Notes

AWS Project Documentation

Serverless Chat Application Using WebSocket API

@sAnandha

Notes

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:

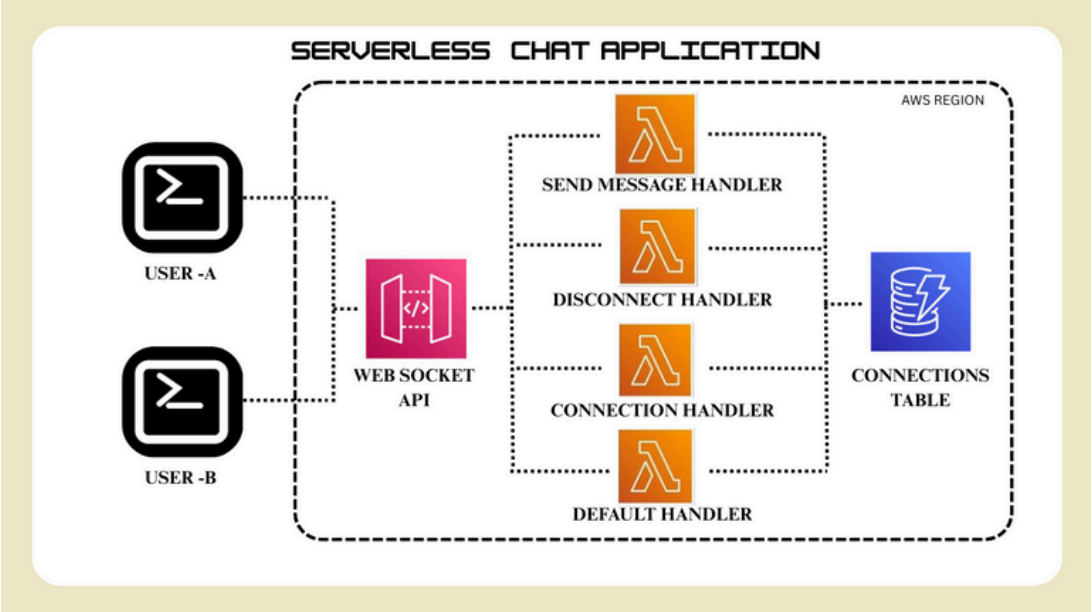
1. **Automation**
   - Set up everything (like EC2, S3, RDS, IAM, etc.) with just one template.
   - No need to click around in the AWS Console.
2. **Consistency**
   - Ensures the same setup every time — great for dev, test, and production environments.

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).

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.

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.

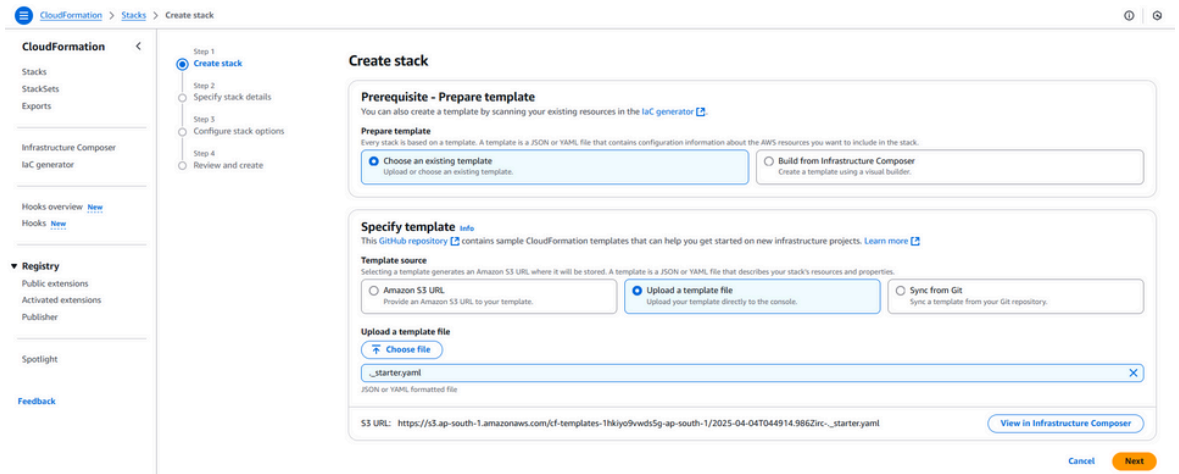
Notes

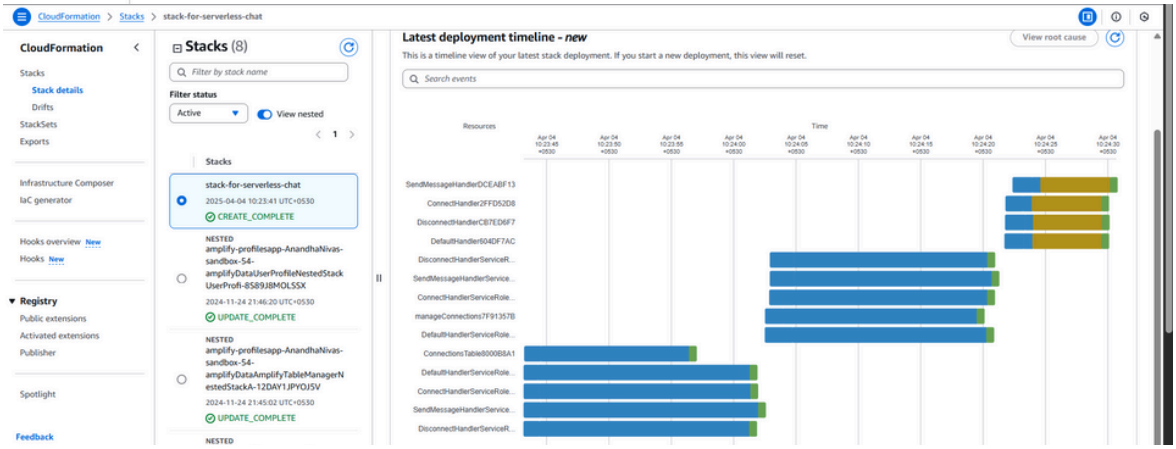
🚀 Steps to Build the Serverless Chat App:

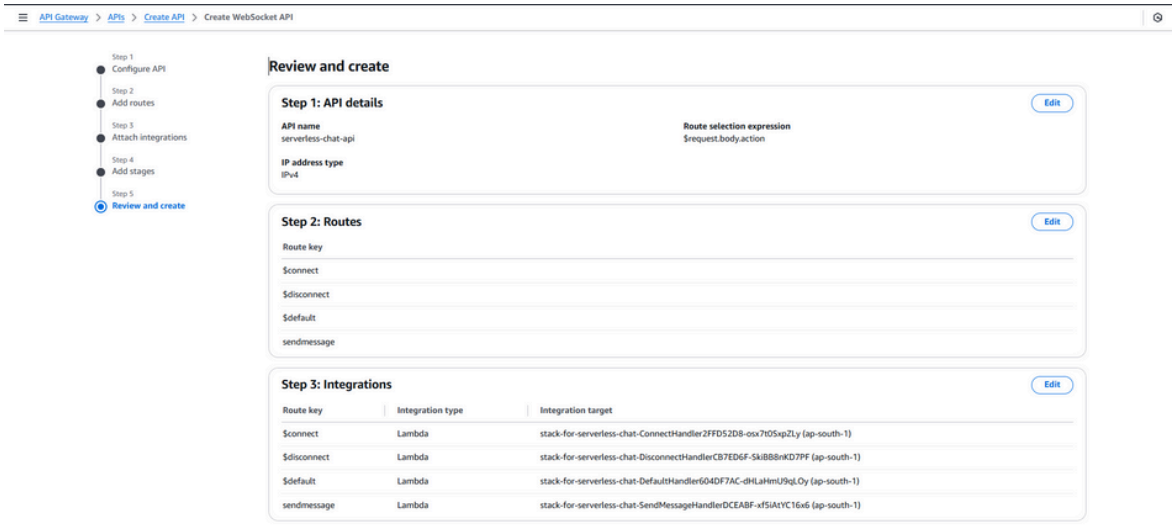
1. **Create a WebSocket API in Amazon API Gateway**
   - Define routes: $connect, $disconnect, $default, and sendMessage.
2. **Create AWS Lambda Functions**
   - One for each route:
     - ConnectionHandler, DisconnectHandler, DefaultHandler, SendMessageHandler.
3. **Set Up DynamoDB Table**
   - Create 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.

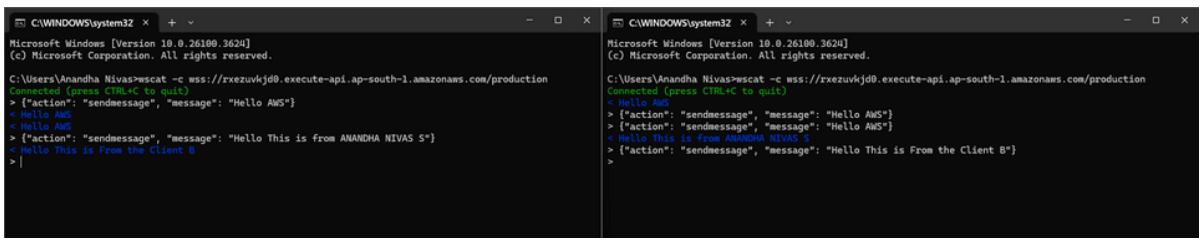
Notes

SAMPLE OUTPUT :









For More References :

https://www.linkedin.com/posts/khushi-nandwani_serverless-chat-application-project-documentation-activity-7311258250077765633-d9tn?utm_source=share&utm_medium=member_desktop&rcm=ACoAAD2n0psBzAPpvyJdcQUkYnmpxxr7gXInHKs