**Lab: Node.js Producer & Consumer for Redpanda Cloud**

**Goal:** Use the kafkajs library in a Node.js application to programmatically create a topic, produce messages to it, and consume them in real-time from a Redpanda Cloud cluster.

# **Purpose of the Lab**

This lab is designed for JavaScript/Node.js developers who want to integrate Redpanda directly into their applications. You will write and run a series of scripts to perform the full lifecycle of a data pipeline from the application layer. This lab provides a practical, code-first understanding of how to use a standard Kafka client library in the Node.js ecosystem to build powerful, cloud-native streaming applications on Redpanda.

# **Prerequisites**

* A Redpanda Cloud account with a running cluster.
* An rpk profile configured to connect to your cloud cluster (e.g., rpk-cloud) for easy verification.
* **Node.js and npm installed on your local machine.** Follow the instructions for your operating system below.

## **macOS (using Homebrew)**

|  |
| --- |
| # Install Homebrew if you don't have it: /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"  # Install Node.js brew install node |

## Windows (using the Official Installer)

1. Go to the official [Node.js download page](https://nodejs.org/en/download/).
2. Download the "LTS" (Long Term Support) version for Windows.
3. Run the downloaded .msi installer and follow the on-screen prompts.

## **Linux (using Node Version Manager - nvm)**

|  |
| --- |
| curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash  wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash  source ~/.bashrc  nvm –version  nvm install --lts |

After installation, verify it was successful by running node -v and npm -v in your terminal.

Project Layout

You will create a single directory for this lab.

|  |
| --- |
| rp-nodejs-cloud-lab/ ├── package.json ├── admin.js ├── producer.js ├── consumer.js └── .env |

# **Part 1: Setting up the Project and Environment**

## Step 1: Get Cloud Credentials and Create a User

1. **Navigate to Security:** In the [Redpanda Cloud UI](https://cloud.redpanda.com/), click on the **Security** tab.
2. **Create a User:** Go to the **Users** sub-tab. Click **Create user**. Give the user a name (e.g., nodejs-user) and save the generated **username** and **password**.
3. **Go to ACLs:** Click on the **ACLs** sub-tab.
4. **Grant Permissions:** Click **Create ACL**. Select the nodejs-user. For the lab, select the **Allow All** preset, then click **Create**.
5. **Get Broker Address:** Go to the **Clusters** page, select your cluster. On the **Overview** page, click the **Kafka API** tab and copy the **Brokers** address.

## **Step 2: Prepare the Project and Node.js Environment**

1. **Create the project directory:**

|  |
| --- |
| mkdir rp-nodejs-cloud-lab cd rp-nodejs-cloud-lab |

1. **Initialize a Node.js project:** This creates a package.json file.

|  |
| --- |
| npm init -y |

1. **Install the necessary libraries:** We need kafkajs to talk to Redpanda and dotenv to handle credentials securely.

|  |
| --- |
| npm install kafkajs dotenv |

1. **Create the Environment File (.env):** This file will securely hold your secrets.

|  |
| --- |
| # .env REDPANDA\_BROKERS="<YOUR\_BROKERS\_URL>" REDPANDA\_USER="<YOUR\_USERNAME>" REDPANDA\_PASS="<YOUR\_PASSWORD>" |

Populate the file with the credentials you saved in Step 1.

# 

# **Part 2: Building the Node.js Application**

## **Step 3: Create the Admin Script**

Create a file named admin.js. This script will connect to your cluster and create a new topic.

**admin.js**

// admin.js  
require('dotenv').config();  
const { Kafka } = require('kafkajs');  
  
const kafka = new Kafka({  
 clientId: 'my-admin-app',  
 brokers: [process.env.REDPANDA\_BROKERS],  
 ssl: true,  
 sasl: {  
 mechanism: 'scram-sha-256',  
 username: process.env.REDPANDA\_USER,  
 password: process.env.REDPANDA\_PASS,  
 },  
});  
  
const admin = kafka.admin();  
const topicName = 'user-signups-nodejs';  
  
const createTopic = async () => {  
 console.log(`Connecting to Redpanda and creating topic "${topicName}"...`);  
 await admin.connect();  
 try {  
 await admin.createTopics({  
 topics: [{  
 topic: topicName,  
 numPartitions: 1,  
 replicationFactor: 3,  
 }],  
 });  
 console.log(`Topic "${topicName}" created successfully.`);  
 } catch (error) {  
 if (error.type === 'TOPIC\_ALREADY\_EXISTS') {  
 console.log(`Topic "${topicName}" already exists.`);  
 } else {  
 console.error('Error creating topic:', error);  
 }  
 } finally {  
 await admin.disconnect();  
 }  
};  
  
createTopic();

## **Step 4: Create the Producer Script** Create a file named producer.js. This script will send a few sample messages to your new topic.

// producer.js  
require('dotenv').config();  
const { Kafka } = require('kafkajs');  
  
const kafka = new Kafka({  
 clientId: 'my-producer-app',  
 brokers: [process.env.REDPANDA\_BROKERS],  
 ssl: true,  
 sasl: {  
 mechanism: 'scram-sha-256',  
 username: process.env.REDPANDA\_USER,  
 password: process.env.REDPANDA\_PASS,  
 },  
});  
  
const producer = kafka.producer();  
const topicName = 'user-signups-nodejs';  
  
const produceMessages = async () => {  
 await producer.connect();  
 console.log('Producer connected. Sending messages...');  
  
 const messages = [  
 { id: 1, email: 'alice@example.com', country: 'USA' },  
 { id: 2, email: 'bob@example.com', country: 'CA' },  
 { id: 3, email: 'carol@example.com', country: 'UK' },  
 ];  
  
 for (const message of messages) {  
 await producer.send({  
 topic: topicName,  
 messages: [{ value: JSON.stringify(message) }],  
 });  
 console.log('Sent:', message);  
 }  
  
 await producer.disconnect();  
 console.log('Producer finished and disconnected.');  
};  
  
produceMessages().catch(console.error);

**Step 5: Create the Consumer Script**

Create a file named consumer.js. This script will connect, subscribe, and run forever, printing any messages it receives.

|  |
| --- |
| // consumer.js require('dotenv').config(); const { Kafka } = require('kafkajs');  const kafka = new Kafka({  clientId: 'my-consumer-app',  brokers: [process.env.REDPANDA\_BROKERS],  ssl: true,  sasl: {  mechanism: 'scram-sha-256',  username: process.env.REDPANDA\_USER,  password: process.env.REDPANDA\_PASS,  }, });  const consumer = kafka.consumer({ groupId: 'nodejs-consumer-group' }); const topicName = 'user-signups-nodejs';  const consumeMessages = async () => {  await consumer.connect();  await consumer.subscribe({ topic: topicName, fromBeginning: true });  console.log(`Consumer connected and subscribed to topic "${topicName}". Waiting for messages...`);   await consumer.run({  eachMessage: async ({ topic, partition, message }) => {  console.log({  action: 'Received message',  value: message.value.toString(),  partition,  offset: message.offset,  });  },  }); };  consumeMessages().catch(e => console.error('Consumer error:', e)); |

# **Part 3: Running the Pipeline**

## **Step 6: Run the Scripts in Order**

1. **Create the topic:**

|  |
| --- |
| node admin.js |

1. **Run the consumer in one terminal:** This will start, connect, and wait for messages.

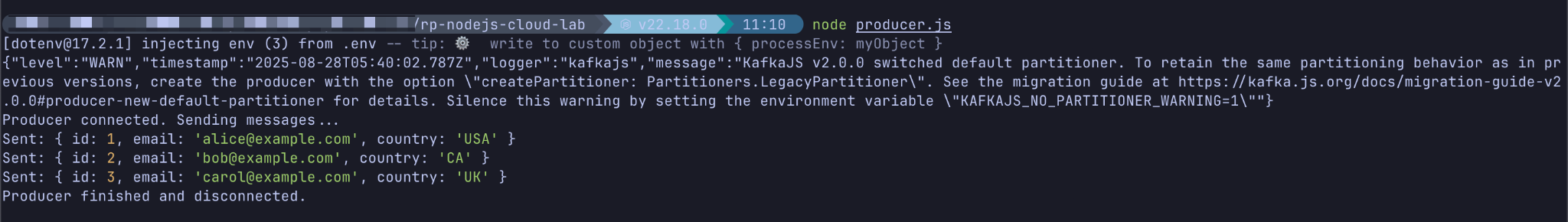
|  |
| --- |
| node consumer.js |

1. **Open a new terminal window** and navigate to the same project directory.
2. **Run the producer in the new terminal:**

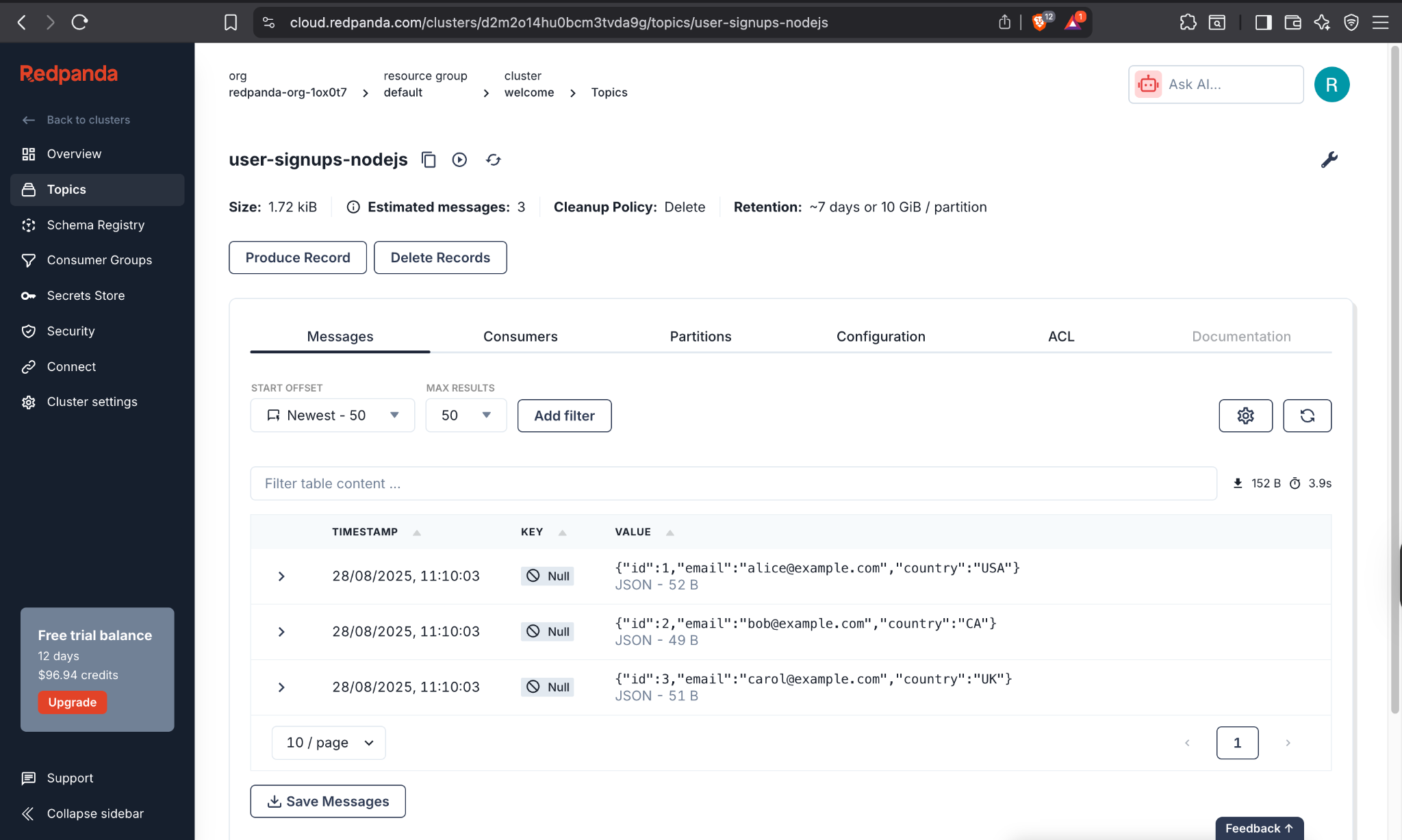
|  |
| --- |
| node producer.js |

## **Step 7: Verify the Results**

1. **Verify in the Consumer Terminal:** Switch back to your first terminal where consumer.js is running. You should see the three messages from the producer printed to the console almost instantly.



1. **Verify in Cloud UI:** Refresh the **Messages** tab for the user-signups-nodejs topic in the Redpanda Cloud UI. You will see the three records.



# **Cleanup**

1. Stop the consumer script by pressing Ctrl+C.
2. Delete the topic from Redpanda Cloud:

|  |
| --- |
| rpk topic delete user-signups-nodejs --profile rpk-cloud |