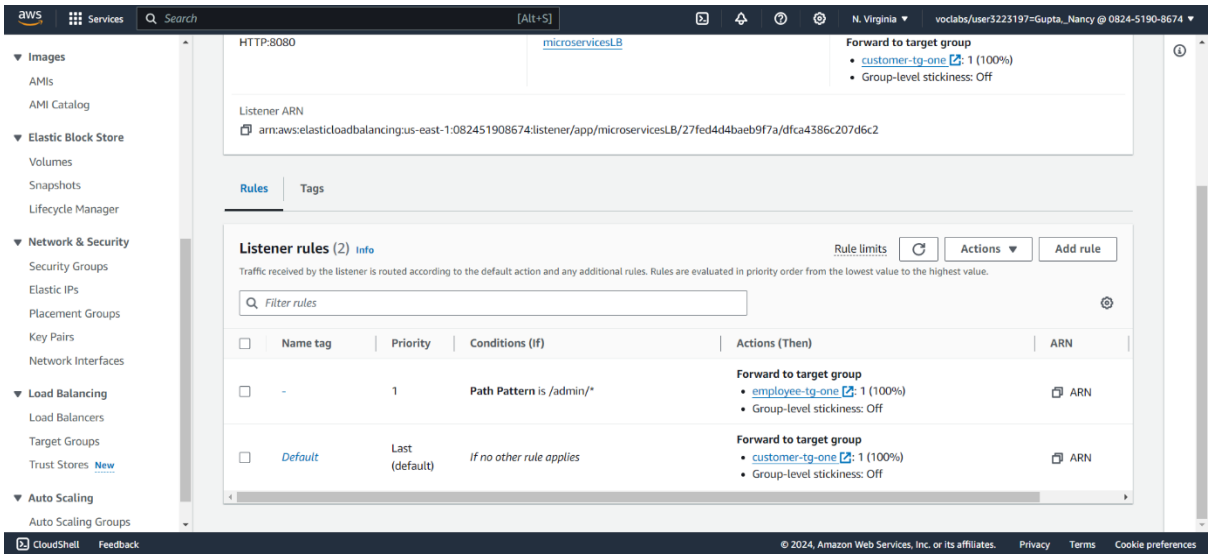




- Observe the HTTP:8080 listener rules.

The two rules still forward to the "one" target groups.



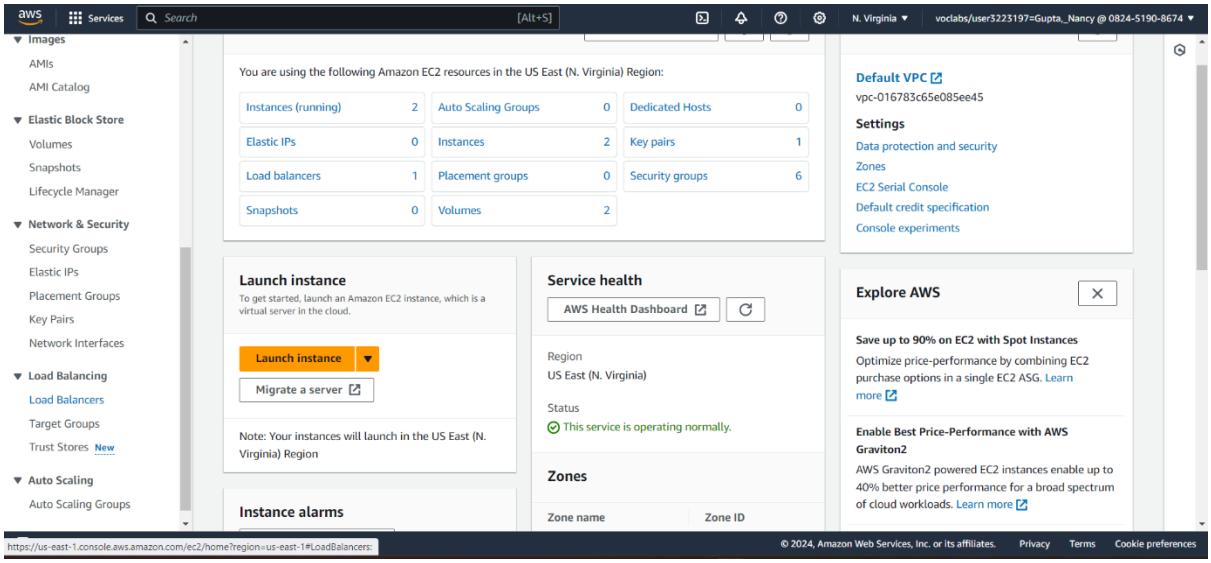


Phase 9: Adjusting the microservice code to cause a pipeline to run again

In this phase, you will experience the benefits of the microservices architecture and the CI/CD pipeline that you built. You will begin by adjusting the load balancer listener rules that are related to the *employee* microservice. You will also update the source code of the *employee* microservice, generate a new Docker image, and push that image to Amazon ECR, which will cause the pipeline to run and update the production deployment. You will also scale up the number of containers that support the *customer* microservice.

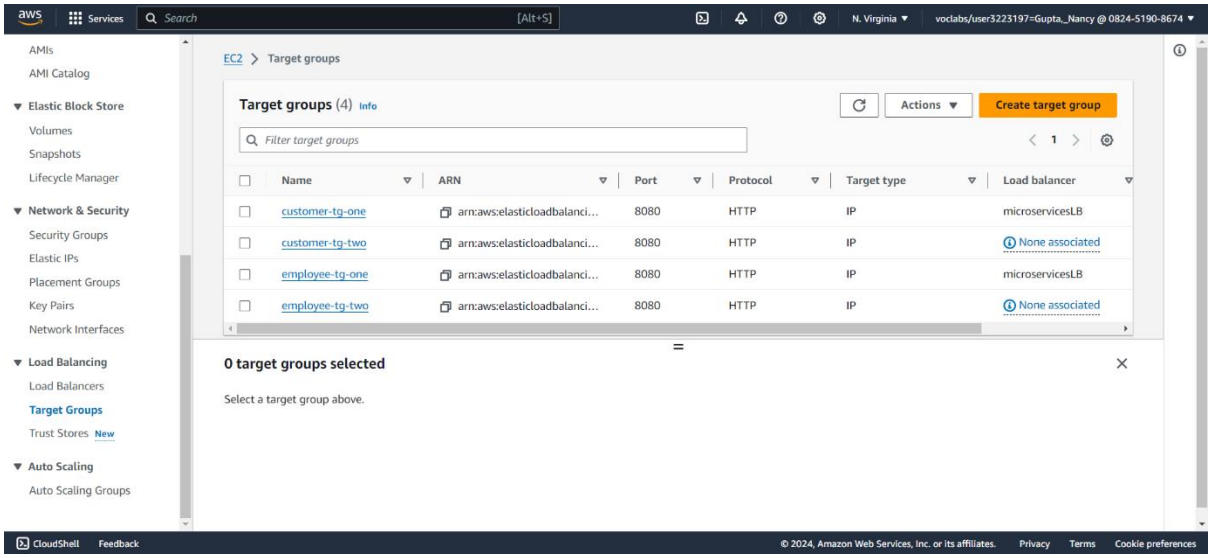
Task 9.1: Limit access to the *employee* microservice

In this task, you will limit access to the *employee* microservice to only people who try to connect to it from a specific IP address. By limiting the source IP to a specific IP address, only users who access the application from that IP can access the pages, and edit or delete supplier entries.



1. Confirm that all target groups are still associated with the Application Load Balancer.

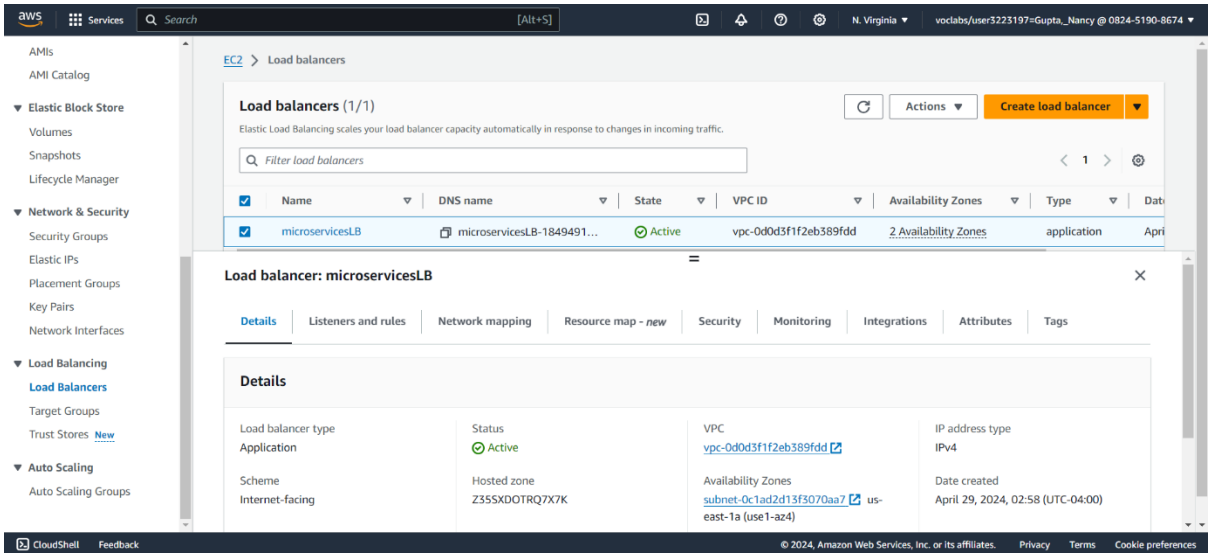
In the Amazon EC2 console, check that all four target groups are still associated with the load balancer. Reassociate target groups as needed before going to the next step.

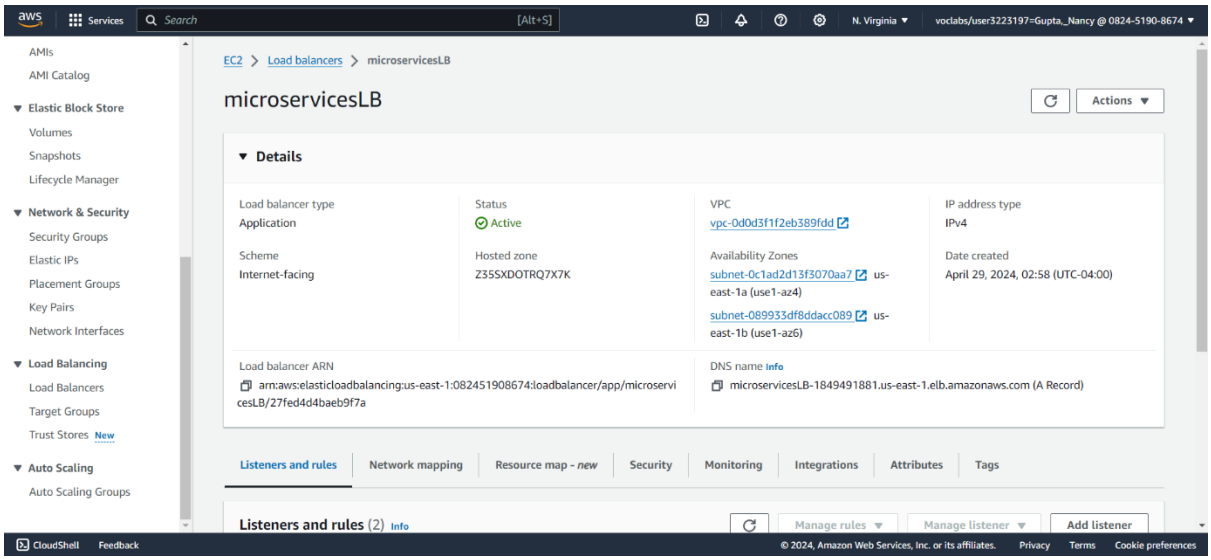


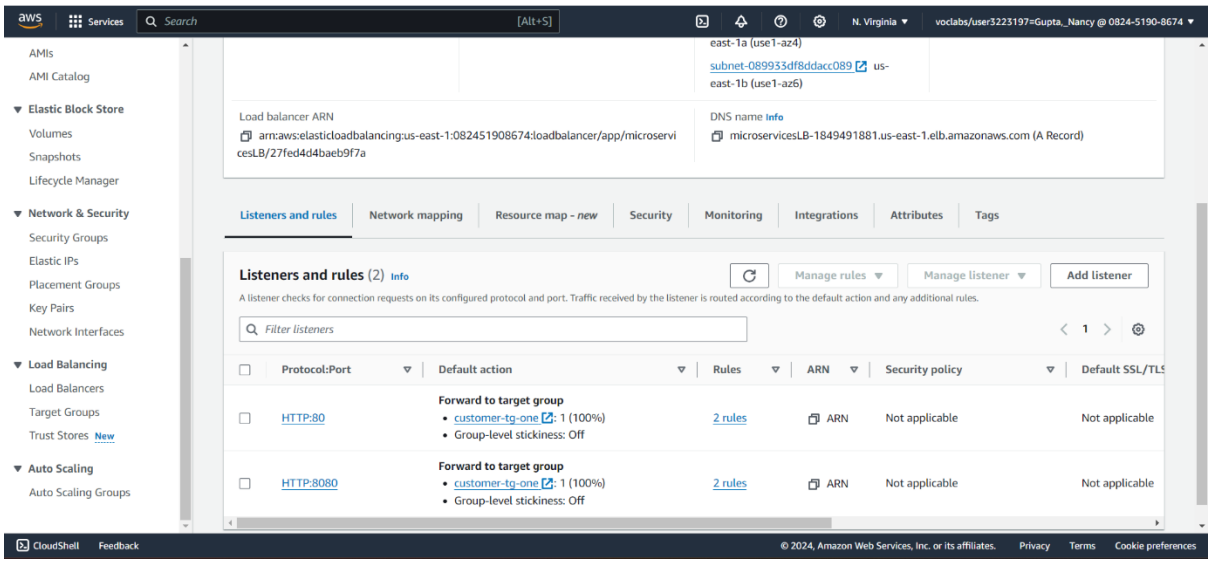
Since the target groups are not associated with the Load balancer, we will reassociate target groups with the load balancer.

Tip: For details, see Reassociating target groups with the load balancer in the appendix.

Reassociating target groups with the load balancer





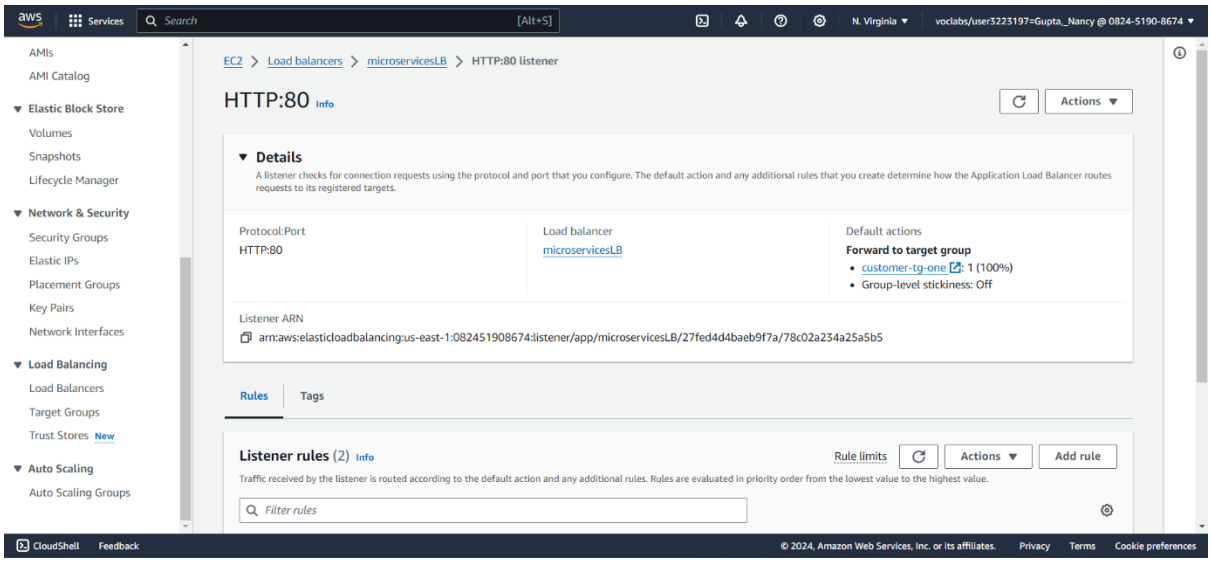


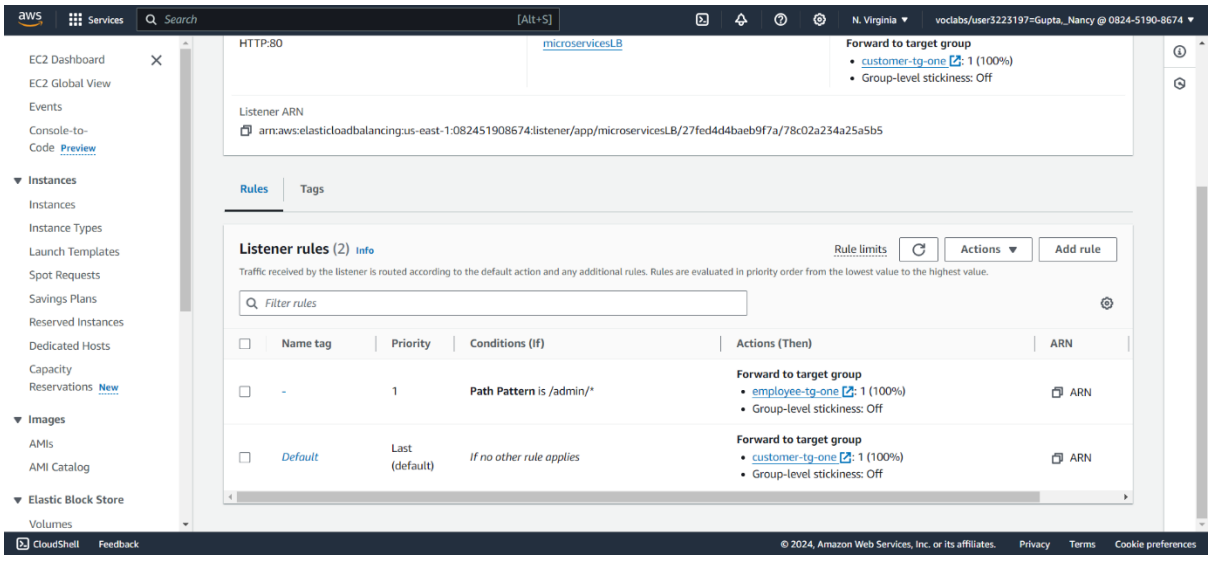
Because CodeDeploy manages the Application Load Balancer rules, you might periodically find that some target groups are no longer associated with the load balancer. The following steps explain how to reassociate the target groups.

1. Note the original configuration that you made for the **HTTP:80** listener.

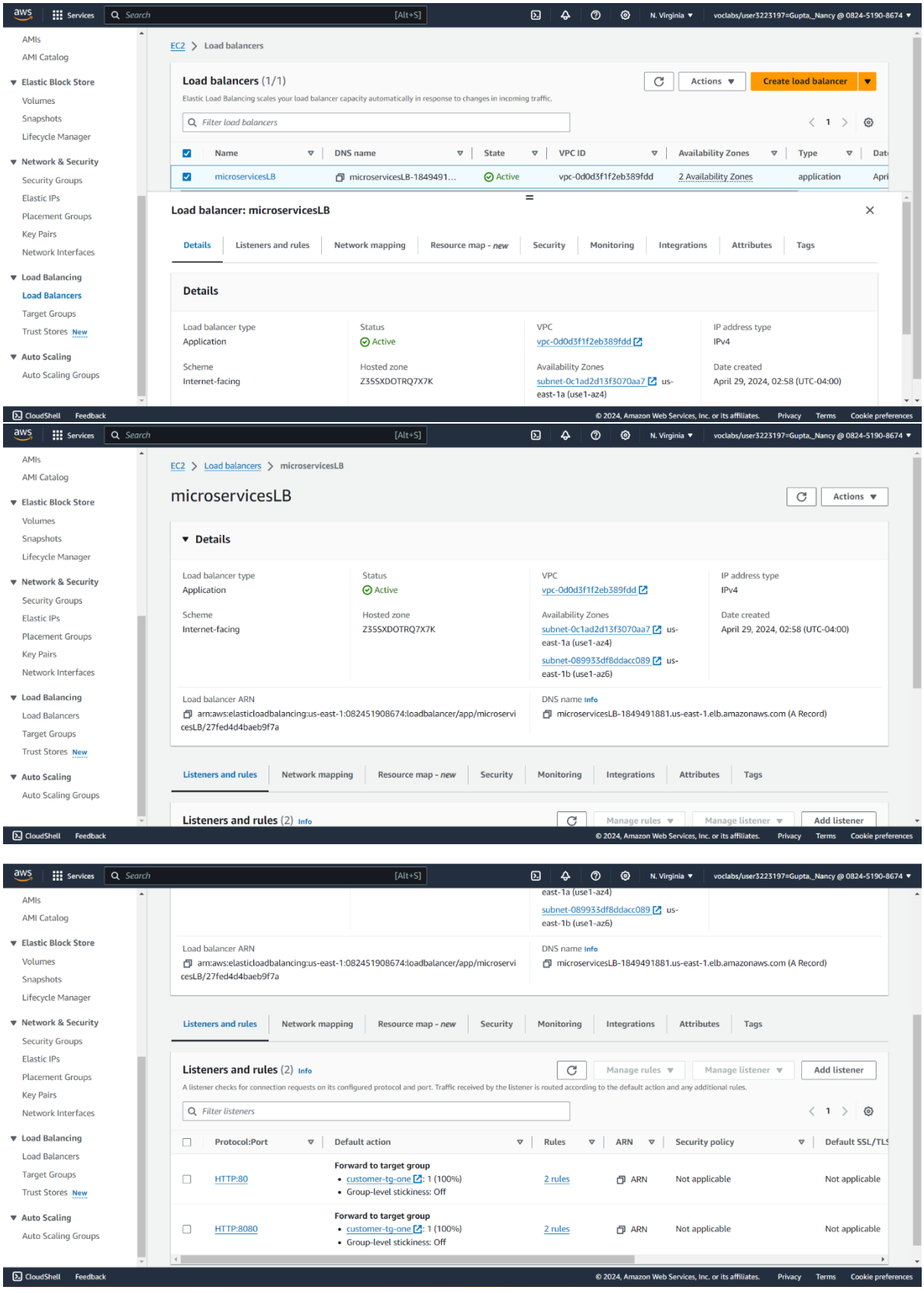
The following image shows the original configuration for this listener:

1. If you successfully deployed the microservices previously, and the settings that are shown in the previous image are currently in use for the deployment (where the target groups in use by the HTTP:80 listener have "two" in their names), then this time you want to set the HTTP:80 listener to use target groups that have "one" in their names. Effectively, the green task set has become the blue task set, and the blue task set has become the green task set.



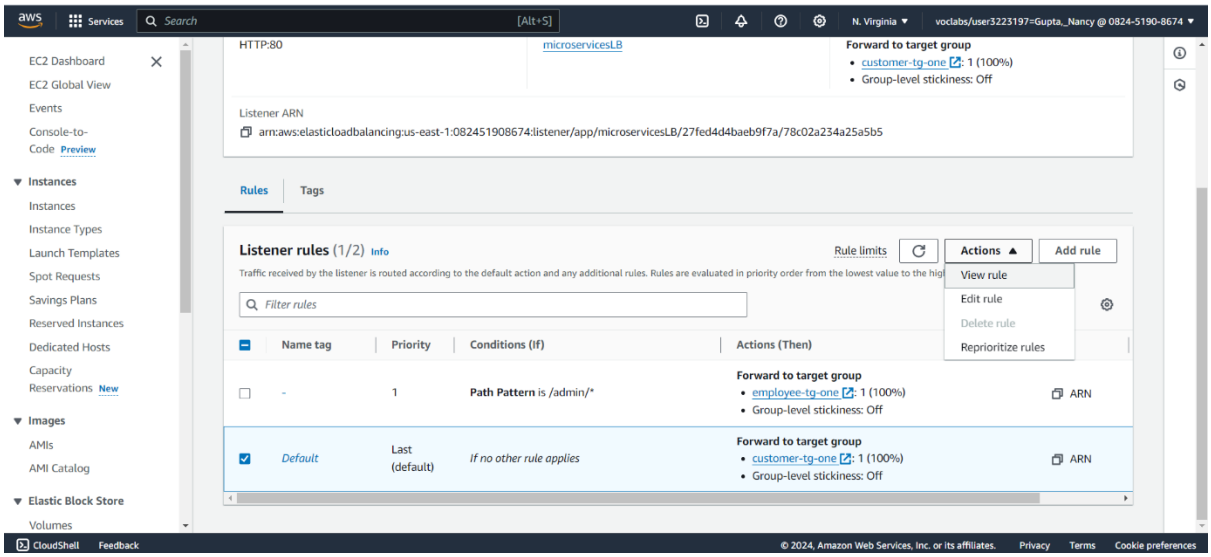


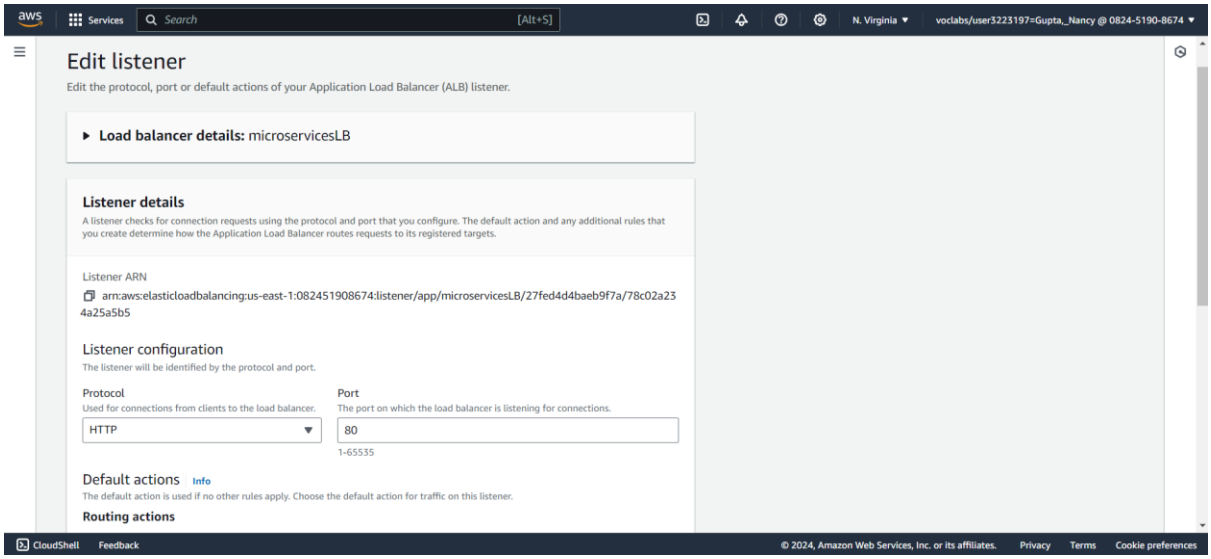
2. Edit the **HTTP:80** load balancer rules to make sure that they still match the previous image. If they don't, adjust them:
    - Navigate to the Amazon EC2 console.
    - In the navigation pane, choose **Load Balancers**.
    - Choose the link for the **microservicesLB** load balancer.

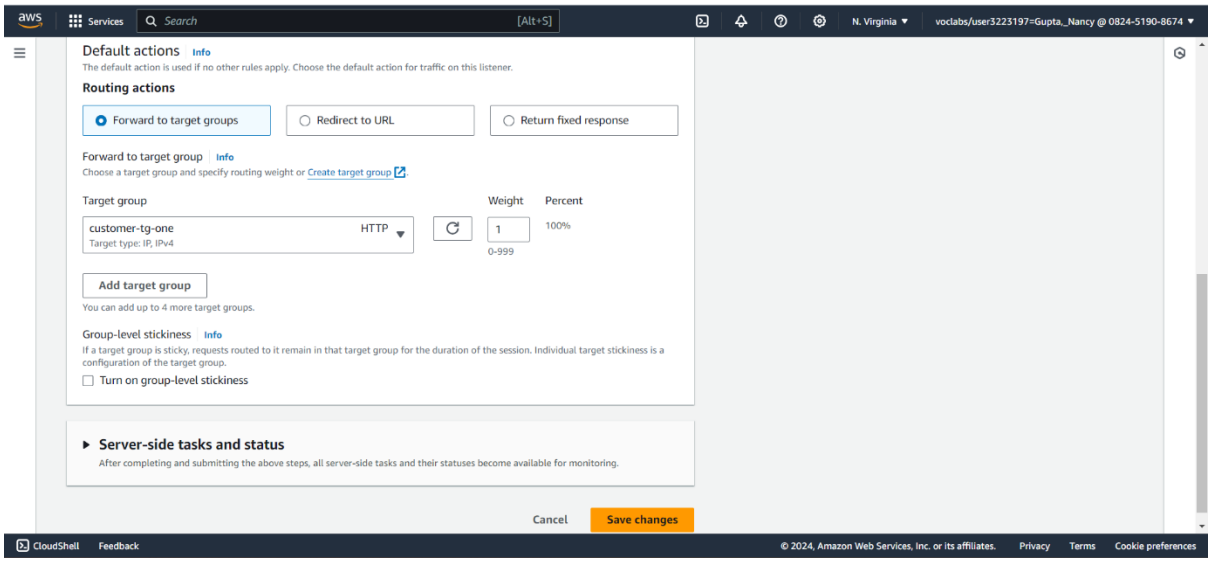


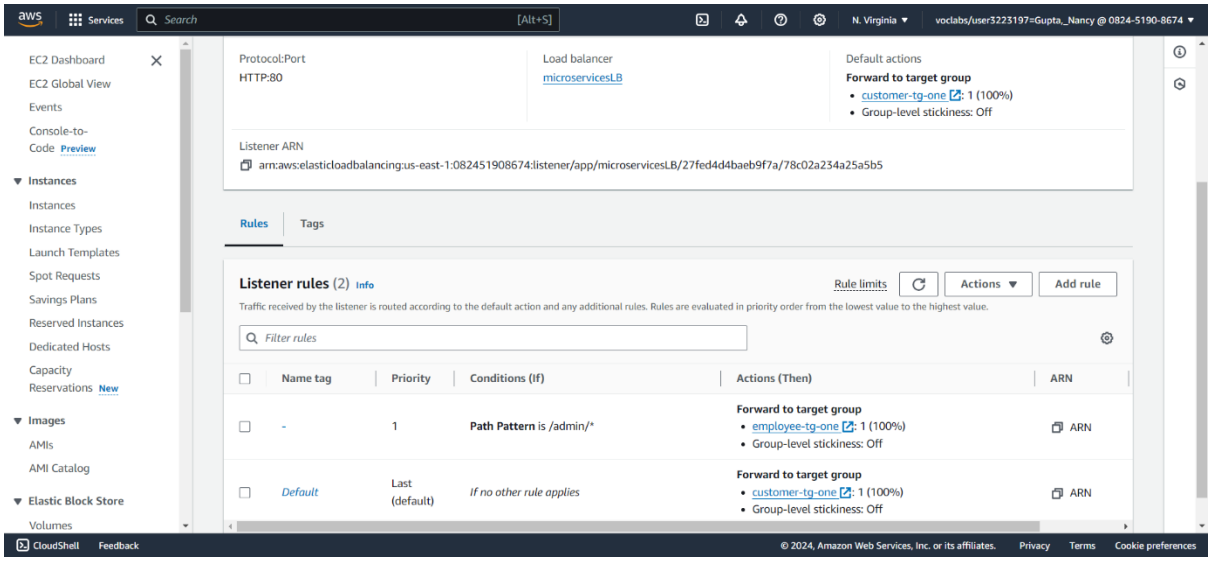
- On the **Listeners and rules** tab, choose the **HTTP:80** link.
- In the **Listener rules** panel, verify that the **Default** rule forwards to **customer-tg-one**. If it doesn't:

- Select the rule.
- From the **Actions** menu, choose **Edit rule**.
- Set **Forward to target group** to **customer-tg-one**, and choose **Save changes**.

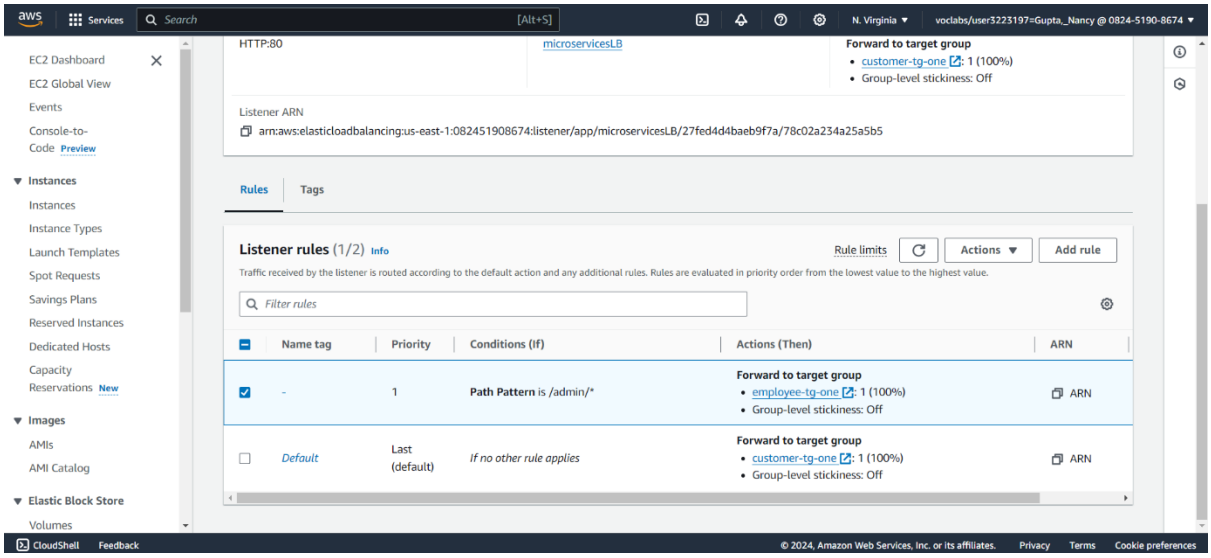






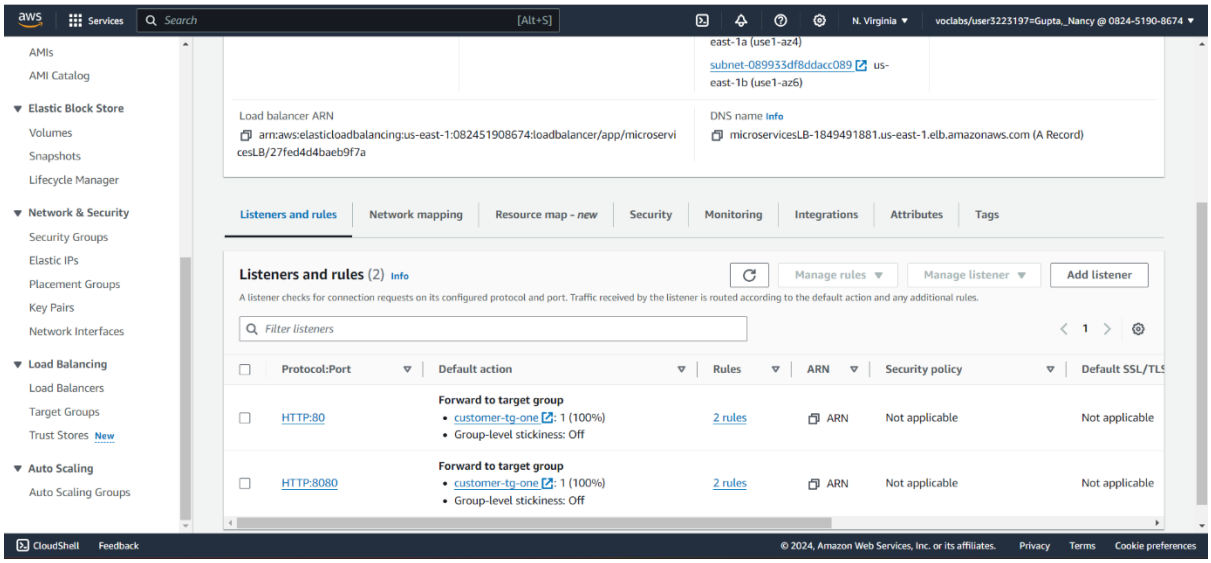


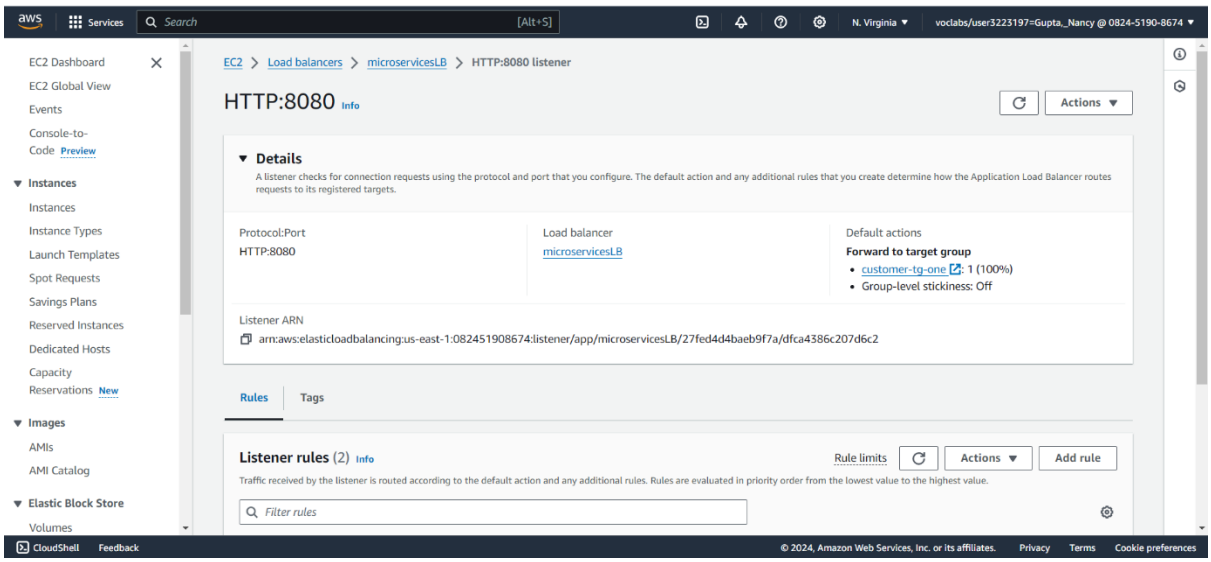
- Still on the **HTTP:80** page, in the **Listener rules** panel, verify that the rule with **Path Pattern is /admin/** forwards to **employee-tg-one**. If it doesn't:
  - Select the rule.
  - From the **Actions** menu, choose **Edit rule**, and then choose **Next**.
  - In the **Actions** panel, set **Forward to target group** to **employee-tg-one**.
  - Choose **Next**, and then choose **Save changes**.

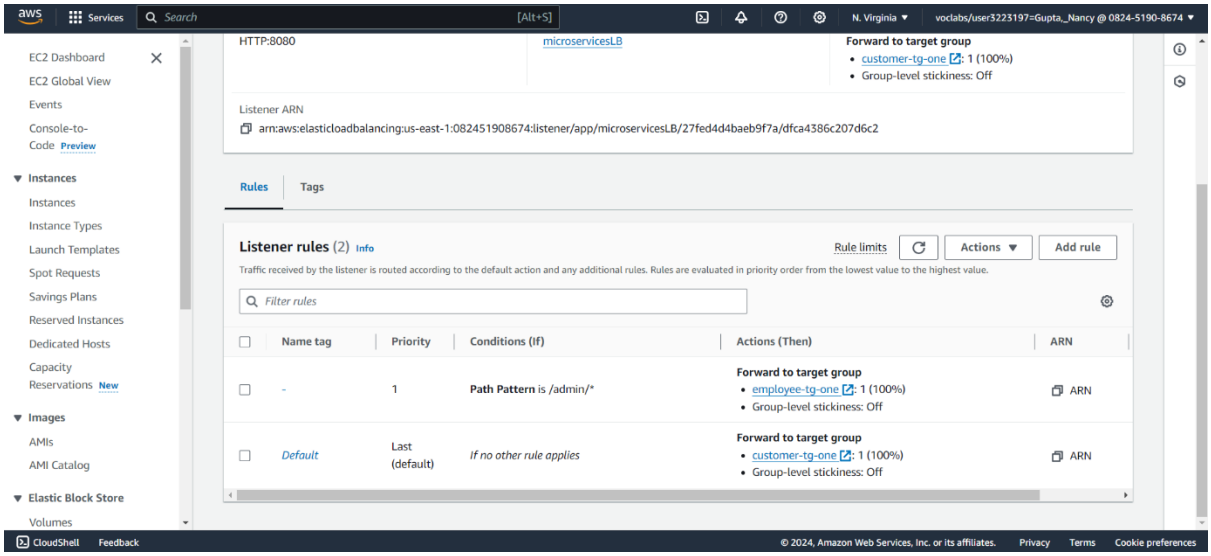


3. Note the original configuration that you made for the **HTTP:8080** listener.

The following image shows the original configuration for this listener:



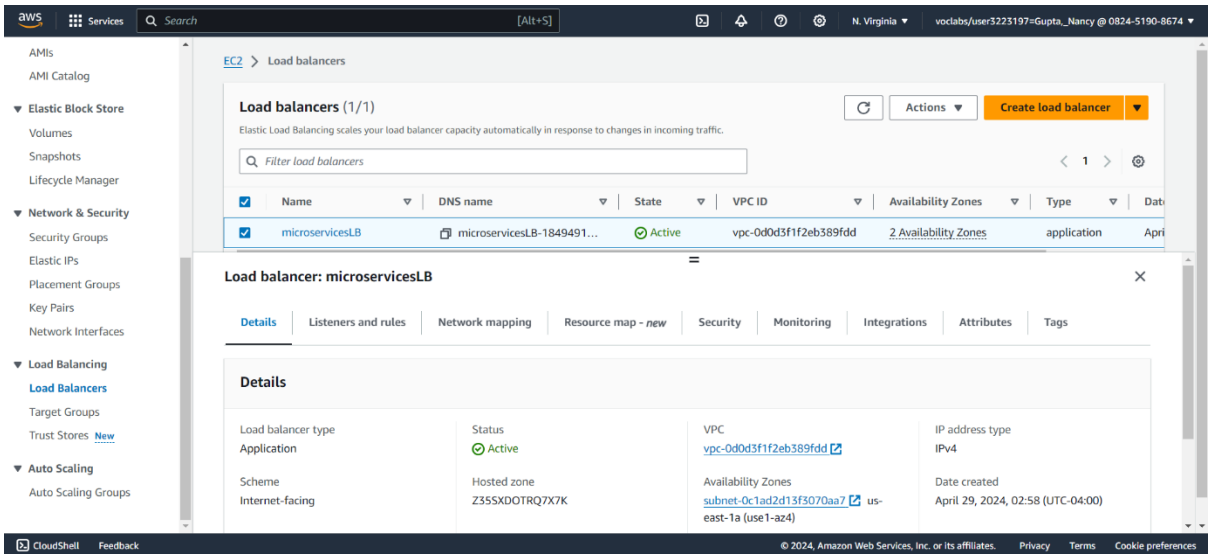




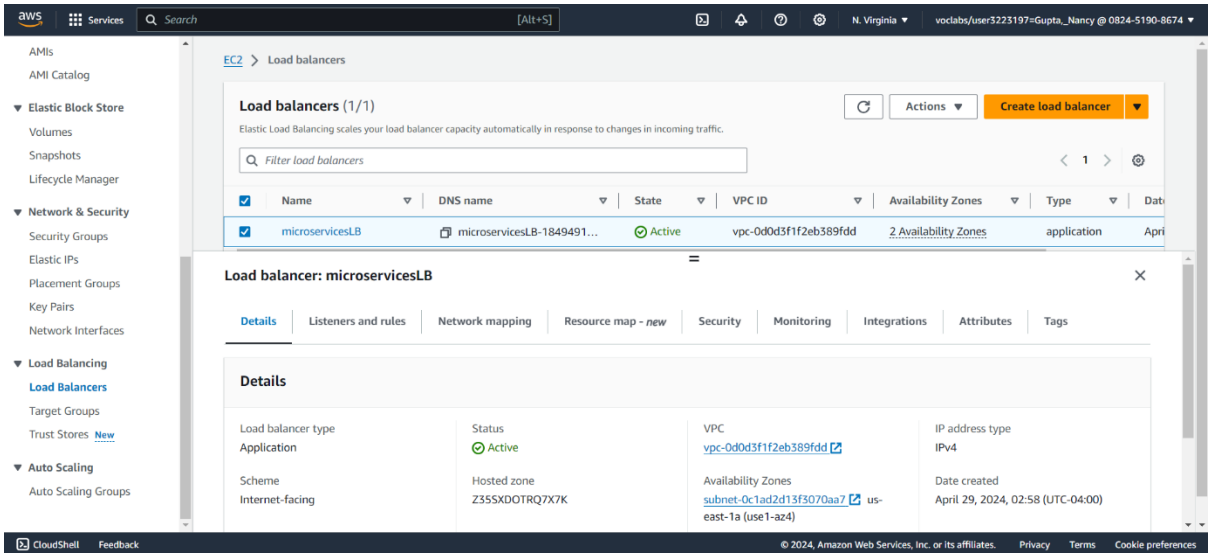
3. If you successfully deployed the microservices previously, and the settings that are shown in the previous image are currently in use for the deployment (where the target groups in use by the HTTP:8080 listener have "one" in their names), then this time you want to set the HTTP:8080 listener to use target groups that have "two" in their names.

4. Edit the **HTTP:8080** load balancer rules to make sure that they still match the previous image. If they don't, adjust them:

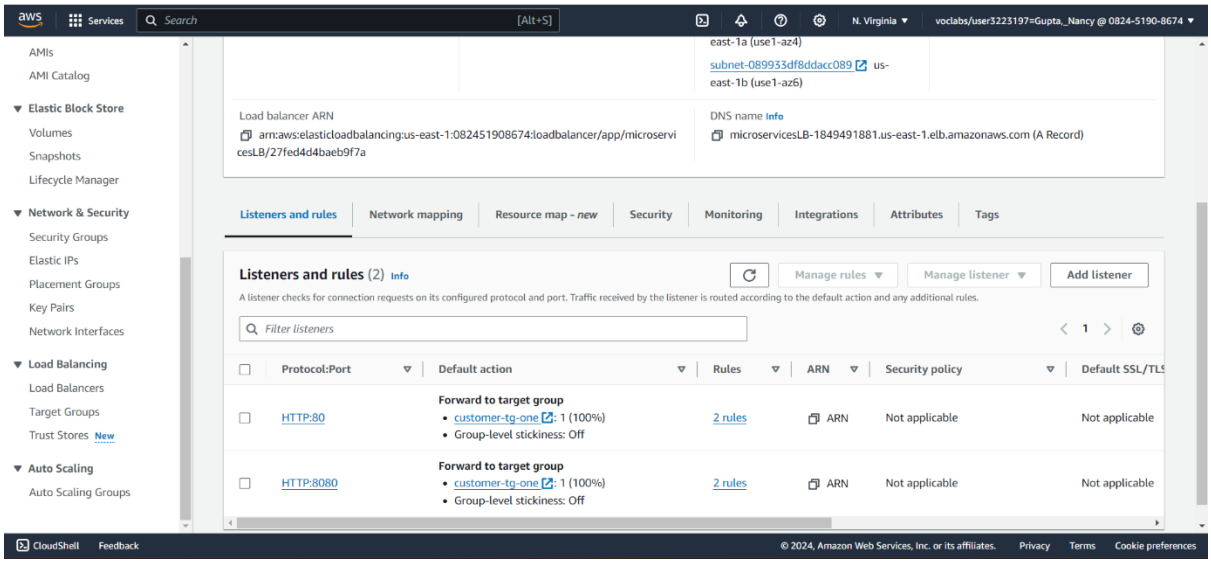
   - Navigate to the Amazon EC2 console.
   - In the navigation pane, choose **Load Balancers**.



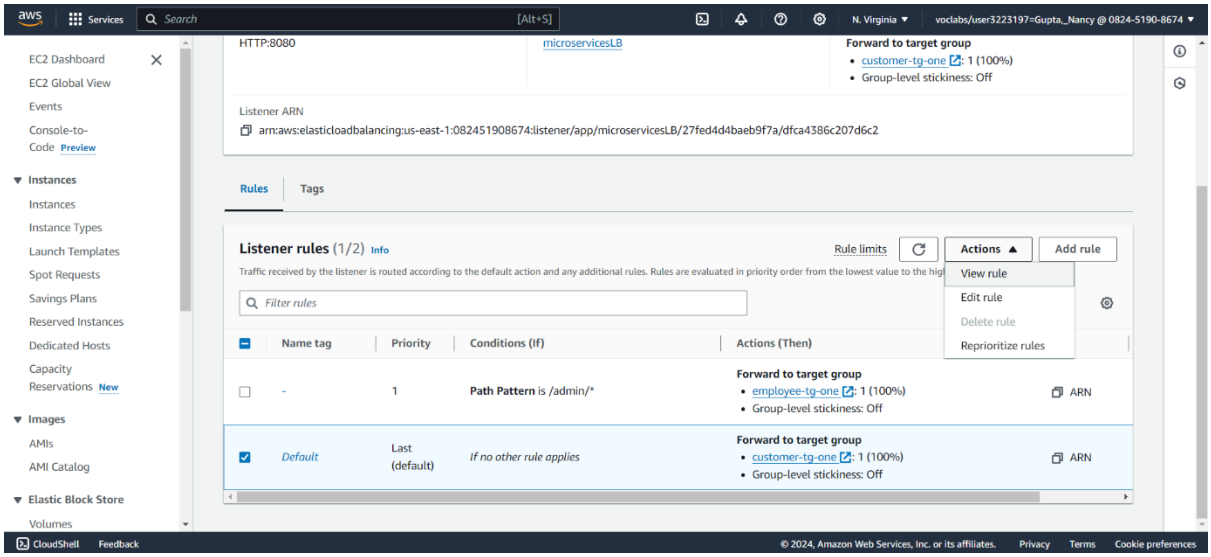
   - Choose the link for the **microservicesLB** load balancer.

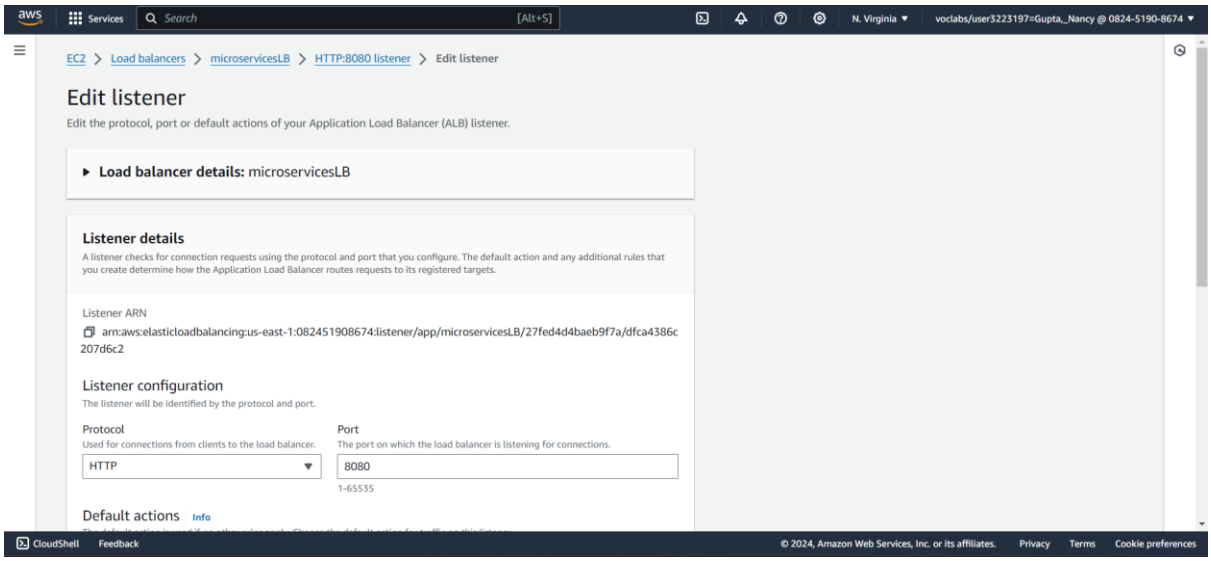


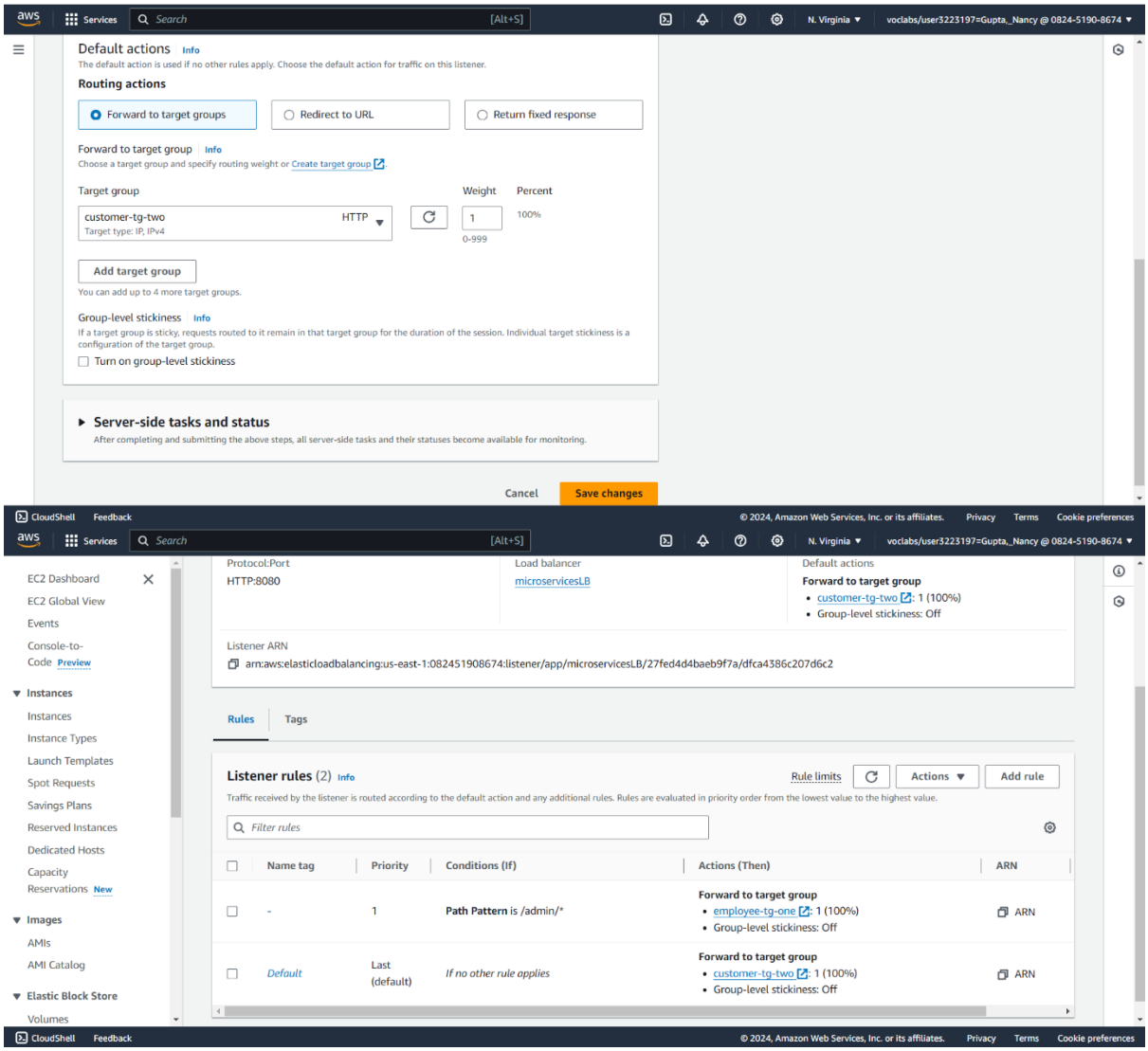
- On the **Listeners and rules** tab, choose the **HTTP:8080** link.



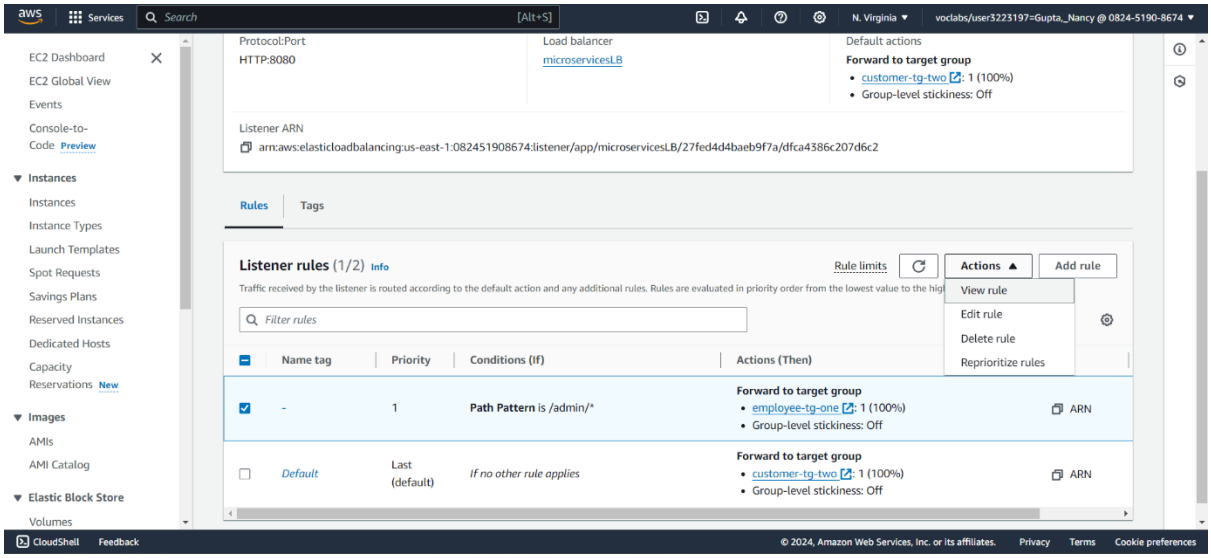
- In the **Listener rules** panel, verify that the **Default** rule forwards to **customer-tg-two**. If it doesn't:
  - Select the rule.
  - From the **Actions** menu, choose **Edit rule**.
  - Set **Forward to target group** to **customer-tg-two**, and choose **Save changes**.

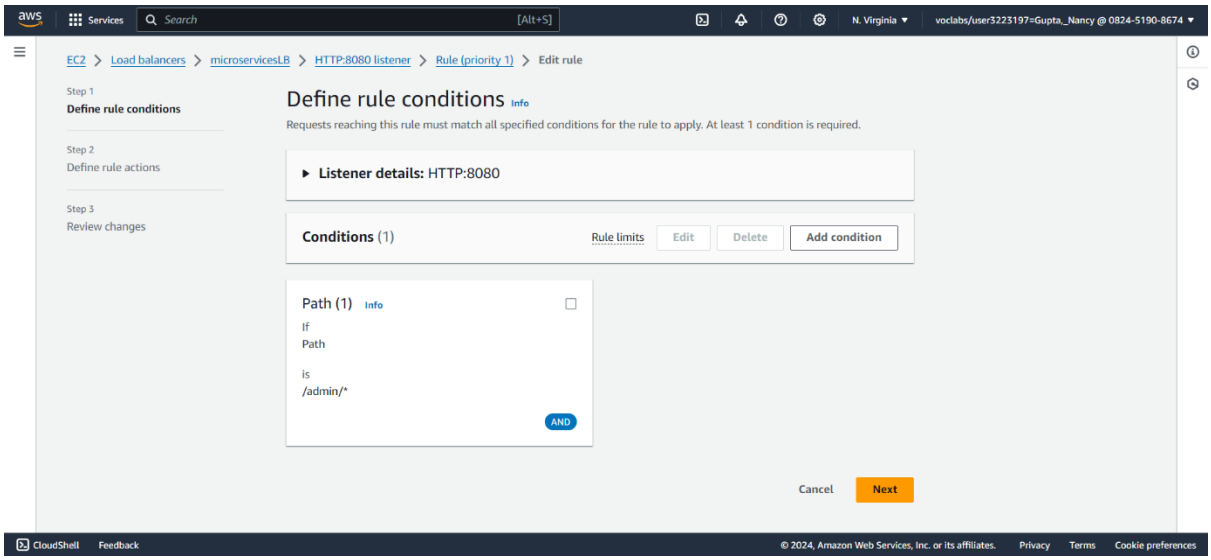

Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
HTTP:8080
microservicesLB
Forward to target group
customer-tg-one: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2
Rules
Tags
Listener rules (1/2) Info
Rule limits
Actions
Add rule
Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the hig
View rule
Edit rule
Delete rule
Reprioritize rules
Filter rules
Name tag
Priority
Conditions (If)
Actions (Then)
-
1
Path Pattern is /admin/*
Forward to target group
employee-tg-one: 1 (100%)
Group-level stickiness: Off
ARN
Default
Last (default)
If no other rule applies
Forward to target group
customer-tg-one: 1 (100%)
Group-level stickiness: Off
ARN
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

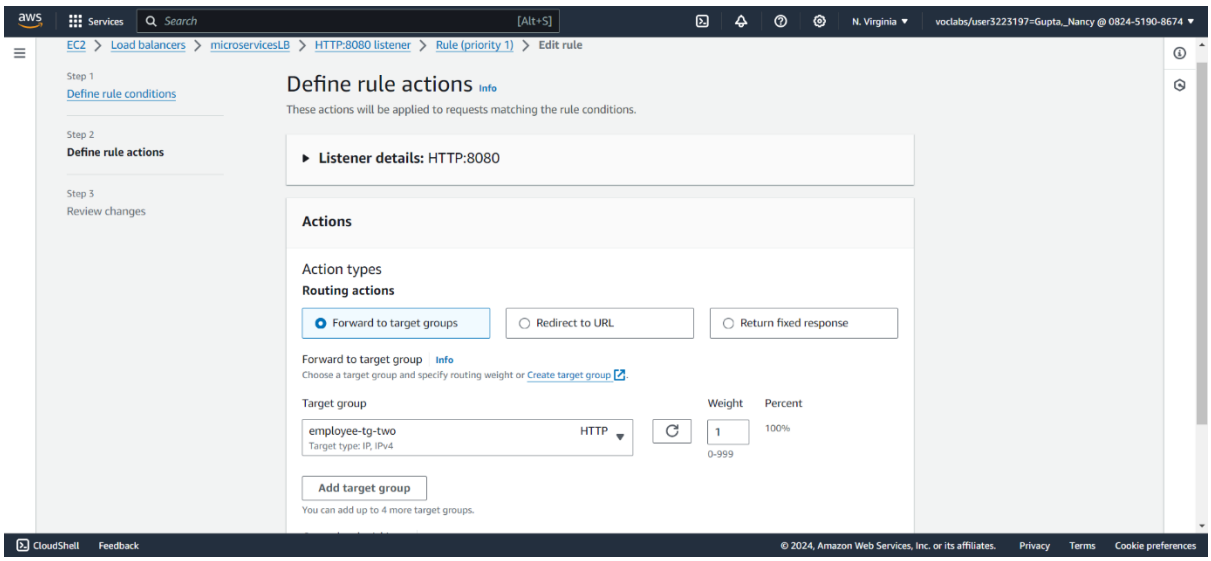

Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
EC2 > Load balancers > microservicesLB > HTTP:8080 listener > Edit listener
Edit listener
Edit the protocol, port or default actions of your Application Load Balancer (ALB) listener.
Load balancer details: microservicesLB
Listener details
A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2
Listener configuration
The listener will be identified by the protocol and port.
Protocol
Used for connections from clients to the load balancer.
HTTP
Port
The port on which the load balancer is listening for connections.
8080
1-65535
Default actions Info
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

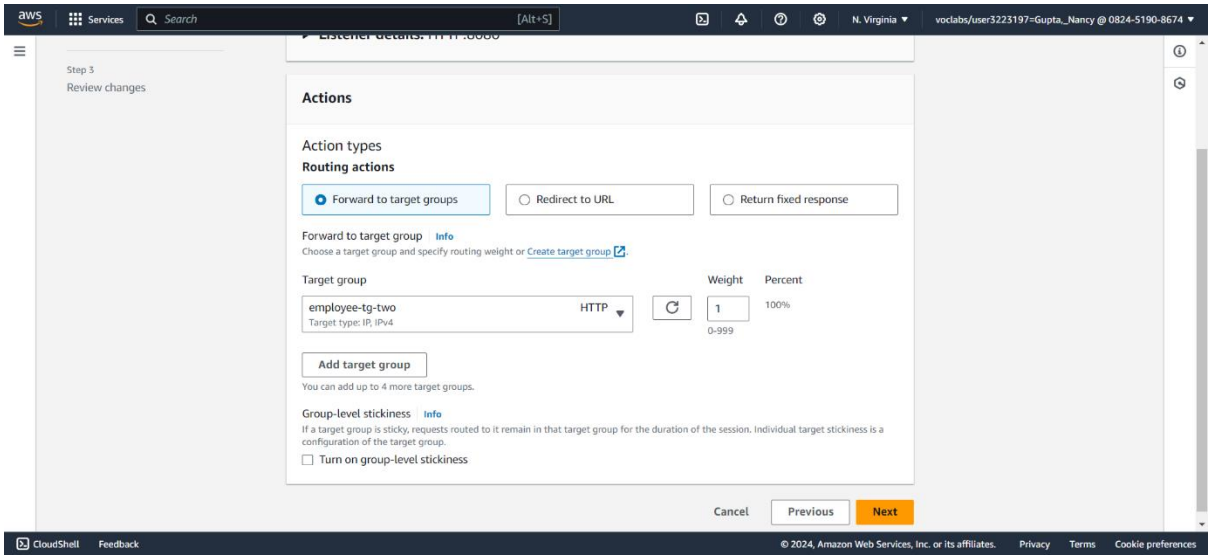


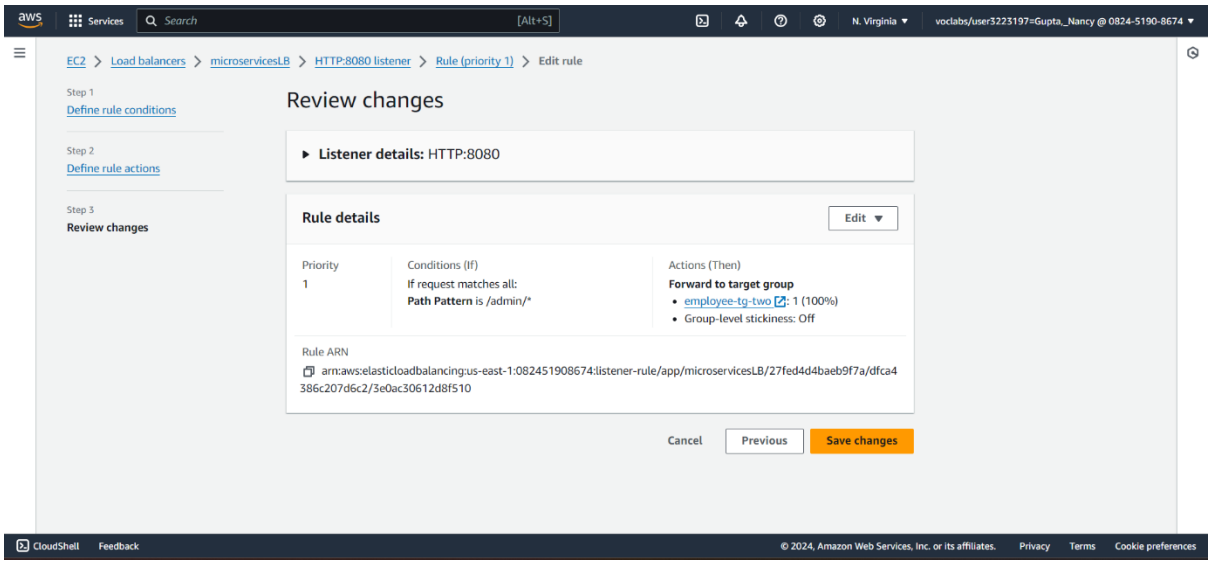
- Still on the **HTTP:8080** page, in the **Listener rules** panel, verify that the rule with **Path Pattern is /admin/** forwards to **employee-tg-two**. If it doesn't:
  - Select the rule.
  - From the **Actions** menu, choose **Edit rule**, and then choose **Next**.
  - In the **Actions** panel, set **Forward to target group** to **employee-tg-two**.
  - Choose **Next**, and then choose **Save changes**.

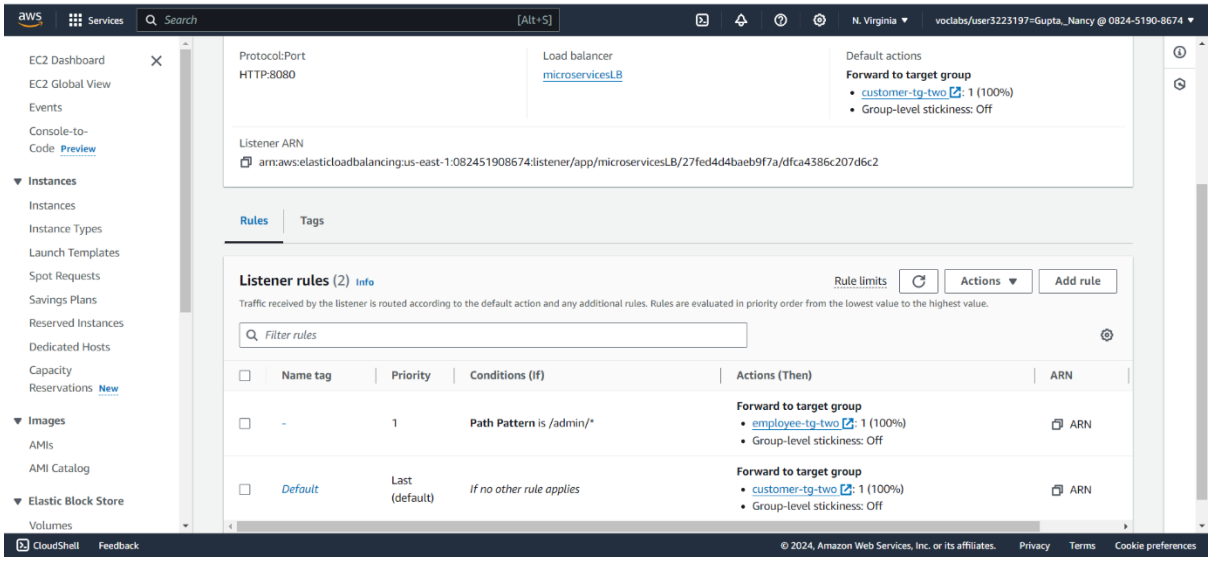

Protocol:Port
HTTP:8080
Load balancer
microservicesLB
Default actions
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2
Rules
Tags
Listener rules (1/2) Info
Rule limits
Actions
Add rule
View rule
Edit rule
Delete rule
Reprioritize rules
Filter rules
Name tag
Priority
Conditions (If)
Actions (Then)
Path Pattern is /admin/*
Forward to target group
employee-tg-one: 1 (100%)
Group-level stickiness: Off
ARN
Default
Last (default)
If no other rule applies
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
ARN


EC2 > Load balancers > microservicesLB > HTTP:8080 listener > Rule (priority 1) > Edit rule
Step 1
Define rule conditions
Step 2
Define rule actions
Step 3
Review changes
Define rule conditions Info
Requests reaching this rule must match all specified conditions for the rule to apply. At least 1 condition is required.
Listener details: HTTP:8080
Conditions (1)
Rule limits
Edit
Delete
Add condition
Path (1) Info
If
Path
is
/admin/*
AND
Cancel
Next


EC2 > Load balancers > microservicesLB > HTTP:8080 listener > Rule (priority 1) > Edit rule
Step 1
Define rule conditions
Step 2
Define rule actions
Step 3
Review changes
Define rule actions Info
These actions will be applied to requests matching the rule conditions.
Listener details: HTTP:8080
Actions
Action types
Routing actions
Forward to target groups
Redirect to URL
Return fixed response
Forward to target group Info
Choose a target group and specify routing weight or Create target group.
Target group
Weight
Percent
employee-tg-two
Target type: IP, IPv4
HTTP
1
0-999
100%
Add target group
You can add up to 4 more target groups.



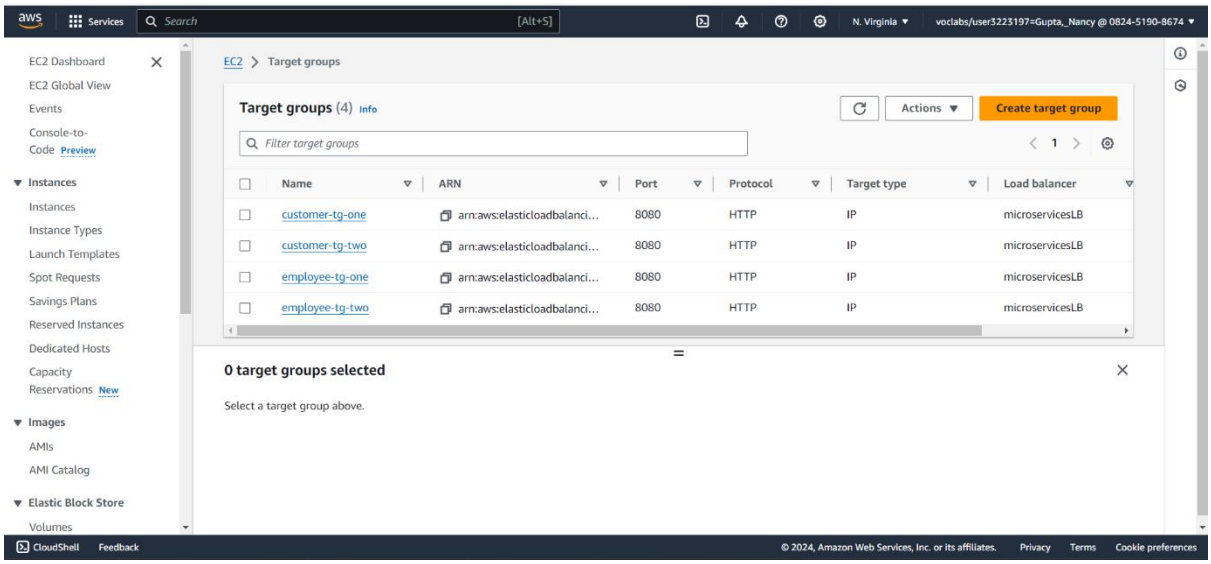




5. Confirm that all target groups are reassociated.

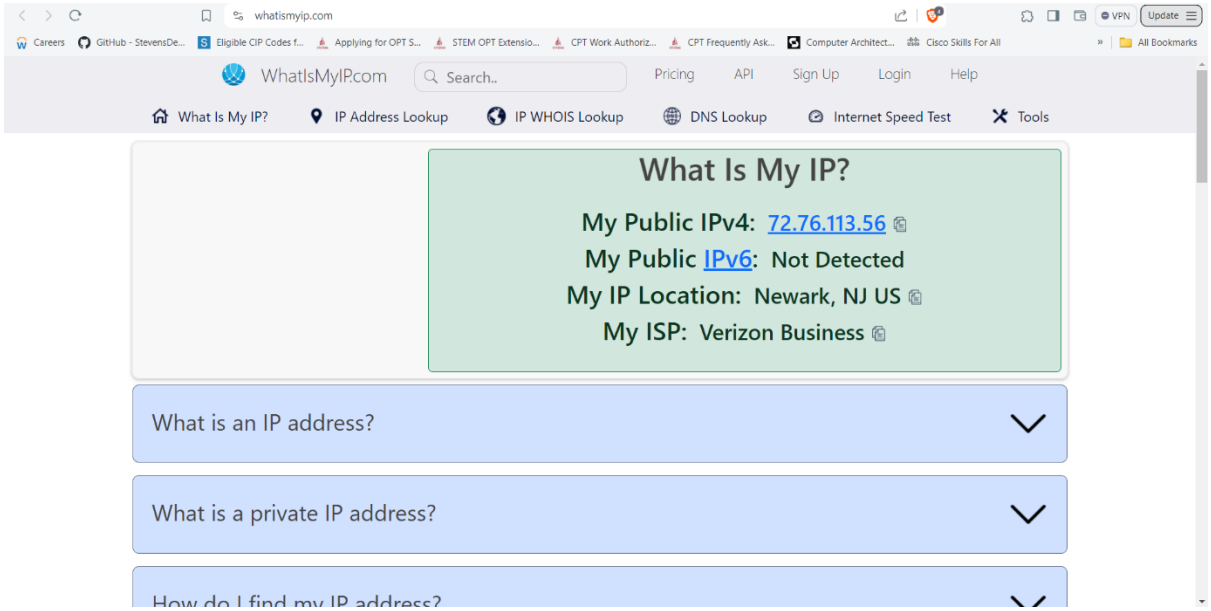
   - In the navigation pane of the Amazon EC2 console, choose **Target Groups**.

- Confirm that all four target groups are now associated with the *microservicesLB* load balancer.



2. Discover your public IPv4 address.

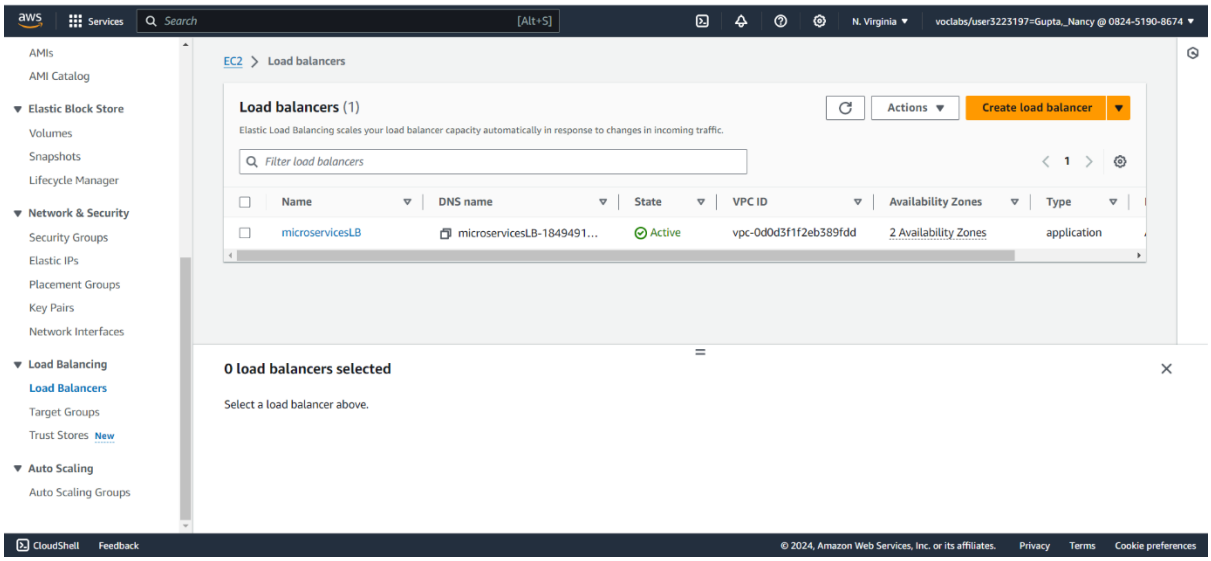
Tip: One resource you can use to do this is https://www.whatismyip.com.

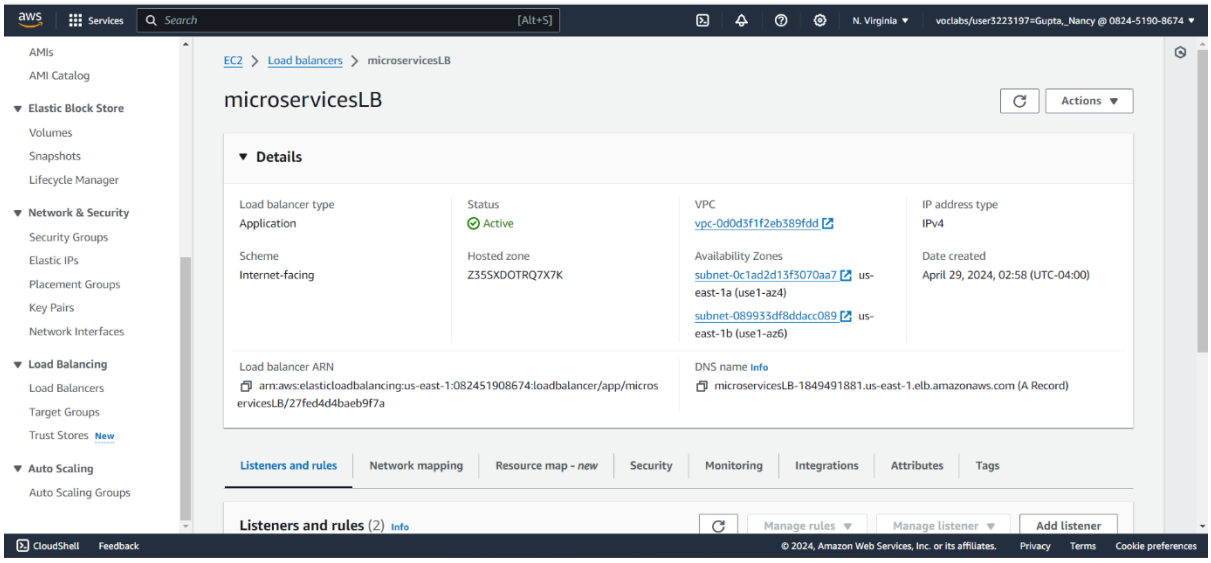


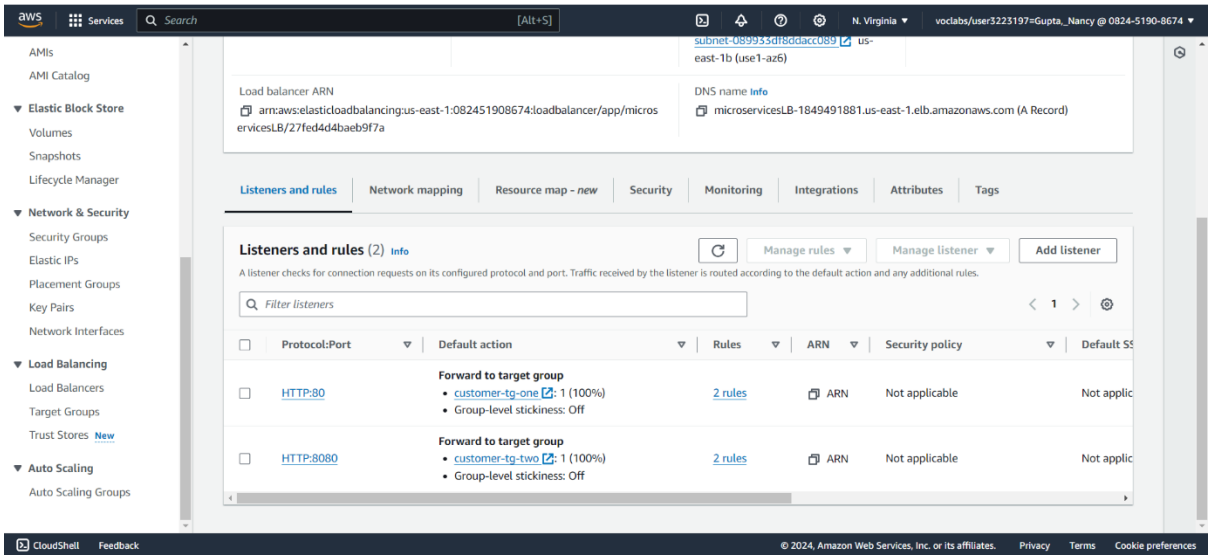
3. Edit the rules for the **HTTP:80** listener.

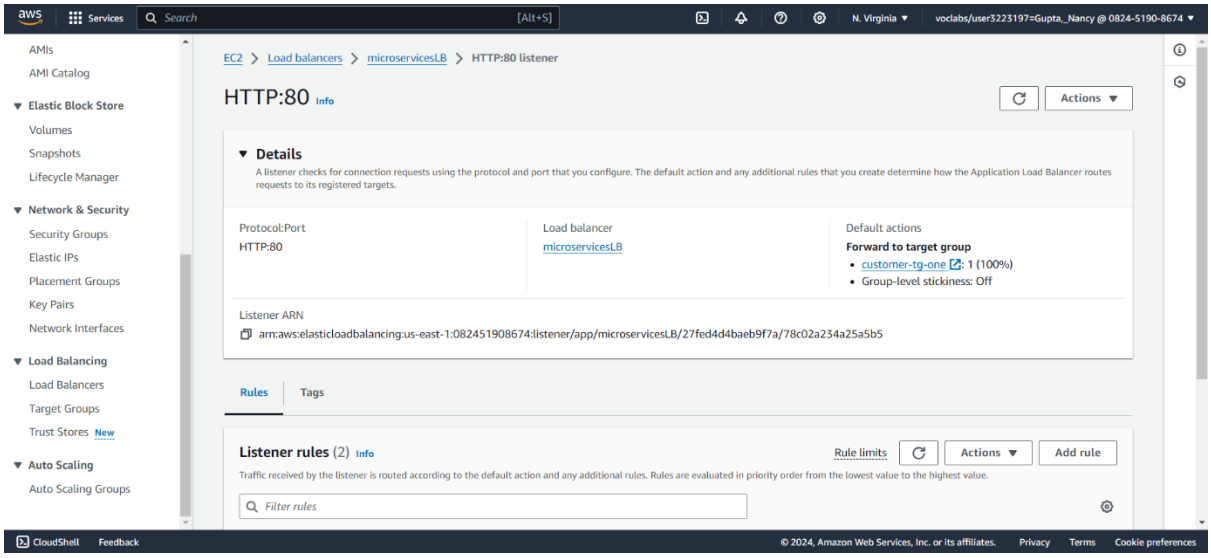
For the rule that currently has "IF Path is /admin/*" in the details, add a second condition to route the user to the target groups only if the source IP of the request is your IP address.

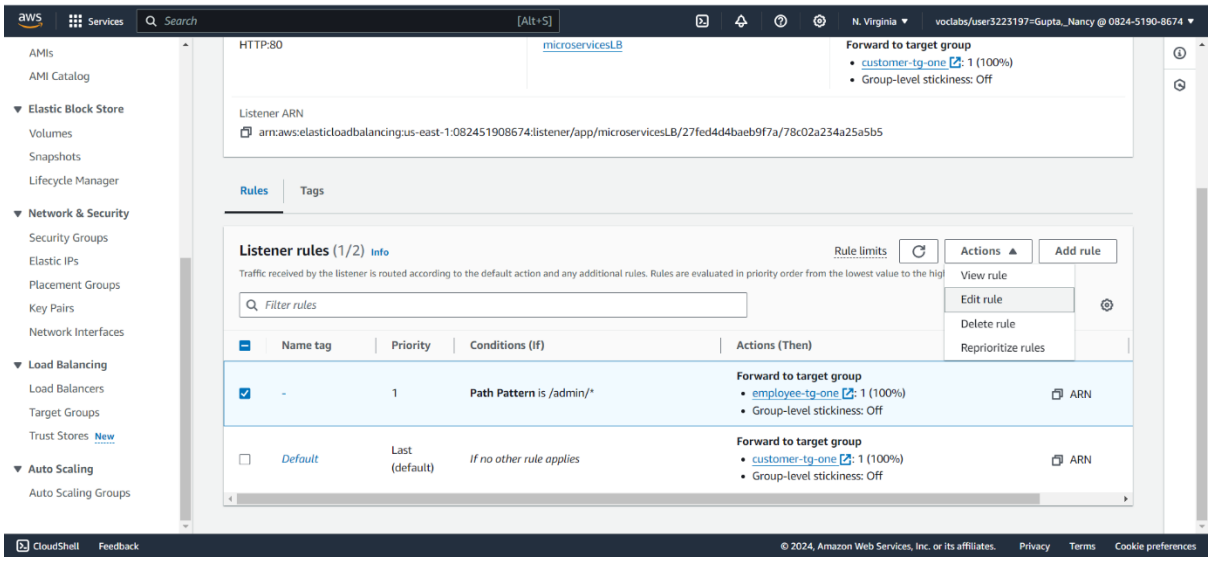
Tip: For the source IP, paste in your public IPv4 address and then add /32. The following image shows an example:

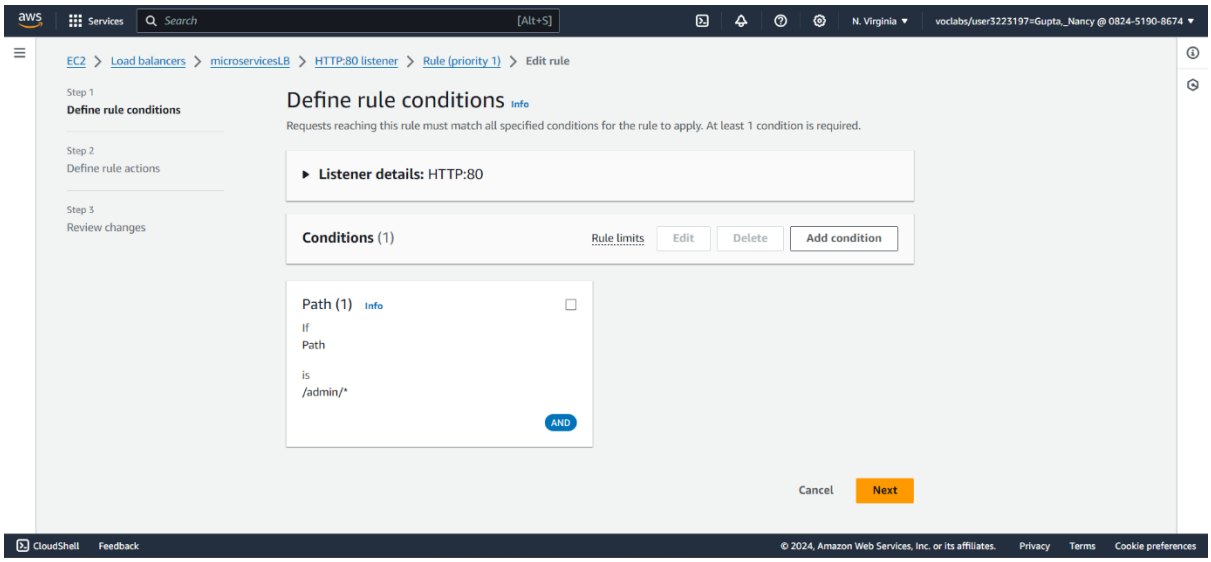

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Trust Stores New
Auto Scaling
Auto Scaling Groups
EC2 > Load balancers
Load balancers (1)
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.
Actions
Create load balancer
Filter load balancers
Name
DNS name
State
VPC ID
Availability Zones
Type
microservicesLB
microservicesLB-1849491...
Active
vpc-0d0d3f1f2eb389fdd
2 Availability Zones
application
0 load balancers selected
Select a load balancer above.
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

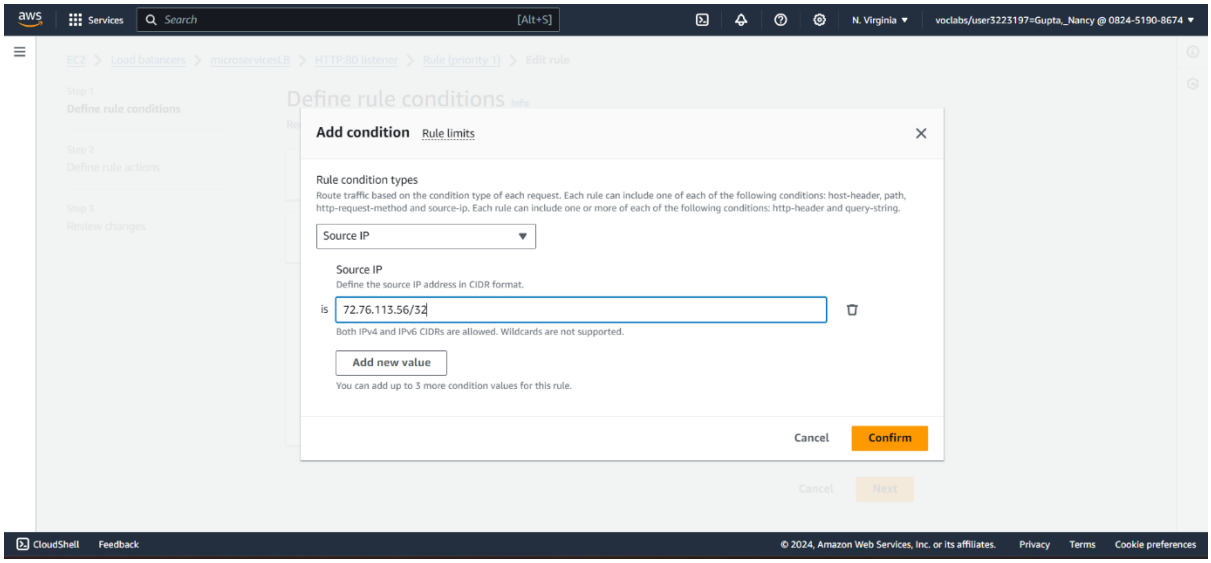

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
EC2 > Load balancers > microservicesLB
microservicesLB
Actions
Details
Load balancer type
Application
Status
Active
VPC
vpc-0d0d3f1f2eb389fdd
IP address type
IPv4
Scheme
Internet-facing
Hosted zone
Z35SXDOTRQ7X7K
Availability Zones
subnet-0c1ad2d13f3070aa7 us-east-1a (use1-az4)
subnet-089933df8ddacc089 us-east-1b (use1-az6)
Date created
April 29, 2024, 02:58 (UTC-04:00)
Load balancer ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:loadbalancer/app/microservicesLB/27fed4d4baeb9f7a
DNS name Info
microservicesLB-1849491881.us-east-1.elb.amazonaws.com (A Record)
Listeners and rules
Network mapping
Resource map - new
Security
Monitoring
Integrations
Attributes
Tags
Listeners and rules (2) Info
Manage rules
Manage listener
Add listener
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

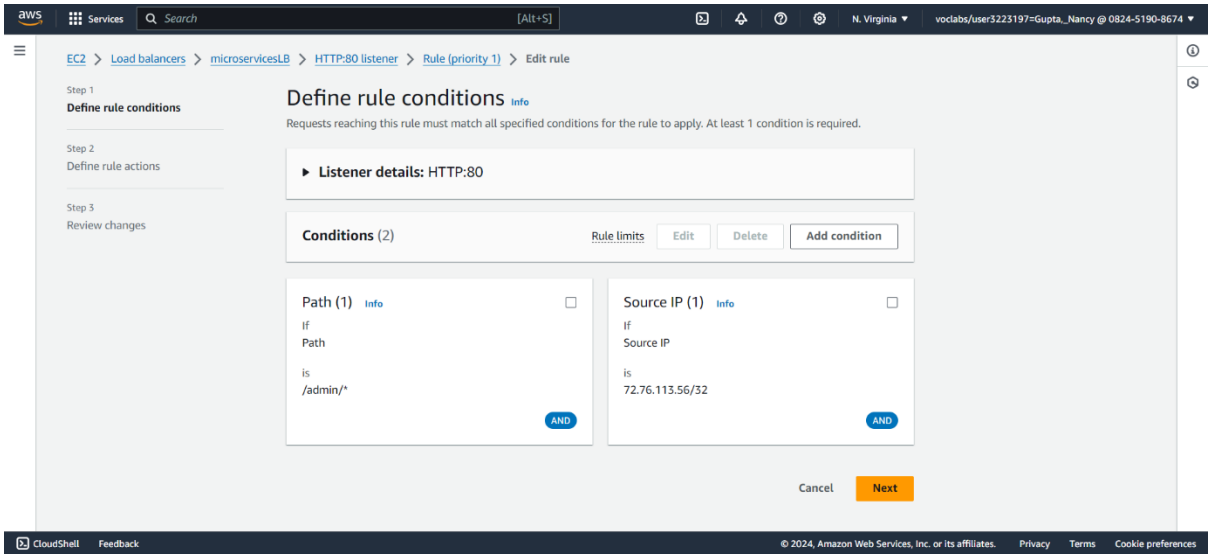

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
subnet-089933df8ddacc089 us-east-1b (use1-az6)
Load balancer ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:loadbalancer/app/microservicesLB/27fed4d4baeb9f7a
DNS name Info
microservicesLB-1849491881.us-east-1.elb.amazonaws.com (A Record)
Listeners and rules
Network mapping
Resource map - new
Security
Monitoring
Integrations
Attributes
Tags
Listeners and rules (2) Info
Manage rules
Manage listener
Add listener
A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.
Filter listeners
Protocol:Port
Default action
Rules
ARN
Security policy
HTTP:80
Forward to target group
customer-tg-one: 1 (100%)
Group-level stickiness: Off
2 rules
ARN
Not applicable
HTTP:8080
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
2 rules
ARN
Not applicable
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

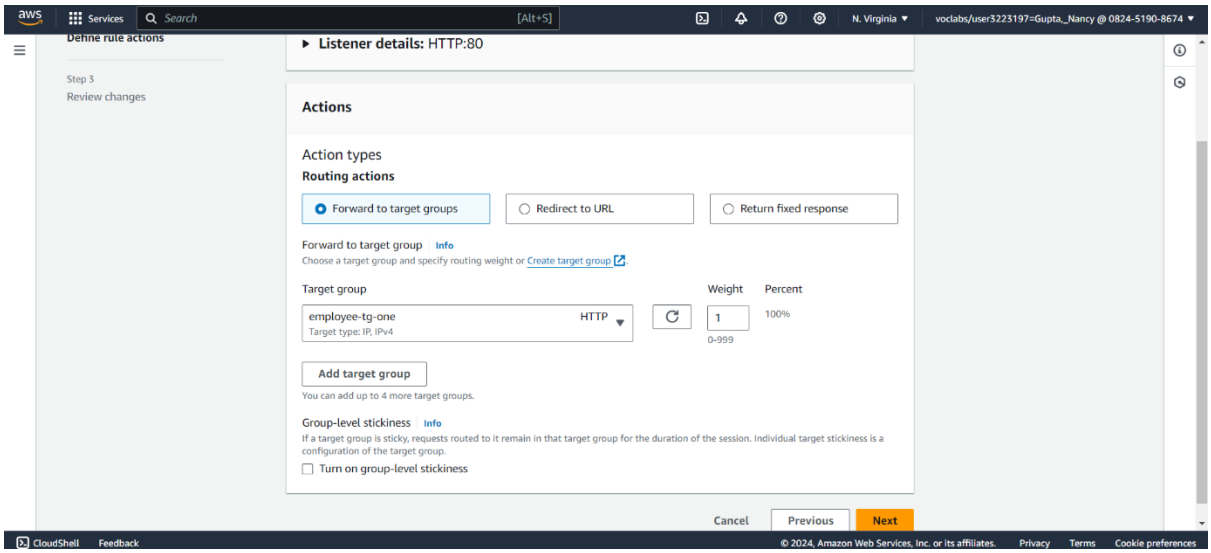

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Trust Stores New
Auto Scaling
Auto Scaling Groups
EC2 > Load balancers > microservicesLB > HTTP:80 listener
HTTP:80 Info
Actions
Details
A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.
Protocol:Port
HTTP:80
Load balancer
microservicesLB
Default actions
Forward to target group
customer-tg-one: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/78c02a234a25a5b5
Rules
Tags
Listener rules (2) Info
Rule limits
Actions
Add rule
Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.
Filter rules
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

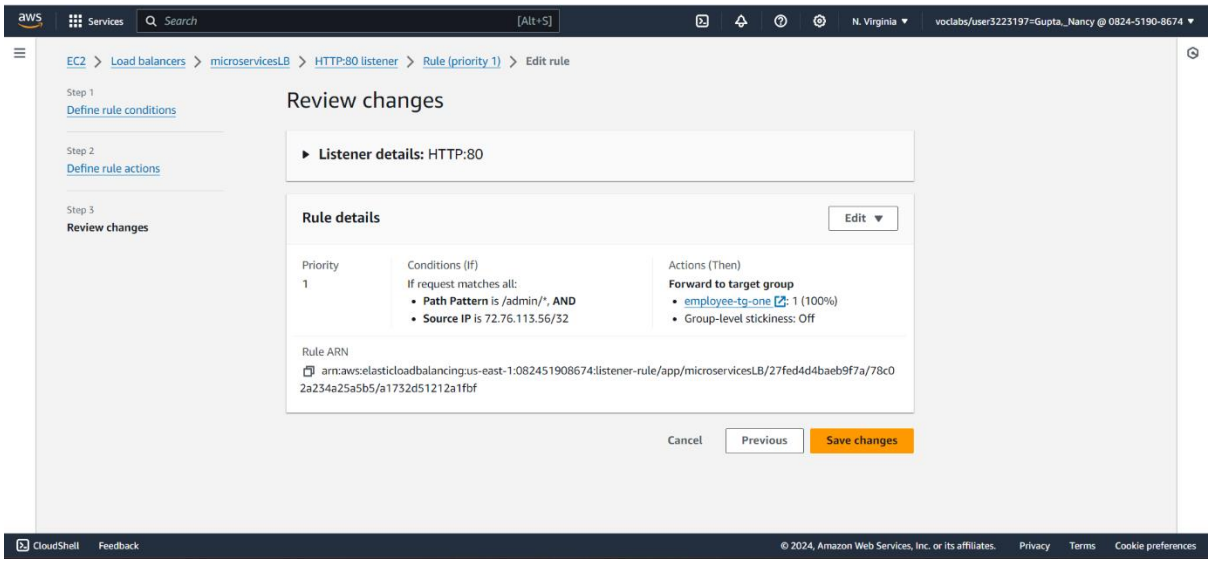

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
HTTP:80
microservicesLB
Forward to target group
customer-tg-one: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/78c02a234a25a5b5
Rules
Tags
Listener rules (1/2) Info
Rule limits
Actions
Add rule
View rule
Edit rule
Delete rule
Reprioritize rules
Filter rules
Name tag
Priority
Conditions (If)
Actions (Then)
-
1
Path Pattern is /admin/*
Forward to target group
employee-tg-one: 1 (100%)
Group-level stickiness: Off
ARN
Default
Last (default)
If no other rule applies
Forward to target group
customer-tg-one: 1 (100%)
Group-level stickiness: Off
ARN
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

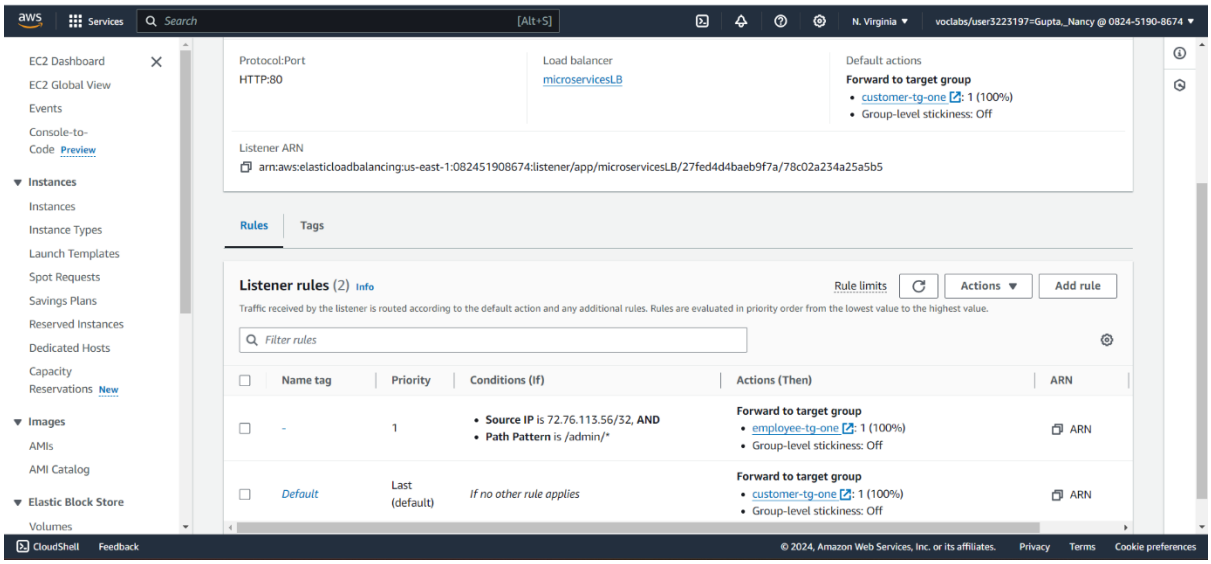

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
EC2 > Load balancers > microservicesLB > HTTP:80 listener > Rule (priority 1) > Edit rule
Step 1
Define rule conditions
Step 2
Define rule actions
Step 3
Review changes
Define rule conditions Info
Requests reaching this rule must match all specified conditions for the rule to apply. At least 1 condition is required.
Listener details: HTTP:80
Conditions (1)
Rule limits
Edit
Delete
Add condition
Path (1) Info
If
Path
is
/admin/*
AND
Cancel
Next
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Add condition
Rule limits
Rule condition types
Route traffic based on the condition type of each request. Each rule can include one of each of the following conditions: host-header, path, http-request-method and source-ip. Each rule can include one or more of each of the following conditions: http-header and query-string.
Source IP
Source IP
Define the source IP address in CIDR format.
is
72.76.113.56/32
Both IPv4 and IPv6 CIDRs are allowed. Wildcards are not supported.
Add new value
You can add up to 3 more condition values for this rule.
Cancel
Confirm


EC2 > Load balancers > microservicesLB > HTTP:80 listener > Rule (priority 1) > Edit rule
Step 1
Define rule conditions
Step 2
Define rule actions
Step 3
Review changes
Define rule conditions
Info
Requests reaching this rule must match all specified conditions for the rule to apply. At least 1 condition is required.
Listener details: HTTP:80
Conditions (2)
Rule limits
Edit
Delete
Add condition
Path (1)
Info
If
Path
is
/admin/*
AND
Source IP (1)
Info
If
Source IP
is
72.76.113.56/32
AND
Cancel
Next

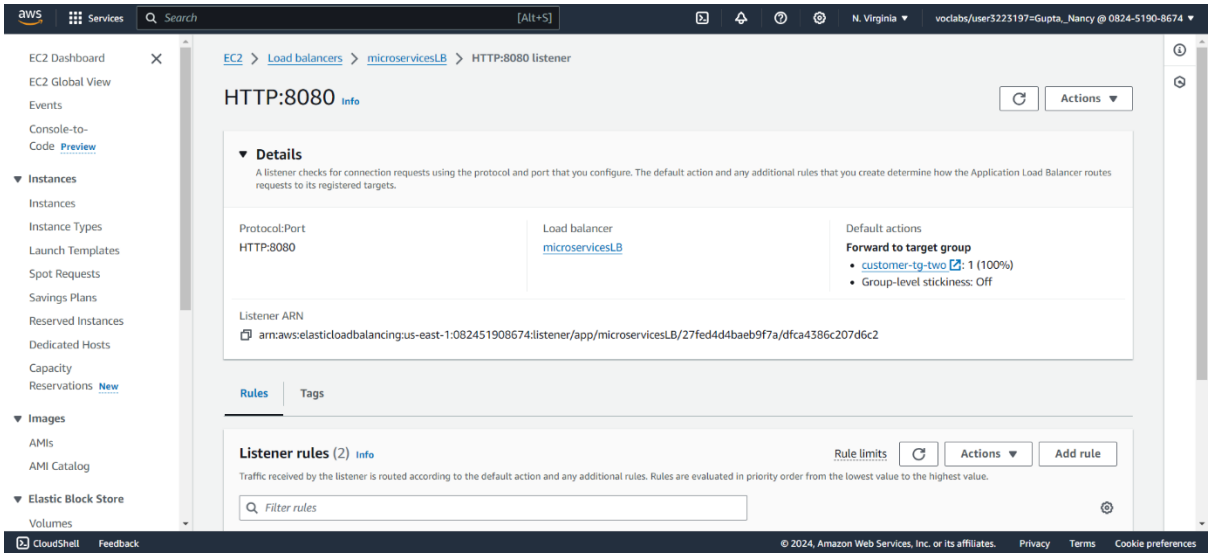

Define rule actions
Step 3
Review changes
Listener details: HTTP:80
Actions
Action types
Routing actions
Forward to target groups
Redirect to URL
Return fixed response
Forward to target group
Info
Choose a target group and specify routing weight or Create target group.
Target group
Weight
Percent
employee-tg-one
HTTP
Target type: IP, IPv4
1
100%
0-999
Add target group
You can add up to 4 more target groups.
Group-level stickiness
Info
If a target group is sticky, requests routed to it remain in that target group for the duration of the session. Individual target stickiness is a configuration of the target group.
Turn on group-level stickiness
Cancel
Previous
Next

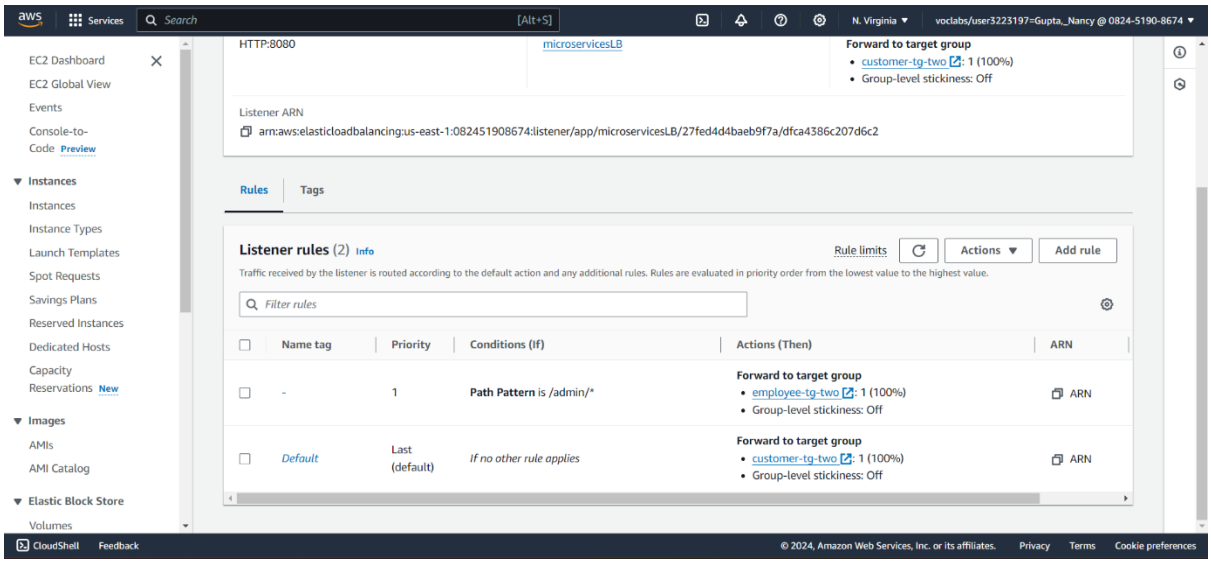


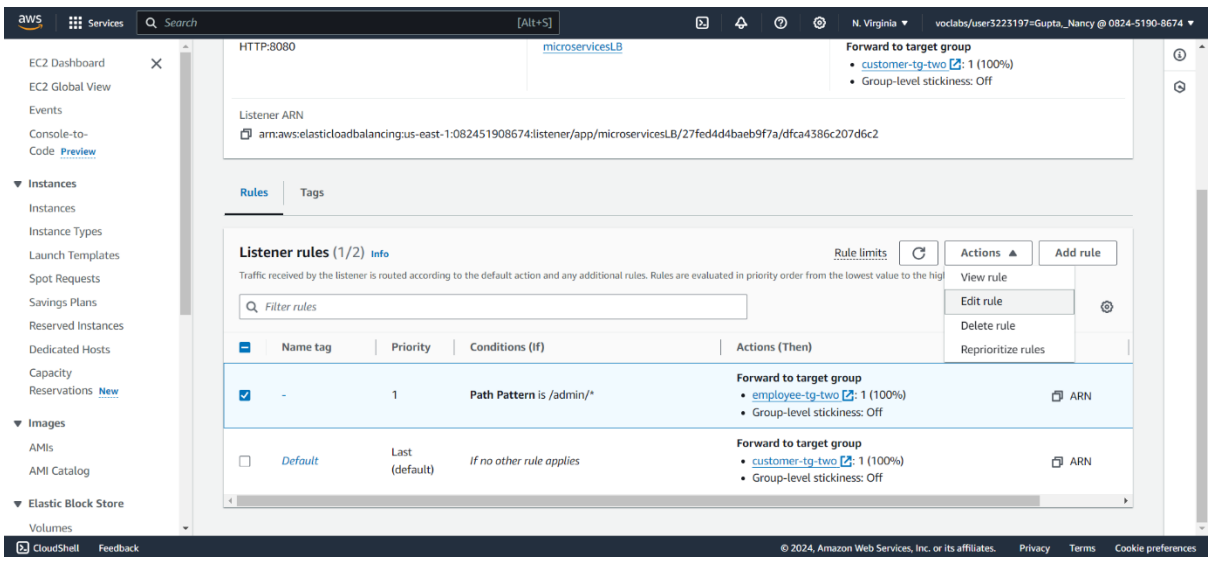


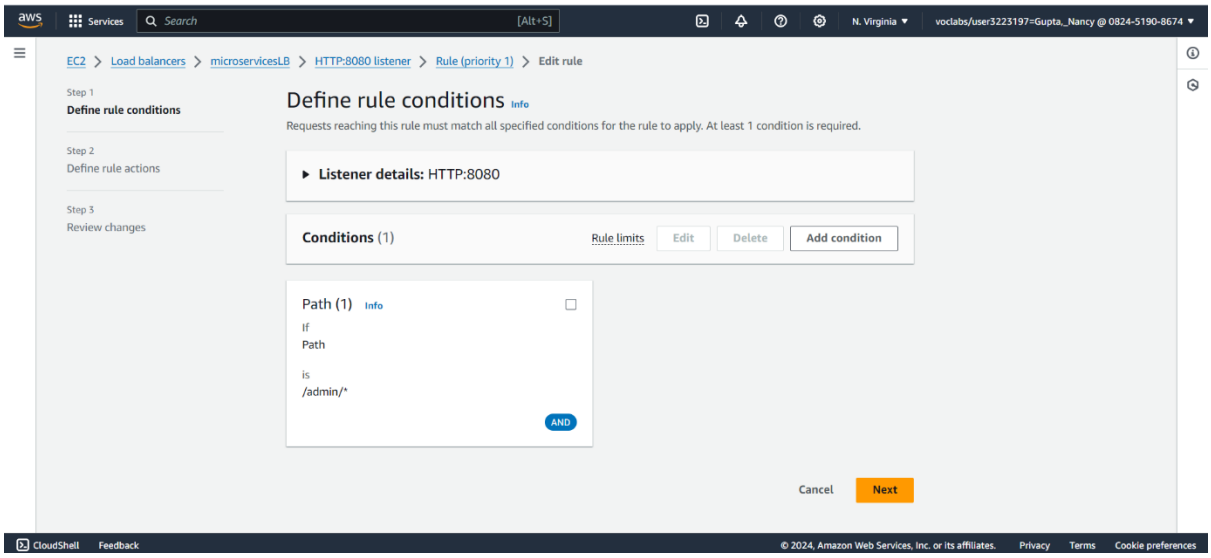
4. Edit the rules for the **HTTP:8080** listener.

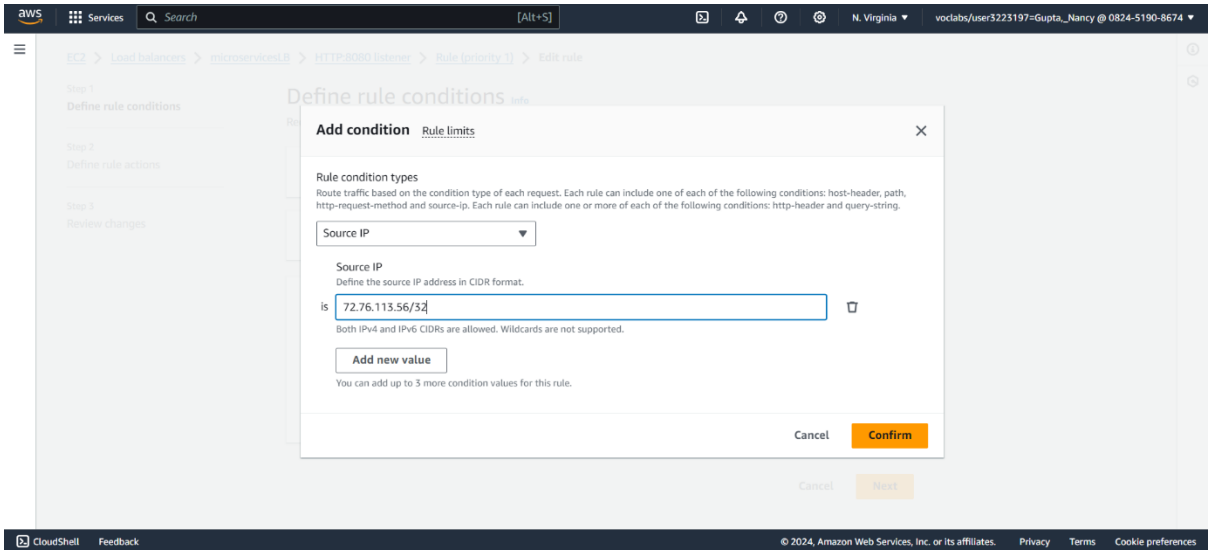
Edit the rules in the same way that you edited the rules for the **HTTP:80** listener. You want access to the employee target groups to be limited to your IP address.

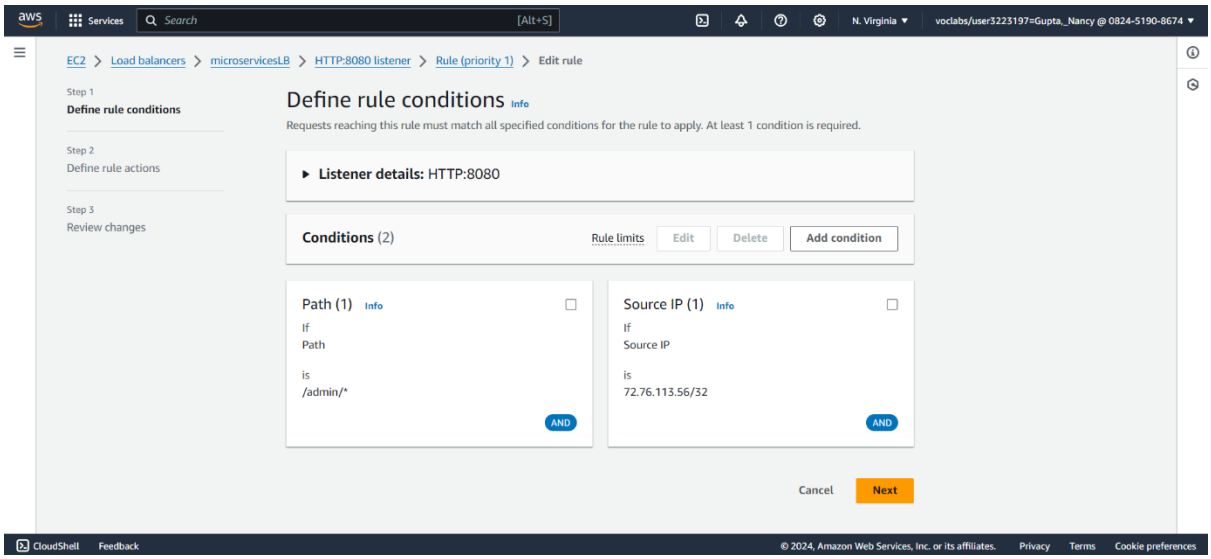

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
EC2 > Load balancers > microservicesLB > HTTP:8080 listener
HTTP:8080 Info
Actions
Details
A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.
Protocol:Port
HTTP:8080
Load balancer
microservicesLB
Default actions
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2
Rules
Tags
Listener rules (2) Info
Rule limits
Actions
Add rule
Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.
Filter rules
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

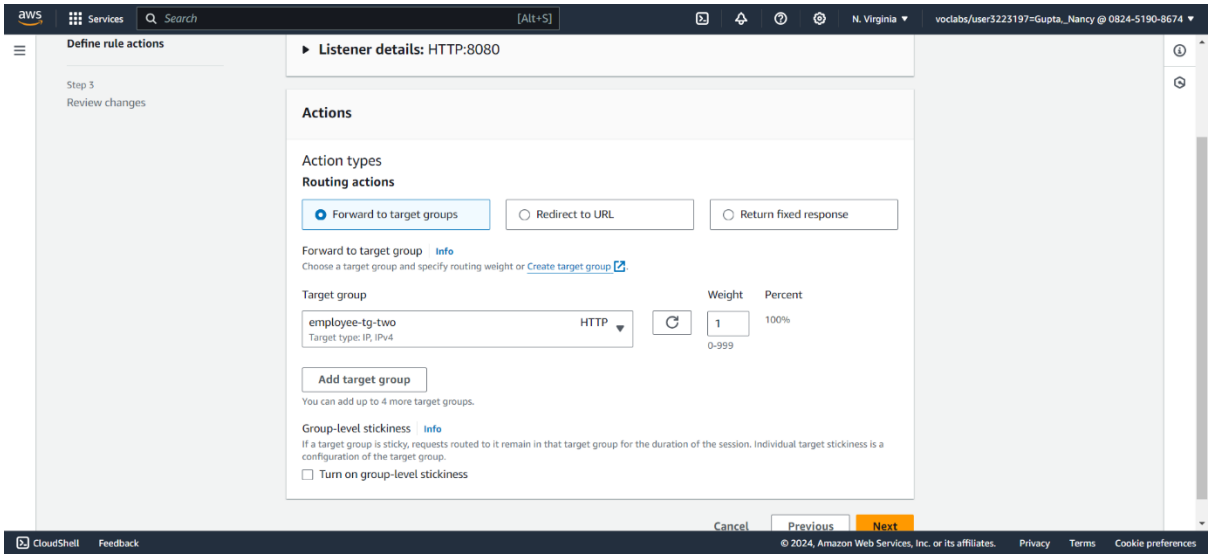

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
HTTP:8080
microservicesLB
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2
Rules
Tags
Listener rules (2) Info
Rule limits
Actions
Add rule
Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.
Filter rules
Name tag
Priority
Conditions (If)
Actions (Then)
ARN
-
1
Path Pattern is /admin/*
Forward to target group
employee-tg-two: 1 (100%)
Group-level stickiness: Off
ARN
Default
Last (default)
If no other rule applies
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
ARN
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

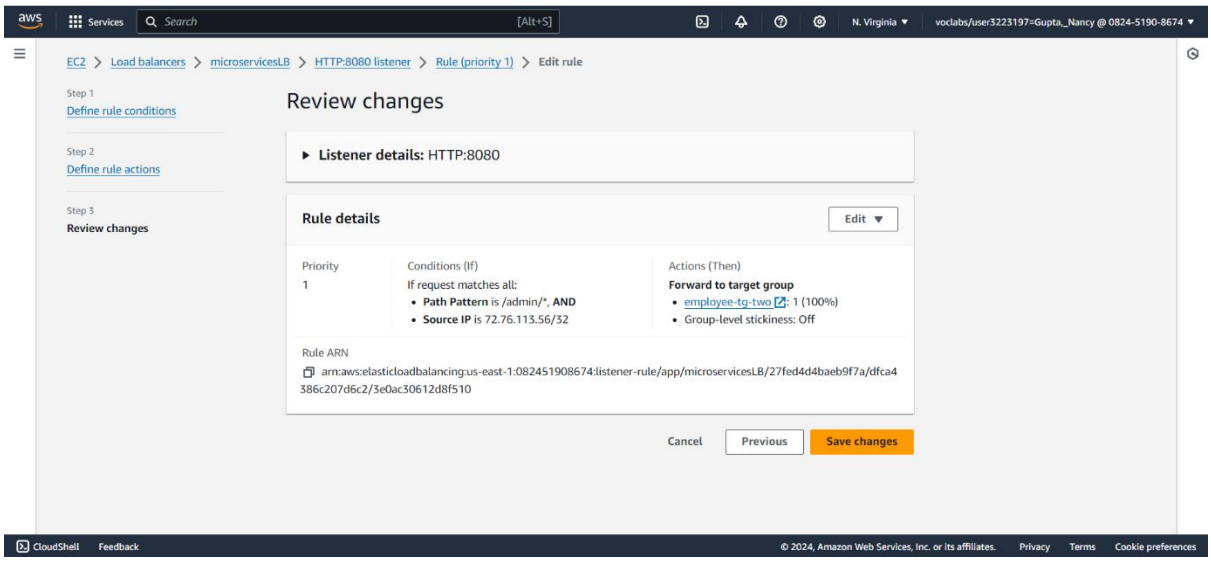

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
HTTP:8080
microservicesLB
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2
Rules
Tags
Listener rules (1/2) Info
Rule limits
Actions
Add rule
View rule
Edit rule
Delete rule
Reprioritize rules
Filter rules
Name tag
Priority
Conditions (If)
Actions (Then)
ARN
-
1
Path Pattern is /admin/*
Forward to target group
employee-tg-two: 1 (100%)
Group-level stickiness: Off
ARN
Default
Last (default)
If no other rule applies
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
ARN
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

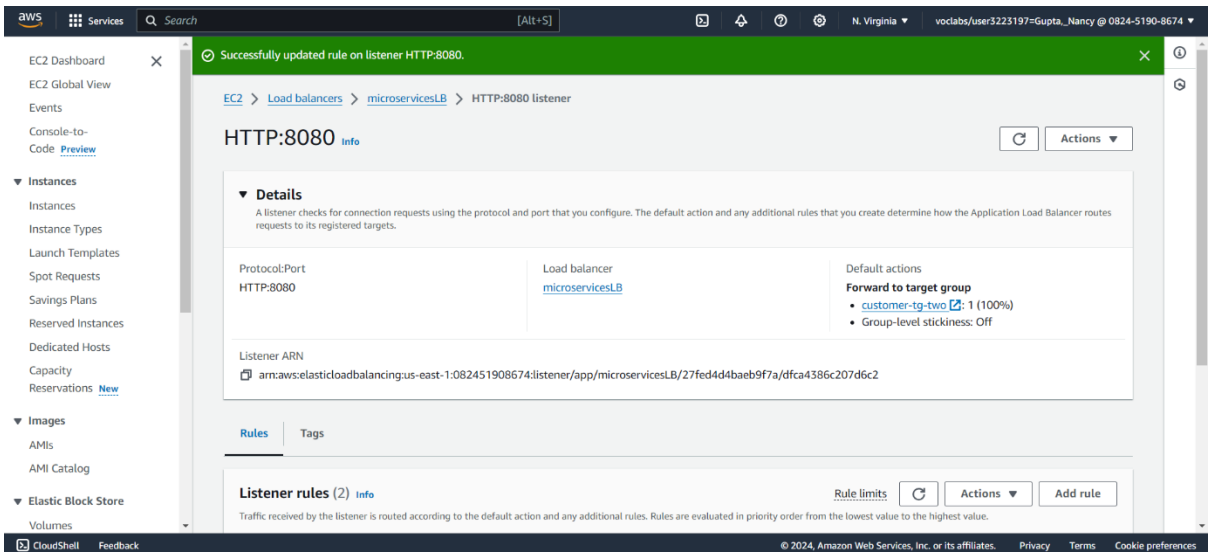

Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
EC2 > Load balancers > microservicesLB > HTTP:8080 listener > Rule (priority 1) > Edit rule
Step 1
Define rule conditions
Step 2
Define rule actions
Step 3
Review changes
Define rule conditions Info
Requests reaching this rule must match all specified conditions for the rule to apply. At least 1 condition is required.
Listener details: HTTP:8080
Conditions (1)
Rule limits
Edit
Delete
Add condition
Path (1) Info
If
Path
is
/admin/*
AND
Cancel
Next
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

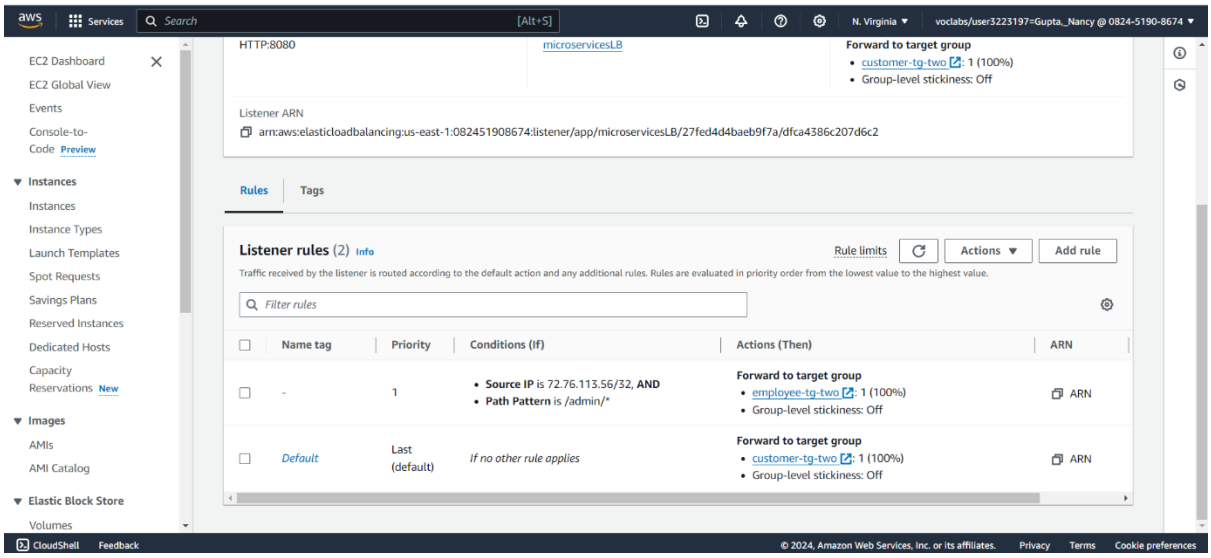

Add condition
Rule limits
Rule condition types
Route traffic based on the condition type of each request. Each rule can include one of each of the following conditions: host-header, path, http-request-method and source-ip. Each rule can include one or more of each of the following conditions: http-header and query-string.
Source IP
Source IP
Define the source IP address in CIDR format.
is
72.76.113.56/32
Both IPv4 and IPv6 CIDRs are allowed. Wildcards are not supported.
Add new value
You can add up to 3 more condition values for this rule.
Cancel
Confirm


Services
Search
[Alt+S]
N. Virginia
voclabs/user3223197=Gupta,_Nancy @ 0824-5190-8674
EC2 > Load balancers > microservicesLB > HTTP:8080 listener > Rule (priority 1) > Edit rule
Step 1
Define rule conditions
Step 2
Define rule actions
Step 3
Review changes
Define rule conditions Info
Requests reaching this rule must match all specified conditions for the rule to apply. At least 1 condition is required.
Listener details: HTTP:8080
Conditions (2)
Rule limits
Edit
Delete
Add condition
Path (1) Info
If
Path
is
/admin/*
AND
Source IP (1) Info
If
Source IP
is
72.76.113.56/32
AND
Cancel
Next
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Define rule actions
Step 3
Review changes
Listener details: HTTP:8080
Actions
Action types
Routing actions
Forward to target groups
Redirect to URL
Return fixed response
Forward to target group Info
Choose a target group and specify routing weight or Create target group.
Target group
Weight
Percent
employee-tg-two
Target type: IP, IPv4
HTTP
1
100%
0-999
Add target group
You can add up to 4 more target groups.
Group-level stickiness Info
If a target group is sticky, requests routed to it remain in that target group for the duration of the session. Individual target stickiness is a configuration of the target group.
Turn on group-level stickiness
Cancel
Previous
Next
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

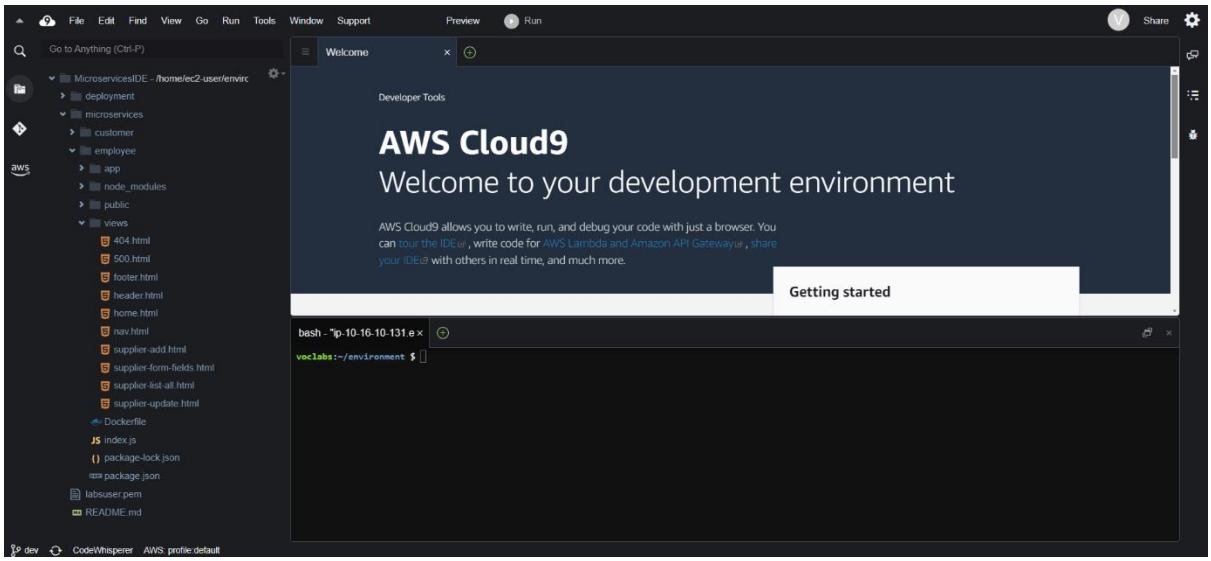

EC2 > Load balancers > microservicesLB > HTTP:8080 listener > Rule (priority 1) > Edit rule
Step 1
Define rule conditions
Step 2
Define rule actions
Step 3
Review changes
Review changes
Listener details: HTTP:8080
Rule details
Edit
Priority
1
Conditions (If)
If request matches all:
Path Pattern is /admin/*, AND
Source IP is 72.76.113.56/32
Actions (Then)
Forward to target group
employee-tg-two: 1 (100%)
Group-level stickiness: Off
Rule ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener-rule/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2/3e0ac30612d8f510
Cancel
Previous
Save changes
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Successfully updated rule on listener HTTP:8080.
EC2 > Load balancers > microservicesLB > HTTP:8080 listener
HTTP:8080 Info
Actions
Details
A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.
Protocol:Port
HTTP:8080
Load balancer
microservicesLB
Default actions
Forward to target group
customer-tg-two: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2
Rules
Tags
Listener rules (2) Info
Rule limits
Actions
Add rule
Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

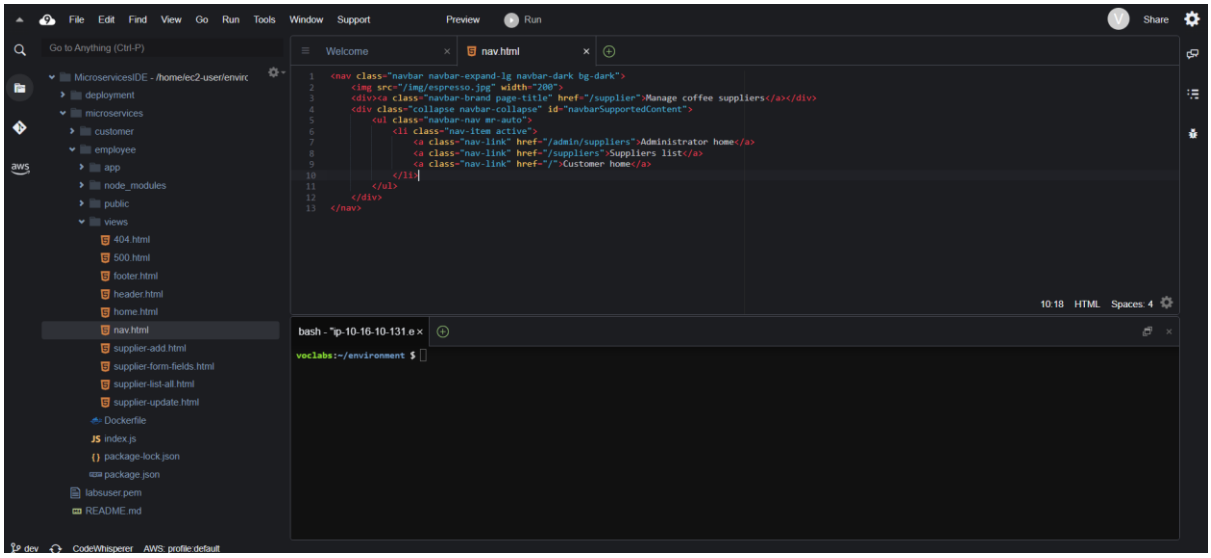


Task 9.2: Adjust the UI for the *employee* microservice and push the updated image to Amazon ECR

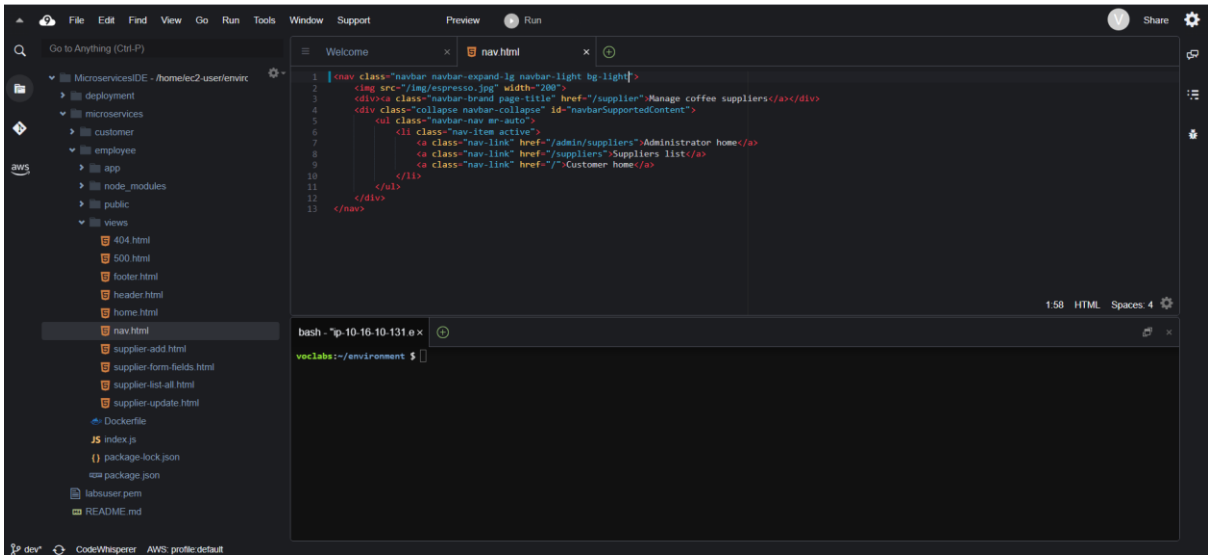
In this task, you will adjust the deployed microservices.



1. Edit the employee/views/**nav.html** file.

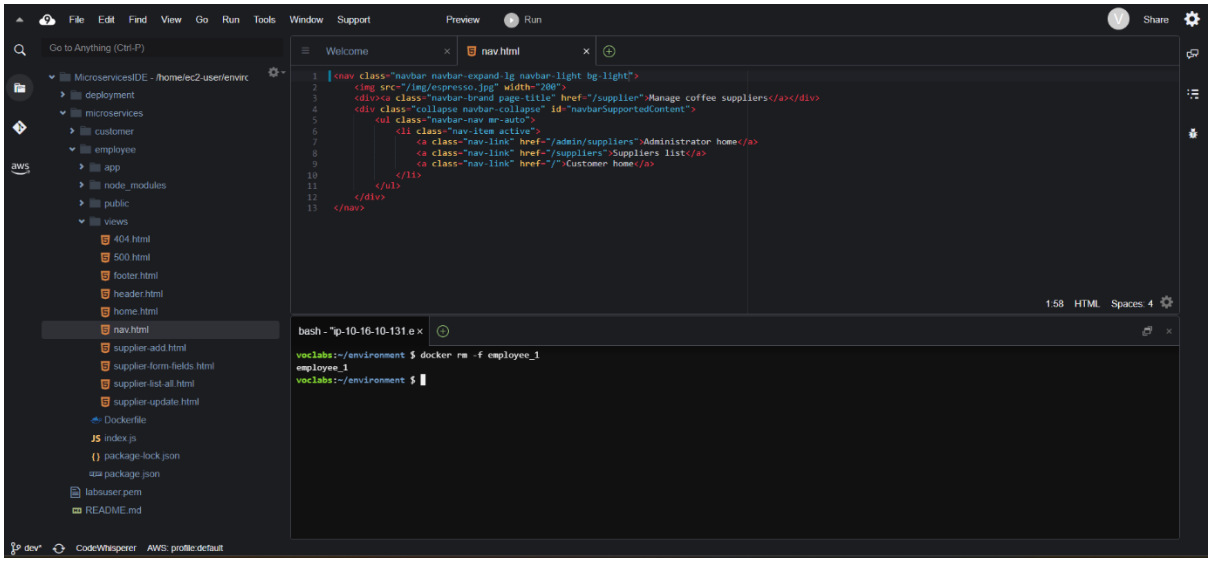


- On line 1, change navbar-dark bg-dark to navbar-light bg-light
- Save the change.

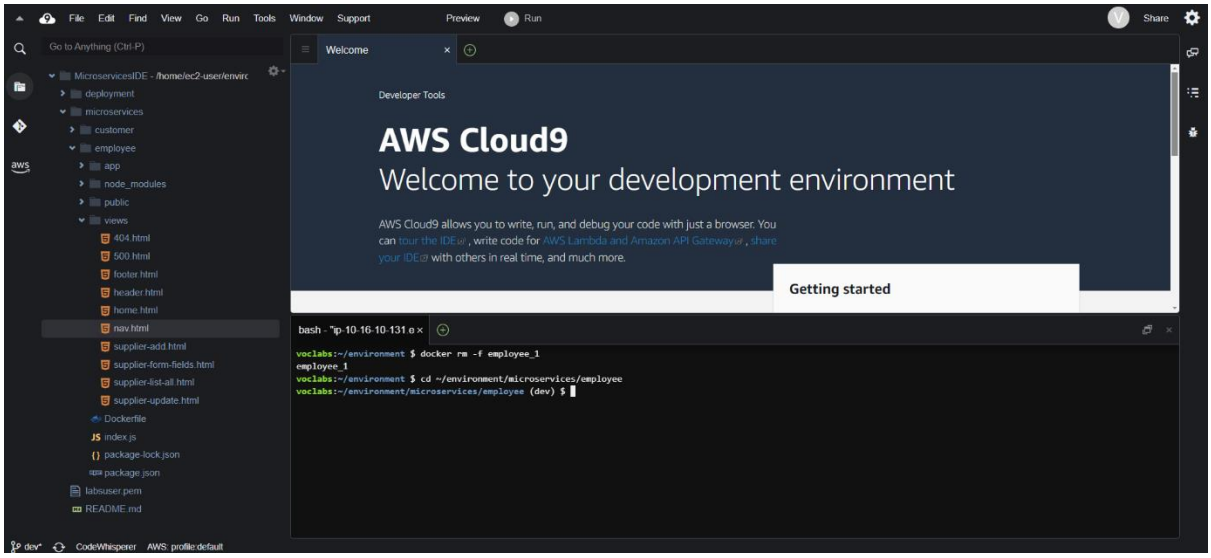


2. To generate a new Docker image from the *employee* microservice source files that you modified and to label the image, run the following commands:

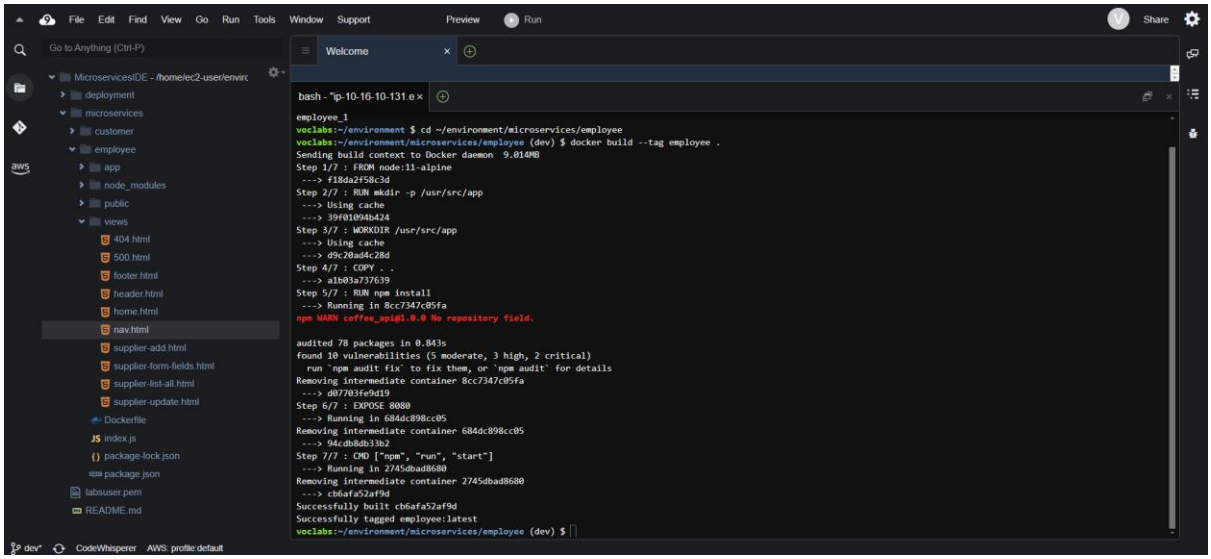
```
docker rm -f employee_1
```



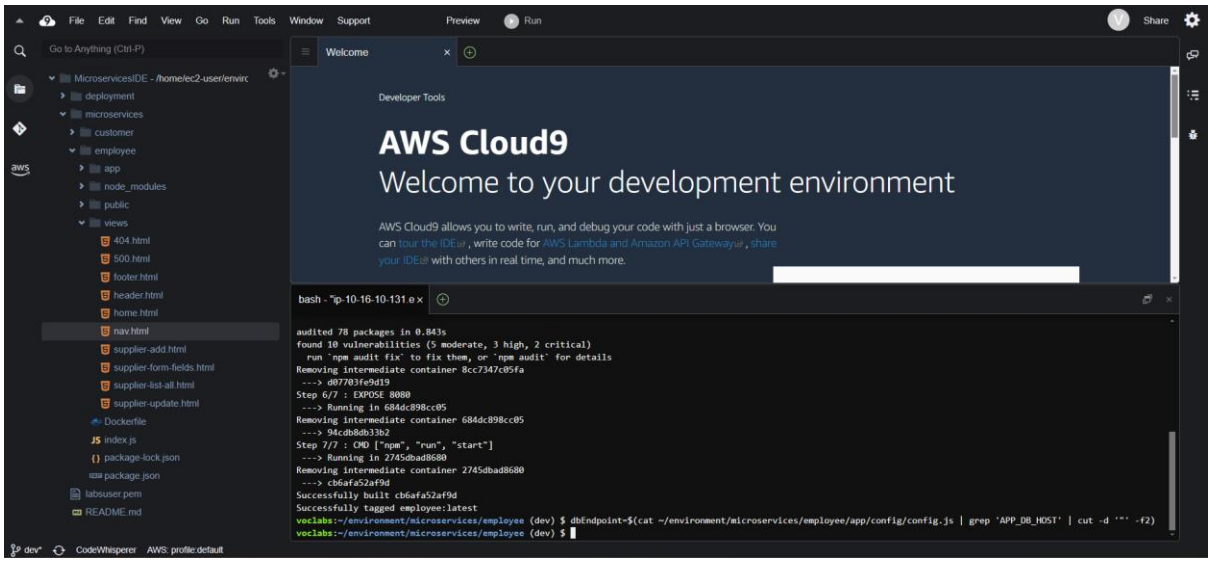
cd ~/environment/microservices/employee



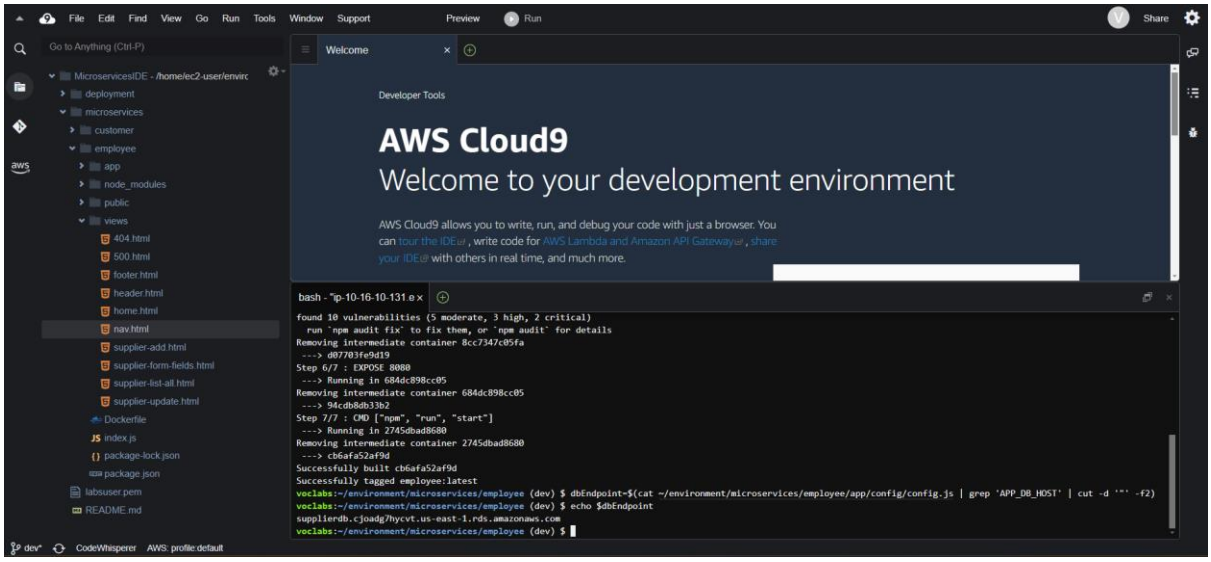
docker build --tag employee .



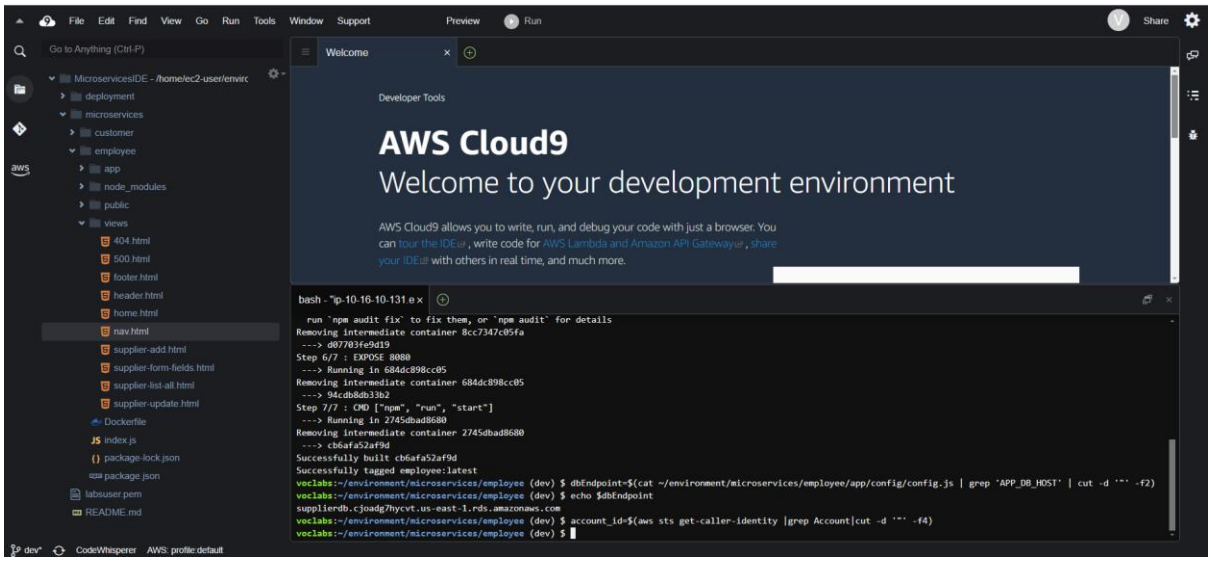
dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)



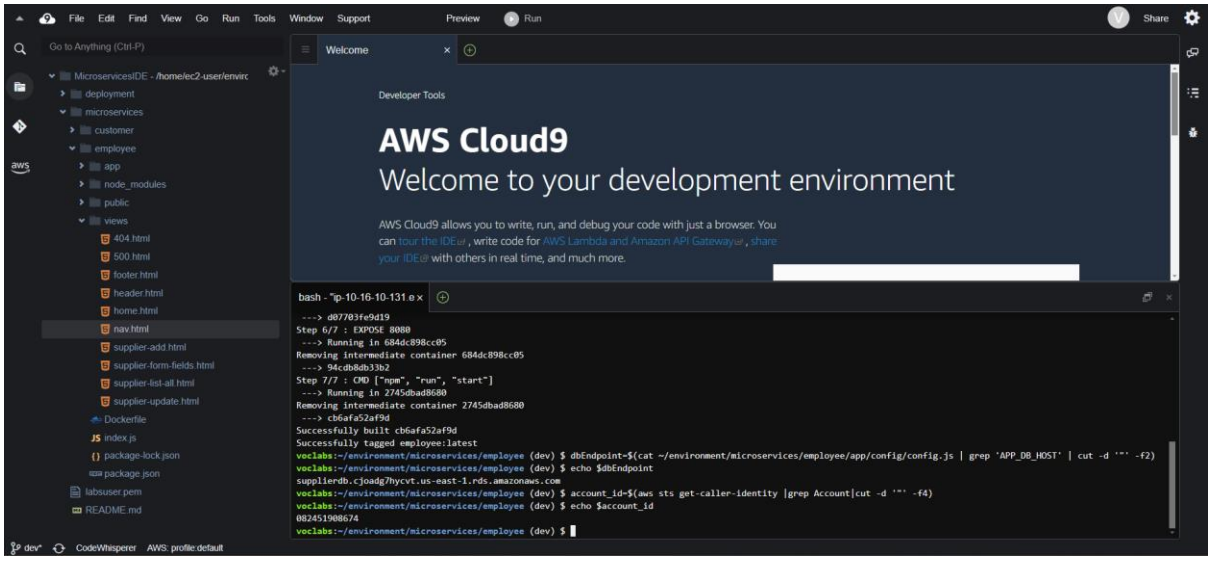
echo $dbEndpoint



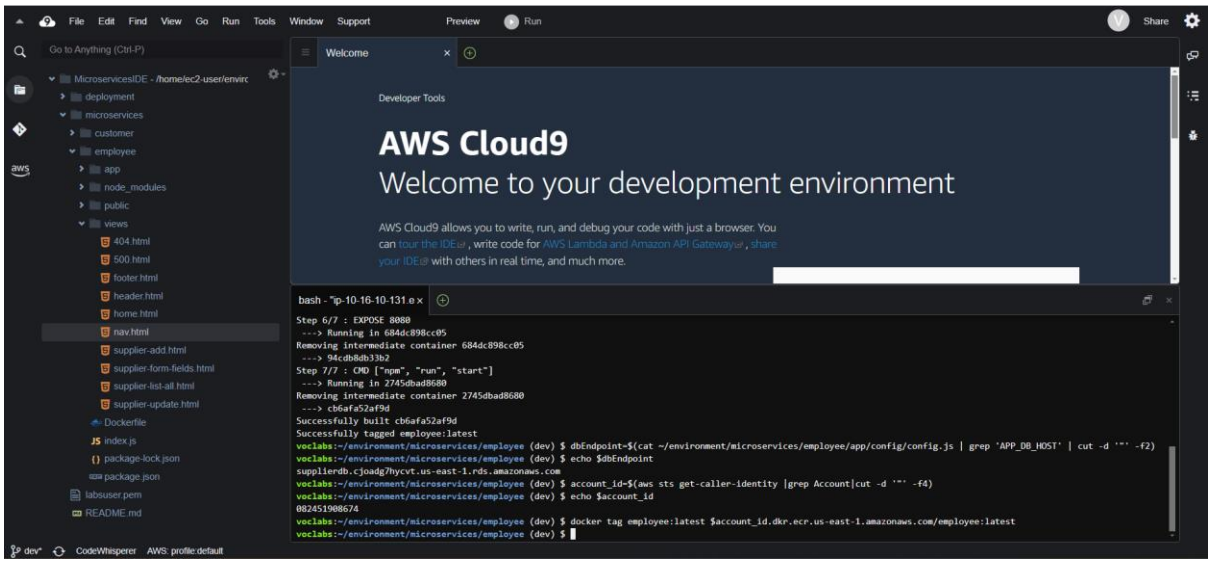
account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)



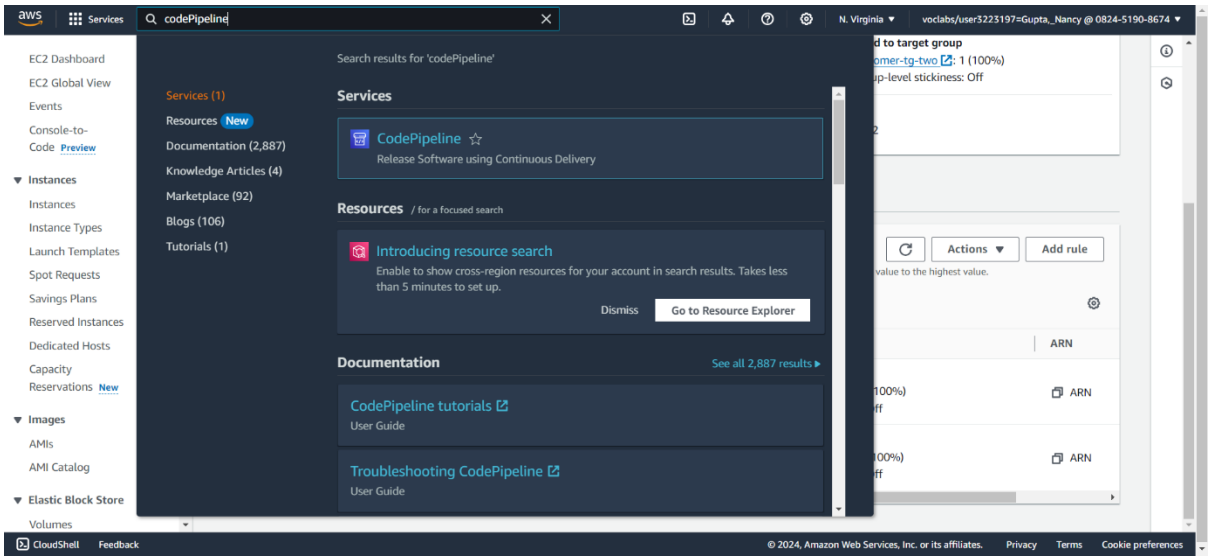
echo $account_id



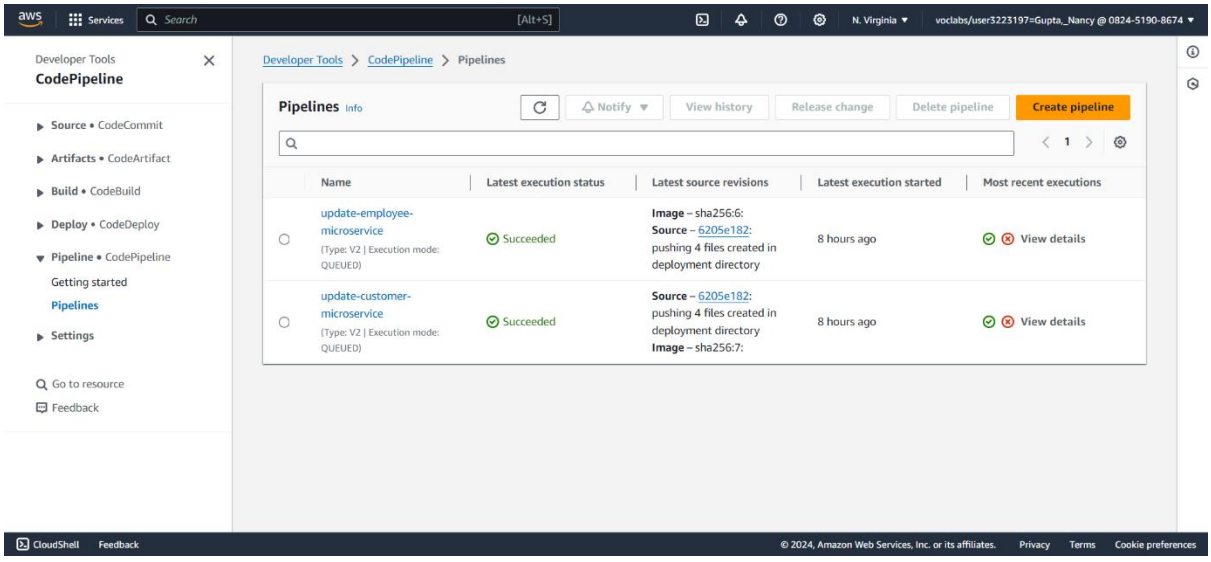
docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest

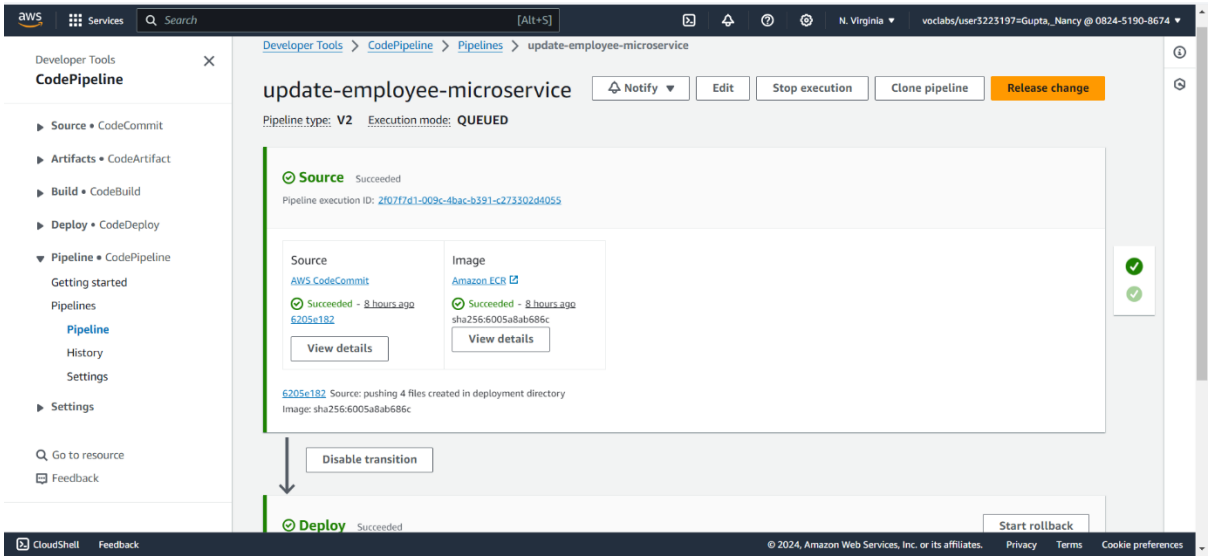


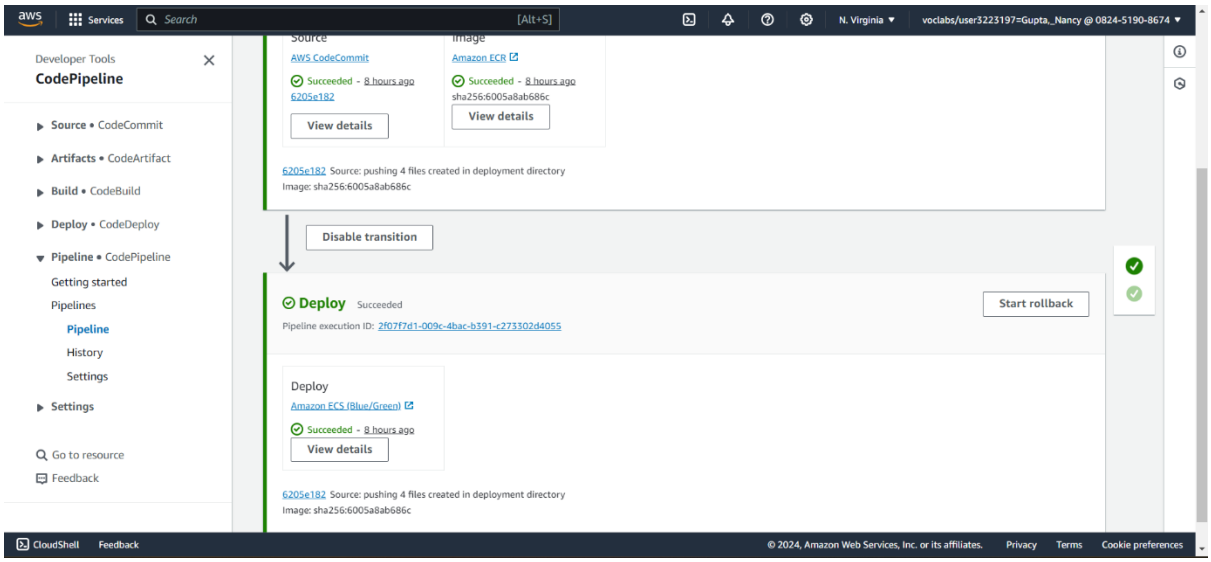
3. Push an updated image to Amazon ECR so that the *update-employee-microservice* pipeline will be invoked.



- In the CodePipeline console, navigate to the details page for the *update-employee-microservice* pipeline. Keep this page open.



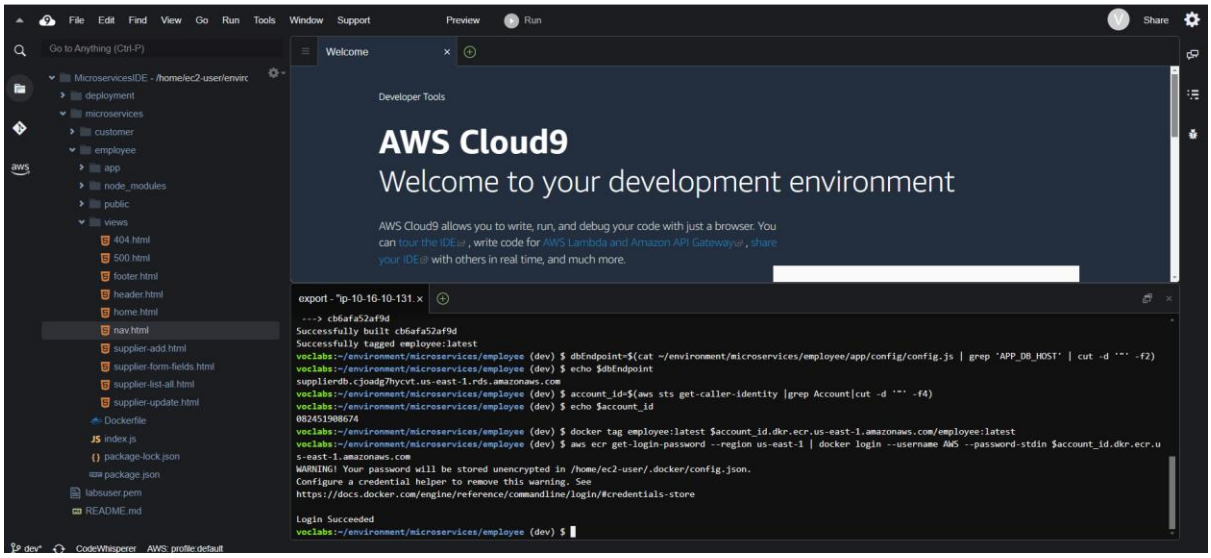




- To push the new employee microservice Docker image to Amazon ECR, run the following commands in your AWS Cloud9 IDE:

#refresh credentials in case needed

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com



#push the image

docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest

- Confirm that the output is similar to the following.

b20e9725db21: Pushed

715b0f96c609: Pushed

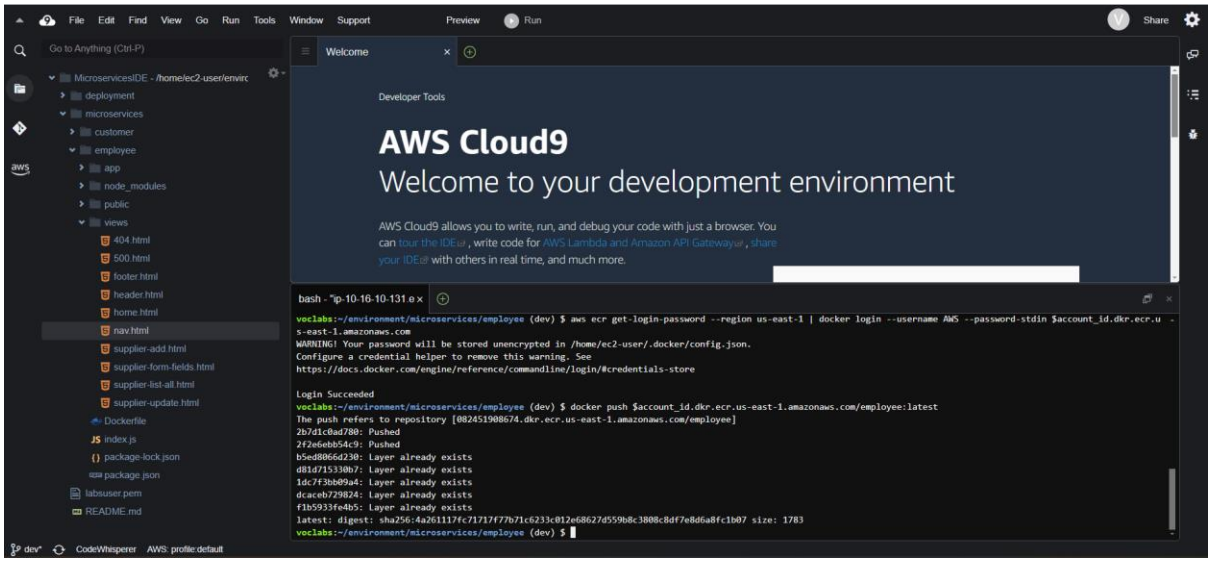
44141cf9260f: Layer already exists

d81d715330b7: Layer already exists

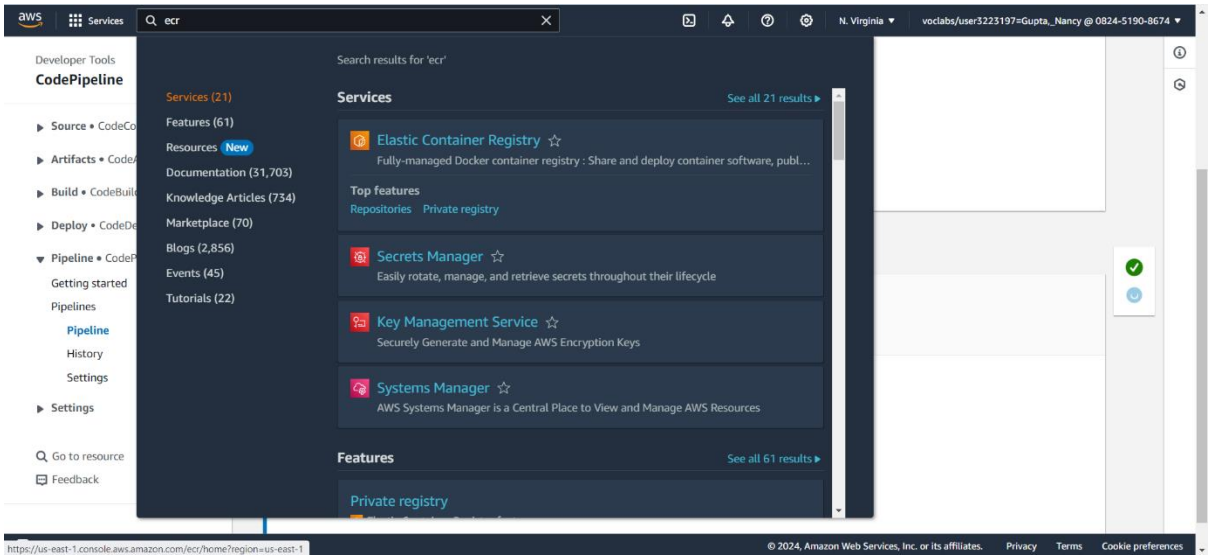
1dc7f3bb09a4: Layer already exists

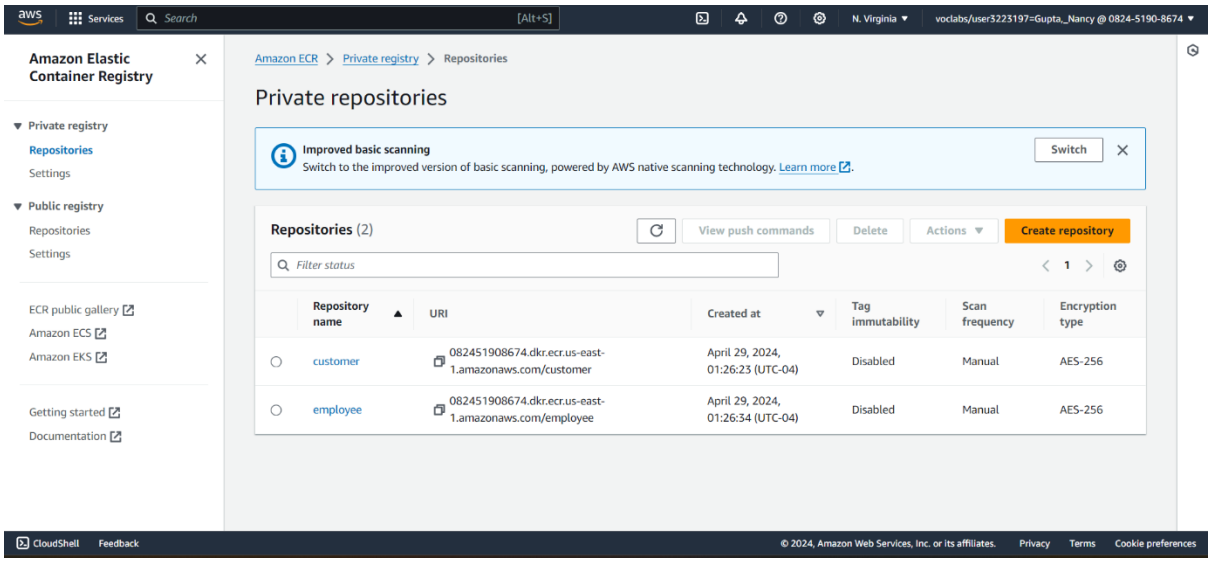
dcaceb729824: Layer already exists

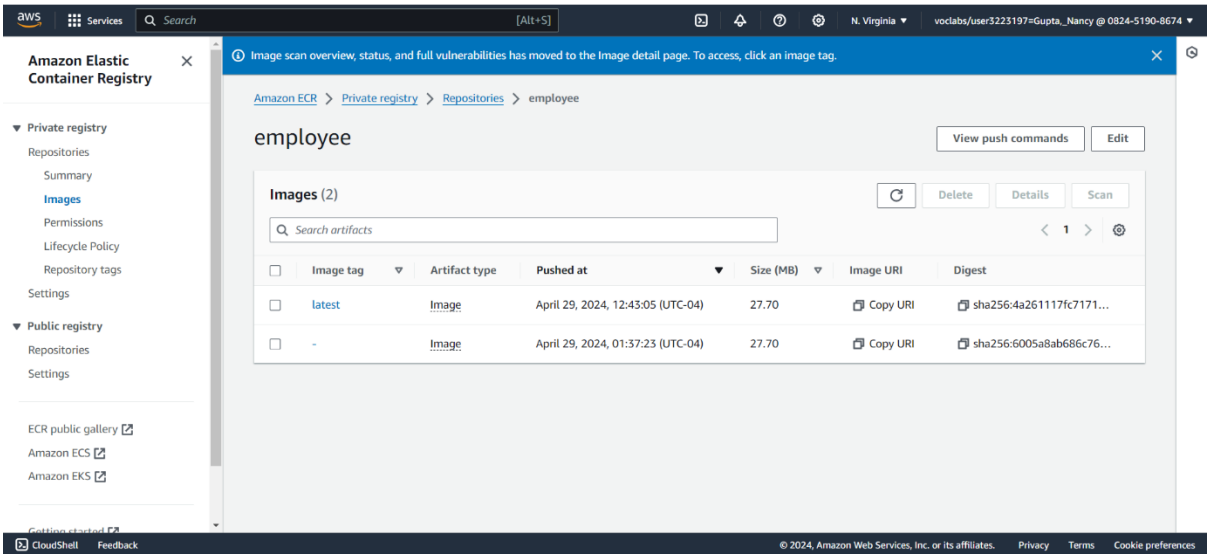
f1b5933fe4b5: Layer already exists



At least one layer should indicate that it was pushed, which indicates that the image was modified since it was last pushed to Amazon ECR. You could also look in the Amazon ECR repository to confirm the last modified timestamp of the image that is tagged as the latest.



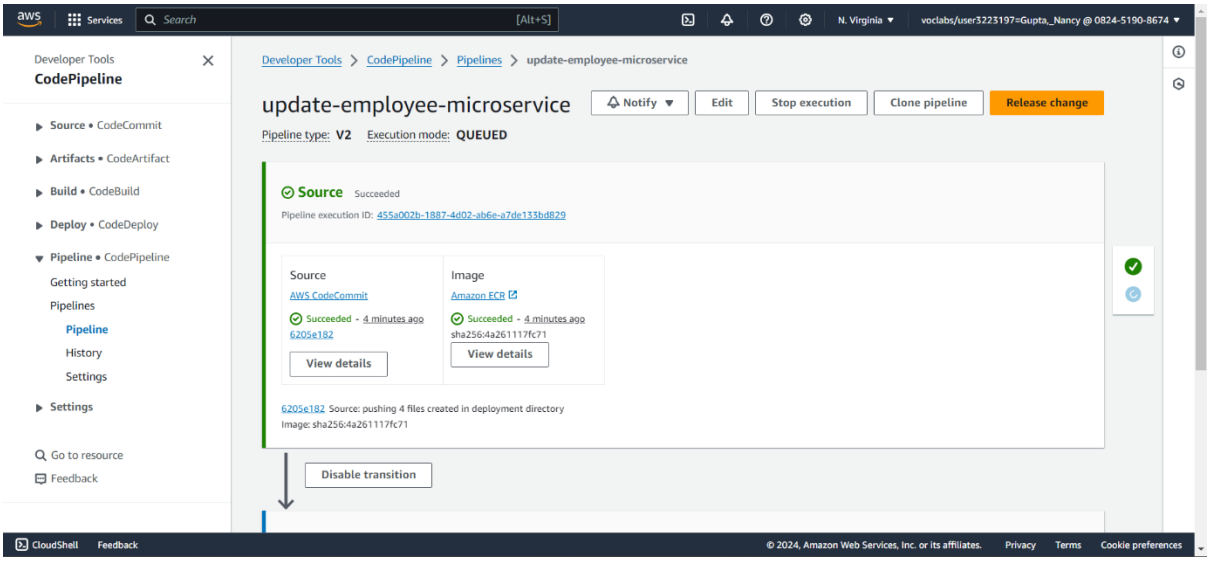


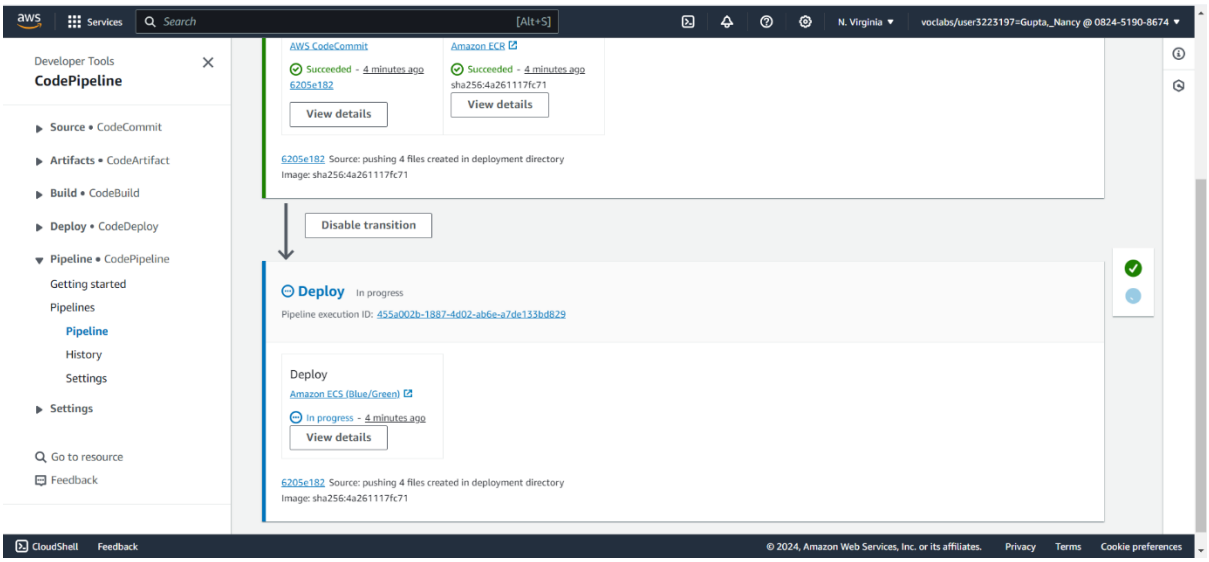


Task 9.3: Confirm that the *employee* pipeline ran and the microservice was updated

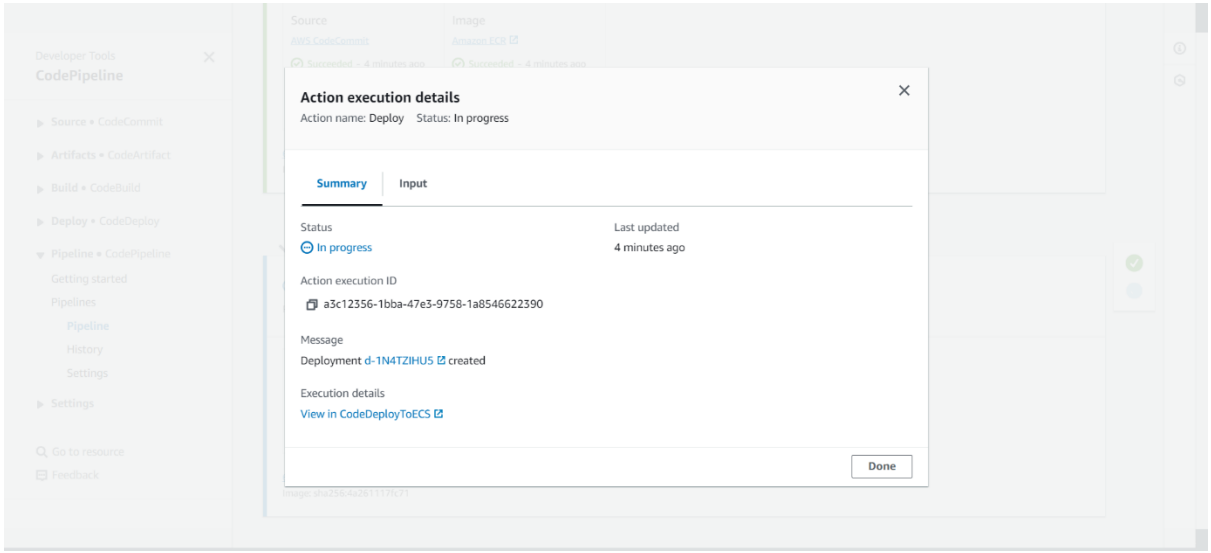
1. **Observe the *update-employee-microservice* pipeline details in the CodePipeline console.**

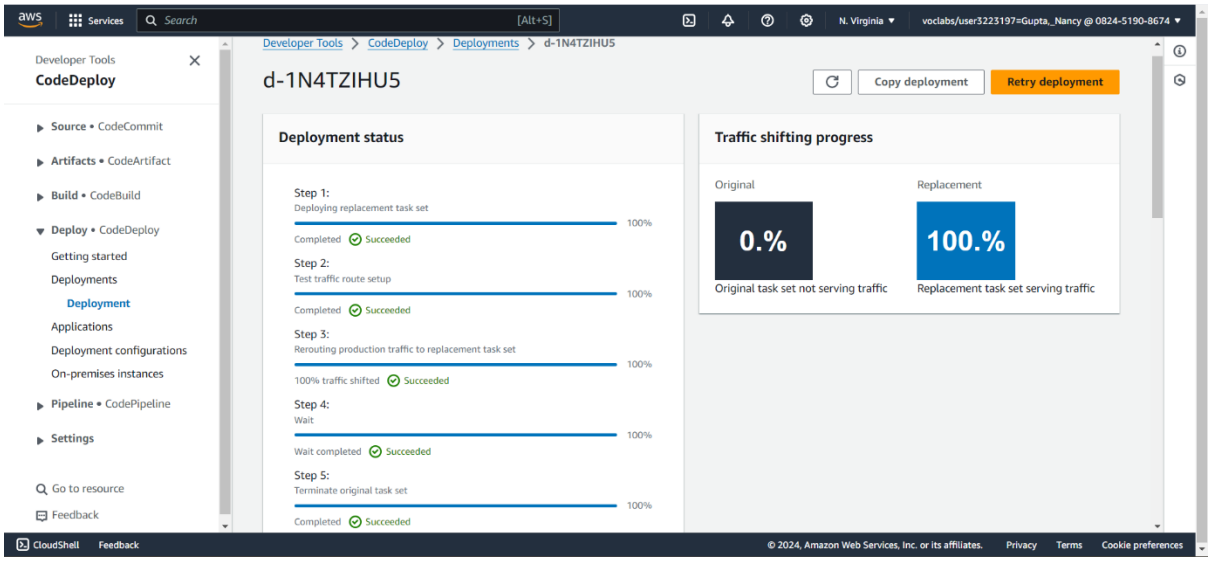
Notice that when you uploaded a new Docker image to Amazon ECR, the pipeline was invoked and ran. Note that the pipeline might take a minute or two to notice that the Docker image was updated before the pipeline is invoked.

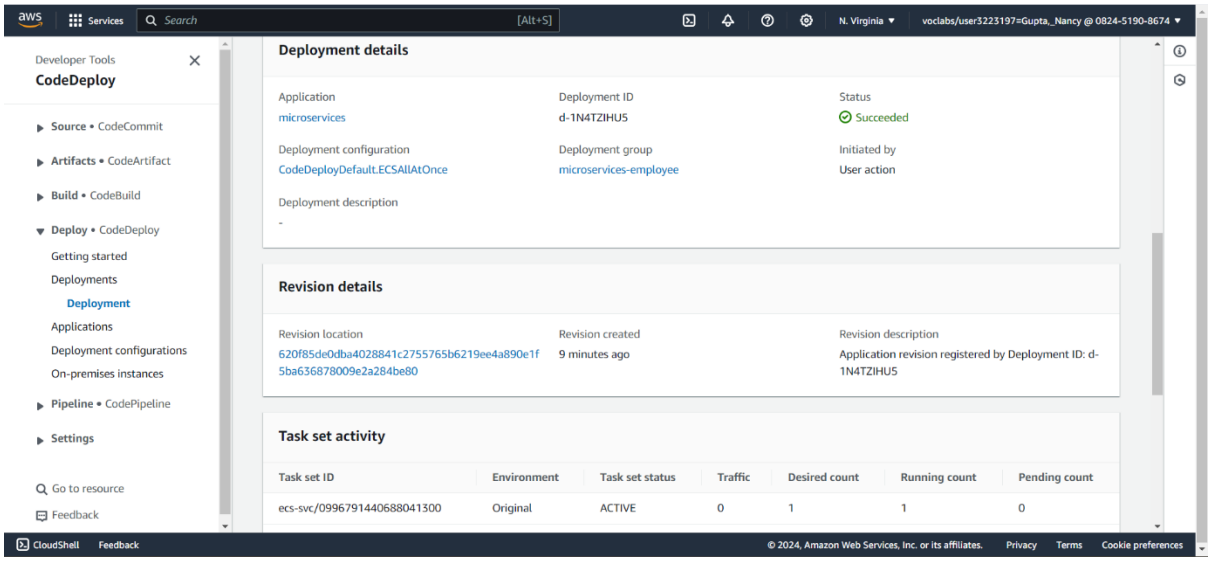


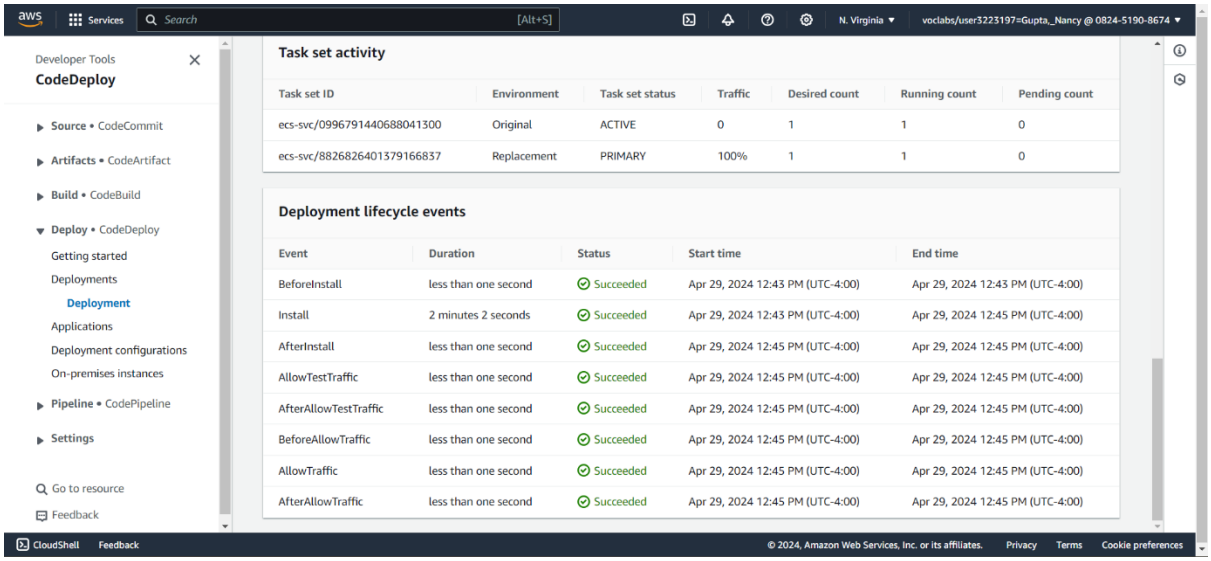


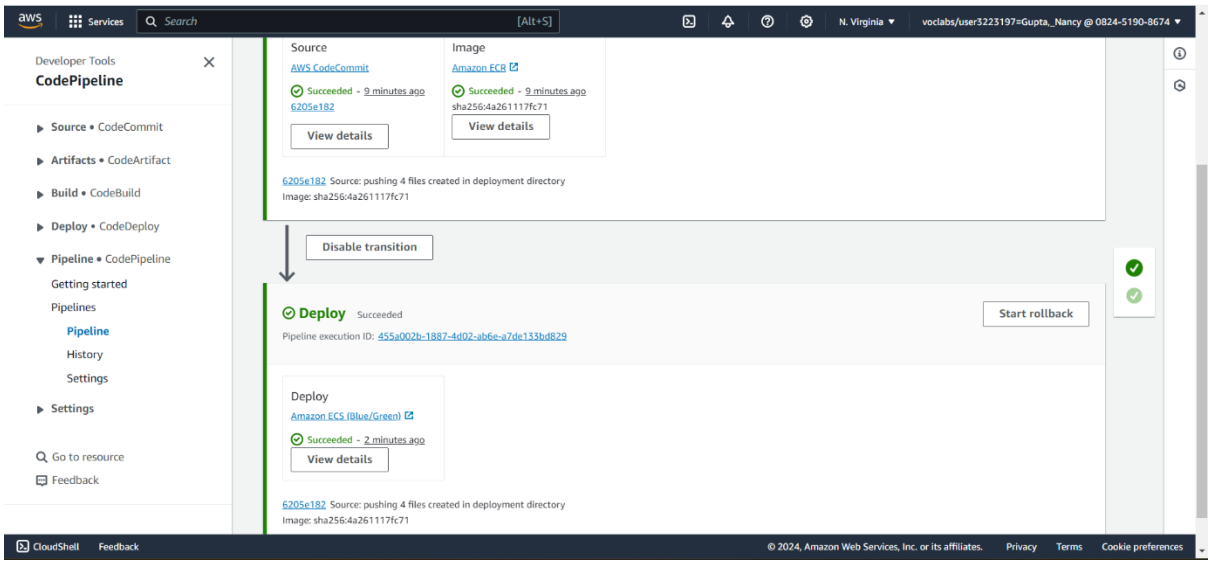
2. **Observe the details in the CodeDeploy console.**









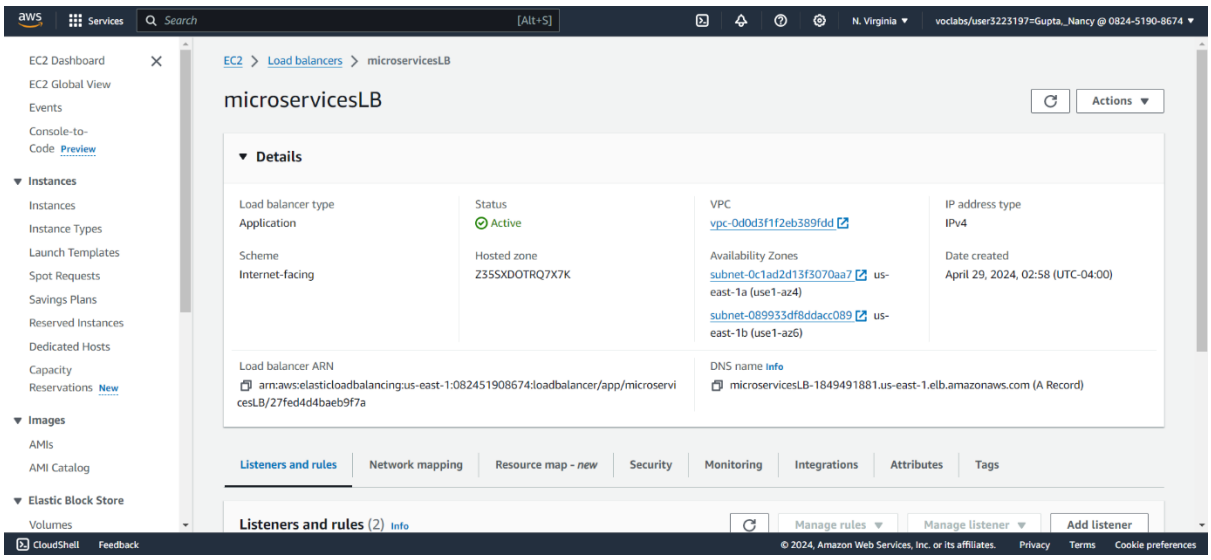


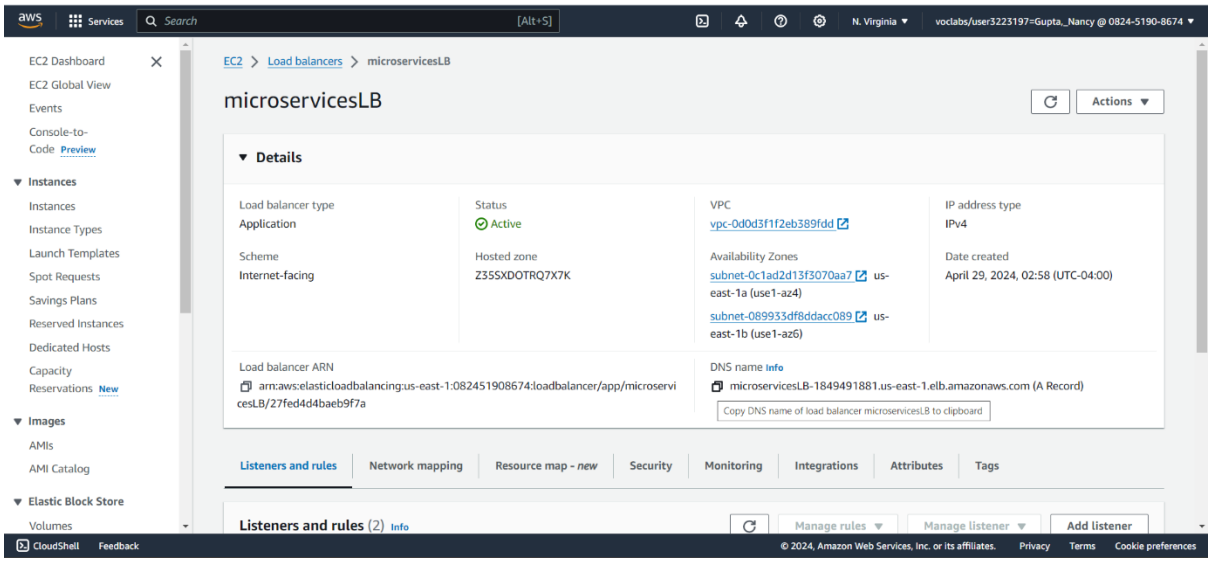
Task 9.4: Test access to the *employee* microservice

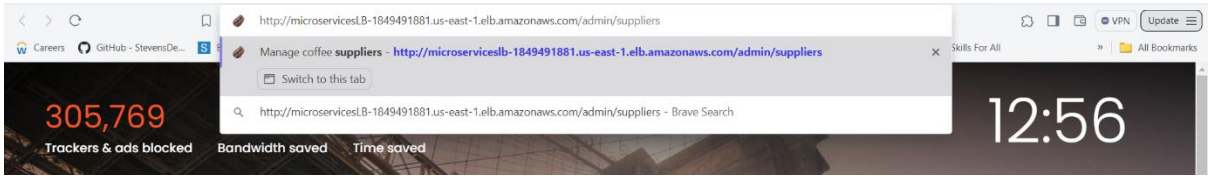
In this task, you will test access to the *employee* microservice.

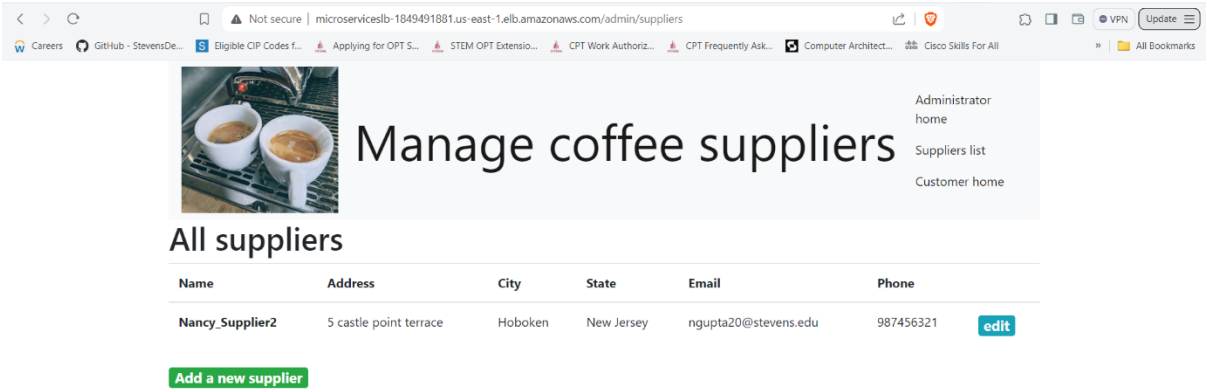
1. Test access to the *employee* microservice pages at http://<alb-endpoint>/admin/suppliers and http://<alb-endpoint>:8080/admin/suppliers from the same device that you have used for this project so far. Replace *<alb-endpoint>* with the DNS name of the *microservicesLB* load balancer.

At least one of the two pages should load successfully.



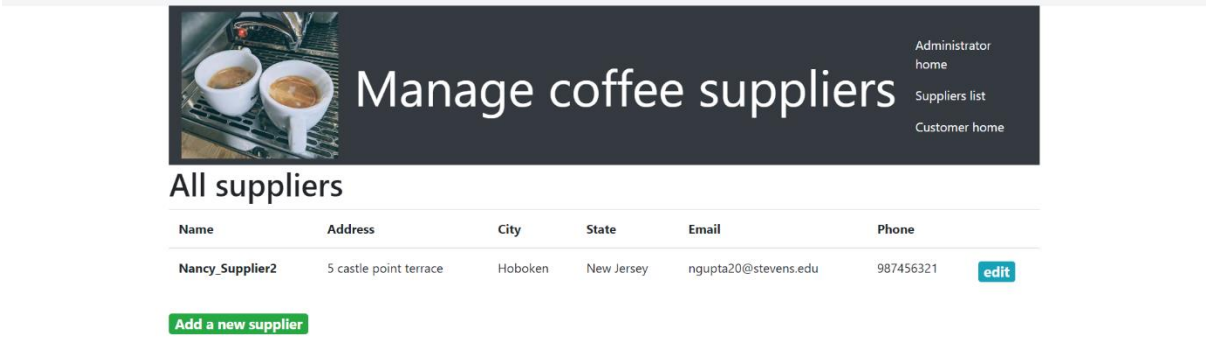




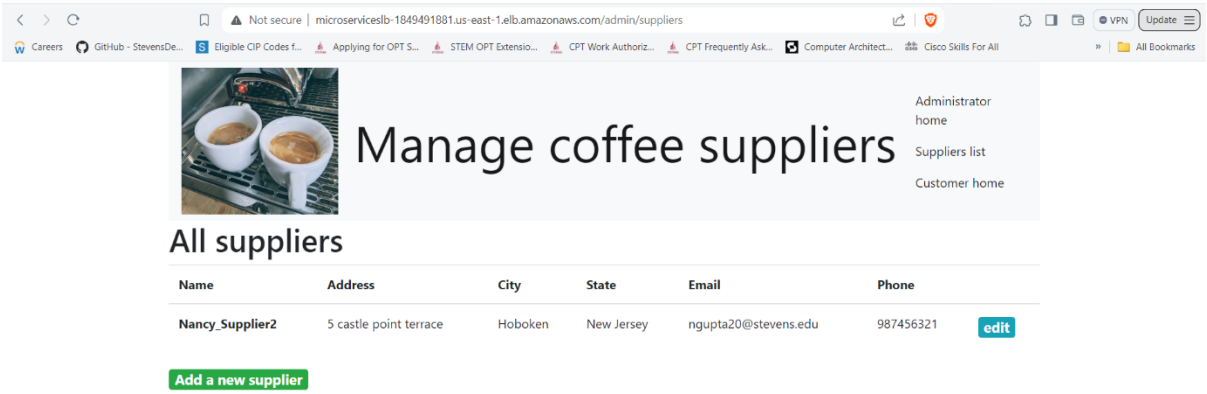


Notice that the banner with the page title is a light color now because of the change that you made to the nav.html file. Pages that are hosted by the customer microservice still have the dark banner. This demonstrates that by using a microservices architecture, you could independently modify the UI or features of each microservice without affecting others.

Earlier, the banner was dark color like this:



Now the banner with the page title is a light color now because of the change that you made to the nav.html file.

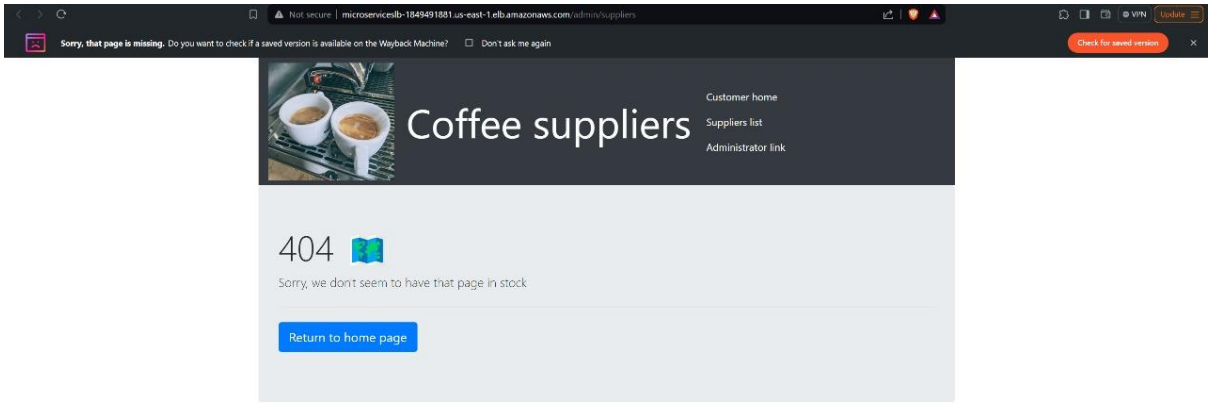


2. Test access to the same *employee* microservice pages from a different device.

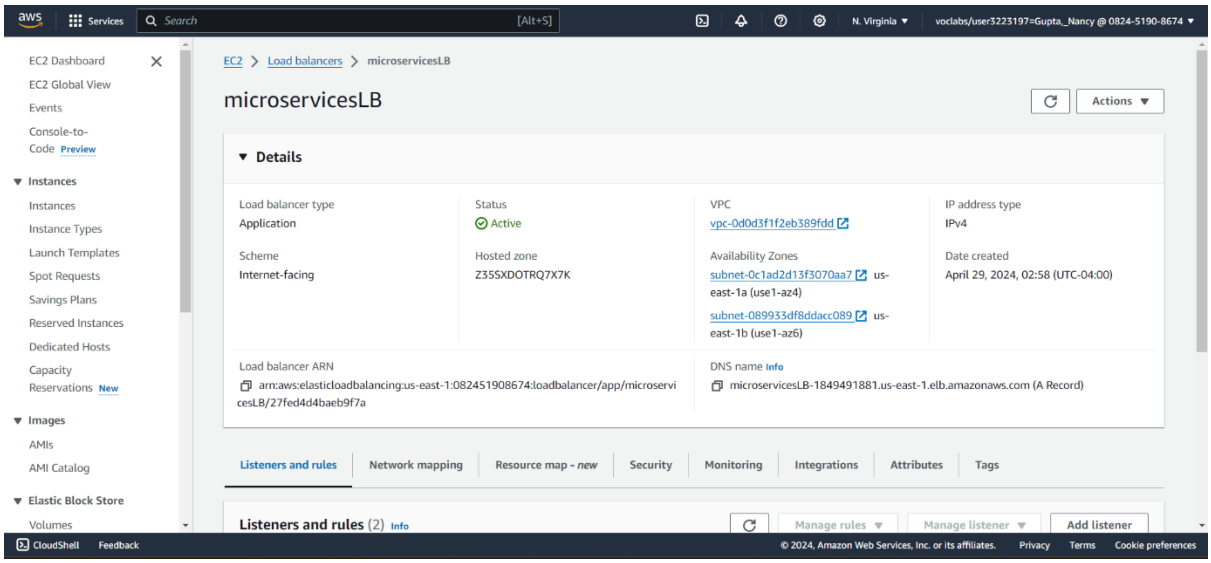
For example, you could use your phone to connect from the cellular network and not the same Wi-Fi network that your computer uses. You want the device to use a different IP address to connect to the internet than your computer.

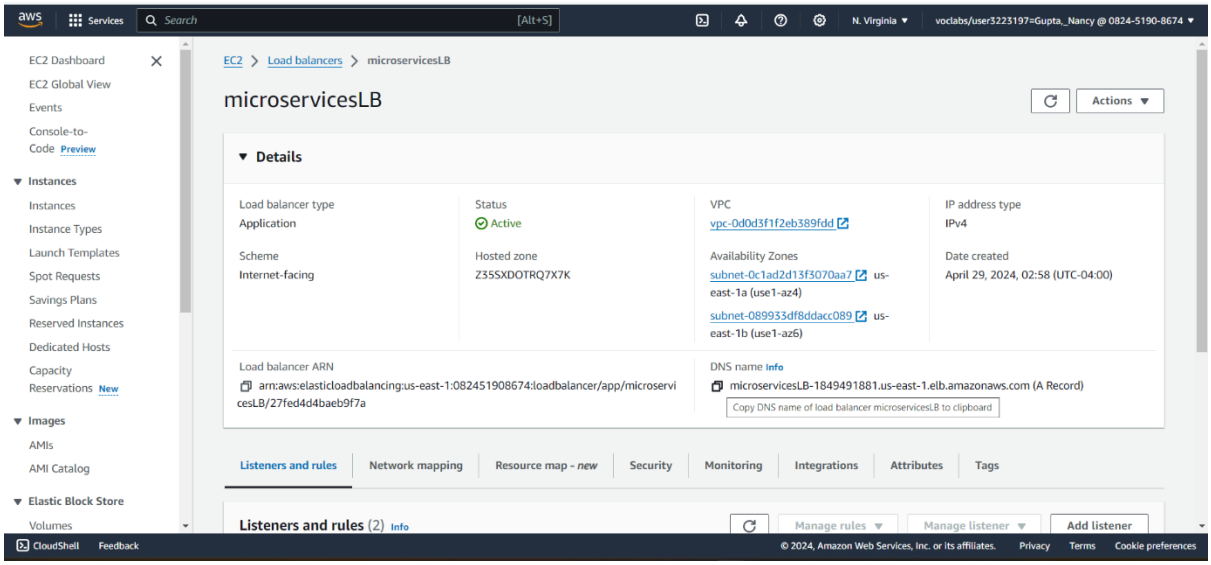
You should get a 404 error on any page that loads, and the page should say "Coffee suppliers" instead of "Manage coffee suppliers." This is evidence that you cannot successfully connect to the employee microservice from another IP address.

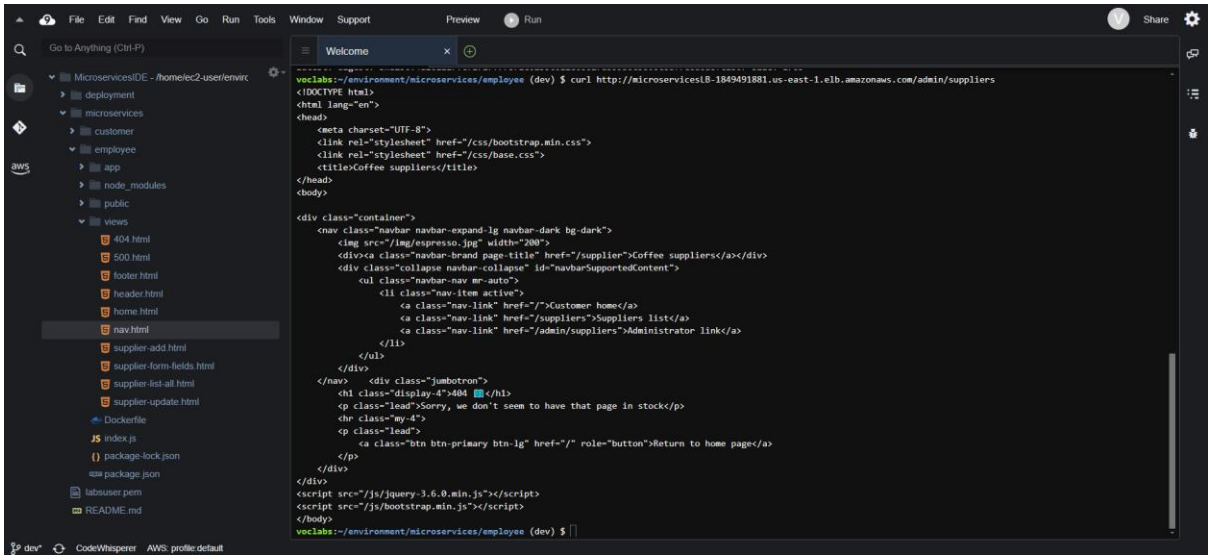
This screenshot is taken using some other IP address as mentioned.



Tip: If you don't have another network available, run the following command in the AWS Cloud9 terminal: curl http://<alb-endpoint>/admin/suppliers. The source IP address of the AWS Cloud9 instance is different than your browser's source IP. The result should include Sorry, we don't seem to have that page in stock .







This proves that the updated rules on the load balancer listener are working as intended.

Tip: If the update doesn't function as intended, you could go to the Deployments page in the CodeDeploy console within 5 minutes to stop the deployment and roll back to the previous version. You would choose Stop and roll back deployment and then choose Stop and rollback. This action would reroute production traffic to the original task set and then delete the replacement task set. You configured the 5-minute setting in phase 8, task 1.

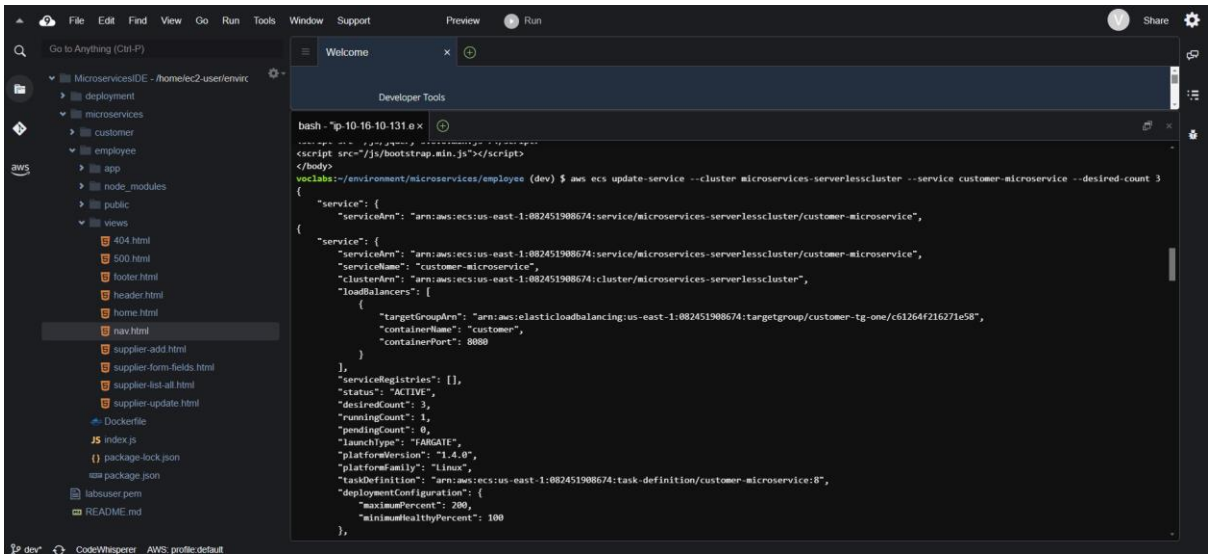
Task 9.5: Scale the *customer* microservice

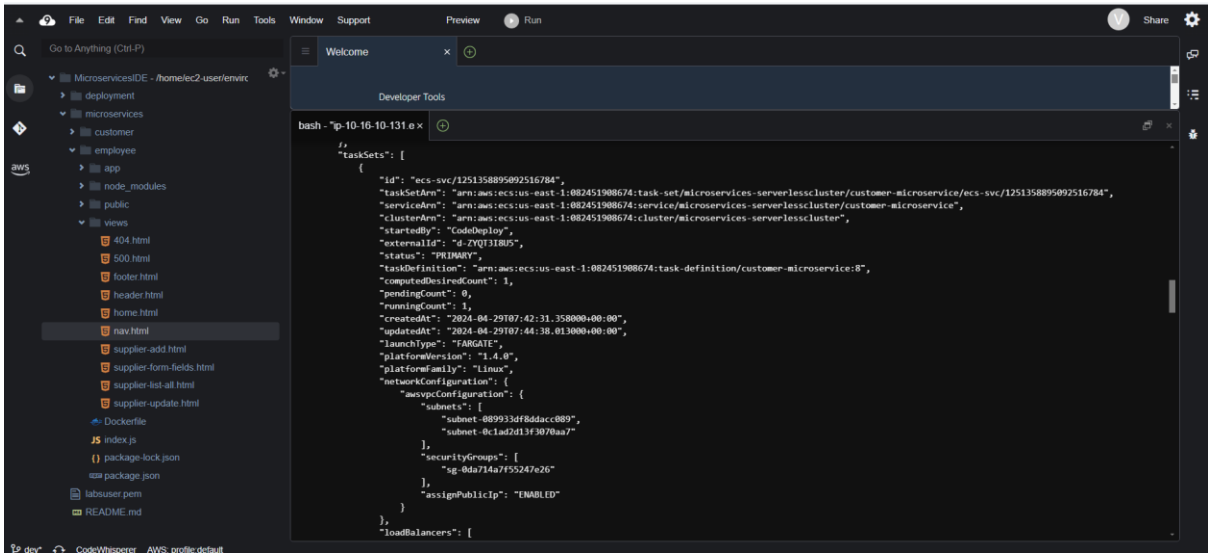
In this task, you will scale up the number of containers that run to support the *customer* microservice. You can make this change without causing the *update-customer-microservice* pipeline to run.

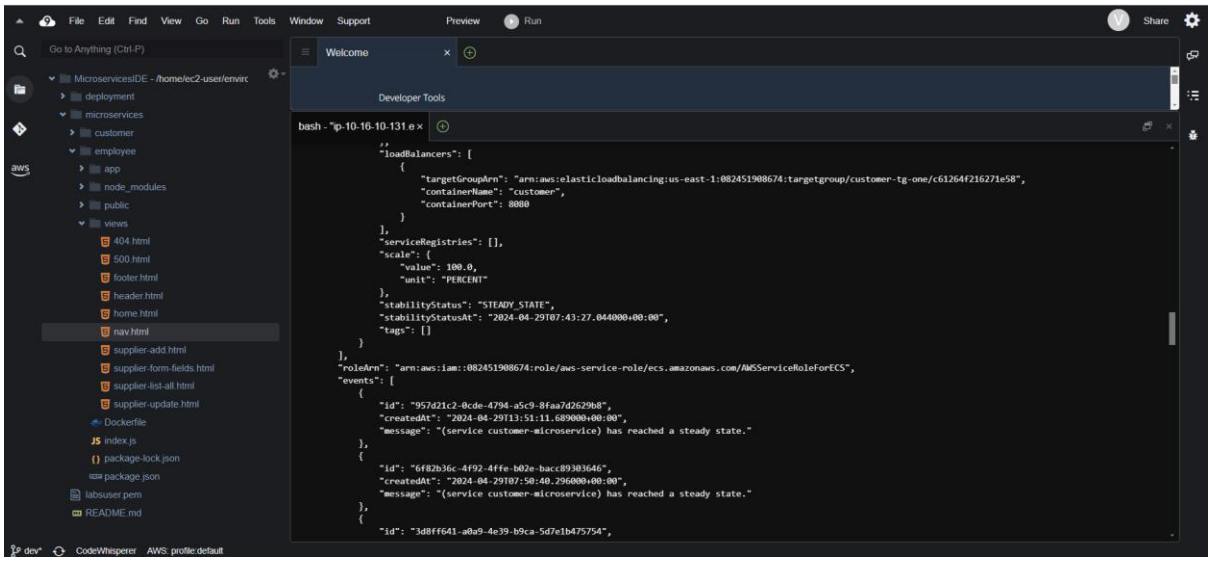
1. Update the *customer* service in Amazon ECS.
    - Run the following command:

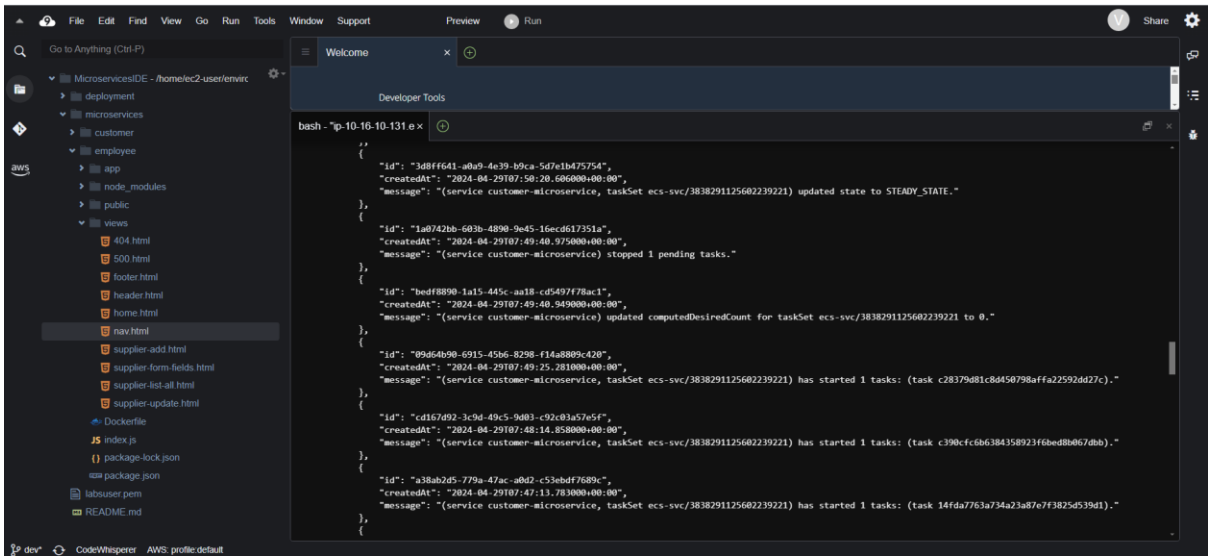
aws ecs update-service --cluster microservices-serverlesscluster --service customer-microservice --desired-count 3

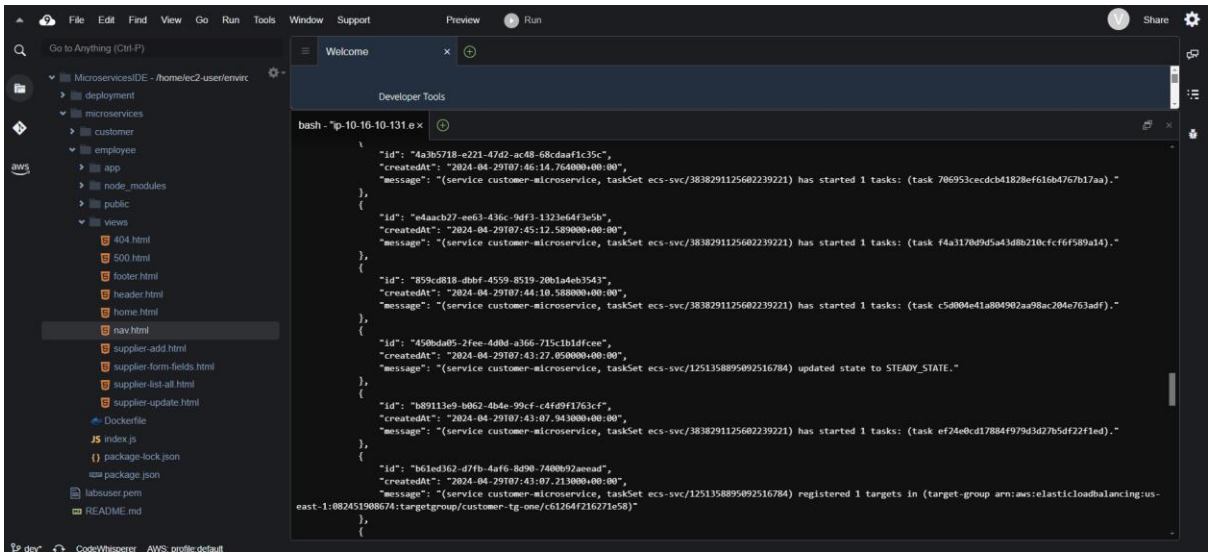
A large JSON-formatted response is returned.

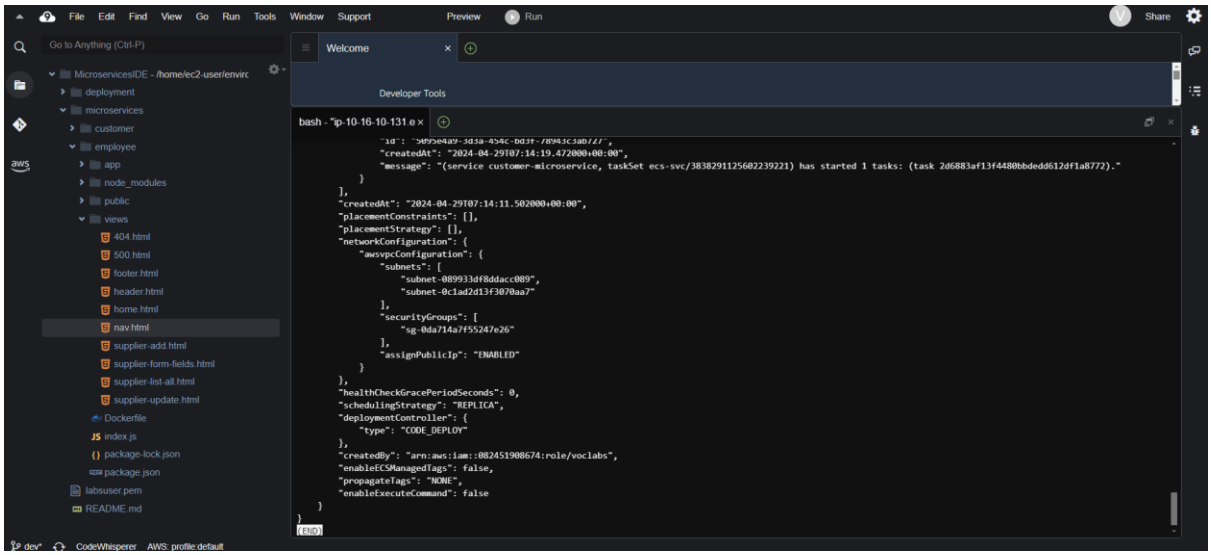




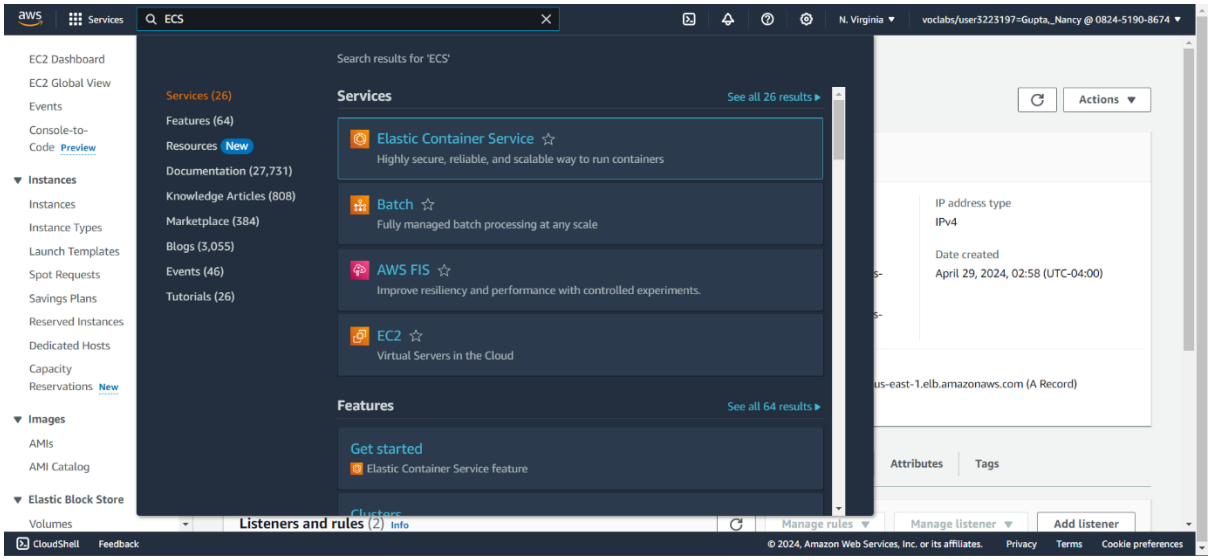


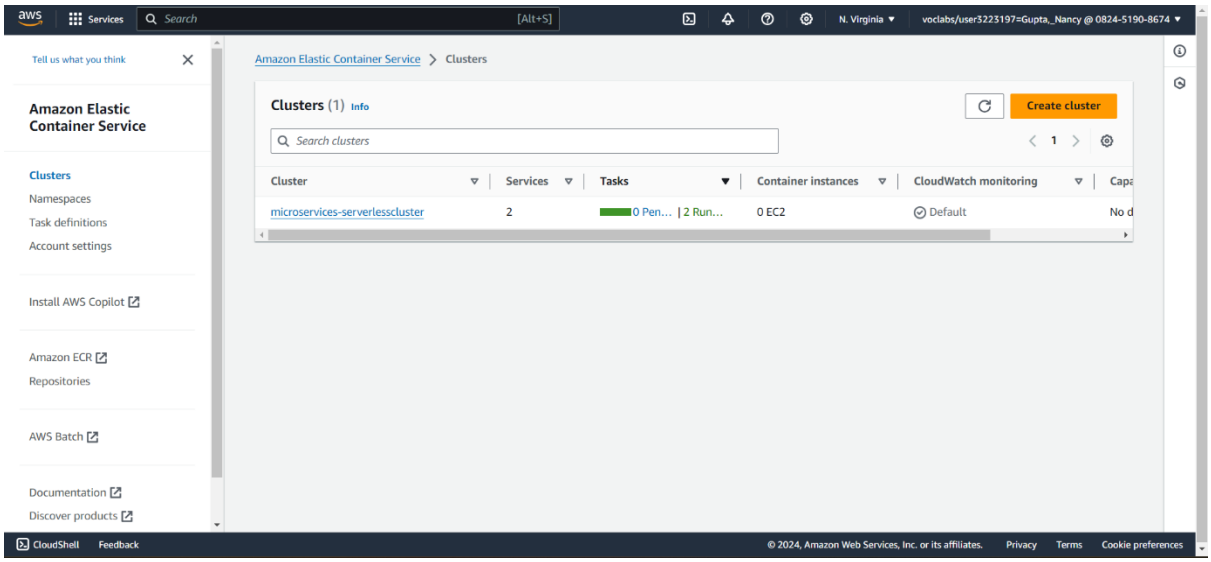


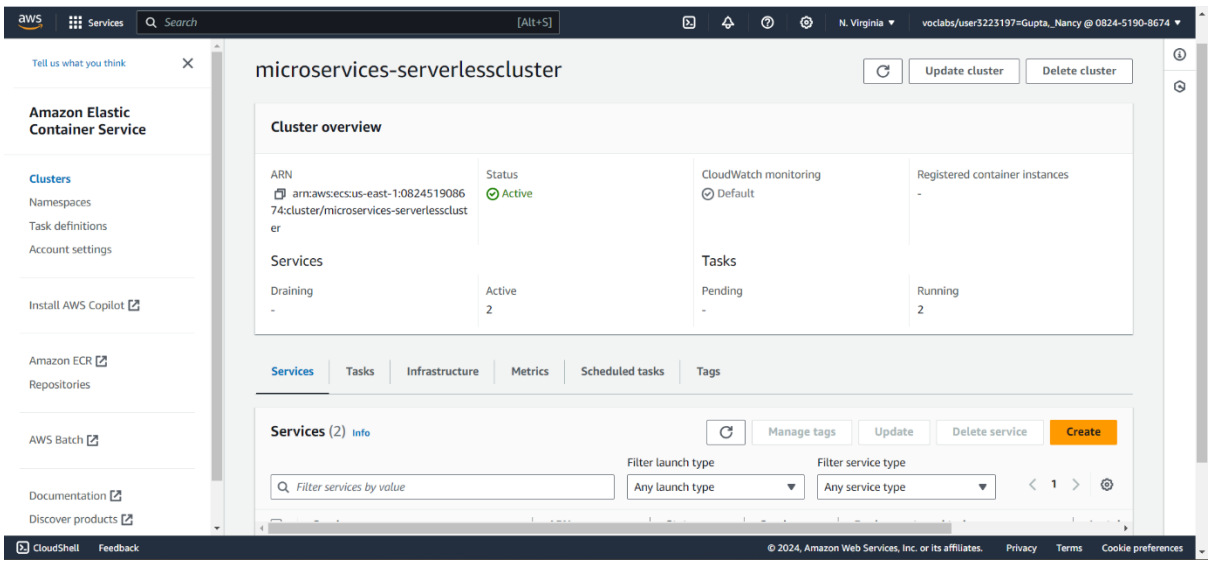




- Go to the Amazon ECS services view.



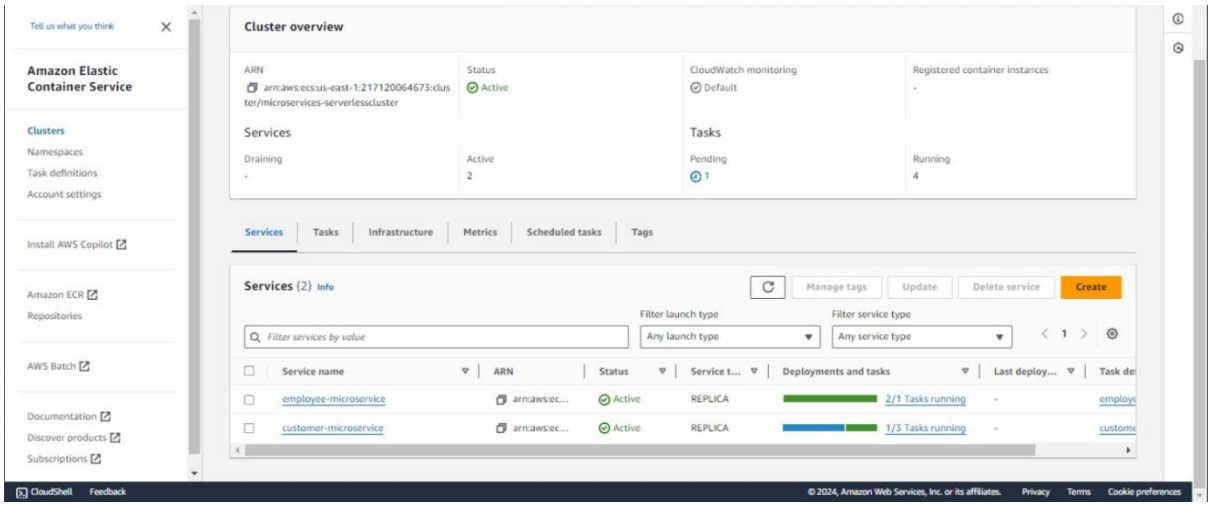


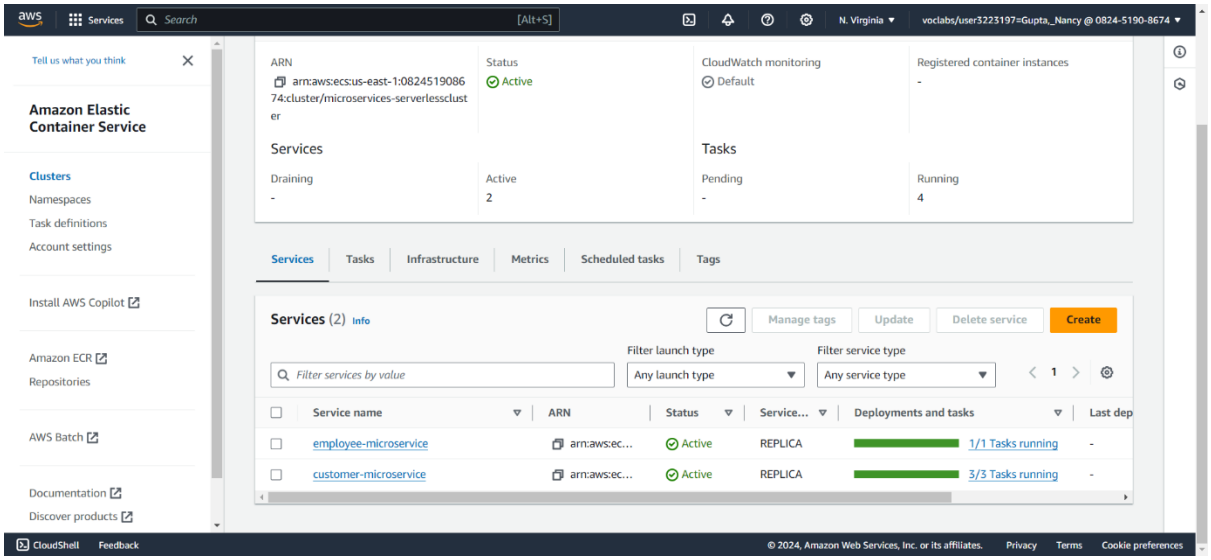


It might look similar to the following image

In this example, the *customer-microservice* shows 1/3 tasks running because you increased the desired count from 1 to 3, but the two new containers are still being started. If you wait long enough and look at the **Tasks** tab, you will see that three containers run to support the customer microservice. This demonstrates how you can scale microservices independently of one another.

Note: The *employee-microservice* might show 2/1 tasks running. This could happen because a replacement task set (which has one container) has been created, but the original task set remains active for the first 5 minutes in case you decide to roll back.





Part 2: Analysis

Detailed Explanation:
o Provide a thorough analysis of each phase of the project.
o Explain the significance and outcomes of each phase.

Analysis and Significance of Phase 1:

Laying the groundwork: Design and Cost

This initial phase is all about figuring out how the system will be built and how much it will cost. It's like sketching out a map of the entire project, showing how the different pieces fit together. This map, called an architecture diagram, is essential for everyone working on the project to be on the same page. It also helps us estimate how much money we'll need to spend to bring this project to life.

Why it matters:

This phase sets the stage for the whole project. By planning the design and estimating the cost upfront, we make sure we're starting off on the right foot. We have clear goals for what we're building and a realistic idea of how much it will cost.

What we'll achieve:

- A clear picture: We'll create a visual representation, the architecture diagram, that shows how all the parts of the system work together.
- Budgeting for success: We'll develop a cost estimate so we can plan our budget and make sure the project stays financially on track.

Analysis of Phase 2:

The Importance of Phase 2:

Transitioning to a microservices architecture is like building a new house. Before laying the foundation for the new structure, we need to thoroughly understand the existing one – the monolithic application in this case. Phase 2 acts as our architectural inspection, analyzing the infrastructure, testing functionality, and uncovering how the monolithic application operates. This knowledge is critical for making informed decisions during the microservices development process.

What We'll Do:

- **Task 2.1:** Verifying Monolith Availability: This initial step ensures the monolithic application is accessible and operational. We'll verify users can reach the application and it responds as expected. This involves checking the application URL and confirming it returns the appropriate HTTP status codes (e.g., 200 OK).
- **Task 2.2:** Testing the Monolith's Functionality: Moving beyond basic availability, we'll delve deeper into how well the application actually works. This involves functional testing to confirm each feature operates as intended. Additionally, performance testing might be conducted to assess the application's responsiveness and ability to handle varying user loads. Identifying any limitations or issues in the current monolithic architecture allows us to make informed decisions about future improvements or how to best break down functionalities into microservices.
- **Task 2.3:** Analyzing the Monolith's Inner Workings: This task dives deeper, examining the core operations of the monolithic application. We'll utilize performance profiling tools to pinpoint areas for optimization by analyzing CPU, memory, and network usage. Additionally, dependency analysis will be conducted to understand how different parts of the monolithic application interact and how these dependencies might impact performance. This in-depth analysis provides valuable insights into the application's inner workings, guiding decisions on how to potentially refactor or redesign components during the transition to microservices.

Outcomes of Phase 2:

By thoroughly analyzing the monolithic application, we gain a comprehensive understanding of its functionality, performance characteristics, and potential bottlenecks. This knowledge serves as a crucial foundation for the microservices development process. We can identify areas for improvement, optimize the design of our new architecture, and make informed decisions about how to best break down the monolithic application into smaller, independent services. Phase 2 essentially equips us with a roadmap for the successful migration to a microservices architecture.

Analysis and Significance of Phase 3:

The Significance of Phase 3:

Before diving headfirst into microservices development, we need a well-equipped workshop. Phase 3 focuses on establishing a collaborative development environment and implementing a version control system. This ensures a smooth workflow for the team while safeguarding the codebase throughout the project lifecycle.

The Steps Involved:

- **Task 3.1:** Constructing the Cloud9 IDE: We'll leverage AWS Cloud9 to create a cloud-based Integrated Development Environment (IDE). This will serve as our central workspace, accessible from anywhere with an internet connection, offering all the necessary tools for coding, debugging, and deploying microservices. Cloud9 provides a consistent development experience for all team members, eliminating the need for individual setup variations.

- **Task 3.2:** Transferring the Codebase: The existing application code will be meticulously transferred into the newly established Cloud9 IDE. This ensures developers have immediate access to the necessary files and resources for development activities within the Cloud9 environment. This eliminates the need for local setup and facilitates streamlined code editing, testing, and debugging.

- **Task 3.3:** Organizing for Microservices Success: To promote a structured approach to microservices development, we'll establish dedicated working directories within the codebase. Each microservice will have its own designated directory, populated with initial code templates. This fosters modularity and maintainability of the codebase, allowing developers to focus on specific components while maintaining a clear overall structure.

- **Task 3.4**: Implementing Version Control with Git: For effective collaboration and code management, we'll establish a Git repository using AWS CodeCommit. This serves as the central repository for the microservices codebase. By pushing the code to CodeCommit, we ensure proper version control and tracking of all changes made by team members. This enables features like collaborative development, reverting to previous versions if needed, and maintaining a single source of truth for the project's code.

The Outcomes of Phase 3:

By successfully completing Phase 3, we'll have established a robust foundation for collaborative development and effective version control. This streamlines the development process, empowers teamwork, and ensures the codebase remains stable and traceable throughout the project's life cycle. With a centralized development environment and a well-implemented version control system

in place, the team is now prepared to embark on the exciting journey of building the microservices architecture.

Analysis of Phase 4: Transitioning to Microservices Architecture and Docker Testing

Phase 4 involves migrating the application from a monolithic architecture to a microservices architecture and testing them using Docker containers. This phase validates the viability of the microservices approach and ensures correct interaction between components.

Outcomes:

Task 4.1: Configure AWS Cloud9 Security Group Settings

Analysis: Configure security group settings of the AWS Cloud9 instance to enable communication between Docker containers on the same EC2 instance.

Significance: Facilitates testing and development activities by allowing Docker containers to communicate internally and with external services.

Task 4.2: Refactor Customer Microservice Source Code

Analysis: Refactor customer microservice source code to separate customer-related functionality from the monolithic application, creating a standalone microservice.

Significance: Initiates the decomposition of the monolithic application into manageable microservices, promoting modularity and maintainability.

Task 4.3: Create Dockerfile for Customer Microservice and Launch Test Container

Analysis: Develop a Dockerfile for the customer microservice, defining dependencies and configurations, and launch a test container to validate functionality.

Significance: Encapsulates the customer microservice in a Docker container, ensuring consistency and portability across different environments.

Task 4.4: Refactor Employee Microservice Source Code

Analysis: Extract employee-related functionality from the monolithic application and implement it as a separate microservice.

Significance: Continues the transition to microservices architecture, promoting single responsibility and autonomy of components.

Task 4.5: Create Dockerfile for Employee Microservice and Launch Test Container

Analysis: Develop a Dockerfile for the employee microservice, build its Docker image, and launch a test container to validate functionality.

Significance: Ensures encapsulation of the employee microservice within a Docker container, facilitating deployment and resource isolation.

Task 4.6: Adjust Employee Microservice Port and Rebuild Image

Analysis: Adjust port configuration of the employee microservice to avoid conflicts and rebuild its Docker image with updated settings.

Significance: Prevents port conflicts when running multiple microservices on the same EC2 instance, ensuring smooth operation.

Task 4.7: Version Control Microservice Code in CodeCommit

Analysis: Store modified microservice source code in AWS CodeCommit for version control and collaboration.

Significance: Establishes a version-controlled repository for microservices codebase, enabling tracking, collaboration, and rollback if needed.

Phase 4 is pivotal in transitioning from a monolithic to a microservices architecture. By modularizing functionality and testing in Docker containers, it validates the microservices approach and sets the foundation for deployment and automation in subsequent phases.

Analysis of Phase 5: Establishing ECS Deployment Infrastructure

Phase 5 focuses on transitioning from single-instance Docker container deployment to a scalable architecture using AWS ECS and AWS CodeDeploy. It addresses scalability, load balancing, and automated deployment requirements.

Outcomes:

Task 5.1: Set up ECR Repositories and Upload Docker Images

Analysis: Establish Amazon ECR repositories for secure Docker image storage. Uploading Docker images to ECR ensures accessibility for ECS deployment.

Significance: Centralized Docker image storage ensures version control and secure deployment for ECS.

Task 5.2: Create an ECS Cluster

Analysis: Creating an ECS cluster provides a scalable environment for running containerized applications. It enables efficient resource utilization and automatic scaling.

Significance: Establishes a foundation for scalable and fault-tolerant microservices deployment.

Task 5.3: Configure CodeCommit Repository for Deployment Files

Analysis: Setting up a CodeCommit repository facilitates version control and collaboration for deployment files like task definitions and AppSpec files.

Significance: Centralized management of deployment artifacts ensures consistency and traceability throughout the deployment process.

Task 5.4: Define Task Definitions for Each Microservice and Register with ECS

Analysis: Task definitions specify parameters for running Docker containers within ECS. Defining task definitions for each microservice enables effective deployment and management.

Significance: Specifies container configurations, resource requirements, and networking settings for reliable microservices deployment.

Task 5.5: Create AppSpec Files for CodeDeploy

Analysis: AppSpec files define deployment actions and lifecycle hooks for AWS CodeDeploy. They specify how to deploy updates to microservices.

Significance: Enables automated deployment updates, ensuring consistency and minimizing downtime during deployments.

Task 5.6: Update and Version Control Deployment Files in CodeCommit

Analysis: Updating and version controlling deployment files in CodeCommit ensures availability of the latest configurations for deployment.

Significance: Facilitates version control and traceability of deployment artifacts, allowing rollback to previous versions if needed.

Phase 5 represents a significant advancement, transitioning from basic Docker deployment to a scalable architecture using AWS ECS and CodeDeploy. It addresses scalability, reliability, and efficient management of microservices in production environments.

Analysis of Phase 6: Implementing Load Balancing and Target Groups

Phase 6 focuses on deploying an Application Load Balancer (ALB) and creating target groups to enhance the scalability and reliability of the application.

Outcomes:

Task 6.1: Establish Target Groups:

Purpose: Group instances or containers handling similar traffic.

Outcome: Efficient traffic routing and load distribution among microservices or instances, boosting scalability and reliability.

Task 6.2: Configure ALB and Routing Rules:

Purpose: Set up an ALB as a centralized entry point for users.

Outcome: Enhanced accessibility and reliability by distributing traffic to appropriate target groups based on defined criteria.

Overall, Phase 6 is pivotal in improving application accessibility, scalability, and reliability through the implementation of an ALB and target groups. It ensures optimal performance and high availability by efficiently managing incoming traffic across multiple microservices or instances.

Analysis of Phase 7: Deploying Amazon ECS Services

Phase 7 focuses on creating ECS services for each microservice, enabling independent management and scaling to simplify operations and ensure component isolation.

Outcomes:

Task 7.1: Establish ECS Service for Customer Microservice

Analysis: Defining parameters like task definition and desired task count for deploying the customer microservice within its ECS service.

Significance: Enables autonomous scaling, management, and monitoring of the customer microservice, enhancing flexibility and isolation.

Task 7.2: Deploy Amazon ECS Service for Employee Microservice

Analysis: Similar to Task 7.1, creating an ECS service for the employee microservice with tailored configurations.

Significance: Facilitates separate lifecycle management and scaling for the employee microservice, ensuring resilience and autonomy in handling workload variations.

Overall, Phase 7 streamlines the deployment of microservices as distinct ECS services, fostering modularity, scalability, and ease of management in the application architecture. By segregating microservices into dedicated services, the project team enhances operational efficiency, fault tolerance, and adaptability to evolving needs.

Analysis of Phase 8: Setting Up CodeDeploy and CodePipeline

Phase 8 focuses on configuring a CI/CD (Continuous Integration/Continuous Deployment) pipeline using AWS CodePipeline and AWS CodeDeploy. This phase automates the deployment process, ensuring seamless updates to the microservices in the production environment.

Outcomes:

Task 8.1: Establish CodeDeploy Application and Deployment Groups

Analysis: Defining a CodeDeploy application and deployment groups to specify deployment targets (e.g., ECS services) and configurations (e.g., deployment type, settings).

Significance: Sets up the infrastructure for deploying microservice updates using CodeDeploy, ensuring consistency and reliability.

Task 8.2: Create a Pipeline for the Customer Microservice

Analysis: Setting up a CodePipeline for the customer microservice, defining stages (e.g., source, build, deploy) and actions (e.g., fetching source code, building Docker images, deploying to ECS) to automate deployment.

Significance: Enables automated deployment of customer microservice updates, reducing manual effort and ensuring fast and reliable delivery.

Task 8.3: Test the CI/CD Pipeline for the Customer Microservice

Analysis: Validating the functionality of the CI/CD pipeline for the customer microservice by triggering a deployment and verifying successful updates.

Significance: Ensures correct configuration and operation of the pipeline, instilling confidence in the deployment process.

Task 8.4: Create a Pipeline for the Employee Microservice

Analysis: Similar to Task 8.2, setting up a CodePipeline for the employee microservice tailored to its deployment requirements.

Significance: Automates deployment of employee microservice updates, ensuring consistency and reliability across the application.

Task 8.5: Test the CI/CD Pipeline for the Employee Microservice

Analysis: Validating the functionality of the CI/CD pipeline for the employee microservice, ensuring successful deployment of updates.

Significance: Confirms reliability of the pipeline, minimizing downtime and ensuring smooth deployment process.

Task 8.6: Monitor CodeDeploy's Impact on Load Balancer Listener Rules

Analysis: Observing CodeDeploy's modification of load balancer listener rules to ensure correct routing of traffic to updated microservice versions.

Significance: Validates seamless integration with the load balancer, maintaining uninterrupted access to the application during updates.

Phase 8 facilitates the implementation of a robust CI/CD pipeline, automating microservice deployment and enhancing agility, reliability, and scalability in the application deployment process. It streamlines release cycles, reduces manual effort, and accelerates time-to-market for new features and updates.

Analysis of Phase 9: Triggering Automated Deployments and Scaling Microservices

Phase 9 illustrates the effectiveness of microservices architecture and CI/CD pipelines by showcasing how updates to microservices code trigger automated deployments and scaling actions. It involves adjusting load balancer rules, updating microservice code, pushing new Docker images to ECR, and scaling microservice instances.

Outcomes:

Task 9.1: Implement Access Controls for the Employee Microservice

Analysis: Modifying load balancer listener rules to restrict access to the employee microservice.

Significance: Demonstrates the flexibility of microservices architecture in managing access control and routing traffic based on security and business requirements.

Task 9.2: Update UI for the Employee Microservice and Trigger CI/CD Pipeline

Analysis: Modifying the microservice code, generating a new Docker image, and pushing it to Amazon ECR to automatically trigger the CI/CD pipeline.

Significance: Illustrates the seamless integration between code changes, Docker image builds, and deployment updates facilitated by the CI/CD pipeline.

Task 9.3: Verify Pipeline Execution and Microservice Update

Analysis: Confirming successful execution of the CI/CD pipeline and deployment of the updated Docker image to the production environment.

Significance: Validates the reliability and effectiveness of the CI/CD pipeline in automating deployment updates without manual intervention.

Task 9.4: Test Access to the Employee Microservice

Analysis: Verifying accessibility and functionality of the updated UI for the employee microservice.

Significance: Ensures that microservice updates maintain application integrity and user experience without introducing regressions or issues.

Task 9.5: Scale the Customer Microservice

Analysis: Increasing the number of containers or instances supporting the customer microservice to accommodate increased demand or workload.

Significance: Highlights the scalability advantages of microservices architecture, enabling dynamic resource scaling to ensure optimal performance and responsiveness.

Phase 9 exemplifies the agility, scalability, and automation capabilities inherent in microservices architecture and CI/CD pipelines. It demonstrates how the architecture efficiently handles updates and scaling, ensuring continuous delivery of new features and enhancements while maintaining reliability and performance.