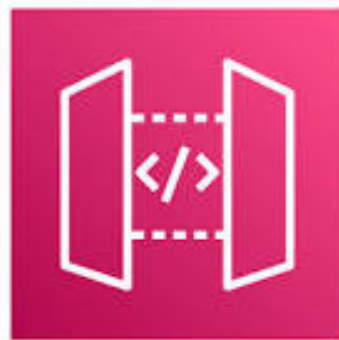




AWS Project Documentation

Serverless Chat Application Using WebSocket API





WHAT IS CLOUD FORMATION :

- AWS CloudFormation is an Infrastructure as Code (IaC) service by AWS.
- It allows you to define and provision cloud resources using YAML or JSON templates.
- With CloudFormation, infrastructure can be deployed consistently and repeatedly.
- It automates resource creation, updates, and rollback in case of errors.
- This simplifies cloud management and ensures reliable deployments.

Why is it used?

Here's why CloudFormation is powerful and widely used:

1. Automation

- Set up everything (like EC2, S3, RDS, IAM, etc.) with just one template.
- No need to click around in the AWS Console.

2. Consistency

- Ensures the same setup every time — great for dev, test, and production environments.

3. Version Control

- Since it's code (in YAML/JSON), you can store it in Git and track changes over time.

4. Easy Updates

- You can update resources safely using change sets, without breaking things.

5. Rollback on Failure

- If something goes wrong during setup, CloudFormation can roll back changes automatically.

Here are some common use cases of AWS CloudFormation:

1. Automated Infrastructure Deployment

- Quickly launch servers, databases, networks, and other AWS resources using templates.

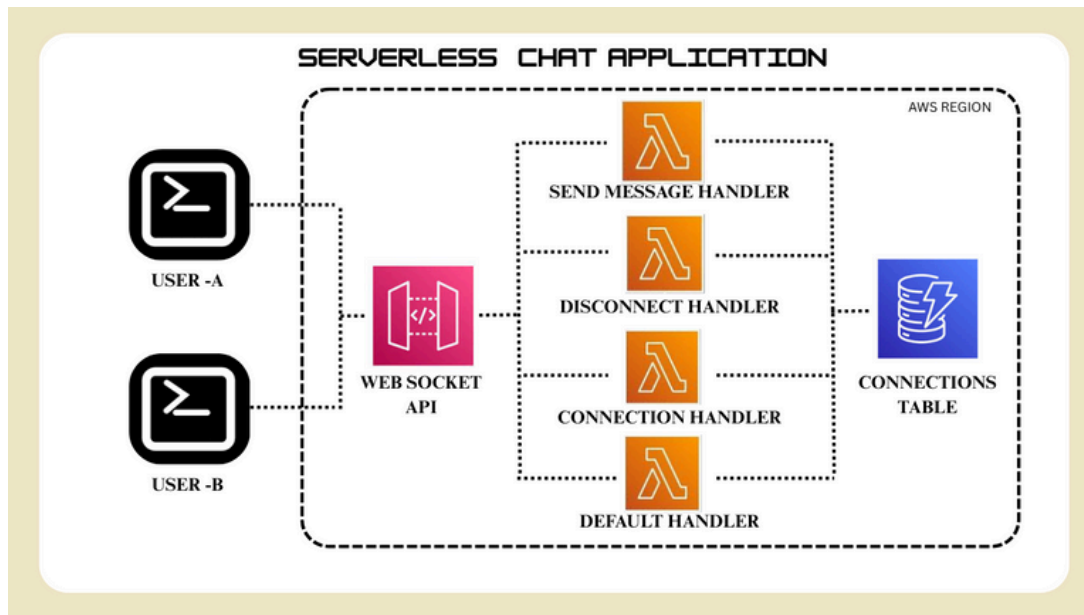
2. Multi-Environment Setup

- Create consistent dev, test, and production environments from the same code.

3. Disaster Recovery Setup

- Rebuild your infrastructure in a different region quickly using the same templates.

ARCHITECTURE DIAGRAM:



Architecture Explanation :

1.Users Connect via WebSocket API:

User-A and User-B connect through an API Gateway configured for WebSocket communication.

2.Connection Handling with Lambda:

When users connect, a Connection Handler (Lambda function) stores their connection info in the Connections Table (likely DynamoDB).

3.Message Sending:

When a user sends a message, the Send Message Handler Lambda retrieves recipient connection info from the table and delivers the message.

4.Disconnection Handling:

If a user disconnects, the Disconnect Handler removes their entry from the Connections Table.

5.Default Handler for Other Events:

The Default Handler processes any unexpected or custom WebSocket messages.

Notes

Key Components:

1. DynamoDBTable:

(ConnectionsTable8000B8A1)

- Stores active WebSocket connections.
- Primary key: connectionId (String).
- Uses ProvisionedThroughput (5 reads/writes per second).
- DeletionPolicy: Delete (removes table when stack is deleted).

2. IAM Roles & Policies

AWS Lambda needs permissions to interact with DynamoDB and API Gateway.

- ConnectHandlerServiceRole,
DisconnectHandlerServiceRole,
SendMessageHandlerServiceRole,
DefaultHandlerServiceRole
 - These IAM roles allow Lambda functions to assume execution permissions.
- IAM Policies
 - Grant permissions for DynamoDB operations (PutItem, DeleteItem, Scan, Query).
 - Allow WebSocket message handling (execute-api:ManageConnections).

Notes

3.Lambda Functions:

Each function handles different WebSocket events.

ConnectHandler (Handles client connections)

- Triggered when a WebSocket client connects.
- Stores connectionId in DynamoDB.
- Returns 200 on success.

DisconnectHandler (Handles client disconnections)

- Triggered when a WebSocket client disconnects.
- Deletes connectionId from DynamoDB.
- Returns 200 on success.

SendMessageHandler (Handles messaging)

- Fetches all active connections from DynamoDB.
- Sends messages to all clients except the sender.
- Uses API Gateway Management API (PostToConnectionCommand).
- If sending fails (e.g., client disconnected), logs the error.

DefaultHandler (Handles unknown requests)

- Replies with instructions when a client sends an invalid route.
- Retrieves client connection info and sends a response.

Notes

Process Flow

- Client connects via WebSocket:
 - API Gateway triggers ConnectHandler.
 - ConnectHandler saves connectionId in DynamoDB.
- Client sends a message:
 - API Gateway triggers SendMessageHandler.
 - SendMessageHandler retrieves all active connections.
 - Sends the message to all clients except the sender.
- Client disconnects:
 - API Gateway triggers DisconnectHandler.
 - DisconnectHandler removes connectionId from DynamoDB.
- Unknown WebSocket requests:
 - API Gateway triggers DefaultHandler.
 - DefaultHandler sends a default response.

Final Summary:

- WebSocket API with API Gateway.
- AWS Lambda handles connect, disconnect, and messaging.
- DynamoDB stores active connections.
- IAM roles/policies secure access.
- Scalable and real-time WebSocket communication.

Notes



Steps to Build the Serverless Chat App:

1. Create a WebSocket API in Amazon API Gateway

- Define routes: \$connect, \$disconnect, \$default, and sendMessage.

2. Create AWS Lambda Functions

- One for each route:
 - ConnectionHandler, DisconnectHandler, DefaultHandler, SendMessageHandler.

3. Set Up DynamoDB Table

- Create a ConnectionsTable to store connected client IDs (connectionId as primary key).

4. Integrate Lambda with WebSocket Routes

- Attach each Lambda function to its respective WebSocket route.

5. Deploy and Test

- Deploy the WebSocket API and use a WebSocket client (like Postman or a frontend) to connect and test real-time chat.

Notes

SAMPLE OUTPUT :

CloudFormation > Stacks > Create stack

Step 1: Create stack

Prerequisite - Prepare template

You can also create a template by scanning your existing resources in the [IaC generator](#).

Prepare template

Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

☒ Choose an existing template
Upload or choose an existing template.

☐ Build from Infrastructure Composer
Create a template using a visual builder.

Specify template

This [GitHub repository](#) contains sample CloudFormation templates that can help you get started on new infrastructure projects. [Learn more](#)

Template source

Selecting a template generates an Amazon S3 URL, where it will be stored. A template is a JSON or YAML file that describes your stack's resources and properties.

☐ Amazon S3 URL
Provide an Amazon S3 URL to your template.

☒ Upload a template file
Upload your template directly to the console.

☐ Sync from Git
Sync a template from your Git repository.

Upload a template file

[Choose file](#)

`_starter.yaml`

JSON or YAML formatted file

S3 URL: `https://s3.ap-south-1.amazonaws.com/cf-templates-1hky09wds5g-ap-south-1/2025-04-04T044914.986Zrc-_starter.yaml`

[View in Infrastructure Composer](#)

stack-for-serverless-chat

Stacks (3)

Filter by stack name

Filter status: Active View nested

Stacks

- stack-for-serverless-chat
2025-04-04 10:23:41 UTC+0530
CREATE_COMPLETE
- NESTED amplify-profilesapp-AnandhaNivas-sandbox-54-amplifyDataUserProfileNestedStack
UserProf-8589J8MOL55K
2024-11-24 21:46:30 UTC+0530
UPDATE_COMPLETE
- NESTED amplify-profilesapp-AnandhaNivas-sandbox-54-amplifyDataAmplifyTableManagerNestedStackA-12DAY1JPVOJSV
2024-11-24 21:45:02 UTC+0530
UPDATE_COMPLETE
- NESTED

Latest deployment timeline - new

This is a timeline view of your latest stack deployment. If you start a new deployment, this view will reset.

Search events

Resources

Timeline view showing deployment of resources over time.

Review and create

Step 1: API details

API name: serverless-chat-api

Route selection expression: \$request.body.action

IP address type: IPv4

Step 2: Routes

Route key: \$connect

\$disconnect

\$default

sendMessage

Step 3: Integrations

Route key	Integration type	Integration target
\$connect	Lambda	stack-for-serverless-chat-ConnectHandler2FFD52D8-0sx705pZly (ap-south-1)
\$disconnect	Lambda	stack-for-serverless-chat-DisconnectHandlerCB7ED6F-5kBB8nKD7PF (ap-south-1)
\$default	Lambda	stack-for-serverless-chat-DefaultHandler604DF7AC-dHLAHLmU9qL (ap-south-1)
sendMessage	Lambda	stack-for-serverless-chat-SendMessageHandlerDCEABF-xf5IA1YC16x6 (ap-south-1)

Terminal Output 1:

```
C:\WINDOWS\system32 x + -  
Microsoft Windows [Version 10.0.26100.3624]  
(c) Microsoft Corporation. All rights reserved.  
C:\Users\Anandha Nivas>aws --profile rxezvukjd0 execute-api ap-south-1.amazonaws.com/production  
Connected (press CTRL+C to quit)  
> {"action": "sendMessage", "message": "Hello AWS"}  
< Hello AWS  
> {"action": "sendMessage", "message": "Hello This is from ANANDHA NIVAS S"}  
< Hello This is from the client 0  
>
```

Terminal Output 2:

```
C:\WINDOWS\system32 x + -  
Microsoft Windows [Version 10.0.26100.3624]  
(c) Microsoft Corporation. All rights reserved.  
C:\Users\Anandha Nivas>aws --profile rxezvukjd0 execute-api ap-south-1.amazonaws.com/production  
Connected (press CTRL+C to quit)  
> {"action": "sendMessage", "message": "Hello AWS"}  
< Hello AWS  
> {"action": "sendMessage", "message": "Hello This is from ANANDHA NIVAS S"}  
< Hello This is from the client 0  
>
```

For More References :

[https://www.linkedin.com/posts/khushi-nandwani_serverless-chat-application-project-documentation-activity-731125825007765633-d9tn?](https://www.linkedin.com/posts/khushi-nandwani_serverless-chat-application-project-documentation-activity-731125825007765633-d9tn?utm_source=share&utm_medium=member_desktop&rcm=ACoAAD2n0psBzAPpvyJdcQUkYnmpxxr7gXInHKs)
[utm_source=share&utm_medium=member_desktop&rcm=ACoAAD2n0psBzAPpvyJdcQUkYnmpxxr7gXInHKs](https://www.linkedin.com/posts/khushi-nandwani_serverless-chat-application-project-documentation-activity-731125825007765633-d9tn?utm_source=share&utm_medium=member_desktop&rcm=ACoAAD2n0psBzAPpvyJdcQUkYnmpxxr7gXInHKs)