

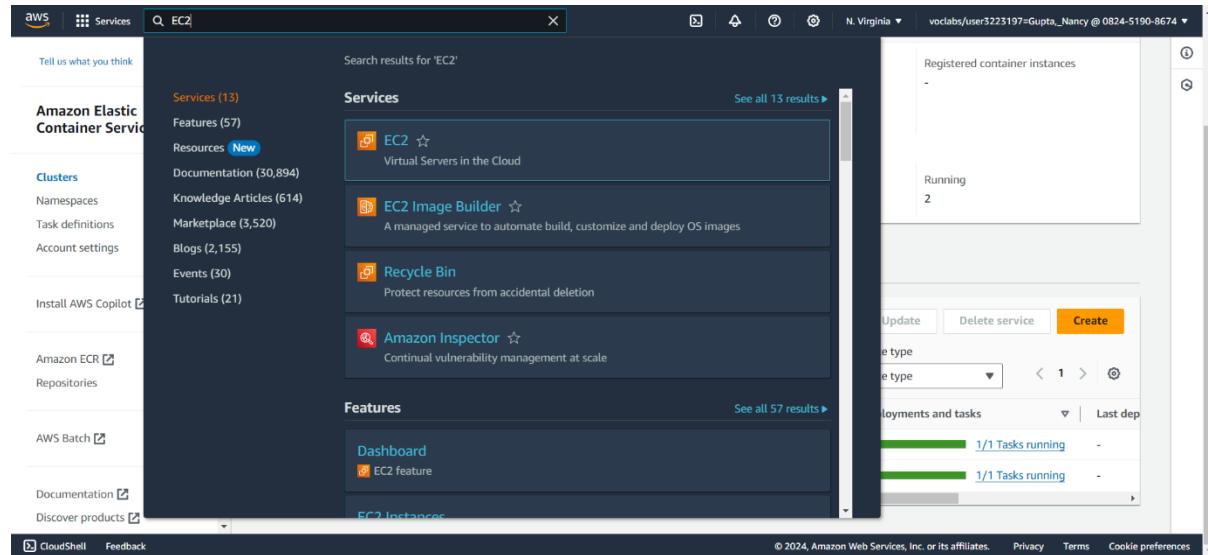
4. Return to the CodeDeploy page to confirm that all five steps of the deployment succeeded and the replacement task set is now serving traffic.

The screenshots illustrate the progression of a deployment:

- Deployment status:** Shows the five steps of the deployment: Step 1 (Deploying replacement task set) completed at 100% success; Step 2 (Test traffic route setup) completed at 100% success; Step 3 (Rerouting production traffic to replacement task set) completed at 100% success; Step 4 (Wait) completed at 100% success; Step 5 (Terminate original task set) completed at 100% success. The Traffic shifting progress shows 0% for the Original task set and 100% for the Replacement task set.
- Deployment details:** Provides deployment metadata: Application (microservices), Deployment ID (d-NMB93F9U5), Status (Succeeded), Deployment configuration (CodeDeployDefault.ECSAllAtOnce), Deployment group (microservices-employee), and Deployment description (none).
- Task set activity:** Lists task sets: ecs-svc/0996791440688041300 (Environment: Replacement, Task set status: PRIMARY, Traffic: 100%, Desired count: 1, Running count: 1, Pending count: 0) and ecs-svc/0164575005052814059 (Environment: Original, Task set status: ACTIVE, Traffic: 0, Desired count: 1, Running count: 0, Pending count: 0).

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



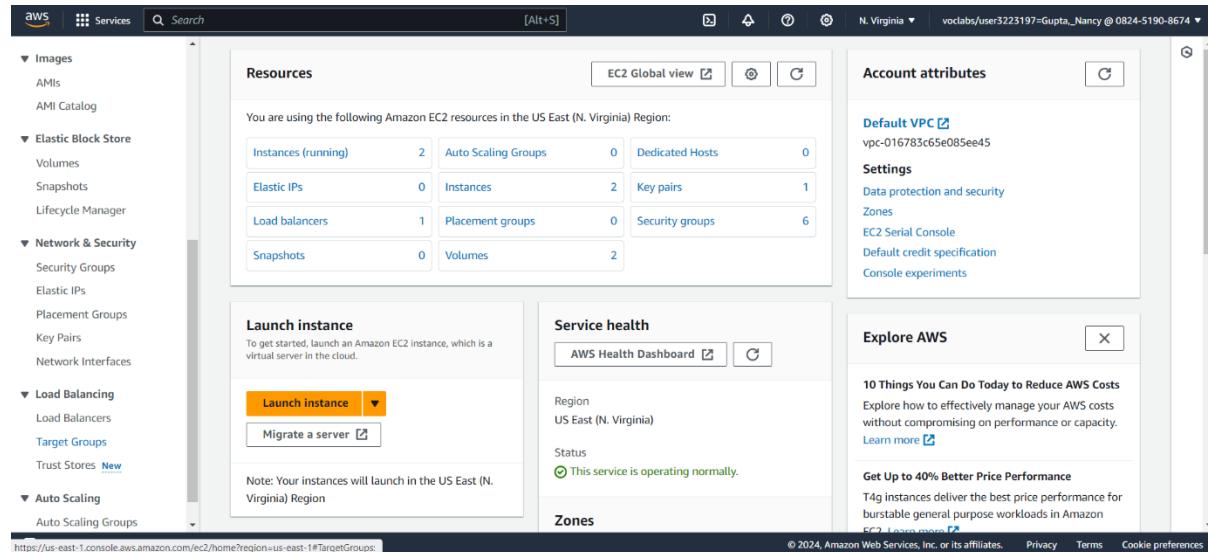
1. Observe the load balancer and target group settings.

- o 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.



The screenshot shows the AWS EC2 Target Groups page. The left sidebar includes sections for Images, Elastic Block Store, Network & Security, Load Balancing, Auto Scaling, and CloudShell. Under Load Balancing, 'Target Groups' is selected. The main content area displays a table titled 'Target groups (4)'. The table columns are Name, ARN, Port, Protocol, Target type, and Load balancer. The rows show:

Name	ARN	Port	Protocol	Target type	Load balancer
customer-tg-one	arn:aws:elasticloadbalancing:us-east-1:082451908674:targetgroup/customer-tg-one/27fed4d4baeb9f7a	8080	HTTP	IP	microservicesLB
customer-tg-two	arn:aws:elasticloadbalancing:us-east-1:082451908674:targetgroup/customer-tg-two/27fed4d4baeb9f7a	8080	HTTP	IP	None associated
employee-tg-one	arn:aws:elasticloadbalancing:us-east-1:082451908674:targetgroup/employee-tg-one/27fed4d4baeb9f7a	8080	HTTP	IP	microservicesLB
employee-tg-two	arn:aws:elasticloadbalancing:us-east-1:082451908674:targetgroup/employee-tg-two/27fed4d4baeb9f7a	8080	HTTP	IP	None associated

Below the table, a message says '0 target groups selected' and 'Select a target group above.'

The screenshot shows the AWS EC2 Load Balancers page. The left sidebar includes sections for Images, Elastic Block Store, Network & Security, Load Balancing, Auto Scaling, and CloudShell. Under Load Balancing, 'Load Balancers' is selected. The main content area displays a table titled 'Load balancers (1)'. The table columns are Name, DNS name, State, VPC ID, Availability Zones, Type, and Date created. The row shows:

Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
microservicesLB	microservicesLB-1849491...	Active	vpc-0d0d3f1f2eb389fdd	2 Availability Zones	application	April 29, 2024, 02:58 (UTC-04:00)

Below the table, a message says '0 load balancers selected' and 'Select a load balancer above.'

- 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*.

The screenshot shows the AWS EC2 Load Balancers details page for the 'microservicesLB' load balancer. The left sidebar includes sections for Images, Elastic Block Store, Network & Security, Load Balancing, Auto Scaling, and CloudShell. Under Load Balancing, 'Load Balancers' is selected. The main content area shows the 'Details' section for the 'microservicesLB' load balancer. The table includes fields for 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-a2z4) subnet-089933df8ddacc089 us-east-1b (use1-a2z6)), and DNS name (microservicesLB-1849491881.us-east-1.elb.amazonaws.com (A Record)). Below the table, tabs for 'Listeners and rules', 'Network mapping', 'Resource map - new', 'Security', 'Monitoring', 'Integrations', 'Attributes', and 'Tags' are visible. The 'Listeners and rules' tab is selected, showing 'Listeners and rules (2)'. At the bottom, there are buttons for 'Manage rules', 'Manage listener', 'Add listener', and copyright information.

Screenshot of the AWS CloudFront console showing the configuration of a Load balancer ARN and DNS name.

Load balancer ARN: arn:aws:elasticloadbalancing:us-east-1:082451908674:loadbalancer/app/microservicesLB/27fed4d4baeb9f7a

DNS name: microservicesLB-1849491881.us-east-1.elb.amazonaws.com (A Record)

Listeners and rules:

- Protocol:Port:** HTTP:80
 - Default action:** Forward to target group
 - customer-tg-one: 1 (100%)
 - Group-level stickiness: Off
 - Rules:** 2 rules
 - ARN:** Not applicable
 - Security policy:** Not applicable
 - Default SSL/TLS:** Not applicable
- Protocol:Port:** HTTP:8080
 - Default action:** Forward to target group
 - customer-tg-one: 1 (100%)
 - Group-level stickiness: Off
 - Rules:** 2 rules
 - ARN:** Not applicable
 - Security policy:** Not applicable
 - Default SSL/TLS:** Not applicable

Screenshot of the AWS CloudFront console showing the configuration of an HTTP:80 listener.

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

Listener rules:

- Protocol:Port:** HTTP:80
 - Name tag:** -
 - Priority:** 1
 - Conditions (If):** Path Pattern is /admin/
 - Actions (Then):** Forward to target group
 - employee-tg-one: 1 (100%)
 - Group-level stickiness: Off
- Default:** Last (default)
 - Conditions (If):** If no other rule applies
 - Actions (Then):** Forward to target group
 - customer-tg-one: 1 (100%)
 - Group-level stickiness: Off

Screenshot of the AWS CloudFront console showing the configuration of an HTTP:80 listener.

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

Listener rules:

- Protocol:Port:** HTTP:80
 - Name tag:** -
 - Priority:** 1
 - Conditions (If):** Path Pattern is /admin/
 - Actions (Then):** Forward to target group
 - employee-tg-one: 1 (100%)
 - Group-level stickiness: Off
- Default:** Last (default)
 - Conditions (If):** If no other rule applies
 - Actions (Then):** Forward to target group
 - customer-tg-one: 1 (100%)
 - Group-level stickiness: Off

- Observe the HTTP:8080 listener rules.

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

HTTP:8080

Details

Protocol: Port
HTTP:8080

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/dfca4386c207d6c](#)

Rules

Listener rules (2)

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

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/dfca4386c207d6c](#)

Rules

Listener rules (2)

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.

Name tag	Priority	Conditions (If)	Actions (Then)	ARN
-	1	Path Pattern is /admin/*	Forward to target group	ARN
Default	Last (default)	If no other rule applies	Forward to target group	ARN

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.

The screenshot shows the AWS EC2 console home page. On the left, a navigation sidebar lists various services like AMIs, Elastic Block Store, Network & Security, Load Balancing, Auto Scaling, and more. The main content area displays a summary of resources in the US East (N. Virginia) Region:

Instances (running)	2	Auto Scaling Groups	0	Dedicated Hosts	0
Elastic IPs	0	Instances	2	Key pairs	1
Load balancers	1	Placement groups	0	Security groups	6
Snapshots	0	Volumes	2		

Below the summary, there are sections for "Launch instance" (with "Launch instance" and "Migrate a server" buttons) and "Service health" (showing "US East (N. Virginia)" region and "This service is operating normally"). A sidebar on the right provides links for "Default VPC", "Settings", and "Explore AWS" (with information about Spot Instances and Graviton2).

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.

The screenshot shows the "Target groups" page under the EC2 service. The sidebar includes "Target Groups" under the "Load Balancing" section. The main content displays a table of target groups:

Name	ARN	Port	Protocol	Target type	Load balancer
customer-tg-one	arn:aws:elasticloadbalanc...	8080	HTTP	IP	microservicesLB
customer-tg-two	arn:aws:elasticloadbalanc...	8080	HTTP	IP	None associated
employee-tg-one	arn:aws:elasticloadbalanc...	8080	HTTP	IP	microservicesLB
employee-tg-two	arn:aws:elasticloadbalanc...	8080	HTTP	IP	None associated

At the bottom, it says "0 target groups selected" and "Select a target group above."

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

Screenshot of the AWS CloudShell interface showing the EC2 Load Balancers page. A single load balancer named "microservicesLB" is listed as active. The VPC is "vpc-0d0d3f1f2eb389fdd" and it has two availability zones: "us-east-1a (use1-az4)" and "us-east-1a (use1-az2)". The IP address type is IPv4.

Screenshot of the AWS CloudShell interface showing the details of the load balancer "microservicesLB". The load balancer type is Application, status is Active, and the scheme is Internet-facing. The VPC is "vpc-0d0d3f1f2eb389fdd". The hosted zone is "Z355XDOTRQ7X7K". Availability zones include "subnet-0c1ad2d13f3070aa7" (us-east-1a) and "subnet-089933df8ddacc089" (us-east-1b). The DNS name is "microservicesLB-1849491881.us-east-1.elb.amazonaws.com".

Screenshot of the AWS CloudShell interface showing the listeners and rules for the load balancer "microservicesLB". There are two listeners: one for "HTTP:80" and one for "HTTP:8080". Both listeners forward traffic to target groups. The "HTTP:80" listener has a default action of "Forward to target group" with a rule for "customer-tg-one" (100% weight) and group-level stickiness off. The "HTTP:8080" listener also has a default action of "Forward to target group" with a rule for "customer-tg-one" (100% weight) and group-level stickiness off.

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.**

The screenshot shows the AWS EC2 Load Balancers console. In the navigation pane, under 'Load Balancing', 'Target Groups' is selected. In the main content area, the 'microservicesLB' load balancer is selected, and the 'HTTP:80' listener is viewed. The 'Details' tab shows the protocol as 'HTTP:80' and the load balancer as 'microservicesLB'. The 'Default actions' section shows 'Forward to target group' with one rule: 'customer-tg-one' (100%) and 'Group-level stickiness: Off'. The 'Listener ARN' is listed as 'arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/78c02a234a25a5b5'. Below this, the 'Rules' tab is selected, showing two rules: 'customer-tg-one' (Priority 1, Path Pattern '/admin/*') and 'Default' (Priority Last (default), If no other rule applies). Both rules forward to the target group 'customer-tg-one'.

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.

The screenshot shows three sequential views of the AWS CloudFront console for managing a load balancer named "microservicesLB".

View 1: Load balancers (1/1)

- Shows a table with one row for "microservicesLB".
- Details: Application load balancer, Active, VPC vpc-0d0d3f1f2eb389fdd, IPv4, Internet-facing.
- Listeners and rules tab is selected.

View 2: microservicesLB - Details

- Shows detailed information about the load balancer.
- Listeners and rules tab is selected.

View 3: microservicesLB - Listener and rules (2)

- Shows two listeners: HTTP:80 and HTTP:8080.
- HTTP:80 listener details:
 - Protocol:Port: HTTP:80
 - Default action: Forward to target group (customer-tg-one, 100%)
 - Rules: 2 rules
 - ARN: Not applicable
 - Security policy: Not applicable
- HTTP:8080 listener details:
 - Protocol:Port: HTTP:8080
 - Default action: Forward to target group (customer-tg-one, 100%)
 - Rules: 2 rules
 - ARN: Not applicable
 - Security policy: Not applicable

- 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.

Listener rules (1/2) Info

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.

Name tag	Priority	Conditions (If)	Actions (Then)
-	1	Path Pattern is /admin/*	Forward to target group • customer-tg-one: 1 (100%) • Group-level stickiness: Off
Default	Last (default)	If no other rule applies	Forward to target group • customer-tg-one: 1 (100%) • Group-level stickiness: Off

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/78c02a234a25a5b5

Listener configuration

The listener will be identified by the protocol and port.

Protocol Used for connections from clients to the load balancer.	Port The port on which the load balancer is listening for connections.
HTTP	80

Default actions Info
The default action is used if no other rules apply. Choose the default action for traffic on this listener.

Routing actions

The screenshot shows the AWS Elastic Load Balancing console. On the left, there's a sidebar with navigation links like EC2 Dashboard, Instances, Images, and Elastic Block Store. The main panel is titled 'Default actions' and 'Info'. It says 'The default action is used if no other rules apply. Choose the default action for traffic on this listener.' Under 'Routing actions', there are three options: 'Forward to target groups' (selected), 'Redirect to URL', and 'Return fixed response'. Below this, it says 'Forward to target group' and 'Info'. It asks to choose a target group and specify routing weight or 'Create target group'. A table shows a single target group entry: 'customer-tg-one' (Target type: IP, IPv4), 'HTTP' (Protocol), '1' (Weight), and '100%' (Percent). There's also a link to 'Add target group' and a note about adding up to 4 more target groups. A section for 'Group-level stickiness' is present with an info link and a checkbox for 'Turn on group-level stickiness'. At the bottom, a box contains '▶ Server-side tasks and status' with a note about monitoring completed steps. At the very bottom, there are 'Cancel' and 'Save changes' buttons.

This screenshot continues from the previous one, showing the 'Listener rules' section. It lists two rules: 'Path Pattern is /admin/' (Priority 1) which forwards to 'employee-tg-one', and a 'Default' rule (Last, default) which forwards to 'customer-tg-one'. The right side of the screen shows the 'Default actions' panel, which is currently set to 'Forward to target group' with 'customer-tg-one' selected at 100% weight. Group-level stickiness is turned off. The bottom of the screen has standard AWS footer links.

- 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**.

HTTP:80

microservicesLB

Listener ARN
arnaws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/78c02a234a25a5bs

Rules

Listener rules (1/2) Info

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.

Name tag	Priority	Conditions (If)	Actions (Then)	ARN
<input checked="" type="checkbox"/>	1	Path Pattern is /admin/*	Forward to target group • employee-tg-one 1 (100%) • Group-level stickiness: Off	<input type="button" value="ARN"/>
<input type="checkbox"/>	Last (default)	If no other rule applies	Forward to target group • customer-tg-one 1 (100%) • Group-level stickiness: Off	<input type="button" value="ARN"/>

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

The following image shows the original configuration for this listener:

Load balancer ARN
arnaws:elasticloadbalancing:us-east-1:082451908674:loadbalancer/app/microservicesLB/27fed4d4baeb9f7a

DNS name info
subnet-089953df8ddacc089 us-east-1b (use1-az6)

Listeners and rules (2) Info

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.

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS
HTTP:80	Forward to target group • customer-tg-one 1 (100%) • Group-level stickiness: Off	2 rules	<input type="button" value="ARN"/>	Not applicable	Not applicable
HTTP:8080	Forward to target group • customer-tg-one 1 (100%) • Group-level stickiness: Off	2 rules	<input type="button" value="ARN"/>	Not applicable	Not applicable

EC2 > Load balancers > microservicesLB > HTTP:8080 listener

HTTP:8080 Info

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-one 1 (100%) • Group-level stickiness: Off
----------------------------	--	---

Listener ARN
arnaws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c

Rules

Listener rules (2) Info

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.

Name tag	Priority	Conditions (If)	Actions (Then)	ARN
<input type="checkbox"/>	1	Path Pattern is /admin/*	Forward to target group • employee-tg-one 1 (100%) • Group-level stickiness: Off	<input type="button" value="ARN"/>
<input type="checkbox"/>	Last (default)	If no other rule applies	Forward to target group • customer-tg-one 1 (100%) • Group-level stickiness: Off	<input type="button" value="ARN"/>

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:

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

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

The screenshot shows the AWS EC2 Load Balancers page. On the left sidebar, under the 'Load Balancing' section, 'Load Balancers' is selected. In the main content area, there is a table titled 'Load balancers (1/1)'. The table has columns for Name, DNS name, State, VPC ID, Availability Zones, Type, and Date created. One row is selected, showing 'microservicesLB' as the Name, 'microservicesLB-1849491...' as the DNS name, 'Active' as the state, 'vpc-0d0d3f1f2eb389fd' as the VPC ID, '2 Availability Zones' as the availability zones, 'application' as the type, and 'April 29, 2024, 02:58 (UTC-04:00)' as the date created.

- On the **Listeners and rules** tab, choose the [HTTP:8080](#) link.

The screenshot shows the AWS Load Balancer ARN and Listener rules panel. At the top, it displays the ARN: arnaws:elasticloadbalancing:us-east-1:082451908674:loadbalancer/app/microservicesLB/27fed4d4baeb9f7a. Below this, the 'Listeners and rules' tab is selected, showing two rules: 'HTTP:80' and 'HTTP:8080'. Both rules forward traffic to the 'customer-tg-one' target group at 100% weight. The 'Default SSL/TLS' column indicates 'Not applicable' for both rules.

- 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**.

aws Services Search [Alt+S] N. Virginia v oclabs/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

CloudShell Feedback

HTTP:8080 microservicesLB

Listener ARN arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2

Forward to target group

- customer-tg-one 1 (100%)
- Group-level stickiness: Off

Rules Tags

Listener rules (1/2) Info

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.

Filter rules

Name tag	Priority	Conditions (If)	Actions (Then)
	1	Path Pattern is /admin/*	Forward to target group
<input checked="" type="checkbox"/> Default	Last (default)	If no other rule applies	Forward to target group

Rule limits Actions ▲ Add rule View rule Edit rule Delete rule Reprioritize rules

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

aws Services Search [Alt+S] N. Virginia v oclabs/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.	Port	The port on which the load balancer is listening for connections.
HTTP		8080	1-65535

Default actions Info

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Default actions [Info](#)
The default action is used if no other rules apply. Choose the default action for traffic on this listener.

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
customer-tg-two Target type: IP, IPv4	HTTP	<input type="text" value="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

Server-side tasks and status
After completing and submitting the above steps, all server-side tasks and their statuses become available for monitoring.

Cancel **Save changes**

CloudShell Feedback Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta_Nancy @ 0824-5190-8674

Protocol:Port: HTTP:8080 Load balancer: microservicesLB Default actions: Forward to target group: customer-tg-two (100%) Group-level stickiness: Off

Listener ARN: arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c

Rules Tags

Listener rules (2) [Info](#)
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-one (100%) Group-level stickiness: Off	ARN
	Default	Last (default)	If no other rule applies	Forward to target group: customer-tg-two (100%) Group-level stickiness: Off	ARN

Rule limits: [Edit](#) Actions: [Edit](#) Add rule

© 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/dfca4386c207d6c

Rules [Tags](#)

Listener rules (1/2) Info

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.

Name tag	Priority	Conditions (If)	Actions (Then)
<input checked="" type="checkbox"/> -	1	Path Pattern is /admin/*	Forward to target group <ul style="list-style-type: none"> employee-tg-one: 1 (100%) Group-level stickiness: Off
<input type="checkbox"/> Default	Last (default)	If no other rule applies	Forward to target group <ul style="list-style-type: none"> customer-tg-two: 1 (100%) Group-level stickiness: Off

Actions ▲ [View rule](#) [Edit rule](#) [Delete rule](#) [Reprioritize rules](#)

© 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

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)

Path (1) Info

If Path is /admin/*

Next

© 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

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
1	100%
0-999	

Add target group

You can add up to 4 more target groups.

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

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
employee-tg-two HTTP
Target type: IP, IPv4

Weight	Percent
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

Priority	Conditions (If)	Actions (Then)
1	If request matches all: Path Pattern is /admin/*	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 X

Protocol:Port
HTTP:8080

Load balancer
[microservicesLB](#)

Listener ARN
[arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c2](#)

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

Rules Tags

Listener rules (2) [Info](#)
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.

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

5. Confirm that all target groups are reassociated.

- o 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.

The screenshot shows the AWS EC2 Target Groups page. The left sidebar includes links for EC2 Dashboard, EC2 Global View, Events, Console-to-Code Preview, Instances, Images, AMIs, AMI Catalog, and Elastic Block Store. The main content area is titled "Target groups (4) Info" and displays a table with columns: Name, ARN, Port, Protocol, Target type, and Load balancer. The table lists four entries, each with a checkbox next to it. The "Load balancer" column for all entries is set to "microservicesLB". At the bottom of the table, it says "0 target groups selected" and "Select a target group above." The top right of the page has a "Create target group" button. The bottom right corner of the page includes copyright information: "© 2024, Amazon Web Services, Inc. or its affiliates." and links for Privacy, Terms, and Cookie preferences.

2. Discover your public IPv4 address.

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

The screenshot shows the WhatIsMyIP.com website. The top navigation bar includes links for Careers, GitHub - StevensDe..., Eligible CIP Codes f..., Applying for OPT S..., STEM OPT Extension..., CPT Work Authoriz..., CPT Frequently Asked..., Computer Architect..., Cisco Skills For All, and All Bookmarks. Below the navigation bar, there are search and sign-in options. The main content area features a large green box with the heading "What Is My IP?". Inside the box, it displays "My Public IPv4: 72.76.113.56", "My Public IPv6: Not Detected", "My IP Location: Newark, NJ US", and "My ISP: Verizon Business". Below this box are three expandable sections with arrows: "What is an IP address?", "What is a private IP address?", and "How do I find my IP address?".

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 vclabs/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

CloudShell Feedback

EC2 > Load balancers

Load balancers (1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Filter load balancers

Name	DNS name	State	VPC ID	Availability Zones	Type
microservicesLB	microservicesLB-1849491...	Active	vpc-0d0d3f1f2eb589fdd	2 Availability Zones	application

0 load balancers selected

Select a load balancer above.

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia vclabs/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

CloudShell Feedback

EC2 > Load balancers > microservicesLB

microservicesLB

▼ Details

Load balancer type Application	Status Active	VPC vpc-0d0d3f1f2eb589fdd	IP address type IPv4
Scheme Internet-facing	Hosted zone Z355XD0TRQ7X7K	Availability Zones subnet-0c1ad2d13f3070aa7 us-east-1a (use1-az2) subnet-089933df8ddace089 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 (2) Info

Listeners and rules (2) Manage rules Manage listener Add listener

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia vclabs/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

CloudShell Feedback

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 (2) Info

Listeners and rules (2) 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.

Listeners and rules (2) Manage rules Manage listener Add listener

Protocol:Port Default action Rules ARN Security policy Default SS

HTTP:80	Forward to target group customer-tg-one: 1 (100%) Group-level stickiness: Off	2 rules	ARN	Not applicable	Not applicable
HTTP:8080	Forward to target group customer-tg-two: 1 (100%) Group-level stickiness: Off	2 rules	ARN	Not applicable	Not applicable

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia v vocabs/user\$223197=Gupta_Nancy @ 0824-5190-8674 ▾

AMIls 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

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

EC2 > Load balancers > microservicesLB > HTTP:80 listener

HTTP:80

Details

Protocol:Port Listener ARN

Load balancer: microservicesLB

Default actions: Forward to target group

- customer-tg-one: 1 (100%)
- Group-level stickiness: Off

Listener rules (2)

Rule limits Actions Add rule

Filter rules

aws Services Search [Alt+S] N. Virginia v vocabs/user\$223197=Gupta_Nancy @ 0824-5190-8674 ▾

AMIls 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

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

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)

Rule limits Actions Add rule View rule Edit rule Delete rule Reprioritize rules

Filter rules

Name tag	Priority	Conditions (If)	Actions (Then)
<input checked="" type="checkbox"/> -	1	Path Pattern is /admin/*	Forward to target group
<input type="checkbox"/> Default	Last (default)	If no other rule applies	Forward to target group

aws Services Search [Alt+S] N. Virginia v vocabs/user\$223197=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

aws Services Search [Alt+S] N. Virginia v vocabs/user\$223197=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

Add condition Rule limits Info X

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 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

Cancel Next

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

aws Services Search [Alt+S] N. Virginia v vocabs/user\$223197=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 (2) Rule limits Edit Delete Add condition

Path (1) Info
If Path
is /admin/*

Source IP (1) Info
If Source IP
is 72.76.113.56/32

AND AND

Cancel Next

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

aws Services Search [Alt+S] N. Virginia v vocabs/user\$223197=Gupta_Nancy @ 0824-5190-8674 ▾

Define rule actions Info

► 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 Info

Target group	Type	Weight	Percent
employee-tg-one	HTTP	<input type="text" value="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

Review changes

Listener details: HTTP:80

Rule details

Priority	Conditions (If)	Actions (Then)
1	If request matches all: • Path Pattern is /admin/*, AND • Source IP is 72.76.113.56/32	Forward to target group • employee-tg-one: 1 (100%) • Group-level stickiness: Off

Rule ARN: arn:aws:elasticloadbalancing:us-east-1:082451908674:listener-rule/app/microservicesLB/27fed4d4baeb9f7a/78c02a234a25a5b5/a1732d51212a1fbf

Cancel Previous Save changes

Protocol:Port: HTTP:80

Load balancer: microservicesLB

Listener ARN: arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/78c02a234a25a5b5

Rules

Listener rules (2)

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.

Name tag	Priority	Conditions (If)	Actions (Then)	ARN
-	1	• Source IP is 72.76.113.56/32, AND • 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

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 v vocabs/user\$223197=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

CloudShell Feedback

EC2 > Load balancers > microservicesLB > HTTP:8080 listener

HTTP:8080 [info](#)

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 <ul style="list-style-type: none">customer-tg-two: 1 (100%)Group-level stickiness: Off
Listener ARN arn:aws:elasticloadbalancing:us-east-1:082451908674:listener/app/microservicesLB/27fed4d4baeb9f7a/dfca4386c207d6c		

Rules Tags

Listener rules (2) [info](#)
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

Rule limits Actions Add rule

Name tag	Priority	Conditions (If)	Actions (Then)	ARN
-	1	Path Pattern is /admin/*	Forward to target group <ul style="list-style-type: none">employee-tg-two: 1 (100%)Group-level stickiness: Off	ARN
Default	Last (default)	If no other rule applies	Forward to target group <ul style="list-style-type: none">customer-tg-two: 1 (100%)Group-level stickiness: Off	ARN

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia v vocabs/user\$223197=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

CloudShell Feedback

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/dfca4386c207d6c](#)

Rules Tags

Listener rules (2) [info](#)
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

Rule limits Actions Add rule

Name tag	Priority	Conditions (If)	Actions (Then)	ARN
-	1	Path Pattern is /admin/*	Forward to target group <ul style="list-style-type: none">employee-tg-two: 1 (100%)Group-level stickiness: Off	ARN
Default	Last (default)	If no other rule applies	Forward to target group <ul style="list-style-type: none">customer-tg-two: 1 (100%)Group-level stickiness: Off	ARN

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia v vocabs/user\$223197=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

CloudShell Feedback

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/dfca4386c207d6c](#)

Rules Tags

Listener rules (1/2) [info](#)
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

Rule limits Actions View rule Edit rule Delete rule Reprioritize rules

Name tag	Priority	Conditions (If)	Actions (Then)	ARN
-	1	Path Pattern is /admin/*	Forward to target group <ul style="list-style-type: none">employee-tg-two: 1 (100%)Group-level stickiness: Off	ARN
Default	Last (default)	If no other rule applies	Forward to target group <ul style="list-style-type: none">customer-tg-two: 1 (100%)Group-level stickiness: Off	ARN

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS CloudFront Rule Editor - Step 1: Define rule conditions.

The page shows a navigation path: 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)

Path (1) Info

If Path
is /admin/*

Rule limits Edit Delete Add condition

Cancel Next

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS CloudFront Rule Editor - Step 2: Add condition.

The page shows a navigation path: 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

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

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS CloudFront Rule Editor - Step 3: Review changes.

The page shows a navigation path: 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)

Path (1) Info

If Path
is /admin/*

Source IP (1) Info

If Source IP
is 72.76.113.56/32

Rule limits Edit Delete Add condition

Cancel Next

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

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	HTTP	1 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](#)

Review changes

Listener details: HTTP:8080

Rule details

Priority	Conditions (If)	Actions (Then)
1	If request matches all: <ul style="list-style-type: none"> • Path Pattern is /admin/*, AND • Source IP is 72.76.113.56/32 	Forward to target group <ul style="list-style-type: none"> • 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](#)

Successfully updated rule on listener HTTP:8080.

HTTP:8080 [Info](#)

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 <ul style="list-style-type: none"> • 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](#)
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.

[Rule limits](#) [Actions](#) [Add rule](#)

[Cancel](#) [Actions](#)

The screenshot shows the AWS CloudWatch Metrics interface. A metric named "customer-tg-two" is selected, showing a value of 100%. The interface includes a search bar, navigation tabs, and a detailed view of the metric's configuration.

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.

The screenshot shows the AWS Cloud9 IDE interface. On the left is a file explorer with project files like "MicroservicesIDE", "aws", and "views". The main area displays the "Welcome" screen with the message "Welcome to your development environment". Below it is a "Getting started" section and a terminal window titled "bash - [ip-10-16-10-131.e]" showing a command prompt. The top menu bar includes options like File, Edit, Find, View, Go, Run, Tools, Window, Support, Preview, and Run.

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

```

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  
  <div><a class="navbar-brand page-title" href="/supplier">Manage coffee suppliers</a></div>
  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="/admin/suppliers">Administrator home</a>
      <li class="nav-item">
        <a class="nav-link" href="/suppliers">Suppliers list</a>
      <li class="nav-item">
        <a class="nav-link" href="/">Customer home</a>
      </li>
    </ul>
  </div>
</nav>

```

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

```

<nav class="navbar navbar-expand-lg navbar-light bg-light">
  
  <div><a class="navbar-brand page-title" href="/supplier">Manage coffee suppliers</a></div>
  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="/admin/suppliers">Administrator home</a>
      <li class="nav-item">
        <a class="nav-link" href="/suppliers">Suppliers list</a>
      <li class="nav-item">
        <a class="nav-link" href="/">Customer home</a>
      </li>
    </ul>
  </div>
</nav>

```

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
```

The screenshot shows the AWS Cloud9 IDE interface. On the left, the file tree displays files like `nav.html`, `supplier-add.html`, and `Dockerfile`. The main editor window shows the `nav.html` code:

```

1 <nav class="navbar-expand-lg navbar-light bg-light">
2   
3   <div><a class="navbar-brand page-title" href="/supplier">Manage coffee suppliers</a></div>
4   <div class="collapse navbar-collapse" id="navSupportedContent">
5     <ul class="nav flex-grow-1 justify-content-end">
6       <li class="nav-item active">
7         <a class="nav-link" href="/admin/suppliers">Administrator home</a>
8       <li class="nav-item">
9         <a class="nav-link" href="/suppliers">Suppliers list</a>
10      <a class="nav-link" href="/Customer/home">Customer home</a>
11    </ul>
12  </div>
13 </nav>

```

The terminal window at the bottom shows the command `docker rm -f employee_1` being run.

`cd ~/environment/microservices/employee`

The screenshot shows the AWS Cloud9 IDE interface. The main editor window displays the AWS Cloud9 welcome message: "AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share your IDE with others in real time, and much more." Below it, a "Getting started" button is visible. The terminal window shows the command `docker rm -f employee_1` being run again.

`docker build --tag employee .`

The screenshot shows the AWS Cloud9 IDE interface. The terminal window displays the Docker build process for the `employee` image:

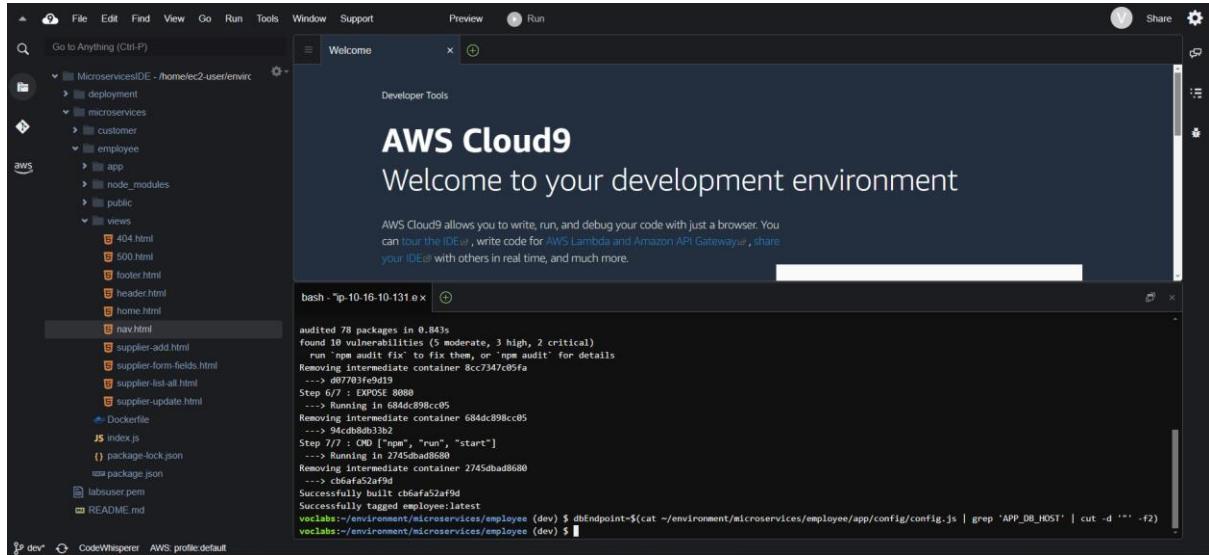
```

bash - *ip-10-16-10-131.e x
employee_1
voclabs:/environment $ cd ~/environment/microservices/employee
voclabs:/environment/microservices/employee (dev) $ docker build --tag employee .
Sending build context to Docker daemon 9.014MB
Step 1/7 : FROM node:11-alpine
--> f18da2f58c3d
Step 2/7 : RUN mkdir -p /usr/src/app
--> Using cache
--> 39f01946424
Step 3/7 : WORKDIR /usr/src/app
--> Using cache
--> 39f01946424
Step 4/7 : COPY .
--> ab03a737639
Step 5/7 : RUN npm install
--> Running in 8cc7347c05fa
npm WARN coffee_apiclient@1.0.0 No repository field.

audited 78 packages in 0.843s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 8cc7347c05fa
--> ab03a737639
Step 6/7 : EXPOSE 8088
--> Running in 684dc899cc05
Removing intermediate container 684dc899cc05
--> 94cdb8db3b2
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in 2745dbad8680
Removing intermediate container 2745dbad8680
--> cb6af52a9d
Successfully built cb6af52a9d
Successfully tagged employee:latest
voclabs:/environment/microservices/employee (dev) $

```

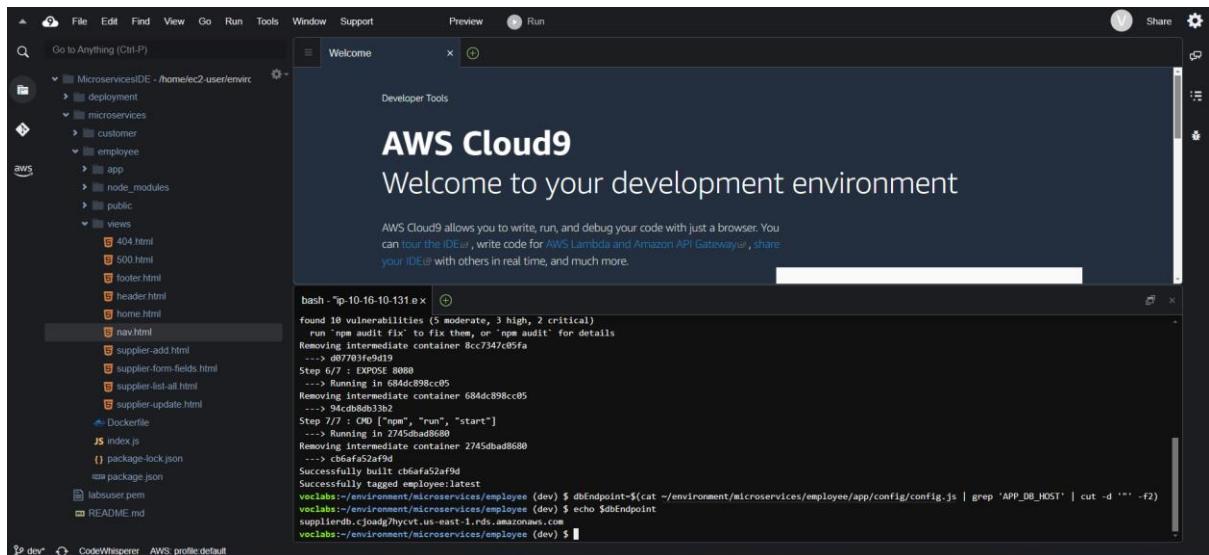
```
dbEndpoint=$(cat ~environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
```



The screenshot shows the AWS Cloud9 IDE interface. On the left is a file explorer with a tree view of files and folders related to a project named "MicroservicesIDE". The current file open in the editor is "nav.html". In the bottom right terminal window, the command `dbEndpoint=\$(cat ~environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)` is run, and the output shows the variable being assigned a value.

```
bash -> ip-10-16-10-131 ~ x
audited 78 packages in 0.843s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run 'npm audit fix' to fix them, or 'npm audit' for details
Removing intermediate container 8cc7347c05fa
--> d6770f0ed919
Step 6/7 : EXPOSE 8088
--> Running in 684dc898cc05
Removing intermediate container 684dc898cc05
--> 94cdb8b33b2
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in 2745dbad8680
Removing intermediate container 2745dbad8680
--> cb6afa52af9d
Successfully built cb6afa52af9d
Successfully tagged employee:latest
voclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:~/environment/microservices/employee (dev) $
```

```
echo $dbEndpoint
```



The screenshot shows the AWS Cloud9 IDE interface. The file explorer on the left shows the same project structure as the previous screenshot. The terminal window at the bottom shows the command `echo \$dbEndpoint` being run, and it outputs the value of the `dbEndpoint` variable.

```
bash -> ip-10-16-10-131 ~ x
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run 'npm audit fix' to fix them, or 'npm audit' for details
Removing intermediate container 8cc7347c05fa
--> d6770f0ed919
Step 6/7 : EXPOSE 8088
--> Running in 684dc898cc05
Removing intermediate container 684dc898cc05
--> 94cdb8b33b2
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in 2745dbad8680
Removing intermediate container 2745dbad8680
--> cb6afa52af9d
Successfully built cb6afa52af9d
Successfully tagged employee:latest
voclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
voclabs:~/environment/microservices/employee (dev) $
```

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

```

bash -ip-10-16-10-131.e.x
  run 'npm audit fix' to fix them, or 'npm audit' for details
  can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share
  your IDEs with others in real time, and much more.

Removing Intermediate container 8cc7347c05fa
-> 80790fed919
Step 6/7 : EXPOSE: 8088
--> Running in 684dc898cc05
Removing intermediate container 684dc898cc05
--> 94cdb8d3b32
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in 2745dbad8680
Removing Intermediate container 2745dbad8680
--> cb6af52af9d
Successfully built cb6af52af9d
Successfully tagged employee:latest
voclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '' -f2)
voclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.cjoadg7hycvt.us-east-1.rds.amazonaws.com
voclabs:~/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '' -f4)
voclabs:~/environment/microservices/employee (dev) $

```

echo \$account_id

```

bash -ip-10-16-10-131.e.x
--> d87705fe9d19
Step 6/7 : EXPOSE: 8088
--> Running in 684dc898cc05
Removing Intermediate container 684dc898cc05
--> 94cdb8d3b32
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in 2745dbad8680
Removing intermediate container 2745dbad8680
--> cb6af52af9d
Successfully built cb6af52af9d
Successfully tagged employee:latest
voclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '' -f2)
voclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.cjoadg7hycvt.us-east-1.rds.amazonaws.com
voclabs:~/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '' -f4)
voclabs:~/environment/microservices/employee (dev) $ echo $account_id
082451908674
voclabs:~/environment/microservices/employee (dev) $

```

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

```

bash -ip-10-16-10-131.e.x
Step 6/7 : EXPOSE: 8088
--> Running in 684dc898cc05
Removing intermediate container 684dc898cc05
--> 94cdb8d3b32
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in 2745dbad8680
Removing Intermediate container 2745dbad8680
--> cb6af52af9d
Successfully built cb6af52af9d
Successfully tagged employee:latest
voclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '' -f2)
voclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.cjoadg7hycvt.us-east-1.rds.amazonaws.com
voclabs:~/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '' -f4)
voclabs:~/environment/microservices/employee (dev) $ echo $account_id
082451908674
voclabs:~/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
voclabs:~/environment/microservices/employee (dev) $

```

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

The screenshot shows the AWS search interface with the query 'codePipeline'. The results are categorized into Services, Resources, Documentation, and Troubleshooting. The 'CodePipeline' service is highlighted in the Services section. To the right, there is a detailed view of a target group named 'omer-tg-two' with a 100% healthy status.

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

Name	Latest execution status	Latest source revisions	Latest execution started	Most recent executions
update-employee-microservice (Type: V2 Execution mode: QUEUED)	Succeeded	Image - sha256:6; Source - 6205e182; pushing 4 files created in deployment directory	8 hours ago	View details
update-customer-microservice (Type: V2 Execution mode: QUEUED)	Succeeded	Source - 6205e182; pushing 4 files created in deployment directory Image - sha256:7;	8 hours ago	View details

Developer Tools **CodePipeline**

Source • CodeCommit
Artifacts • CodeArtifact
Build • CodeBuild
Deploy • CodeDeploy
Pipeline • CodePipeline
Getting started
Pipelines **Pipeline**
History
Settings
Settings

Source • Go to resource
Feedback

CloudShell Feedback

Source Succeeded
Pipeline execution ID: 2f07f7d1-009c-4bac-b391-c273302d4055

Source AWS CodeCommit
Succeeded - 8 hours ago 6205e182
View details

Image Amazon ECR
Succeeded - 8 hours ago sha256:6005a8ab686c
View details

6205e182 Source: pushing 4 files created in deployment directory
Image: sha256:6005a8ab686c

Disable transition

Deploy Succeeded

Start rollback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Developer Tools **CodePipeline**

Source • CodeCommit
Artifacts • CodeArtifact
Build • CodeBuild
Deploy • CodeDeploy
Pipeline • CodePipeline
Getting started
Pipelines **Pipeline**
History
Settings
Settings

Source • Go to resource
Feedback

CloudShell Feedback

Source Succeeded
Pipeline execution ID: 2f07f7d1-009c-4bac-b391-c273302d4055

Source AWS CodeCommit
Succeeded - 8 hours ago 6205e182
View details

Image Amazon ECR
Succeeded - 8 hours ago sha256:6005a8ab686c
View details

6205e182 Source: pushing 4 files created in deployment directory
Image: sha256:6005a8ab686c

Disable transition

Deploy Succeeded

Start rollback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- 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
```

```

export -> ip-10-16-10-131.x
...-> cbafaf52af9d
Successfully built cbafaf52af9d
Successfully tagged employee:latest
veclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '' -f2)
veclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.cjodag7hvcvt.us-east-1.rds.amazonaws.com
veclabs:~/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity | grep Account|cut -d '' -f4)
veclabs:~/environment/microservices/employee (dev) $ echo $account_id
08245198674
veclabs:~/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
veclabs:~/environment/microservices/employee (dev) $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
veclabs:~/environment/microservices/employee (dev) $

```

#push the image

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

- o 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

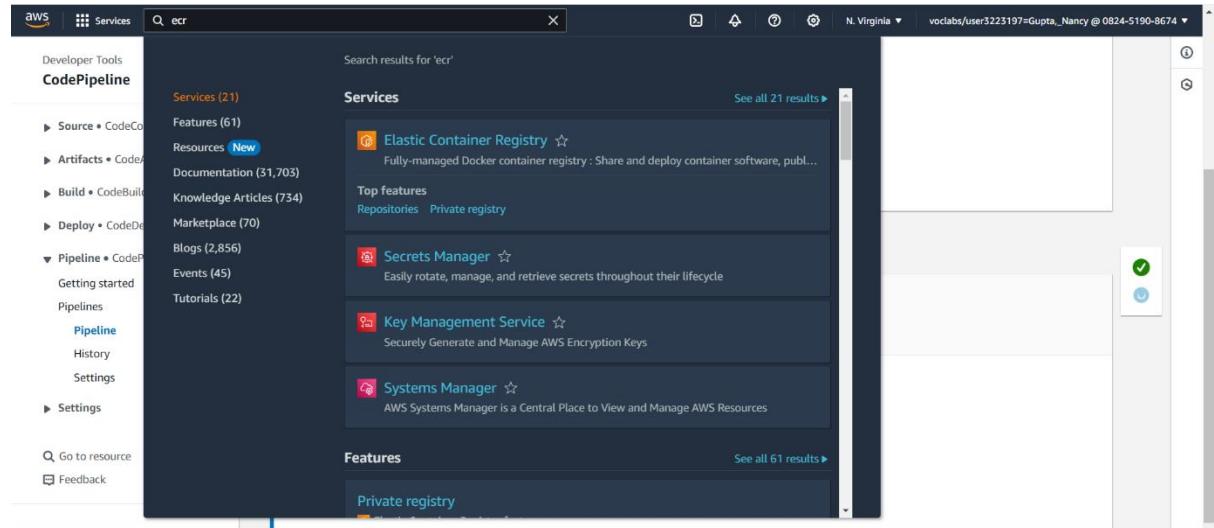
f1b5933fe4b5: Layer already exists

```

bash -> ip-10-16-10-131.x
veclabs:~/environment/microservices/employee (dev) $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
veclabs:~/environment/microservices/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [08245198674.dkr.ecr.us-east-1.amazonaws.com/employee]
2b7d1c0ad780: Pushed
2f26eb6b54c9: Pushed
b5ed8866d230: Layer already exists
d81a715330b7: Layer already exists
1dc7f3bb09a4: Layer already exists
dcaceb729824: Layer already exists
f1b5933fe4b5: Layer already exists
latest: digest: sha256:4a261119fc7171f77b71c6233c012e68627d559b8c3b08c8df7e8d6a8fc1b07 size: 1783
veclabs:~/environment/microservices/employee (dev) $

```

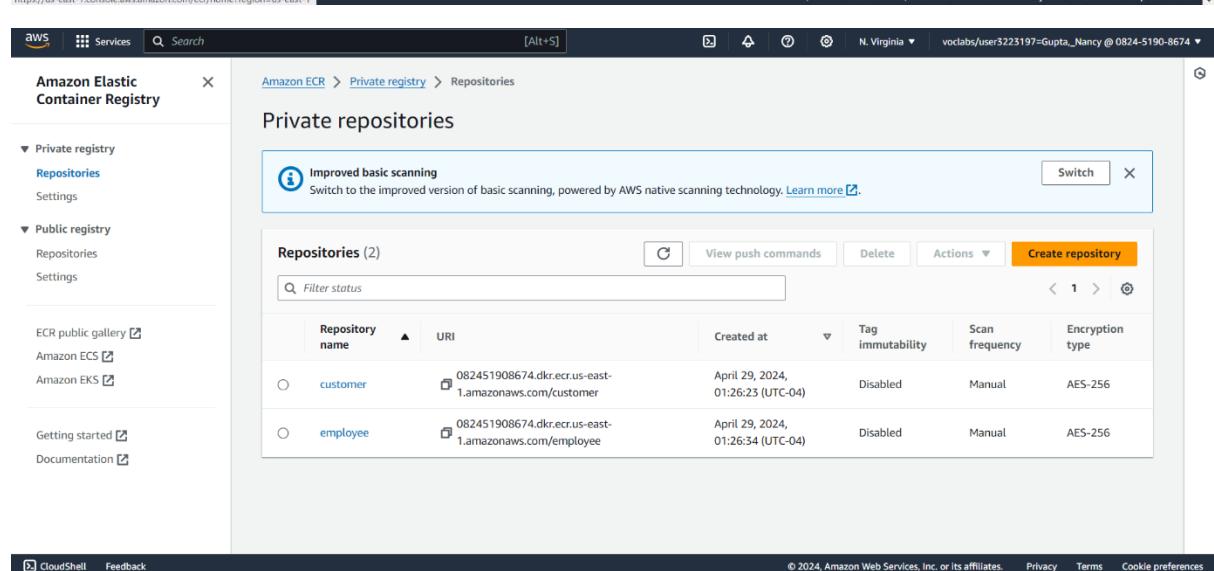
At least one layer should indicate that it was pushed, which indicates that the image was modified since it was last pushed to Amazon ECR. You could also look in the Amazon ECR repository to confirm the last modified timestamp of the image that is tagged as the latest.



The screenshot shows the AWS CloudSearch interface with a search bar containing 'ecr'. The results are categorized under 'Services' and 'Features'.

- Services (21)**
 - Elastic Container Registry: Fully-managed Docker container registry : Share and deploy container software, publ...
 - Secrets Manager: Easily rotate, manage, and retrieve secrets throughout their lifecycle
 - Key Management Service: Securely Generate and Manage AWS Encryption Keys
 - Systems Manager: AWS Systems Manager is a Central Place to View and Manage AWS Resources
- Features**
 - Private registry

Below the search results, there is a note: "See all 21 results >" and "See all 61 results >".



The screenshot shows the AWS ECR interface with a sidebar for 'Amazon Elastic Container Registry' and 'Private registry'.

- Private registry**
 - Repositories**
 - Settings
- Public registry**
 - Repositories
 - Settings

The main area displays 'Private repositories' with a heading 'Improved basic scanning'. It shows two repositories:

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
customer	082451908674.dkr.ecr.us-east-1.amazonaws.com/customer	April 29, 2024, 01:26:23 (UTC-04)	Disabled	Manual	AES-256
employee	082451908674.dkr.ecr.us-east-1.amazonaws.com/employee	April 29, 2024, 01:26:34 (UTC-04)	Disabled	Manual	AES-256

Buttons for 'View push commands', 'Delete', and 'Actions' are visible above the table. A 'Create repository' button is located at the top right of the table area.

Amazon ECR > Private registry > Repositories > employee

Images (2)

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
latest	Image	April 29, 2024, 12:43:05 (UTC-04)	27.70	Copy URI	sha256:a261117fc7171...
-	Image	April 29, 2024, 01:37:23 (UTC-04)	27.70	Copy URI	sha256:6005a8ab686c76...

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.

Developer Tools > CodePipeline > Pipelines > update-employee-microservice

Pipeline type: V2 Execution mode: QUEUED

Source Succeeded Pipeline execution ID: [455a002b-1887-4d02-ab6e-a7de133bd829](#)

Source AWS CodeCommit Succeeded - 4 minutes ago [6205e182](#)

Image Amazon ECR Succeeded - 4 minutes ago sha256:a261117fc71

6205e182 Source: pushing 4 files created in deployment directory
Image: sha256:a261117fc71

Disable transition

The screenshot shows the AWS CodePipeline console. On the left, the navigation pane includes sections for Source, Artifacts, Build, Deploy, and Pipeline. The Pipeline section is expanded, showing 'Getting started', 'Pipelines', and 'History'. Under 'History', the 'Pipeline' section is selected. The main area displays a pipeline execution with three stages:

- Source**: AWS CodeCommit, Status: Succeeded - 4 minutes ago, Task ID: 6205e182. Description: Source: pushing 4 files created in deployment directory. Image: sha256:4a261117fc71.
- Build**: Amazon ECR, Status: Succeeded - 4 minutes ago, Task ID: 6205e182. Description: sha256:4a261117fc71.
- Deploy**: In progress, Pipeline execution ID: 455a002b-1887-4d02-ab6e-a7de133bd829. Sub-step: Deploy (Amazon ECS (Blue/Green)). Status: In progress - 4 minutes ago.

A 'Disable transition' button is located between the Build and Deploy stages. The bottom right corner of the screen shows standard AWS footer links: © 2024, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

2. Observe the details in the CodeDeploy console.

The screenshot shows the AWS CodeDeploy console. The navigation pane includes sections for Source, Artifacts, Build, Deploy, and Pipeline. The Deploy section is expanded, showing 'Getting started', 'Pipelines', and 'History'. Under 'History', the 'Deployment' section is selected. A modal window titled 'Action execution details' is open, showing the following information:

- Action name:** Deploy
- Status:** In progress
- Action execution ID:** a3c12356-1bba-47e3-9758-1a8546622390
- Message:** Deployment d-1N4TZIHU5 created
- Execution details:** View in CodeDeployToECS

A 'Done' button is at the bottom right of the modal. The bottom right corner of the screen shows standard AWS footer links: © 2024, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

The screenshot shows the AWS CodeDeploy console. The navigation pane includes sections for Source, Artifacts, Build, Deploy, and Pipeline. The Deploy section is expanded, showing 'Getting started', 'Deployments', and 'Deployment'. Under 'Deployment', the 'd-1N4TZIHU5' deployment is selected. The main area displays the deployment status and traffic shifting progress:

Deployment status:

- Step 1: Deploying replacement task set (Completed: Succeeded)
- Step 2: Test traffic route setup (Completed: Succeeded)
- Step 3: Rerouting production traffic to replacement task set (Completed: Succeeded)
- Step 4: Wait (Completed: Succeeded)
- Step 5: Terminate original task set (Completed: Succeeded)

Traffic shifting progress:

- Original: 0%
- Replacement: 100%

The bottom right corner of the screen shows standard AWS footer links: © 2024, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta,_Nancy @ 0824-5190-8674

Developer Tools **CodeDeploy**

- Source • CodeCommit
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Getting started
- Deployments
- Deployment**
- Applications
- Deployment configurations
- On-premises instances
- Pipeline • CodePipeline
- Settings
- Go to resource
- Feedback

Deployment details

Application	Deployment ID	Status
microservices	d-1N4TZIHU5	Succeeded
Deployment configuration	Deployment group	Initiated by
CodeDeployDefault.ECSAllAtOnce	microservices-employee	User action
Deployment description		
-		

Revision details

Revision location	Revision created	Revision description
620f85de0dba4028841c2755765b6219ee4a890e1f5ba6368780092a284be80	9 minutes ago	Application revision registered by Deployment ID: d-1N4TZIHU5

Task set activity

Task set ID	Environment	Task set status	Traffic	Desired count	Running count	Pending count
ecs-svc/0996791440688041300	Original	ACTIVE	0	1	1	0

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta,_Nancy @ 0824-5190-8674

Developer Tools **CodeDeploy**

- Source • CodeCommit
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Getting started
- Deployments
- Deployment**
- Applications
- Deployment configurations
- On-premises instances
- Pipeline • CodePipeline
- Settings
- Go to resource
- Feedback

Task set activity

Task set ID	Environment	Task set status	Traffic	Desired count	Running count	Pending count
ecs-svc/0996791440688041300	Original	ACTIVE	0	1	1	0
ecs-svc/8826826401379166837	Replacement	PRIMARY	100%	1	1	0

Deployment lifecycle events

Event	Duration	Status	Start time	End time
BeforeInstall	less than one second	Succeeded	Apr 29, 2024 12:43 PM (UTC-4:00)	Apr 29, 2024 12:43 PM (UTC-4:00)
Install	2 minutes 2 seconds	Succeeded	Apr 29, 2024 12:43 PM (UTC-4:00)	Apr 29, 2024 12:45 PM (UTC-4:00)
AfterInstall	less than one second	Succeeded	Apr 29, 2024 12:45 PM (UTC-4:00)	Apr 29, 2024 12:45 PM (UTC-4:00)
AllowTestTraffic	less than one second	Succeeded	Apr 29, 2024 12:45 PM (UTC-4:00)	Apr 29, 2024 12:45 PM (UTC-4:00)
AfterAllowTestTraffic	less than one second	Succeeded	Apr 29, 2024 12:45 PM (UTC-4:00)	Apr 29, 2024 12:45 PM (UTC-4:00)
BeforeAllowTraffic	less than one second	Succeeded	Apr 29, 2024 12:45 PM (UTC-4:00)	Apr 29, 2024 12:45 PM (UTC-4:00)
AllowTraffic	less than one second	Succeeded	Apr 29, 2024 12:45 PM (UTC-4:00)	Apr 29, 2024 12:45 PM (UTC-4:00)
AfterAllowTraffic	less than one second	Succeeded	Apr 29, 2024 12:45 PM (UTC-4:00)	Apr 29, 2024 12:45 PM (UTC-4:00)

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta,_Nancy @ 0824-5190-8674

Developer Tools **CodePipeline**

- Source • CodeCommit
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
- Getting started
- Pipelines
- Pipeline**
- History
- Settings
- Settings
- Go to resource
- Feedback

Source

AWS CodeCommit	Image
6205e182	Amazon ECR
Succeeded - 9 minutes ago	
sha256:4a261117fc71	
View details	View details

6205e182 Source: pushing 4 files created in deployment directory
Image: sha256:4a261117fc71

Disable transition

Deploy Succeeded Pipeline execution ID: 455a002b-1887-4d02-ab6e-a7de133bd829

Deploy	Amazon ECS (Blue/Green)
6205e182	Succeeded - 2 minutes ago
View details	

6205e182 Source: pushing 4 files created in deployment directory
Image: sha256:4a261117fc71

Start rollback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

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.

microservicesLB

Listeners and rules (2) Info

Listener	Protocol	Port	Target Groups
HTTP (8080)	HTTP	8080	microservices
HTTPS (443)	HTTPS	443	microservices

microservicesLB

Listeners and rules (2) Info

Listener	Protocol	Port	Target Groups
HTTP (8080)	HTTP	8080	microservices
HTTPS (443)	HTTPS	443	microservices

Manage coffee suppliers - <http://microservicesLB-1849491881.us-east-1.elb.amazonaws.com/admin/suppliers>

The screenshot shows a web browser window with a light gray header bar containing various links like 'Careers', 'GitHub - StevensDe...', 'Eligible CIP Codes f...', 'Applying for OPT S...', 'STEM OPT Extension...', 'CPT Work Authoriz...', 'CPT Frequently Ask...', 'Computer Architect...', 'Cisco Skills For All', 'VPN', and 'Update'. The main content area has a white background. At the top left is a small image of two cups of coffee on a espresso machine. To the right of the image, the title 'Manage coffee suppliers' is displayed in a large, dark font. Below the title is a navigation menu with three items: 'Administrator home', 'Suppliers list', and 'Customer home'. A section titled 'All suppliers' follows, featuring a table with one row of data:

Name	Address	City	State	Email	Phone
Nancy_Supplier2	5 castle point terrace	Hoboken	New Jersey	ngupta20@stevens.edu	987456321

A green 'edit' button is located at the end of the table row. At the bottom of this section is a green 'Add a new supplier' button.

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:

The screenshot shows a web browser window with a dark gray header bar containing the same set of links as the previous screenshot. The main content area has a dark gray background. The title 'Manage coffee suppliers' is centered in a large, white font. Below the title is a navigation menu with three items: 'Administrator home', 'Suppliers list', and 'Customer home'. A section titled 'All suppliers' follows, featuring a table with one row of data:

Name	Address	City	State	Email	Phone
Nancy_Supplier2	5 castle point terrace	Hoboken	New Jersey	ngupta20@stevens.edu	987456321

A green 'edit' button is located at the end of the table row. At the bottom of this section is a green 'Add a new supplier' button.

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

The screenshot shows a web browser window with a light gray header bar containing the same set of links as the previous screenshots. The main content area has a white background. The title 'Manage coffee suppliers' is centered in a large, dark font. Below the title is a navigation menu with three items: 'Administrator home', 'Suppliers list', and 'Customer home'. A section titled 'All suppliers' follows, featuring a table with one row of data:

Name	Address	City	State	Email	Phone
Nancy_Supplier2	5 castle point terrace	Hoboken	New Jersey	ngupta20@stevens.edu	987456321

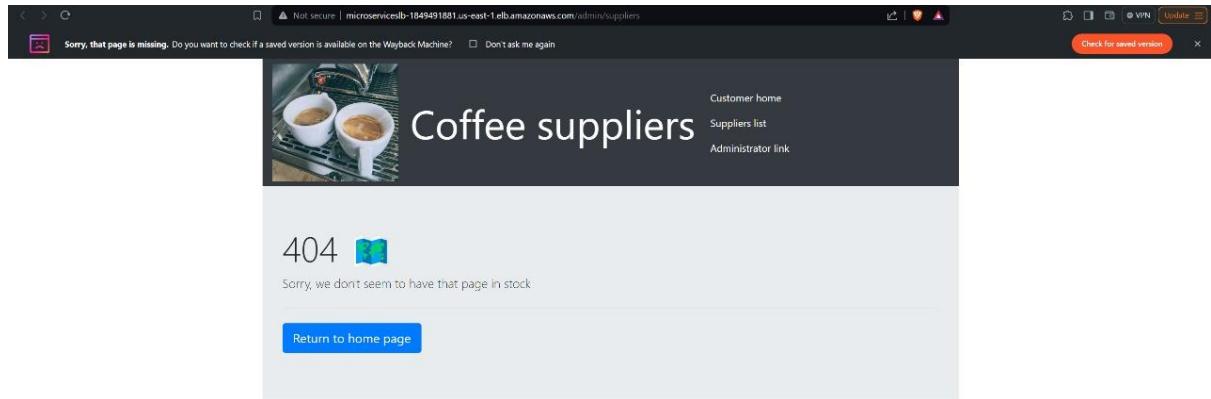
A green 'edit' button is located at the end of the table row. At the bottom of this section is a green 'Add a new supplier' button.

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 <p class="lead">Sorry, we don't seem to have that page in stock</p>.

A screenshot of the AWS EC2 Load Balancers console. The left sidebar shows navigation options like EC2 Dashboard, Services, and a search bar. The main panel shows a load balancer named 'microservicesLB'. The 'Details' tab is selected, displaying information such as the Load balancer type (Application), Status (Active), Scheme (Internet-facing), Hosted zone (Z35SXDOTRQ7X7K), Availability Zones (us-east-1a, us-east-1b), and IP address type (IPv4). It also shows the Load balancer ARN and DNS name. Below the details, there are tabs for 'Listeners and rules', 'Network mapping', 'Resource map - new', 'Security', 'Monitoring', 'Integrations', 'Attributes', and 'Tags'. At the bottom, there are buttons for 'Manage rules', 'Manage listener', and 'Add listener'. The footer includes copyright information for Amazon Web Services and links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

The screenshot shows the AWS Lambda function configuration for the 'customer' microservice. It includes sections for Basic information, Handler, Runtime, and Configuration. The Handler is set to 'index.handler' and the Runtime to 'Node.js 16.x'. The Configuration section shows the 'customer' layer selected. The Lambda function is triggered by an API Gateway event.

The screenshot shows the AWS CloudShell interface with the command 'curl http://microservicesLB-1849491881.us-east-1.elb.amazonaws.com/admin/suppliers' entered. The output shows a 404 error page for the 'supplier' endpoint.

```
vocabs:~/environment/microservices/employee (dev) $ curl http://microservicesLB-1849491881.us-east-1.elb.amazonaws.com/admin/suppliers
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="/css/bootstrap.min.css">
    <link rel="stylesheet" href="/css/base.css">
    <title>Coffee suppliers</title>
</head>
<body>

<div class="container">
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="/">Customer home</a>
                <li class="nav-item">
                    <a class="nav-link" href="/suppliers">Suppliers list</a>
                <li class="nav-item">
                    <a class="nav-link" href="/admin/suppliers">Administrator link</a>
                </li>
            </ul>
        </div>
    </nav>
    <div class="jumbotron">
        <h1>Sorry, we don't seem to have that page in stock</h1>
        <p>My 4</p>
        <p>Lead</p>
        <a class="btn btn-primary btn-lg" href="/" role="button">Return to home page</a>
    </div>
</body>
</html>
```

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.

- 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.

The screenshot shows three vertically stacked terminal windows, each displaying a portion of a large JSON-formatted response. The JSON is related to AWS ECS and Lambda services, specifically regarding task definitions and service configurations. The first window shows the beginning of the JSON, including service definitions and task definitions. The second window continues the JSON, focusing on task sets and network configurations. The third window concludes the JSON, mentioning role ARNs and events. The terminal interface includes a file explorer on the left and developer tools on the right.

```
bash -ip-10-16-10-131.e x [+] 
script src="/js/bootstrap.min.js">
</body>
vocabs:/environment/microservices/employee (dev) $ aws ecs update-service --cluster microservices-serverlesscluster --service customer-microservice --desired-count 3
{
  "service": {
    "serviceName": "arn:aws:ecs:us-east-1:082451988674:service/microservices-serverlesscluster/customer-microservice",
    "taskDefinition": "arn:aws:ecs:us-east-1:082451988674:task-definition/customer-microservice:8",
    "clusterName": "arn:aws:ecs:us-east-1:082451988674:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      {
        "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:082451988674:targetgroup/customer-tg-one/c61264f216271e58",
        "containerName": "customer",
        "containerPort": 8080
      }
    ],
    "serviceRegistries": [],
    "status": "ACTIVE",
    "desiredCount": 3,
    "runningCount": 1,
    "pendingCount": 0,
    "lastUpdateStatus": "UPDATING",
    "platformVersion": "1.4.0",
    "platformFamily": "LINUX",
    "taskDefinition": "arn:aws:ecs:us-east-1:082451988674:task-definition/customer-microservice:8",
    "deploymentConfiguration": {
      "maximumPercent": 200,
      "minimumHealthyPercent": 100
    },
  },
}

{
  "taskSets": [
    {
      "id": "ecs-svc/1251358895092516784",
      "taskSetArn": "arn:aws:ecs:us-east-1:082451988674:task-set/microservices-serverlesscluster/customer-microservice/ecs-svc/1251358895092516784",
      "clusterArn": "arn:aws:ecs:us-east-1:082451988674:service/microservices-serverlesscluster/customer-microservice",
      "externalId": "CodeDeployTaskSet-4-ZYQF3IBUS",
      "status": "PRIMARY",
      "taskDefinition": "arn:aws:ecs:us-east-1:082451988674:task-definition/customer-microservice:8",
      "computedDesiredCount": 1,
      "pendingCount": 0,
      "runningCount": 1,
      "lastUpdatedAt": "2024-04-29T07:42:31.358000+00:00",
      "updatedAt": "2024-04-29T07:44:38.013000+00:00",
      "lastSyncType": "FORCED",
      "platformVersion": "1.4.0",
      "platformFamily": "LINUX",
      "networkConfiguration": {
        "awsvpcConfiguration": {
          "subnets": [
            "subnet-089933df8dacc089",
            "subnet-0c1ad2d1f3070aa7"
          ],
          "securityGroups": [
            "sg-0d4714a7155247e26"
          ],
          "assignPublicIp": "ENABLED"
        }
      },
      "loadBalancers": [
        {
          "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:082451988674:targetgroup/customer-tg-one/c61264f216271e58",
          "containerName": "customer",
          "containerPort": 8080
        }
      ],
      "serviceRegistries": []
    }
  ],
  "scale": {
    "value": 100.0,
    "unit": "PERCENT"
  },
  "stabilityStatus": "STADY_STATE",
  "stabilityStatusAt": "2024-04-29T07:43:27.044000+00:00",
  "tags": []
}
]

{
  "roleArn": "arn:aws:iam::082451988674:role/aws-service-role/ecs.amazonaws.com/AMSServiceRoleForECS",
  "events": [
    {
      "id": "957d21c2-8cd6-4794-a5c9-8faa/d262908",
      "createdAt": "2024-04-29T13:51:11.089000+00:00",
      "message": "(service customer-microservice) has reached a steady state."
    },
    {
      "id": "6f82b36c-4f92-4ffe-b02e-bacc89303646",
      "createdAt": "2024-04-29T07:58:49.296000+00:00",
      "message": "(service customer-microservice) has reached a steady state."
    },
    {
      "id": "3d8ff641-a0a9-4e39-b9ca-5d7e1b475754",
      "createdAt": "2024-04-29T07:58:49.296000+00:00",
      "message": "(service customer-microservice) has reached a steady state."
    }
  ]
}
```

```

bash - "ip-10-16-10-131.e.x"
{
  "id": "3d8ff641-a0a9-4e39-b9ca-5d7e1b475754",
  "createdAt": "2024-04-29T07:49:20.660000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) updated state to STEADY_STATE."
},
{
  "id": "1a0742bb-683b-4898-9e45-1ec6d17351a",
  "createdAt": "2024-04-29T07:49:40.975000+00:00",
  "message": "(service customer-microservice) stopped 1 pending tasks."
},
{
  "id": "b6df8099-1a1d-445c-aa18-cd5407f78ac1",
  "createdAt": "2024-04-29T07:49:40.946000+00:00",
  "message": "(service customer-microservice) updated computedDesiredCount for taskSet ecs-svc/3838291125602239221 to 0."
},
{
  "id": "09d6db90-6915-45b6-8298-f14a8889c420",
  "createdAt": "2024-04-29T07:49:25.281000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) has started 1 tasks: (task c28379d81c8d450798ffa22592dd27c)."
},
{
  "id": "cd167d92-3-c94-49c-9a0-92c03a57e6f",
  "createdAt": "2024-04-29T07:48:14.764000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) has started 1 tasks: (task c390fc6b6384358923f6bed8b67d8b)."
},
{
  "id": "a38ab2d5-779a-47ac-abd2-c53ebdf7689c",
  "createdAt": "2024-04-29T07:47:13.783000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) has started 1 tasks: (task 14fd7763a73a23a87e7f3825d539d1)."
},
{
}

```

```

bash - "ip-10-16-10-131.e.x"
{
  "id": "4a3b5718-e221-47d2-ac48-68cdmaaf1c5c",
  "createdAt": "2024-04-29T07:46:14.764000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) has started 1 tasks: (task 706953cecdb41828ef616b4767b17aa)."
},
{
  "id": "e6aaacb7-e6d3-436c-94f3-1323e64f3e5b",
  "createdAt": "2024-04-29T07:45:12.589000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) has started 1 tasks: (task f4a3170d9d5a4d88b210ccfc6f589a14)."
},
{
  "id": "859cd818-dbbf-4559-8519-2061a4eb3543",
  "createdAt": "2024-04-29T07:44:10.588000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) has started 1 tasks: (task c5d8004e41a004902a98ac204e763adf)."
},
{
  "id": "450ba0b5-2-fee-4d04-a366-715c1b1dfce",
  "createdAt": "2024-04-29T07:43:27.050000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/1251358895092516784) updated state to STEADY_STATE."
},
{
  "id": "b89113e9-b062-4b46-99cf-c4fd9f1763cf",
  "createdAt": "2024-04-29T07:43:07.213000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) has started 1 tasks: (task ef24e0cd17884f979d3d27b5df22f1ed)."
},
{
  "id": "b61ed362-d7fb-4a46-8d90-7400092acead",
  "createdAt": "2024-04-29T07:43:07.213000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/1251358895092516784) registered 1 targets in (target-group arn:aws:elasticloadbalancing:us-east-1:082451908674:targetgroup/customer-tg-one/c61264f21627fe58)"
},
{
}

```

```

bash - "ip-10-16-10-131.e.x"
{
  "id": "9b9584a9-4d3a-454c-0d3f-78943c3ab02f",
  "createdAt": "2024-04-29T07:14:11.502000+00:00",
  "message": "(service customer-microservice, taskSet ecs-svc/3838291125602239221) has started 1 tasks: (task 2d6883a13f4480bbded612df1a8772)."
},
{
  "createdAt": "2024-04-29T07:14:11.502000+00:00",
  "placementConstraints": [],
  "placementStrategy": [],
  "networkConfiguration": {
    "awsvpcConfiguration": {
      "subnets": [
        "subnet-089933d18ddacc009",
        "subnet-0c1ad2d13f0708a7"
      ],
      "securityGroups": [
        "sg-dfa71a7f55247e26"
      ],
      "assignPublicIp": "ENABLED"
    }
  },
  "healthCheckGracePeriodSeconds": 0,
  "schedulingStrategy": "REPLICAS",
  "deploymentController": {
    "type": "CODE_DEPLOY"
  },
  "createdBy": "arn:aws:lambda:082451908674:role/voclabs",
  "enableECSManagedTags": false,
  "propagateTags": "NONE",
  "enableExecuteCommand": false
}

```

- Go to the [Amazon ECS services view](#).

The screenshot shows the AWS Services Catalog interface. The search bar at the top contains the text 'ECS'. The left sidebar lists various services and features under categories like EC2 Dashboard, Instances, Images, and Elastic Block Store. The main pane displays search results for 'ECS' under the 'Services' category, listing 'Elastic Container Service', 'Batch', 'AWS FIS', and 'EC2'. A detailed view of the 'Elastic Container Service' card is shown on the right, including its description, star rating, and creation date (April 29, 2024, 02:58 UTC-04:00). The bottom of the screen shows navigation links for CloudShell and Feedback.

The screenshot shows the 'Clusters' page within the Amazon Elastic Container Service (ECS) console. The left sidebar includes links for Clusters, Namespaces, Task definitions, Account settings, and other services like ECR and Batch. The main content area shows a table titled 'Clusters (1) Info' with one entry: 'microservices-serverlesscluster' (Status: Active, Services: 2, Tasks: 0 Pending | 2 Running, Container instances: 0 EC2, CloudWatch monitoring: Default). A 'Create cluster' button is visible in the top right. The bottom of the screen shows navigation links for CloudShell and Feedback.

The screenshot shows the details for the 'microservices-serverlesscluster' within the ECS console. The left sidebar is identical to the previous screenshot. The main content area is titled 'Cluster overview' and shows the ARN (arn:aws:ecs:us-east-1:082451908674:cluster/microservices-serverlesscluster), Status (Active), CloudWatch monitoring (Default), and Registered container instances (None). Below this is a 'Services' section with two entries: 'Draining' (Status: Active, Tasks: 2 Pending) and 'Running' (Status: Pending, Tasks: 2 Running). Navigation tabs include Services, Tasks, Infrastructure, Metrics, Scheduled tasks, and Tags. The bottom of the screen shows navigation links for CloudShell and Feedback.

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.

The screenshot shows the AWS Elastic Container Service Cluster overview. In the Services section, the *customer-microservice* is listed with a status of Active and 1/3 Tasks running. The *employee-microservice* is also listed with a status of Active and 2/1 Tasks running. The navigation bar at the bottom includes Services, Tasks, Infrastructure, Metrics, Scheduled tasks, and Tags.

The screenshot shows the AWS Elastic Container Service Cluster overview after some time has passed. Both the *customer-microservice* and *employee-microservice* now have 1/1 Tasks running, indicating they are fully scaled up. The navigation bar at the bottom includes Services, Tasks, Infrastructure, Metrics, Scheduled tasks, and Tags.

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.