message sent for jenkins pipeline stage event ... getting error code 40301 message : Topic authorization failed .... wht could be the debugging solution stepwsie how to resolve pls guide

Debugging and Resolving Jenkins Pipeline 40301 "Topic Authorization Failed" Error

This error typically occurs when your Jenkins pipeline lacks proper permissions to publish messages to a messaging topic or event bus. Here's a step-by-step guide to resolve it:

Step 1: Understand the Error

* Error code 40301 indicates an authorization failure
* The pipeline is trying to send a message/event but doesn't have permissions
* This could be in Jenkins itself or an external messaging system (like RabbitMQ, Kafka, etc.)

Step 2: Identify the Target System

Determine where the message is being sent:

1. Check your Jenkinsfile for any messaging-related steps
2. Look for plugins like:
   * CloudBees Event Broker
   * RabbitMQ Publisher
   * Kafka Plugin
   * Webhook steps
   * Any custom messaging integration

Step 3: Verify Credentials and Permissions

For Jenkins Internal Systems:

1. Check Jenkins credentials store:
   * Go to Manage Jenkins > Manage Credentials
   * Verify credentials used in the pipeline exist and are correct
2. Check pipeline permissions:
   * Ensure the Jenkins job/user has "Run" permissions
   * Check "Overall/Read" permissions if using script security

For External Messaging Systems:

1. Verify connection details:
   * Hostname/URL
   * Port
   * Topic/exchange name
   * Credentials (username/password or

message sent for jenkins pipeline stage event ... getting error code 40301 message : Topic authorization failed to kafka .... wht could be the debugging solution stepwsie how to resolve or if you can share extra care or more logs from jenkins pipeline to find root cause pls guide

Debugging Kafka "Topic Authorization Failed" (40301) in Jenkins Pipeline

This error occurs when your Jenkins pipeline lacks proper permissions to produce messages to a Kafka topic. Here's a comprehensive guide to diagnose and resolve the issue:

Step 1: Gather More Information

First, enable detailed logging in your Jenkins pipeline to get more context:

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Produce to Kafka') {

steps {

script {

try {

*// Your Kafka publishing step here*

sh '''

echo "Attempting to publish to Kafka..."

# Add your actual Kafka command here

'''

} catch (Exception e) {

echo "Full error details: ${e.toString()}"

echo "Stack trace: ${e.getStackTrace().join('\n')}"

*// If using a plugin, check for additional error details*

if (e.metaClass.respondsTo(e, 'getCause')) {

echo "Root cause: ${e.getCause()}"

}

throw e

}

}

}

}

}

}

Step 2: Verify Kafka Connection Details

Check these critical parameters in your pipeline:

1. **Bootstrap servers**: Correct host:port combination
2. **Topic name**: Exactly matches the topic in Kafka (case-sensitive)
3. **Authentication**:
   * SASL/SCRAM credentials
   * SSL certificates if using TLS
4. **ACLs**: Your service account has proper permissions

Step 3: Check Kafka-side Configuration

1. Verify Topic Exists

bash

Copy

Download

kafka-topics.sh --bootstrap-server <your\_broker> --list

2. Check ACLs (Access Control Lists)

bash

Copy

Download

kafka-acls.sh --bootstrap-server <your\_broker> --list --topic <your\_topic>

3. Verify Producer Permissions

bash

Copy

Download

kafka-acls.sh --bootstrap-server <your\_broker> --command-config admin.properties \

--list --principal User:<your\_service\_account> --topic <your\_topic>

Step 4: Test Connection Outside Jenkins

Create a test script to isolate the issue:

bash

Copy

Download

#!/bin/bash

*# test-kafka-producer.sh*

BOOTSTRAP\_SERVERS="your-broker:9092"

TOPIC="your-topic"

CLIENT\_ID="jenkins-test-producer"

MESSAGE="test message from Jenkins"

echo "Sending message to Kafka..."

echo "$MESSAGE" | kafka-console-producer.sh \

--bootstrap-server $BOOTSTRAP\_SERVERS \

--topic $TOPIC \

--producer.config kafka-client.properties

Step 5: Common Solutions

1. Add Missing ACLs

bash

Copy

Download

kafka-acls.sh --bootstrap-server <your\_broker> --command-config admin.properties \

--add --allow-principal User:<service\_account> \

--operation Write --topic <your\_topic> \

--producer

2. Fix Authentication Issues

Ensure your kafka-client.properties contains:

properties

Copy

Download

security.protocol=SASL\_SSL

sasl.mechanism=SCRAM-SHA-512

sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required \

username="your\_username" \

password="your\_password";

3. Update Jenkins Plugin Configuration

If using a Kafka plugin:

1. Go to Manage Jenkins > Configure System
2. Find the Kafka plugin section
3. Verify all connection parameters
4. Test the connection if the plugin provides a test button

Step 6: Additional Debugging Commands

Check Kafka Broker Logs

bash

Copy

Download

*# On Kafka brokers:*

tail -f /var/log/kafka/server.log | grep -i "authorization failed"

Describe Topic Details

bash

Copy

Download

kafka-topics.sh --bootstrap-server <your\_broker> --describe --topic <your\_topic>

Step 7: Enable Debug Logging in Kafka Client

Add these JVM options to your Jenkins agent:

Copy

Download

-Dlogging.level.org.apache.kafka=DEBUG

-Dlogging.level.kafka=DEBUG

This will provide detailed logs about the authentication and authorization process.

Would you like me to provide specific guidance for any particular Kafka plugin you're using in Jenkins, or help interpret any specific error messages you're seeing in the logs?

they are usinng http request no kafka plugins

Debugging HTTP Request with Kafka "Topic Authorization Failed" (40301) in Jenkins Pipeline

Since you're using HTTP requests (not Kafka plugins) to interact with Kafka, likely through a REST proxy or custom API, here's a specialized debugging approach:

Step 1: Capture Full HTTP Exchange

Modify your pipeline to log the complete HTTP request/response:

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Produce to Kafka via HTTP') {

steps {

script {

try {

def response = httpRequest(

consoleLogResponseBody: true,

contentType: 'APPLICATION\_JSON',

httpMode: 'POST',

requestBody: '{"records":[{"value":"test message"}]}',

url: 'http://your-kafka-rest-proxy/topics/your-topic',

customHeaders: [

[name: 'Authorization', value: 'Bearer your-token'],

[name: 'Content-Type', value: 'application/vnd.kafka.json.v2+json']

],

validResponseCodes: '100:599' *// Capture all responses*

)

echo "Status: ${response.status}"

echo "Headers: ${response.headers}"

echo "Content: ${response.content}"

} catch (Exception e) {

echo "Full error trace:"

echo e.toString()

echo e.getStackTrace().join('\n')

if (e.getCause()) {

echo "Root cause: ${e.getCause()}"

}

*// Get the HTTP response even if it failed*

def response = e.getResponse()

if (response) {

echo "Failed response content: ${response.content}"

}

throw e

}

}

}

}

}

}

Step 2: Verify HTTP Request Components

Check each element of your HTTP request:

1. **Endpoint URL**:
   * Correct REST proxy address
   * Correct topic name in URL path
   * Example: http://kafka-rest:8082/topics/your-topic
2. **Headers**:
   * Required headers for your Kafka REST proxy:

http

Copy

Download

Content-Type: application/vnd.kafka.json.v2+json

Accept: application/vnd.kafka.v2+json

Authorization: Bearer <token> (or other auth scheme)

1. **Request Body**:
   * Proper JSON format for your proxy:

json

Copy

Download

{

"records": [

{

"key": null,

"value": "your-message-content"

}

]

}

Step 3: Test Connection with CURL

Run this directly on your Jenkins server to isolate the issue:

bash

Copy

Download

curl -v -X POST \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-H "Accept: application/vnd.kafka.v2+json" \

-H "Authorization: Bearer your-token" \

"http://your-kafka-rest-proxy/topics/your-topic" \

-d '{

"records": [

{

"value": "test message from curl"

}

]

}'

Step 4: Common HTTP-Specific Solutions

1. Fix Authentication Issues

* Verify your auth token is valid and not expired
* Check if you need to refresh the token
* Verify the correct auth scheme (Bearer, Basic, etc.)

2. Correct API Version

Some proxies require specific API versions in headers:

http

Copy

Download

Content-Type: application/vnd.kafka.avro.v2+json

Accept: application/vnd.kafka+json, application/vnd.kafka.v2+json

3. Check Proxy-Specific Requirements

Some proxies need:

* Special headers like X-Request-Id
* Different message format (Avro, binary, etc.)
* Schema registry references

Step 5: Examine REST Proxy Logs

If you have access to the Kafka REST proxy, check:

bash

Copy

Download

*# Typical log locations:*

tail -f /var/log/kafka-rest/kafka-rest.log

grep -i "403" /var/log/kafka-rest/\*.log

Step 6: Verify Proxy Configuration

Check these proxy configurations:

1. **Authentication**:
   * OAuth2 settings
   * Basic auth configurations
   * IP whitelisting
2. **Authorization**:
   * Role-based access control
   * Topic-level permissions
   * Producer group permissions
3. **CORS** (if calling from browser scripts):
   * Allowed origins
   * Allowed headers

Step 7: Alternative Approaches

If debugging remains difficult:

1. **Use Kafka CLI through HTTP**:

groovy

Copy

Download

sh '''

curl -sS -X POST \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-H "Authorization: Bearer $(cat /path/to/token)" \

http://proxy/topics/your-topic \

-d @message.json

'''

1. **Temporary Disable Auth for Testing**:  
   (Only in development environments)

properties

Copy

Download

*# In proxy configuration*

authentication.method=NONE

authorization.required=false

Would you like me to provide specific debugging steps for your particular REST proxy implementation (Confluent, Landoop, custom, etc.) or help interpret the full HTTP response you're getting?

basic authentication

Debugging Kafka REST Proxy with Basic Authentication in Jenkins Pipeline

Since you're using HTTP Basic Authentication with your Kafka REST proxy, here's a focused approach to resolve the "Topic Authorization Failed" (40301) error:

Step 1: Verify Basic Authentication Setup

In Your Jenkins Pipeline

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Produce to Kafka') {

steps {

script {

withCredentials([usernamePassword(

credentialsId: 'kafka-rest-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

def authString = "${KAFKA\_USER}:${KAFKA\_PASSWORD}".bytes.encodeBase64().toString()

def response = httpRequest(

consoleLogResponseBody: true,

httpMode: 'POST',

url: 'http://your-kafka-rest-proxy/topics/your-topic',

customHeaders: [

[name: 'Authorization', value: "Basic ${authString}"],

[name: 'Content-Type', value: 'application/vnd.kafka.json.v2+json']

],

requestBody: '{"records":[{"value":"test message"}]}',

validResponseCodes: '100:599'

)

echo "Response: ${response.content}"

}

}

}

}

}

}

Step 2: Common Basic Auth Issues and Solutions

1. Verify Credentials Are Correct

Test credentials independently:

bash

Copy

Download

*# Using curl to verify*

curl -u 'username:password' -v http://your-proxy/topics

2. Check for Special Characters in Password

If your password contains special characters:

groovy

Copy

Download

*// URL encode the password first*

import java.net.URLEncoder

def encodedPassword = URLEncoder.encode(KAFKA\_PASSWORD, 'UTF-8')

def authString = "${KAFKA\_USER}:${encodedPassword}".bytes.encodeBase64().toString()

3. Verify Jenkins Credential Format

Ensure your stored credential:

1. Doesn't have trailing whitespace
2. Matches exactly what Kafka expects
3. Has proper permissions in Jenkins credentials store

Step 3: Kafka REST Proxy Configuration Checks

1. Proxy Authentication Setup

Verify these properties in your REST proxy config (kafka-rest.properties):

properties

Copy

Download

*# Basic Auth must be enabled*

authentication.method=BASIC

authentication.realm=KafkaRest

authentication.roles=admin,user

*# If using JAAS*

listeners=http://0.0.0.0:8082

admin.listeners=http://0.0.0.0:8083

2. JAAS Configuration

Check your JAAS config file (typically kafka-rest.jaas):

java

Copy

Download

KafkaRest {

org.eclipse.jetty.jaas.spi.PropertyFileLoginModule required

file="/path/to/password-file"

debug="true";

};

3. Password File Format

Verify your password file:

text

Copy

Download

# Format: username: password[,rolename...]

admin: adminpassword,admin

producer: producerpassword,producer

Step 4: Permission Mapping

Check how roles map to Kafka permissions in your proxy config:

properties

Copy

Download

*# Example role-to-permission mapping*

role.mapper.class=com.example.CustomRoleMapper

admin.roles=admin

user.roles=producer,consumer

Step 5: Testing with Different Tools

1. Using httpie (more readable than curl)

bash

Copy

Download

http --auth user:pass POST http://proxy/topics/your-topic \

Content-Type:application/vnd.kafka.json.v2+json \

records:='[{"value": "test"}]'

2. Using Postman

1. Set auth type to "Basic Auth"
2. Enter credentials
3. Set headers:
   * Content-Type: application/vnd.kafka.json.v2+json
4. Send raw JSON body

Step 6: Advanced Debugging

1. Enable Proxy Debug Logging

properties

Copy

Download

*# In log4j.properties*

log4j.logger.kafka.rest=DEBUG

log4j.logger.org.eclipse.jetty=DEBUG

2. Check Proxy Access Logs

bash

Copy

Download

tail -f /var/log/kafka-rest/access.log | grep ' 403 '

3. Verify SSL/TLS Configuration (if used)

properties

Copy

Download

*# If using HTTPS*

listeners=https://0.0.0.0:8443

ssl.keystore.location=/path/to/keystore

ssl.keystore.password=keystorepass

Step 7: Final Verification

Create a test pipeline stage just for auth verification:

groovy

Copy

Download

stage('Verify Auth') {

steps {

script {

def response = httpRequest(

url: 'http://your-proxy/topics',

customHeaders: [[name: 'Authorization', value: "Basic ${authString}"]],

validResponseCodes: '100:599'

)

echo "Auth test response: ${response.status}"

}

}

}

If you're still encountering issues, please share:

1. The exact HTTP response headers/body you're getting
2. Any relevant proxy configuration (redacted)
3. Whether you're using Confluent, Landoop, or another REST proxy
4. The Jenkins httpRequest plugin version you're using

dont have admin access to jenkins or any kafka server ... devops cicd service id which clones and checkout code and runs jenkins pipeline with service id ... service id is working fine for other projects this new trying kafka topic and rest url ......so guide

Debugging Kafka REST Access Without Admin Privileges

Since you don't have admin access to Jenkins or Kafka servers but have a working service account that fails only with Kafka topics, here's a targeted approach:

Step 1: Isolate the Authentication Issue

Test Credentials Independently

Add this stage to your pipeline to verify basic connectivity:

groovy

Copy

Download

stage('Test Kafka REST Access') {

steps {

script {

withCredentials([usernamePassword(

credentialsId: 'kafka-rest-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

*// Test endpoint connectivity*

def response = sh(returnStdout: true, script: """

curl -sS -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

-X GET \

-H "Content-Type: application/vnd.kafka.v2+json" \

http://your-kafka-rest-proxy/topics \

-w "\nHTTP Status: %{http\_code}"

""")

echo "Connectivity test result:\n${response}"

*// Test topic-specific access*

def topicTest = sh(returnStdout: true, script: """

curl -sS -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

-X POST \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

http://your-kafka-rest-proxy/topics/your-topic \

-d '{"records":[{"value":"test"}]}' \

-w "\nHTTP Status: %{http\_code}"

""")

echo "Topic access test result:\n${topicTest}"

}

}

}

}

Step 2: Analyze the 403 Error Patterns

Common Causes Without Admin Access:

1. **Topic-Specific Permissions**:
   * Your service account may need explicit produce permissions for this specific topic
   * Existing permissions might be for consumer only
2. **Environment Differences**:
   * The new topic might be in a different Kafka cluster
   * REST proxy might have different auth configurations per environment
3. **Message Format Requirements**:
   * This topic might require Avro/binary format instead of JSON
   * Might need schema registry references

Step 3: Workaround Solutions

Option A: Request Proper Permissions

Provide this information to your Kafka admins:

Copy

Download

Service Account: [your\_service\_id]

Required Access: Producer access to topic [topic\_name]

Current Error: 40301 Topic authorization failed

Test Command Used: [paste your curl test from above]

Option B: Verify Topic Name Exactly

groovy

Copy

Download

*// Add this verification step*

def exactTopicName = sh(returnStdout: true, script: """

curl -sS -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

http://your-kafka-rest-proxy/topics \

| grep -i 'your\_topic\_pattern'

""").trim()

echo "Verified topic exists with name: ${exactTopicName}"

Option C: Try Alternative Endpoints

groovy

Copy

Download

*// Sometimes the /topics endpoint works when specific topic fails*

def bulkResponse = httpRequest(

url: 'http://your-kafka-rest-proxy/topics',

customHeaders: [

[name: 'Authorization', value: "Basic ${authString}"],

[name: 'Content-Type', value: 'application/vnd.kafka.json.v2+json']

],

requestBody: '{"your\_topic": {"records":[{"value":"test"}]}}',

validResponseCodes: '100:599'

)

Step 4: Capture Detailed Diagnostics

Add this to your pipeline to gather maximum information:

groovy

Copy

Download

stage('Collect Diagnostics') {

steps {

script {

withCredentials([usernamePassword(*/\* ... \*/*)]) {

*// 1. Get available topics*

def topics = sh(returnStdout: true, script: """

curl -v -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

http://your-kafka-rest-proxy/topics 2>&1

""")

*// 2. Get specific topic metadata*

def topicMeta = sh(returnStdout: true, script: """

curl -v -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

http://your-kafka-rest-proxy/topics/your-topic 2>&1

""")

*// 3. Test with minimal message*

def testProduce = sh(returnStdout: true, script: """

curl -v -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

-X POST \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

http://your-kafka-rest-proxy/topics/your-topic \

-d '{"records":[{"value":"test"}]}' 2>&1

""")

*// Save all diagnostics*

writeFile file: 'kafka\_diagnostics.log',

text: """

AVAILABLE TOPICS:\n${topics}\n

TOPIC METADATA:\n${topicMeta}\n

PRODUCE ATTEMPT:\n${testProduce}\n

"""

archiveArtifacts artifacts: 'kafka\_diagnostics.log'

}

}

}

}

Step 5: Alternative Approaches

If you can't get permissions quickly:

1. **Verify if a different environment works** (dev vs prod)
2. **Check if Kafka Connect is available** as alternative
3. **See if you can use a shared service account** that already has permissions
4. **Request a temporary topic** with simpler permissions

Key Evidence to Collect for Admins

When requesting help, provide:

1. Exact topic name being accessed
2. Full HTTP request/response from diagnostics
3. Any differences from working projects:
   * Topic naming patterns
   * Message formats
   * Environment configurations
4. Timeframes when access worked (if ever)

Would you like me to help craft a specific permission request email for your Kafka admins based on these findings?

using pipeline script can i echo credentid and username and userpassword

Safely Handling Credentials in Jenkins Pipelines

**Important Security Note**: You should **never** directly echo or log credentials in plaintext. Jenkins provides secure ways to handle credentials. Here's how to work with them safely:

Secure Credential Handling Methods

1. Using withCredentials (Recommended)

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Example') {

steps {

withCredentials([usernamePassword(

credentialsId: 'your-credential-id',

usernameVariable: 'USERNAME',

passwordVariable: 'PASSWORD'

)]) {

*// These variables are available only within this block*

sh 'echo "Using credentials for $USERNAME"'

*// The actual password is masked in logs*

sh 'echo "Password is (masked in logs): $PASSWORD"'

}

}

}

}

}

2. What You'll See in Logs

Copy

Download

[Pipeline] withCredentials

Masking supported pattern matches of $USERNAME or $PASSWORD

[Pipeline] {

[Pipeline] sh

+ echo 'Using credentials for \*\*\*\*'

Using credentials for \*\*\*\*

[Pipeline] sh

+ echo 'Password is (masked in logs): \*\*\*\*'

Password is (masked in logs): \*\*\*\*

3. Alternative for Debugging (Still Secure)

groovy

Copy

Download

script {

withCredentials([usernamePassword(

credentialsId: 'your-credential-id',

usernameVariable: 'USERNAME',

passwordVariable: 'PASSWORD'

)]) {

*// This will show the username but mask the password*

echo "Using credentials with username: ${USERNAME}"

*// To verify credentials are working without exposing them:*

def authString = "${USERNAME}:${PASSWORD}".bytes.encodeBase64().toString()

echo "Base64 encoded auth header (first part shows username): ${authString.substring(0,10)}..."

}

}

What NOT To Do

❌ **Never** do this:

groovy

Copy

Download

*// UNSAFE - exposes credentials in plaintext*

def creds = usernamePassword(credentialsId: 'your-credential-id')

echo "Username: ${creds.username}" *// Exposed in logs*

echo "Password: ${creds.password}" *// Exposed in logs*

If You Absolutely Must Verify Credentials

Instead of echoing, test them with a real API call:

groovy

Copy

Download

withCredentials([usernamePassword(

credentialsId: 'kafka-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASS'

)]) {

def response = sh(returnStdout: true, script: """

curl -sS -u "$KAFKA\_USER:$KAFKA\_PASS" \

-X GET \

http://your-kafka-rest-proxy/topics \

-w "\nHTTP Status: %{http\_code}"

""")

echo "Test response: ${response}"

}

Best Practices

1. **Use credential IDs** - reference credentials by ID never by value
2. **Limit scope** - use withCredentials blocks to minimize exposure
3. **Mask sensitive data** - Jenkins automatically masks standard credential patterns
4. **Use credential helper functions** for complex scenarios

Would you like me to show you how to test your Kafka credentials without exposing them in the logs? I can provide a safe verification pipeline stage.

sure will update for now not conerned with payload but it shd clear 40301 messages

Resolving Kafka 40301 Authorization Error in Jenkins (Without Admin Access)

Since you're focused on clearing the 40301 authorization error first (before handling payload), here's a targeted approach using your Jenkins pipeline:

Safe Credential Testing Pipeline

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Test Kafka Authorization') {

steps {

script {

withCredentials([usernamePassword(

credentialsId: 'kafka-rest-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

*// 1. Test basic connectivity*

def connectivity = sh(returnStdout: true, script: """

curl -sS -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

-X GET \

-H "Content-Type: application/vnd.kafka.v2+json" \

http://your-kafka-rest-proxy/topics \

-w "\nHTTP Status: %{http\_code}" 2>&1

""")

echo "Connectivity Test:\n${connectivity}"

*// 2. Test topic authorization (empty payload)*

def authTest = sh(returnStdout: true, script: """

curl -v -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

-X POST \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

http://your-kafka-rest-proxy/topics/YOUR\_TOPIC \

-d '{"records":[]}' \

-w "\nHTTP Status: %{http\_code}" 2>&1

""")

*// 3. Parse the response*

if (authTest.contains("HTTP Status: 401")) {

error "Authentication failed - invalid credentials"

} else if (authTest.contains("HTTP Status: 403")) {

echo "Authorization Test Result:\n${authTest}"

error "Topic authorization failed (40301) - Service account lacks permissions"

} else if (authTest.contains("HTTP Status: 20")) {

echo "SUCCESS: Credentials have basic topic access"

} else {

echo "Full Response:\n${authTest}"

}

}

}

}

}

}

}

Key Modifications to Fix 40301

1. Verify Credential Scope

groovy

Copy

Download

*// Add this inside your withCredentials block*

def credentialCheck = sh(returnStdout: true, script: """

curl -sS -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

http://your-kafka-rest-proxy/ \

-w "\nHTTP Status: %{http\_code}" 2>&1

""")

echo "Credential Scope Test:\n${credentialCheck}"

2. Test Different Content Types

groovy

Copy

Download

*// Try alternative content types if 403 persists*

def contentTypes = [

'application/vnd.kafka.json.v2+json',

'application/vnd.kafka.binary.v2+json',

'application/json'

]

contentTypes.each { contentType ->

def test = sh(returnStdout: true, script: """

curl -sS -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

-X POST \

-H "Content-Type: ${contentType}" \

http://your-kafka-rest-proxy/topics/YOUR\_TOPIC \

-d '{"records":[]}' \

-w "\nHTTP Status: %{http\_code}" 2>&1

""")

echo "Test with ${contentType}:\n${test}"

}

3. Verify Topic Name Case Sensitivity

groovy

Copy

Download

*// Kafka topics are case-sensitive*

def topicVariations = [

'YOUR\_TOPIC',

'your\_topic',

'Your-Topic',

'YOURTOPIC'

]

topicVariations.each { topic ->

def test = sh(returnStdout: true, script: """

curl -sS -u ${KAFKA\_USER}:${KAFKA\_PASSWORD} \

-X GET \

http://your-kafka-rest-proxy/topics/${topic} \

-w "\nHTTP Status: %{http\_code}" 2>&1

""")

echo "Topic variation '${topic}' test:\n${test}"

}

Next Steps When 40301 Persists

1. **Compare with Working Projects**:

groovy

Copy

Download

*// Add this to identify differences*

echo "Comparing with working project configuration:"

echo "1. Credential ID used in working project: [ask team]"

echo "2. REST endpoint used: [compare URLs]"

echo "3. Headers required: [check other pipelines]"

1. **Request Specific Permissions**:

Copy

Download

Provide this to your Kafka admins:

- Service Account: ${KAFKA\_USER} (from credentials)

- Required Access: PRODUCE to topic YOUR\_TOPIC

- Current Behavior: 40301 on empty payload test

- Working Projects: [list projects where same creds work]

1. **Verify Network Path**:

groovy

Copy

Download

*// Add network path verification*

sh """

traceroute your-kafka-rest-proxy

nc -zv your-kafka-rest-proxy 8082

curl -v http://your-kafka-rest-proxy/info

"""

This approach safely tests your authorization without exposing credentials while gathering the maximum diagnostic information to resolve the 40301 error.

sure , for this scenario in future once everhting is ok ........ if want to make this process as jenkins plugin so that instead of adding all these inevery project in jenkins pipeline .... they just call funtion in plugin ans it does wrapping like decorator pattern ... using kafka rest concept

Creating a Jenkins Plugin for Kafka REST Integration

Based on your scenario, here's how you could design a Jenkins plugin to simplify Kafka REST interactions across all your pipelines:

Plugin Design Concept

1. Core Architecture (Decorator Pattern)

java

Copy

Download

public class KafkaRestPublisher implements SimpleBuildStep {

@DataBoundConstructor

public KafkaRestPublisher(String credentialId, String topic, String message) {

*// Initialization*

}

@Override

public void perform(Run<?,?> run, FilePath workspace,

Launcher launcher, TaskListener listener) {

*// Decorator pattern implementation*

withKafkaCredentials(() -> {

publishToKafkaRest(run, listener);

});

}

private void withKafkaCredentials(Runnable action) {

*// Secure credential handling*

}

private void publishToKafkaRest(Run<?,?> run, TaskListener listener) {

*// Core publishing logic*

}

}

2. Plugin Features

Configuration Options

java

Copy

Download

@Extension

public static final class DescriptorImpl extends BuildStepDescriptor<Builder> {

public ListBoxModel doFillCredentialIdItems() {

*// Populate available credentials*

}

public FormValidation doCheckTopic(@QueryParameter String value) {

*// Validate topic name*

}

}

Pipeline Syntax

groovy

Copy

Download

*// Simplified pipeline usage*

pipeline {

agent any

stages {

stage('Publish to Kafka') {

steps {

kafkaRestPublish(

credentialsId: 'kafka-creds',

topic: 'your-topic',

message: '{"key":"value"}',

contentType: 'application/vnd.kafka.json.v2+json'

)

}

}

}

}

3. Implementation Recommendations

Core Components

1. **Credentials Binding**:

java

Copy

Download

public class KafkaRestCredentials {

private final String username;

private final String password;

public static KafkaRestCredentials fromId(String credentialId) {

*// Fetch from Jenkins credential store*

}

}

1. **REST Client**:

java

Copy

Download

public class KafkaRestClient {

public HttpResponse publish(String topic, String message,

KafkaRestCredentials creds) {

*// Handle all HTTP operations with:*

*// - Retry logic*

*// - Error handling*

*// - Logging*

}

}

1. **Error Handling**:

java

Copy

Download

public class KafkaRestException extends Exception {

private int errorCode;

public static void handleError(HttpResponse response)

throws KafkaRestException {

*// Special handling for 40301*

}

}

4. Advanced Features

Configuration Presets

java

Copy

Download

*// Allow predefined configurations in Jenkins global settings*

public class KafkaRestGlobalConfig extends GlobalConfiguration {

private String defaultRestUrl;

private String defaultCredentialId;

*// Getter/setter methods*

}

Metrics Collection

java

Copy

Download

public class KafkaRestMetrics {

public void recordPublishAttempt(String topic) {

*// Track usage metrics*

}

public void recordError(String topic, int errorCode) {

*// Monitor error rates*

}

}

5. Plugin Development Steps

1. **Setup Maven Project**:

xml

Copy

Download

Run

*<!-- pom.xml -->*

<dependency>

<groupId>org.jenkins-ci.plugins</groupId>

<artifactId>credentials</artifactId>

<version>2.1.19</version>

</dependency>

1. **Implement Extension Points**:

java

Copy

Download

@Extension

public class KafkaRestPublisherDescriptor extends BuildStepDescriptor<Builder> {

@Override

public boolean isApplicable(Class<? extends AbstractProject> jobType) {

return true;

}

}

1. **Add UI Configuration**:

xml

Copy

Download

Run

*<!-- config.jelly -->*

<f:entry title="Credentials" field="credentialId">

<c:select/>

</f:entry>

6. Migration Path from Script to Plugin

1. **Phase 1**: Create shared library with your current solution

groovy

Copy

Download

*// vars/kafkaRestPublish.groovy*

def call(Map config) {

*// Your current implementation*

}

1. **Phase 2**: Develop plugin with same interface

java

Copy

Download

*// Maintain same parameter names/types*

1. **Phase 3**: Gradually migrate projects

7. Security Considerations

1. **Credential Masking**:

java

Copy

Download

public class KafkaRestPublisher extends Step implements StepExecution {

@Override

public void start() throws Exception {

getContext().newBodyInvoker()

.withContext(

CredentialsProvider.getCredentials(

getContext().get(Run.class),

credentialsId

)

)

.start();

}

}

1. **Input Validation**:

java

Copy

Download

public FormValidation doCheckTopic(@QueryParameter String value) {

if (!value.matches("[a-zA-Z0-9\\.\_\\-]+")) {

return FormValidation.error("Invalid topic name");

}

return FormValidation.ok();

}

Would you like me to elaborate on any specific aspect of this plugin design? I can provide more detailed code samples for the most critical components or explain the deployment process for such a plugin.

please can you workflow and class daigram , sequqnce diagram can can understand better how this calls and complete code if possible with folder structure

Kafka REST Jenkins Plugin: Complete Design & Implementation

Here's the comprehensive design and implementation for your Jenkins plugin:

1. Workflow Overview

Copy

Download

[Pipeline] → [KafkaRestPublisher Step] → [Credentials Binding] → [REST Client] → [Kafka REST Proxy]

↑

[Configuration] ← [Global Settings]

2. Class Diagram

Diagram

Code

Download

uses

uses

references

requires

KafkaRestPublisher

+String credentialsId

+String topic

+String message

+String contentType

+perform()

KafkaRestCredentials

+String username

+String password

+fromId()

KafkaRestClient

+HttpResponse publish()

KafkaRestGlobalConfig

+String defaultRestUrl

+String defaultCredentialId

3. Sequence Diagram

Diagram

Code

Download

KafkaProxyKafkaRestClientCredentialsStoreKafkaRestPublisherPipelineKafkaProxyKafkaRestClientCredentialsStoreKafkaRestPublisherPipelinekafkaRestPublish(topic, message)Get credentialsCredentialspublish(topic, message, creds)POST /topics/{topic}200 OK / 403 ErrorResponseSuccess/Failure

4. Complete Implementation

Folder Structure

Copy

Download

kafka-rest-plugin/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── io/jenkins/plugins/

│ │ │ └── kafkarest/

│ │ │ ├── KafkaRestPublisher.java

│ │ │ ├── KafkaRestCredentials.java

│ │ │ ├── KafkaRestClient.java

│ │ │ ├── KafkaRestGlobalConfig.java

│ │ │ └── exceptions/

│ │ │ └── KafkaRestException.java

│ │ └── resources/

│ │ ├── io/jenkins/plugins/kafkarest/

│ │ │ ├── KafkaRestPublisher/

│ │ │ │ └── config.jelly

│ │ │ └── Messages.properties

│ │ └── index.jelly

├── pom.xml

└── README.md

Core Implementation Files

1. KafkaRestPublisher.java

java

Copy

Download

package io.jenkins.plugins.kafkarest;

import hudson.Extension;

import hudson.FilePath;

import hudson.Launcher;

import hudson.model.Run;

import hudson.model.TaskListener;

import hudson.util.ListBoxModel;

import jenkins.model.GlobalConfiguration;

import org.jenkinsci.plugins.workflow.steps.Step;

import org.jenkinsci.plugins.workflow.steps.StepContext;

import org.jenkinsci.plugins.workflow.steps.StepExecution;

import org.kohsuke.stapler.DataBoundConstructor;

import org.kohsuke.stapler.QueryParameter;

import javax.annotation.Nonnull;

import java.io.IOException;

import java.util.Set;

public class KafkaRestPublisher extends Step implements Serializable {

private final String credentialsId;

private final String topic;

private final String message;

private String contentType;

private String restUrl;

@DataBoundConstructor

public KafkaRestPublisher(String credentialsId, String topic, String message) {

this.credentialsId = credentialsId;

this.topic = topic;

this.message = message;

}

@Override

public StepExecution start(StepContext context) {

return new Execution(this, context);

}

public static class Execution extends StepExecution {

private final KafkaRestPublisher publisher;

protected Execution(KafkaRestPublisher publisher, StepContext context) {

super(context);

this.publisher = publisher;

}

@Override

public boolean start() throws Exception {

KafkaRestGlobalConfig config = GlobalConfiguration.all().get(KafkaRestGlobalConfig.class);

KafkaRestCredentials creds = KafkaRestCredentials.fromId(publisher.credentialsId);

KafkaRestClient client = new KafkaRestClient(

publisher.restUrl != null ? publisher.restUrl : config.getDefaultRestUrl()

);

try {

HttpResponse response = client.publish(

publisher.topic,

publisher.message,

publisher.contentType,

creds

);

getContext().onSuccess(response);

} catch (KafkaRestException e) {

getContext().onFailure(e);

}

return false;

}

}

@Extension

public static final class DescriptorImpl extends StepDescriptor {

@Override

public String getFunctionName() {

return "kafkaRestPublish";

}

@Override

public Set<? extends Class<?>> getRequiredContext() {

return ImmutableSet.of(Run.class, FilePath.class, TaskListener.class);

}

public ListBoxModel doFillCredentialsIdItems() {

*// Populate from Jenkins credentials store*

}

}

}

2. KafkaRestClient.java

java

Copy

Download

package io.jenkins.plugins.kafkarest;

import okhttp3.\*;

public class KafkaRestClient {

private final OkHttpClient httpClient;

private final String baseUrl;

public KafkaRestClient(String baseUrl) {

this.baseUrl = baseUrl;

this.httpClient = new OkHttpClient.Builder()

.addInterceptor(new KafkaRestInterceptor())

.build();

}

public HttpResponse publish(String topic, String message, String contentType,

KafkaRestCredentials creds) throws KafkaRestException {

Request request = new Request.Builder()

.url(baseUrl + "/topics/" + topic)

.post(RequestBody.create(message,

MediaType.parse(contentType != null ? contentType :

"application/vnd.kafka.json.v2+json")))

.addHeader("Authorization", creds.getAuthHeader())

.build();

try (Response response = httpClient.newCall(request).execute()) {

if (!response.isSuccessful()) {

throw new KafkaRestException(response.code(),

response.message(), response.body().string());

}

return new HttpResponse(response.code(), response.body().string());

} catch (IOException e) {

throw new KafkaRestException(500, e.getMessage(), null);

}

}

}

3. KafkaRestGlobalConfig.java

java

Copy

Download

package io.jenkins.plugins.kafkarest;

import hudson.Extension;

import jenkins.model.GlobalConfiguration;

import org.kohsuke.stapler.DataBoundSetter;

@Extension

public class KafkaRestGlobalConfig extends GlobalConfiguration {

private String defaultRestUrl = "http://kafka-rest:8082";

private String defaultCredentialId;

public static KafkaRestGlobalConfig get() {

return GlobalConfiguration.all().get(KafkaRestGlobalConfig.class);

}

@DataBoundSetter

public void setDefaultRestUrl(String defaultRestUrl) {

this.defaultRestUrl = defaultRestUrl;

save();

}

@DataBoundSetter

public void setDefaultCredentialId(String defaultCredentialId) {

this.defaultCredentialId = defaultCredentialId;

save();

}

*// Getters*

}

5. Configuration Files

config.jelly

xml

Copy

Download

Run

<?jelly escape-by-default='true'?>

<j:jelly xmlns:j="jelly:core" xmlns:f="/lib/form">

<f:entry title="Credentials" field="credentialsId">

<f:select />

</f:entry>

<f:entry title="Topic" field="topic">

<f:textbox />

</f:entry>

<f:entry title="Message" field="message">

<f:textarea />

</f:entry>

<f:entry title="Content Type" field="contentType">

<f:textbox default="application/vnd.kafka.json.v2+json" />

</f:entry>

<f:entry title="REST URL (override)" field="restUrl">

<f:textbox />

</f:entry>

</j:jelly>

pom.xml

xml

Copy

Download

Run

<project>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.jenkins-ci.plugins</groupId>

<artifactId>plugin</artifactId>

<version>4.16</version>

</parent>

<artifactId>kafka-rest-publisher</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>hpi</packaging>

<dependencies>

<dependency>

<groupId>org.jenkins-ci.plugins</groupId>

<artifactId>credentials</artifactId>

</dependency>

<dependency>

<groupId>org.jenkins-ci.plugins.workflow</groupId>

<artifactId>workflow-step-api</artifactId>

</dependency>

<dependency>

<groupId>com.squareup.okhttp3</groupId>

<artifactId>okhttp</artifactId>

<version>4.9.1</version>

</dependency>

</dependencies>

</project>

6. Usage Examples

Pipeline Usage

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Publish') {

steps {

kafkaRestPublish(

credentialsId: 'kafka-prod-creds',

topic: 'deployment-events',

message: '{"service":"${env.JOB\_NAME}","status":"started"}'

)

}

}

}

}

With Global Defaults

groovy

Copy

Download

kafkaRestPublish(

topic: 'deployment-events',

message: '{"event":"deploy"}'

) *// Uses defaults from Global Configuration*

This implementation provides a complete, secure, and reusable solution for Kafka REST publishing across all your Jenkins pipelines. The plugin handles credential management, error handling, and configuration while providing a simple interface for pipeline authors.

in sequence daigram in you can show ... wht part jenkins does and wht part will kafka server ..so know responsibility

Here's a more detailed sequence diagram that clearly separates Jenkins plugin responsibilities from Kafka server responsibilities:

Enhanced Sequence Diagram with Responsibility Boundaries

Diagram

Code

Download

Kafka BrokersKafka REST ProxyHTTP Client (Plugin)Jenkins Credentials StoreKafka PluginJenkins PipelinePipeline AuthorKafka BrokersKafka REST ProxyHTTP Client (Plugin)Jenkins Credentials StoreKafka PluginJenkins PipelinePipeline AuthorJenkins Pipeline ResponsibilityPlugin ResponsibilityKafka System Responsibility1. Authenticates2. Authorizes3. Validatesalt[Authentication Failed][Authorization Failed][Validation Error]alt[Success Path][Failure Path]Defines pipeline with kafkaRestPublish()Invokes kafkaRestPublish stepRequests credentials (ID: kafka-creds)Returns masked credentialsBuilds request with:POST /topics/your-topicProduce messageAck200 OKSuccessStep succeeds401 Unauthorized403 Forbidden (code 40301)400 Bad RequestErrorStep fails with error detailsPipeline result

Responsibility Breakdown

Jenkins Pipeline Responsibilities:

1. **Pipeline Definition**:
   * User defines when/where to publish messages
   * Provides topic and message content
   * References credential IDs (not actual secrets)
2. **Step Execution**:
   * Invokes plugin at the right stage
   * Handles success/failure outcomes
   * Manages pipeline flow control

Plugin Responsibilities:

1. **Credential Management**:
   * Safely retrieves actual credentials
   * Handles credential masking in logs
   * Manages authentication headers
2. **Request Construction**:
   * Builds proper HTTP request format
   * Applies content-type headers
   * Handles message serialization
3. **Error Handling**:
   * Interprets Kafka error codes
   * Provides user-friendly error messages
   * Implements retry logic (if configured)

Kafka System Responsibilities:

1. **Authentication**:
   * Validates provided credentials
   * Verifies SSL/TLS if enabled
   * Checks IP whitelisting if configured
2. **Authorization**:
   * Verifies topic produce permissions
   * Checks ACLs for the service account
   * Validates quota limits
3. **Message Processing**:
   * Validates message format/schema
   * Handles partitioning
   * Manages producer acks

Key Decision Points Shown:

1. **Credential Flow**:
   * Never exposed to pipeline author
   * Retrieved just-in-time by plugin
2. **Error Classification**:
   * 401 = Authentication failure (wrong credentials)
   * 403 = Authorization failure (need topic permissions)
   * 400 = Validation error (bad message format)
3. **Success Path**:
   * Message flows through REST proxy to Kafka
   * ACK confirms successful write

Typical Error Resolution Paths:

| **Error Code** | **Responsible Party** | **Action Required** |
| --- | --- | --- |
| 401 | Jenkins Admin | Fix credential binding or rotation |
| 40301 | Kafka Admin | Add topic produce permissions |
| 400 | Pipeline Author | Fix message format/schema |
| 502/504 | Infrastructure | Check Kafka proxy health |

This diagram clearly shows where responsibilities transition between the Jenkins environment and the Kafka infrastructure, which is crucial for troubleshooting issues like your original 40301 error. Would you like me to focus on any particular interaction in more detail?

so in my case authentication was suceed using credentialID , username, userpassword but failed for authentication for the topic ...am i analyzing correct

Yes, your analysis is correct. Based on the **40301 (Topic Authorization Failed)** error and successful authentication, here's the breakdown:

**Your Scenario (Exactly What Happened)**

1. ✅ **Authentication Success**
   * Jenkins correctly retrieved credentials (credentialID, username, password)
   * Kafka REST Proxy accepted the credentials (HTTP 200/401 would fail at this stage)
2. ❌ **Authorization Failure (40301)**
   * Your service account **authenticated successfully** but **lacks permissions** for:
     + **Producing** to the topic
     + **Writing** to the specified partition
     + **Using** the REST Proxy’s configured API

**Why This Happens in Kafka**

| **Step** | **System Check** | **Error If Failed** |
| --- | --- | --- |
| 1️⃣ | Can the user authenticate? | 401 Unauthorized |
| 2️⃣ | Does the user have **topic-level permissions**? | 403 Forbidden (40301) |

**Your error (**40301**) occurs at Step 2.**  
→ *Credentials worked, but Kafka’s ACLs (Access Control Lists) blocked the action.*

**How to Fix This (Without Admin Access)**

Since you lack Kafka admin rights, try these steps:

**1. Verify the Topic Exists (Read-Only Check)**

groovy

Copy

Download

sh """

curl -s -u $KAFKA\_USER:$KAFKA\_PASSWORD \

"${KAFKA\_REST\_PROXY}/topics" | grep -i "${TARGET\_TOPIC}"

"""

* If the topic **doesn’t appear**, it may be misspelled or in another cluster.
* If it **appears**, your credentials have **read** but not **write** access.

**2. Test Minimal Write (Diagnostic Request)**

groovy

Copy

Download

def testAuth = sh(returnStatus: true, script: """

curl -s -o /dev/null -w "%{http\_code}" -u $KAFKA\_USER:$KAFKA\_PASSWORD \

-X POST \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

"${KAFKA\_REST\_PROXY}/topics/${TARGET\_TOPIC}" \

-d '{"records":[]}' # Empty payload

""")

echo "HTTP Status: ${testAuth}" *// 403 = Confirms authorization issue*

**3. Request Permissions (Template for Admins)**

Provide this to your Kafka team:

Copy

Download

Service Account: ${KAFKA\_USER}

Required Access: \*\*Produce\*\* to topic `${TARGET\_TOPIC}`

Error Observed: HTTP 403 (Code 40301)

Working Example: Credentials work for topic `old\_topic` but not `new\_topic`

CURL Test:

curl -u USER:PASS -X POST ${REST\_PROXY}/topics/${TARGET\_TOPIC} -d '{"records":[]}'

**4. Temporary Workaround (If Possible)**

* Use a **different topic** with existing permissions.
* Ask if the team has a **shared topic** for testing.

**Key Takeaways**

1. **Authentication ≠ Authorization**
   * 401 = Bad credentials
   * 403 = Correct credentials, but insufficient permissions
2. **Next Steps**
   * Confirm the topic name is **exactly correct** (case-sensitive).
   * Provide your team with the **CURL test output**.
   * Compare with a **working topic** to identify ACL differences.

Would you like a script to **compare permissions** between a working/non-working topic?

but the same service Id when running thorugh pipeline library is able to register events ......that is the same jenkins controller but when same code in jenkinsfile pipleline not working fails authentication ...... can you explain wht might be issue where i am going wrong

This inconsistency suggests there are subtle but critical differences between how your **pipeline library** and **Jenkinsfile** are handling the authentication. Here’s a structured way to debug and resolve the issue:

**1. Key Differences to Investigate**

| **Pipeline Library (Works)** | **Jenkinsfile (Fails)** |
| --- | --- |
| Uses a **predefined shared library** | Directly embeds code |
| May have **implicit credential binding** | Explicit credentials might be misconfigured |
| Could be using **different credential IDs** | Hardcoded or incorrect credentials |
| Might modify **HTTP headers/authentication** | Missing headers or auth method |

**2. Debugging Steps**

**A. Compare Credential Handling**

**In the Library (Working):**  
Check how credentials are fetched:

groovy

Copy

Download

*// Library code likely does this:*

@Library('your-shared-library') \_

kafkaPublish(topic: 'your-topic', message: '...') *// Abstracts credentials*

**In the Jenkinsfile (Failing):**  
Explicit credential usage might be problematic:

groovy

Copy

Download

withCredentials([usernamePassword(

credentialsId: 'kafka-creds', *// Could differ from the library*

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASS'

)]) {

sh """

curl -u $KAFKA\_USER:$KAFKA\_PASS ...

"""

}

**Action:**

* Verify the **exact same**credentialsId is used in both cases.
* Check for **typos** or **environment variables** overriding values.

**B. Check HTTP Headers/Request Format**

The library might be adding **critical headers**:

groovy

Copy

Download

*// Library may silently include:*

headers = [

"Content-Type": "application/vnd.kafka.json.v2+json",

"Accept": "application/vnd.kafka+json"

]

**In your Jenkinsfile, explicitly add:**

groovy

Copy

Download

sh """

curl -u $KAFKA\_USER:$KAFKA\_PASS \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-H "Accept: application/vnd.kafka.v2+json" \

-X POST "${KAFKA\_REST\_URL}/topics/${TOPIC}" \

-d '${MESSAGE}'

"""

**C. Environment/Context Differences**

* **Jenkins Agent Context**:  
  The library might run on a specific agent with **preconfigured environment variables** (e.g., KAFKA\_REST\_URL).  
  **Fix:** Add debug logging to check vars:

groovy

Copy

Download

sh 'env | grep KAFKA\_'

* **Credential Scope**:  
  Library credentials might be **system-level**, while Jenkinsfile uses **pipeline-level**.  
  **Fix:** Verify credential scopes in **Jenkins → Credentials → Store**.

**D. Timeout/Proxy Issues**

* The Jenkinsfile might be missing **proxy settings** configured in the library.  
  **Fix:** Add proxy flags if needed:

groovy

Copy

Download

sh """

curl -x http://proxy:port ...

"""

**3. Quick Diagnostic Script**

Add this to your failing Jenkinsfile to compare environments:

groovy

Copy

Download

stage('Debug Kafka Auth') {

steps {

script {

echo "### Credential Test ###"

sh '''

echo "Testing credentials for Kafka REST..."

curl -v -u $KAFKA\_USER:$KAFKA\_PASS \

-X GET "${KAFKA\_REST\_URL}/topics" \

-w "\nStatus: %{http\_code}"

'''

echo "### Library vs. Jenkinsfile Diff ###"

sh '''

echo "Library path: $(ls -l @Library)"

echo "Jenkinsfile working dir: $(pwd)"

env | sort > env.txt

echo "Environment variables saved to env.txt"

'''

archiveArtifacts 'env.txt'

}

}

}

**4. Common Fixes**

1. **Mismatched Credential IDs**
   * Ensure credentialsId matches the library’s ID exactly.
   * Check for **typos** (e.g., kafka-creds vs. kafka\_creds).
2. **Missing Headers**
   * The library might default to application/json, while Kafka REST requires application/vnd.kafka.json.v2+json.
3. **SSL/TLS Issues**
   * If the library forces HTTPS and your Jenkinsfile uses HTTP, Kafka may reject it.  
     **Fix:** Use https:// and ensure certificates are trusted.
4. **Library Wrappers**
   * The library might silently retry on failures or modify the message format.  
     **Fix:** Replicate its logic exactly.

**5. Final Answer: What’s Likely Happening?**

* **Scenario 1**: Your Jenkinsfile is missing **required headers** (e.g., Content-Type).
* **Scenario 2**: The credentials in the Jenkinsfile are **scoped differently** (e.g., folder-level vs. global).
* **Scenario 3**: The library **pre-processes the message** (e.g., base64 encoding) that your raw Jenkinsfile doesn’t.

**Immediate Action:**

1. Compare the **exact HTTP requests** from both paths (library vs. Jenkinsfile).
2. Use curl -v in both to log headers and auth.
3. Check the **library’s source code** for hidden logic.

Let me know if you’d like a **side-by-side comparison script** to trace the differences!

if i have jenkins console output of sucess run kafka stages event .... how to make the more verbose by using replay or wht to look for in that console output .... if want to make verbose in console output .... wht can try

To make your Jenkins pipeline's Kafka-related stages more verbose in the console output (especially when comparing a **successful run** vs. a failing one), here are the key steps and areas to focus on:

**1. Enable Verbose Logging in Your Pipeline**

Add these debugging techniques to your Jenkinsfile:

**A. For**curl**Requests (HTTP/REST)**

groovy

Copy

Download

sh """

curl -v -u \$KAFKA\_USER:\$KAFKA\_PASSWORD \ # `-v` for verbose

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-X POST "${KAFKA\_REST\_URL}/topics/${TOPIC}" \

-d '${PAYLOAD}' \

-w "\nStatus: %{http\_code}\n"

"""

* -v: Shows full HTTP request/response headers (critical for auth/403 errors).
* -w: Explicitly prints the HTTP status code.

**B. For Kafka Library/Plugins**

If using a Kafka library (e.g., kafka-python), enable debug logging:

groovy

Copy

Download

sh """

export KAFKA\_LOGLEVEL="DEBUG" # For Java/Scala clients

python3 -m your\_kafka\_script --log-level DEBUG

"""

**2. Use Jenkins Replay to Add Debugging**

1. **Go to the successful build** → Click **"Replay"**.
2. **Modify the pipeline** to add verbose logging:

groovy

Copy

Download

stage('Produce to Kafka') {

steps {

script {

echo "### DEBUG: Credential Environment ###"

sh 'echo "Username: $KAFKA\_USER"' *// Check if variables are set*

sh 'env | grep KAFKA\_' *// Verify all Kafka-related env vars*

echo "### DEBUG: HTTP Request Details ###"

sh """

curl -v -u \$KAFKA\_USER:\$KAFKA\_PASSWORD \

"${KAFKA\_REST\_URL}/topics" 2>&1 | tee curl.log

Sucessful run jenkins caonsole output are from Shared Libraray how to get more verbos and strudy so can check why failing in code written in pipeline

To analyze the **successful runs from your Shared Library** and compare them with failing pipeline code, follow this structured approach to extract maximum verbosity and insights:

**1. Capture Debug Output from Shared Library Runs**

**Method A: Replay with Enhanced Logging**

1. **Locate a successful build** using the Shared Library.
2. Click **"Replay"** and modify the library call to add debug flags:

groovy

Copy

Download

@Library('your-library@branch') \_

kafkaPublish(

topic: 'your-topic',

message: '{"event":"test"}',

logLevel: "DEBUG" *// Add this parameter if supported*

)

*If your library doesn’t support logLevel, inject debug code temporarily:*

groovy

Copy

Download

*// Add to your library's Kafka publishing function*

echo "DEBUG: Full Kafka REST URL: ${config.restUrl}"

echo "DEBUG: Using credentials ID: ${config.credentialsId}"

sh 'env | sort | grep KAFKA\_ > env.txt && cat env.txt'

**Method B: Archive Raw HTTP Traffic**

Add this to your library’s code (temporarily):

groovy

Copy

Download

sh """

curl -v --trace-ascii curl.trace \

-u \$KAFKA\_USER:\$KAFKA\_PASSWORD \

-X POST "${REST\_URL}/topics/${topic}" \

-d '${message}' 2>&1

"""

archiveArtifacts artifacts: 'curl.trace'

**2. Key Areas to Compare in Console Output**

**A. Credential Binding**

| **Successful (Library)** | **Failing (Pipeline)** |
| --- | --- |
| [withCredentials] Masking supported patterns | Check if masking is applied |
| Credentials ID: kafka-prod-creds | Verify same ID is used |

**B. HTTP Headers**

Look for these in curl -v output:

Copy

Download

> POST /topics/your-topic HTTP/1.1

> Authorization: Basic [masked]

> Content-Type: application/vnd.kafka.json.v2+json # Must match

**C. Kafka REST Proxy Response**

Copy

Download

< HTTP/1.1 200 OK

< X-Request-Id: 12345 # Present in success, missing in failure

**D. Environment Variables**

Copy

Download

DEBUG: KAFKA\_REST\_URL=https://kafka-proxy:8082

DEBUG: KAFKA\_USER=svc\_account # Compare with failing job

**3. Pipeline Code Audit Checklist**

When comparing your standalone pipeline code to the Shared Library:

1. **Credential Binding**

groovy

Copy

Download

*// Shared Library likely uses:*

withCredentials([usernamePassword(

credentialsId: 'kafka-prod-creds', *// Verify ID matches*

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

*// ...*

}

1. **Message Formatting**  
   The library might preprocess the payload:

groovy

Copy

Download

*// Library may add these silently:*

def message = [

records: [

[ key: null, value: payload ]

]

]

1. **Headers/Content-Type**

groovy

Copy

Download

*// Library may force this:*

headers = [

"Content-Type": "application/vnd.kafka.json.v2+json",

"Accept": "application/vnd.kafka.v2+json"

]

**4. Side-by-Side Test Script**

Add this to **both** your library and pipeline to compare behavior:

groovy

Copy

Download

stage('Kafka Debug Test') {

steps {

script {

sh '''

echo "=== ENV VARS ==="

env | grep -E 'KAFKA|TOPIC|REST' | sort

echo "=== CREDENTIAL TEST ==="

curl -v -u "$KAFKA\_USER:$KAFKA\_PASSWORD" \

"${KAFKA\_REST\_URL}/topics" \

-w "\nStatus: %{http\_code}\n" 2>&1

echo "=== TOPIC TEST ==="

curl -v -u "$KAFKA\_USER:$KAFKA\_PASSWORD" \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-X POST "${KAFKA\_REST\_URL}/topics/${TOPIC}" \

-d '{"records":[]}' \

-w "\nStatus: %{http\_code}\n" 2>&1

'''

}

}

}

**5. Common Library vs. Pipeline Differences**

| **Issue Type** | **Library Behavior** | **Your Pipeline Fix** |
| --- | --- | --- |
| **Credentials** | Uses Jenkins credential binding | Ensure identical credentialsId |
| **Message Format** | Wraps payload in records array | Match the exact JSON structure |
| **Content-Type** | Forces application/vnd.kafka.json.v2+json | Add header explicitly |
| **Error Handling** | Retries on 403/network issues | Implement retry logic |

**Next Steps**

1. **Archive artifacts** from both runs (env.txt, curl.trace).
2. **Diff the outputs**:

bash

Copy

Download

diff library-curl.log pipeline-curl.log

1. **Check Jenkins credentials** for mismatches:

Copy

Download

Manage Jenkins → Credentials → System → Global

Would you like a **specific diff analysis** of your two console outputs? Share redacted snippets, and I’ll identify discrepancies.

how to add retry lgic.. this could be issue

Adding Retry Logic for Kafka REST Calls in Jenkins Pipeline

To handle intermittent failures (like network issues or temporary Kafka unavailability), you can implement retry logic in your Jenkins pipeline. Here's how to add robust retry mechanisms:

1. Basic Retry with Sleep (Simple Approach)

groovy

Copy

Download

def kafkaPublishWithRetry(String url, String topic, String message, int maxRetries = 3) {

int retryCount = 0

boolean success = false

while(retryCount < maxRetries && !success) {

try {

withCredentials([usernamePassword(

credentialsId: 'kafka-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

def response = sh(returnStdout: true, script: """

curl -v -u \$KAFKA\_USER:\$KAFKA\_PASSWORD \\

-H "Content-Type: application/vnd.kafka.json.v2+json" \\

-X POST "${url}/topics/${topic}" \\

-d '${message}' \\

-w "\nHTTP\_STATUS:%{http\_code}" 2>&1

""")

*// Check HTTP status code*

def status = (response =~ /HTTP\_STATUS:(\d+)/)[0][1] as Integer

if (status >= 200 && status < 300) {

success = true

echo "Kafka publish succeeded after ${retryCount} retries"

} else {

throw new Exception("HTTP ${status} - ${response}")

}

}

} catch (Exception e) {

retryCount++

echo "Attempt ${retryCount} failed: ${e.getMessage()}"

if (retryCount < maxRetries) {

sleep(time: 5, unit: 'SECONDS') *// Wait before retry*

echo "Retrying..."

} else {

error("Failed after ${maxRetries} attempts: ${e.getMessage()}")

}

}

}

}

2. Advanced Retry with Exponential Backoff

groovy

Copy

Download

def kafkaPublishWithBackoff(String url, String topic, String message, int maxRetries = 5) {

int retryCount = 0

int delaySeconds = 1

while(true) {

try {

*// Same publish logic as above*

withCredentials([...]) {

def response = sh(...)

def status = (response =~ /HTTP\_STATUS:(\d+)/)[0][1] as Integer

if (status >= 200 && status < 300) {

echo "Success after ${retryCount} retries"

return

}

throw new Exception("HTTP ${status}")

}

} catch (Exception e) {

if (++retryCount >= maxRetries) {

error("Max retries (${maxRetries}) exceeded: ${e.getMessage()}")

}

*// Exponential backoff with jitter*

delaySeconds = Math.min(delaySeconds \* 2, 30) *// Cap at 30s*

def jitter = (Math.random() \* 5) as int *// Add 0-5s jitter*

def waitTime = delaySeconds + jitter

echo "Attempt ${retryCount} failed. Waiting ${waitTime}s before retry..."

sleep(time: waitTime, unit: 'SECONDS')

}

}

}

3. Retry Specific Status Codes Only

groovy

Copy

Download

def isRetryableError(int statusCode) {

*// Retry on these status codes:*

def retryableCodes = [408, 429, 500, 502, 503, 504]

return statusCode in retryableCodes

}

def kafkaPublishWithSelectiveRetry(String url, String topic, String message) {

int retries = 3

int attempt = 0

while(attempt < retries) {

attempt++

try {

def response = sh(...) *// Same as above*

def status = (response =~ /HTTP\_STATUS:(\d+)/)[0][1] as Integer

if (status == 403) {

error("Authorization failed (403) - not retryable")

}

else if (status >= 200 && status < 300) {

return *// Success*

}

else if (isRetryableError(status)) {

echo "Retryable error (${status})"

}

else {

error("Non-retryable error: HTTP ${status}")

}

} catch (Exception e) {

if (attempt >= retries) {

error("Failed after ${retries} attempts: ${e.getMessage()}")

}

sleep(time: 5, unit: 'SECONDS')

}

}

}

4. Using Jenkins Retry Step (Simplest)

For basic cases, you can use Jenkins' built-in retry:

groovy

Copy

Download

stage('Publish to Kafka') {

steps {

retry(3) {

withCredentials([...]) {

sh '''

curl -v -u $KAFKA\_USER:$KAFKA\_PASSWORD \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-X POST "${KAFKA\_URL}/topics/${TOPIC}" \

-d '{"records":[{"value":"test"}]}'

'''

}

}

}

}

Key Considerations:

1. **Don't retry 403 errors** - These are authorization failures that won't resolve without permission changes
2. **Add delays between retries** - Avoid overwhelming the system
3. **Log each attempt** - Critical for debugging
4. **Consider idempotency** - Ensure duplicate messages won't cause issues
5. **Set reasonable limits** - Typically 3-5 retries with increasing delays

Example Usage in Pipeline:

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Publish Event') {

steps {

script {

def message = '{"records":[{"value":"test"}]}'

kafkaPublishWithRetry(

url: 'http://kafka-rest:8082',

topic: 'deploy-events',

message: message,

maxRetries: 5

)

}

}

}

}

}

Choose the approach that best fits your reliability requirements and error handling strategy. The exponential backoff method is generally most robust for production systems.

in pileline script we are encoding payload .. where library code dont encode payload could this be reason

Yes, the **payload encoding difference** between your pipeline script and the shared library code could absolutely be the reason for the authentication/authorization failure (403 error). Here's why and how to diagnose/fix it:

**Why Payload Encoding Matters for Kafka REST**

1. **Content-Type Headers**:
   * If you encode the payload differently, the Content-Type header might not match what Kafka expects.
   * Example: Library sends raw JSON, while your pipeline sends base64-encoded data without updating the header.
2. **Signature Validation**:
   * Some Kafka REST proxies validate message integrity by hashing the raw payload. Encoding changes break this.
3. **Schema Registry**:
   * If using Avro, encoding mismatches cause schema validation failures.

**How to Check for Encoding Issues**

**1. Compare Payloads Side-by-Side**

Add debug output to both flows:

**In Shared Library (Working):**

groovy

Copy

Download

echo "Library Payload (RAW): ${payload}"

echo "Library Headers: ${headers}" *// Check Content-Type*

**In Your Pipeline (Failing):**

groovy

Copy

Download

def encodedPayload = payload.bytes.encodeBase64().toString()

echo "Pipeline Payload (Encoded): ${encodedPayload}"

sh """

curl -v -u \$KAFKA\_USER:\$KAFKA\_PASSWORD \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-X POST "${KAFKA\_URL}/topics/${TOPIC}" \

-d '${encodedPayload}' 2>&1 | tee curl.log

"""

archiveArtifacts artifacts: 'curl.log'

**2. Key Differences to Look For**

|  | **Shared Library (Works)** | **Your Pipeline (Fails)** |
| --- | --- | --- |
| **Payload** | {"records":[{"value":"test"}]} | eyJyZWNvcmRzIjpbeyJ2YWx1ZSI6InRlc3QifV19 (Base64) |
| **Content-Type** | application/vnd.kafka.json.v2+json | Unchanged (mismatched) |
| **Kafka Proxy Logs** | Accepts as valid JSON | Rejects as "malformed" |

**3. Fixes to Try**

**Option A: Remove Encoding (Match Library)**

groovy

Copy

Download

*// Send raw JSON like the library*

def payload = '{"records":[{"value":"test"}]}'

sh """

curl ... -d '${payload}'

"""

**Option B: Update Headers if Encoding is Required**

groovy

Copy

Download

*// If you MUST encode, specify the correct Content-Type*

def encodedPayload = payload.bytes.encodeBase64().toString()

sh """

curl ... \

-H "Content-Type: application/vnd.kafka.binary.v2+json" \

-d '${encodedPayload}'

"""

**Option C: Debug with**base64**Roundtrip**

groovy

Copy

Download

*// Verify encoding reversibility*

def original = '{"records":[{"value":"test"}]}'

def encoded = original.bytes.encodeBase64().toString()

def decoded = new String(encoded.decodeBase64())

echo """

Original: ${original}

Encoded: ${encoded}

Decoded: ${decoded}

"""

*// If decoded != original, your encoding is broken*

**4. Common Encoding Pitfalls**

1. **Double-Encoding**:

groovy

Copy

Download

*// WRONG: Encoding already-encoded data*

def doublyEncoded = payload.bytes.encodeBase64().toString().bytes.encodeBase64()

1. **Charset Issues**:

groovy

Copy

Download

*// Always specify UTF-8*

def encoded = payload.getBytes("UTF-8").encodeBase64().toString()

1. **Newline Handling**:

groovy

Copy

Download

*// Strip newlines that curl might add*

def payload = '{"records":[{"value":"test"}]}'.replaceAll("\n", "")

**5. Final Verification**

After making changes, confirm:

1. The **raw payload** matches the library's version.
2. The **Content-Type header** aligns with the encoding:
   * application/vnd.kafka.json.v2+json → Raw JSON
   * application/vnd.kafka.binary.v2+json → Base64
3. The **HTTP status code** changes from 403 to 200.

**Next Steps**

1. **Temporarily disable encoding** in your pipeline to test if it works like the library.
2. **Check Kafka proxy logs** for messages like:

Copy

Download

"Invalid message format"

"Malformed JSON"

"Schema validation failed"

1. **Update your question** with:
   * A redacted payload from both flows
   * The exact Content-Type headers used
   * Any Kafka proxy error logs

Would you like me to provide a **specific diff script** to compare the library and pipeline payloads byte-for-byte?

Content-Type application/vnd.kafka.json.v2+json i have kept same application/vnd.kafka.json.v2+json , issue might with records we are using ---> key : value: base64Encode(""" { }""").stripIndent() could this be the issue as in pipeline they are not using key pls check and suggest .... also 1 more point .... is the record payload if not then what is payload and wht is record

Yes, the **missing**key**field** and **base64 encoding of the entire payload** (instead of just the value) could absolutely be causing your 403 authorization failure. Let me break this down clearly:

**1. Kafka REST Payload Structure Requirements**

The Kafka REST Proxy expects this **exact JSON structure** for produce requests:

json

Copy

Download

{

"records": [

{

"key": null, *// Optional but must be present*

"value": "your\_actual\_data" *// Can be string, JSON, or base64*

}

]

}

**Your Problem Areas**

| **Issue** | **Impact** | **Fix** |
| --- | --- | --- |
| Missing key field | Some Kafka setups require explicit key: null | Add key: null |
| Base64-encoding the **entire payload** | Proxy expects base64 only on value | Encode just the value |
| stripIndent() altering whitespace | May break JSON parsing | Remove it |

**2. Fix Your Pipeline Payload**

**Broken Code (Current)**

groovy

Copy

Download

*// This encodes the ENTIRE structure (wrong)*

def badPayload = base64Encode("""{

"records": [

{ "value": "test" } // Missing 'key'

]

}""").stripIndent()

**Corrected Code**

groovy

Copy

Download

*// 1. Build proper JSON structure*

def payload = [

records: [

[

key: null, *// Explicit null key*

value: base64Encode("test") *// Only encode the value*

]

]

]

*// 2. Convert to JSON string*

def payloadStr = groovy.json.JsonOutput.toJson(payload)

*// 3. Send (do NOT encode the outer structure)*

sh """

curl -v -u \$KAFKA\_USER:\$KAFKA\_PASSWORD \\

-H "Content-Type: application/vnd.kafka.json.v2+json" \\

-X POST "${KAFKA\_URL}/topics/${TOPIC}" \\

-d '${payloadStr}'

"""

**3. Key Terminology**

| **Term** | **Description** |
| --- | --- |
| **Payload** | The complete HTTP request body ({"records":[...]}) |
| **Record** | A single message in the records array |
| **Value** | The actual data you're sending (can be base64) |
| **Key** | Optional partition key (often null) |

**4. Debugging Steps**

1. **Add this to your pipeline** to validate the payload:

groovy

Copy

Download

echo "Final Payload:\n${payloadStr}"

*// Should look EXACTLY like:*

*// {"records":[{"key":null,"value":"dGVzdA=="}]}*

1. **Compare with Shared Library**:

bash

Copy

Download

*# In library code, find where the payload is built*

echo "Library Payload: ${libraryPayload}"

1. **Test with Minimal Valid Payload**:

groovy

Copy

Download

def testPayload = '{"records":[{"key":null,"value":"dGVzdA=="}]}'

sh """

curl ... -d '${testPayload}'

"""

**5. Common Mistakes Checklist**

* ✅ **Always include**"key": null (even if optional)
* ✅ **Only base64 the**value**field**, not the entire payload
* ✅ **Use raw JSON** for the outer structure
* ❌ Never use stripIndent() on JSON (breaks formatting)
* ❌ Never double-encode (check if your base64Encode() is redundant)

**6. Why This Causes 403 Errors**

The Kafka REST Proxy performs:

1. **Structure Validation** → Rejects payloads missing required fields
2. **Authorization Check** → Fails if the parsed message doesn't match ACLs
3. **Content-Type Verification** → application/vnd.kafka.json.v2+json expects unencoded outer JSON

A malformed payload fails at step 1, often manifesting as a cryptic 403.

**Final Answer**

Your issue is **almost certainly** caused by:

1. Missing key: null in records
2. Incorrect base64 encoding of the **entire payload** instead of just the value

**Fix:**

groovy

Copy

Download

def payload = [

records: [

[ key: null, value: base64Encode("your\_data") ] *// Only encode value*

]

]

Try this and monitor both the Jenkins console and Kafka proxy logs for changes in the error message. The 403 should resolve if the payload structure matches the shared library's.

no ... pipeline code is not using key in records and it is working but we are using in pipeline script ...... and wht is differnce of record and payload are they same

Let me clarify the key concepts and differences between **records** and **payloads** in Kafka REST API calls, and why your pipeline might be failing despite the shared library working without a key field.

**1. Key Definitions**

| **Term** | **Description** | **Example** |
| --- | --- | --- |
| **Payload** | The **complete JSON body** sent to Kafka REST Proxy | {"records": [{...}]} |
| **Record** | An **individual message** inside the records array | {"key":null, "value":"data"} |
| **Value** | The actual data being sent (can be string/JSON/base64) | "Hello" or "eyJkYXRhIjoidGVzdCJ9" (base64) |

**2. Why Your Pipeline Fails (Even Without**key**)**

**Problem Scenarios**

1. **Base64 Encoding the Entire Payload**
   * ❌ Wrong: base64Encode('{"records":[{"value":"data"}]}')
   * ✅ Correct: {"records":[{"value":base64Encode("data")}]}
2. **Whitespace/Formatting Differences**

groovy

Copy

Download

*// stripIndent() may break JSON parsing*

def payload = """{

"records": [{"value":"data"}]

}""".stripIndent() *// Risky!*

1. **Implicit**key**Handling**
   * Some Kafka REST Proxy versions auto-inject key: null if omitted
   * Others **strictly require** it → Causing 403 if missing

**3. How to Debug**

**Step 1: Capture EXACT Working Payload from Shared Library**

groovy

Copy

Download

*// Add this to the library code temporarily*

echo "Library Payload (RAW): ${payload}"

*// Example output: {"records":[{"value":"data"}]}*

**Step 2: Compare with Your Pipeline**

groovy

Copy

Download

*// In your pipeline script*

def yourPayload = """{

"records": [

{"value": ${base64Encode('data')}}

]

}"""

echo "Pipeline Payload (RAW): ${yourPayload}"

**Step 3: Validate with**curl

bash

Copy

Download

*# Test with explicit key (even if null)*

curl -v -u user:pass \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-X POST http://kafka-rest:8082/topics/test \

-d '{"records":[{"key":null,"value":"data"}]}'

*# Test without key*

curl ... -d '{"records":[{"value":"data"}]}'

**4. Critical Differences Table**

|  | **Shared Library (Works)** | **Your Pipeline (Fails)** |
| --- | --- | --- |
| **Payload Structure** | {"records":[{"value":"data"}]} | {"records":[{"value":"<base64>"}]} |
| **Key Field** | Implicitly null | Missing (may trigger 403) |
| **Encoding** | Raw value | Base64-encoded value |
| **Whitespace** | Compact JSON | Pretty-printed with stripIndent() |

**5. Solutions to Try**

**Option A: Match Library Exactly**

groovy

Copy

Download

*// Remove base64 encoding if library doesn't use it*

def payload = '{"records":[{"value":"data"}]}'

**Option B: Explicitly Add**key: null

groovy

Copy

Download

def payload = [

records: [

[ key: null, value: base64Encode("data") ] *// Encode ONLY value*

]

]

def payloadStr = groovy.json.JsonOutput.toJson(payload) *// Compact JSON*

**Option C: Verify Proxy Version**

bash

Copy

Download

*# Check if proxy requires 'key' field*

curl http://kafka-rest:8082/config

*# Look for: `kafka.rest.validation.require.key=false`*

**6. Key Takeaways**

1. **Records ≠ Payload**:
   * A payload **contains** records (like an envelope contains letters)
2. **403 Fixes**:
   * Always include key: null for compatibility
   * Only base64 the value field, not the entire payload
   * Remove stripIndent() to avoid whitespace issues
3. **Debugging**:
   * Compare RAW payloads (not encoded) between library/pipeline
   * Test with curl to isolate Jenkins issues

**Next Steps**

1. **Temporarily make your pipeline payload identical to the library's** (remove base64 encoding).
2. **If it works**, gradually reintroduce your changes (encoding, etc.) one at a time.
3. **Check Kafka REST Proxy logs** for schema validation errors.

Would you like me to provide a **specific payload diff tool** to compare the library and pipeline outputs byte-for-byte?

if ihave replay option of succes lirbaray build ... wht can i add in jenkinsfile which uses pipeline library with gives more details in consoleoutput

To get **detailed debug output** from a successful library build using the **Replay** feature in Jenkins, add these strategic logging statements to your Jenkinsfile. This will help you compare why your standalone pipeline fails while the library succeeds.

**1. Add These Debug Steps in Replay Mode**

Modify your Jenkinsfile during replay to include:

**A. Full Payload Inspection**

groovy

Copy

Download

stage('Debug Kafka Payload') {

steps {

script {

*// 1. Print raw payload before sending*

def payload = [

records: [

[ key: null, value: "your\_data" ]

]

]

echo "Payload Structure:\n${groovy.json.JsonOutput.prettyPrint(groovy.json.JsonOutput.toJson(payload))}"

*// 2. Validate base64 encoding (if used)*

def encodedValue = "your\_data".bytes.encodeBase64().toString()

echo "Base64 Encoded Value:\n${encodedValue}"

}

}

}

**B. HTTP Headers & Credentials**

groovy

Copy

Download

withCredentials([usernamePassword(

credentialsId: 'kafka-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

sh '''

echo "=== Credentials ==="

echo "Username: $KAFKA\_USER"

echo "Password: \*\*\*\*" # Masked by Jenkins

echo "=== HTTP Request ==="

curl -v -u "$KAFKA\_USER:$KAFKA\_PASSWORD" \

-H "Content-Type: application/vnd.kafka.json.v2+json" \

-X POST "${KAFKA\_REST\_URL}/topics/test-topic" \

-d '{"records":[{"key":null,"value":"test"}]}' \

-w "\nStatus: %{http\_code}" 2>&1 | tee curl.log

'''

archiveArtifacts artifacts: 'curl.log'

}

**C. Environment Variables**

groovy

Copy

Download

sh '''

echo "=== Environment ==="

env | sort | grep -E 'KAFKA|TOPIC|REST'

'''

**2. Key Areas to Compare**

| **Checkpoint** | **Library (Working)** | **Your Pipeline (Failing)** |
| --- | --- | --- |
| **Payload Structure** | {"records":[{"key":null,"value":"data"}]} | {"records":[{"value":"data"}]} (missing key) |
| **Base64 Encoding** | Only value field encoded | Entire payload encoded? |
| **Content-Type Header** | application/vnd.kafka.json.v2+json | Same but mismatched with payload? |
| **Whitespace** | Compact JSON | Pretty-printed with stripIndent()? |

**3. Critical Debugging Commands**

Add these to your replay:

**A. Validate JSON Syntax**

groovy

Copy

Download

sh '''

echo '{"records":[{"key":null,"value":"test"}]}' | jq empty

if [ $? -eq 0 ]; then

echo "JSON is valid"

else

echo "JSON is INVALID"

fi

'''

**B. Network Connectivity Test**

groovy

Copy

Download

sh '''

echo "=== Network Check ==="

ping -c 2 $(echo "${KAFKA\_REST\_URL}" | awk -F[/:] '{print $4}')

nc -zv $(echo "${KAFKA\_REST\_URL}" | awk -F[/:] '{print $4}') 8082

'''

**C. Kafka REST API Discovery**

groovy

Copy

Download

sh '''

echo "=== Available Topics ==="

curl -s -u "$KAFKA\_USER:$KAFKA\_PASSWORD" "${KAFKA\_REST\_URL}/topics" | jq .

'''

**4. Sample Jenkinsfile with Full Debugging**

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Debug Kafka') {

steps {

script {

*// ===== PAYLOAD DEBUG =====*

def payload = [

records: [

[ key: null, value: "test" ]

]

]

def payloadStr = groovy.json.JsonOutput.toJson(payload)

echo "Final Payload:\n${payloadStr}"

*// ===== CREDENTIALS & REQUEST =====*

withCredentials([usernamePassword(

credentialsId: 'kafka-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

sh '''

echo "=== Credentials ==="

echo "Username: $KAFKA\_USER"

echo "=== Curl Command ==="

echo "curl -u \$KAFKA\_USER:\$KAFKA\_PASSWORD \\

-H 'Content-Type: application/vnd.kafka.json.v2+json' \\

-X POST '${KAFKA\_REST\_URL}/topics/test-topic' \\

-d '${payloadStr}'"

echo "=== Actual Response ==="

curl -v -u "$KAFKA\_USER:$KAFKA\_PASSWORD" \\

-H "Content-Type: application/vnd.kafka.json.v2+json" \\

-X POST "${KAFKA\_REST\_URL}/topics/test-topic" \\

-d '${payloadStr}' \\

-w "\nStatus: %{http\_code}" 2>&1

'''

}

}

}

}

}

}

**5. What to Look For in Console Output**

1. **Payload Differences**:

diff

Copy

Download

- {"records":[{"value":"test"}]} # Failing

+ {"records":[{"key":null,"value":"test"}]} # Working

1. **HTTP Headers**:

Copy

Download

> Content-Type: application/vnd.kafka.json.v2+json

> Authorization: Basic \*\*\*\*

1. **Response Codes**:

Copy

Download

< HTTP/1.1 403 Forbidden

{"error\_code":40301,"message":"Topic authorization failed"}

1. **Environment Variables**:

Copy

Download

KAFKA\_REST\_URL=http://kafka-proxy:8082

TOPIC=test-topic

**6. Pro Tips**

1. **Use**jq**to Validate JSON**:

bash

Copy

Download

echo '{"records":[{"key":null,"value":"test"}]}' | jq .

1. **Capture Full Traffic** (temporarily):

groovy

Copy

Download

sh '''

curl --trace-ascii trace.log -u "$KAFKA\_USER:$KAFKA\_PASSWORD" ...

archiveArtifacts artifacts: 'trace.log'

'''

1. **Compare with Library**:

groovy

Copy

Download

*// If using @Library, add this to the library code*

echo "Library Payload:\n${JsonOutput.toJson(payload)}"

This approach will give you **side-by-side comparability** between working (library) and failing (pipeline) flows. The key is to eliminate all differences until the 403 disappears. Start by **removing base64 encoding** entirely to test if that’s the root cause.

was checking in 1 response in console output ...... it says [{ "partition":0,"offset":21281881,"erro\_code":null }], "key\_schema\_id":null,"value\_schema\_id":690} can you suggest wht this response means and wht value\_schema\_id am i missing something in pipeline script

The response you're seeing indicates a **successful message publication** to Kafka, but reveals important schema-related metadata. Let me break this down and explain how it impacts your pipeline script:

**1. Decoding the Kafka REST Response**

json

Copy

Download

{

"offsets": [

{

"partition": 0,

"offset": 21281881,

"error\_code": null *// Success!*

}

],

"key\_schema\_id": null, *// No schema for key*

"value\_schema\_id": 690 *// Schema ID for the value*

}

| **Field** | **Meaning** | **Your Case** |
| --- | --- | --- |
| partition | Kafka partition where message was stored | 0 |
| offset | Unique message identifier in the partition | 21281881 |
| error\_code | null means success | ✅ Good |
| key\_schema\_id | Schema ID for the key (if using Avro) | null (no key schema) |
| value\_schema\_id | Schema ID for the value | 690 (critical!) |

**2. Why**value\_schema\_id: 690**Matters**

This indicates:

1. **Schema Registry is Enabled**
   * Your Kafka cluster uses **Avro schemas** (or another schema format)
   * Schema ID 690 validates the message structure
2. **Your Pipeline May Be Missing:**
   * Schema registry URL configuration
   * Proper Avro schema headers
   * Schema-compatible serialization

**3. Required Fixes for Your Pipeline Script**

**A. Add Schema Registry Headers**

groovy

Copy

Download

sh """

curl -v -u \$KAFKA\_USER:\$KAFKA\_PASSWORD \\

-H "Content-Type: application/vnd.kafka.avro.v2+json" \\ // Changed!

-H "Accept: application/vnd.kafka.v2+json" \\

-X POST "${KAFKA\_REST\_URL}/topics/${TOPIC}" \\

-d '{

"value\_schema\_id": 690, // Explicit schema ID

"records": [

{

"key": null,

"value": {"your\_field": "data"} // Must match schema 690

}

]

}'

"""

**B. Or Use Schema Registry Directly**

groovy

Copy

Download

def payload = """

{

"value\_schema\_id": 690,

"records": [

{

"key": null,

"value": {"name": "test", "count": 1} // Example Avro structure

}

]

}

"""

**C. Verify Schema Compatibility**

1. **Check schema ID 690**:

bash

Copy

Download

curl http://schema-registry:8081/schemas/ids/690

1. **Sample output**:

json

Copy

Download

{

"schema": "{\"type\":\"record\",\"name\":\"Event\",\"fields\":[{\"name\":\"name\",\"type\":\"string\"}]}"

}

**4. Key Differences: Library vs Your Script**

|  | **Library (Works)** | **Your Pipeline (Fails)** |
| --- | --- | --- |
| **Content-Type** | application/vnd.kafka.avro.v2+json | application/vnd.kafka.json.v2+json |
| **Schema ID** | Automatically injected | Missing |
| **Value Format** | Avro-compliant JSON | Raw/base64 JSON |

**5. Debugging Steps**

1. **Capture the Library's Exact Request**:

groovy

Copy

Download

*// Add to shared library code temporarily*

echo "Library Headers: ${headers}"

echo "Library Payload: ${payload}"

1. **Test with Hardcoded Schema ID**:

groovy

Copy

Download

def testPayload = '''

{

"value\_schema\_id": 690,

"records": [

{"key": null, "value": {"name": "test"}}

]

}

'''

1. **Schema Registry Check**:

bash

Copy

Download

*# List all subjects*

curl http://schema-registry:8081/subjects

*# Get schema by ID*

curl http://schema-registry:8081/schemas/ids/690

**6. Common Fixes**

1. **Update Content-Type**:

diff

Copy

Download

- "Content-Type: application/vnd.kafka.json.v2+json"

+ "Content-Type: application/vnd.kafka.avro.v2+json"

1. **Include**value\_schema\_id:

groovy

Copy

Download

def payload = [

value\_schema\_id: 690,

records: [

[key: null, value: [name: "test"]] *// Avro-structured data*

]

]

1. **Use Schema Registry Client** (if available):

groovy

Copy

Download

@Grab('io.confluent:kafka-schema-registry-client:6.2.1')

import io.confluent.kafka.schemaregistry.client.CachedSchemaRegistryClient

def client = new CachedSchemaRegistryClient("http://schema-registry:8081", 10)

def schema = client.getByID(690) *// Validate schema*

**7. Final Checklist**

✅ **Add**value\_schema\_id: 690 to your payload  
✅ **Change Content-Type** to application/vnd.kafka.avro.v2+json  
✅ **Structure value** to match the Avro schema (e.g., {"name":"test"})  
✅ **Verify connectivity** to schema registry (http://schema-registry:8081)

If you provide the schema definition (from ID 690), I can help craft an exact payload structure for your pipeline!

customHeaders : [

To

customHeaders[ [name: "Accept", value:"application/vnd.kafka.v2+json" ], name: "Content-Type", value: "application/vnd.kafka.jsonschema.v2+json" ],[name:"Autorization", value:args.outhHeader]], requestBody: jsonPaylod, ... these i am using in pipeline script where i am wrong guide

Based on your custom headers and the Kafka REST Proxy response showing value\_schema\_id: 690, there are a few critical issues to address:

**Key Problems in Your Current Headers/Payload**

1. **Mismatched**Content-Type  
   You're using application/vnd.kafka.jsonschema.v2+json but should use either:
   * application/vnd.kafka.avro.v2+json (for Avro with schema registry)
   * application/vnd.kafka.json.v2+json (for raw JSON without schema)
2. **Typo in**Authorization**Header**  
   "Autorization" → Should be "Authorization" (missing the "h")
3. **Missing**value\_schema\_id  
   Your payload needs to reference schema ID 690 explicitly.

**Corrected Pipeline Script**

groovy

Copy

Download

*// 1. Define headers (fixed typo + correct Content-Type)*

def customHeaders = [

[name: "Accept", value: "application/vnd.kafka.v2+json"],

[name: "Content-Type", value: "application/vnd.kafka.avro.v2+json"], *// For schema registry*

[name: "Authorization", value: args.authHeader] *// Fixed typo*

]

*// 2. Build payload with schema ID*

def jsonPayload = groovy.json.JsonOutput.toJson([

value\_schema\_id: 690, *// From your response*

records: [

[

key: null,

value: [ *// Must match schema 690's structure*

field1: "value1", *// Replace with actual schema fields*

field2: 123

]

]

]

])

*// 3. Send request*

httpRequest(

url: "${kafkaRestUrl}/topics/${topicName}",

httpMethod: 'POST',

customHeaders: customHeaders,

requestBody: jsonPayload,

validResponseCodes: '200:404' *// Allow 200-404 for debugging*

)

**Verification Steps**

1. **Check Schema Structure**  
   Verify what schema ID 690 expects:

bash

Copy

Download

curl http://schema-registry:8081/schemas/ids/690

Example response:

json

Copy

Download

{

"schema": "{\"type\":\"record\",\"name\":\"Event\",\"fields\":[{\"name\":\"field1\",\"type\":\"string\"},{\"name\":\"field2\",\"type\":\"int\"}]}"

}

1. **Test with Minimal Valid Payload**

groovy

Copy

Download

def testPayload = '''

{

"value\_schema\_id": 690,

"records": [

{

"key": null,

"value": {"field1": "test", "field2": 1}

}

]

}

'''

1. **Debug Headers**  
   Add this to your pipeline:

groovy

Copy

Download

echo "Final Headers: ${customHeaders}"

echo "Final Payload: ${jsonPayload}"

**Common Mistakes Checklist**

| **Issue** | **Fix** |
| --- | --- |
| "Autorization" typo | Rename to "Authorization" |
| Wrong Content-Type | Use avro.v2+json for schemas |
| Missing value\_schema\_id | Add to payload |
| Incorrect value structure | Match schema fields exactly |
| Base64-encoded values | Only encode if schema expects bytes |

**If You Still Get 403 Errors**

1. **Verify Credentials**  
   Ensure args.authHeader has a valid token:

groovy

Copy

Download

echo "Auth Header: ${args.authHeader.take(10)}..." *// Log first 10 chars*

1. **Check Kafka ACLs**  
   Your service account needs:

bash

Copy

Download

kafka-acls --list --principal User:your-service-account

Required permission:

Copy

Download

WRITE on TOPIC <your-topic>

1. **Temporarily Disable Schema Validation** (Dev Only)  
   If possible, test with:

groovy

Copy

Download

contentType: "application/vnd.kafka.json.v2+json" *// Bypass schema check*

Let me know if you need help decoding schema ID 690 or crafting the exact value structure!

args.authHeader ahs basic authentication

Since you're using **Basic Authentication** in args.authHeader but getting a 403 error, here's the precise fix for your pipeline script:

**1. Fixing Basic Authentication**

**Problem Identified**

* Your Authorization header is likely malformed when passing args.authHeader
* Basic Auth requires Base64(username:password) but you might be:
  + Double-encoding it
  + Missing the Basic prefix
  + Using incorrect credentials

**Correct Implementation**

groovy

Copy

Download

*// 1. Get credentials PROPERLY (if not already encoded)*

withCredentials([usernamePassword(

credentialsId: 'kafka-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

*// 2. Encode for Basic Auth*

def authString = "${KAFKA\_USER}:${KAFKA\_PASSWORD}".bytes.encodeBase64().toString()

args.authHeader = "Basic ${authString}" *// ← Must include "Basic " prefix*

}

*// 3. Use in headers (fixed typo)*

def customHeaders = [

[name: "Accept", value: "application/vnd.kafka.v2+json"],

[name: "Content-Type", value: "application/vnd.kafka.avro.v2+json"],

[name: "Authorization", value: args.authHeader] *// Now properly formatted*

]

**2. Full Working Script**

groovy

Copy

Download

pipeline {

agent any

stages {

stage('Publish to Kafka') {

steps {

script {

*// A. Get credentials securely*

withCredentials([usernamePassword(

credentialsId: 'kafka-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

*// B. Build headers*

def authString = "${KAFKA\_USER}:${KAFKA\_PASSWORD}".bytes.encodeBase64().toString()

def customHeaders = [

[name: "Accept", value: "application/vnd.kafka.v2+json"],

[name: "Content-Type", value: "application/vnd.kafka.avro.v2+json"],

[name: "Authorization", value: "Basic ${authString}"] *// Proper Basic Auth*

]

*// C. Build schema-aware payload*

def jsonPayload = groovy.json.JsonOutput.toJson([

value\_schema\_id: 690,

records: [

[

key: null,

value: [

field1: "value1", *// Must match schema*

field2: 123

]

]

]

])

*// D. Send request*

httpRequest(

url: "${env.KAFKA\_REST\_URL}/topics/${env.TOPIC}",

httpMethod: 'POST',

customHeaders: customHeaders,

requestBody: jsonPayload,

validResponseCodes: '200:403'

)

}

}

}

}

}

}

**3. Key Verification Steps**

**1. Validate Your Auth Header**

Add this debug step:

groovy

Copy

Download

echo "Auth Header: ${args.authHeader.take(10)}..." *// Check prefix exists*

*// Should look like: "Basic SDFsdGk..."*

**2. Test Credentials Independently**

bash

Copy

Download

*# Replace with your actual credentials*

curl -v -u username:password \

-H "Content-Type: application/vnd.kafka.avro.v2+json" \

http://kafka-rest:8082/topics

**3. Check Schema Compatibility**

bash

Copy

Download

*# Get schema ID 690's requirements*

curl http://schema-registry:8081/schemas/ids/690

*# Response example:*

*# {"schema":"{\"type\":\"record\",\"fields\":[{\"name\":\"field1\",\"type\":\"string\"}]}"}*

**4. Common Pitfalls & Fixes**

| **Error** | **Cause** | **Solution** |
| --- | --- | --- |
| 401 Unauthorized | Malformed Authorization header | Ensure Basic prefix exists |
| 403 Forbidden | Credentials valid but no topic write access | Request WRITE permissions on the topic |
| 404 Not Found | Incorrect topic name | List topics: curl -u user:pass http://kafka-rest:8082/topics |
| 500 Server Error | Value doesn't match schema | Align value structure with schema ID 690 |

**5. Final Checklist**

1. ✅ **Basic Auth Format**: "Basic " + Base64(username:password)
2. ✅ **Schema Compliance**: value matches schema ID 690's structure
3. ✅ **Headers**:

groovy

Copy

Download

Content-Type: application/vnd.kafka.avro.v2+json

Accept: application/vnd.kafka.v2+json

1. ✅ **Permissions**: Service account has WRITE on the topic

If issues persist, share:

* The **exact**value**structure** you’re sending
* **Schema ID 690**'s definition
* Full **error response headers** (use curl -v)

super this looks good will try ..

The server is busy. Please try again later.

super this looks good will try ..

Glad to hear it! Here's a **condensed action plan** with key takeaways to implement immediately:

**1. Critical Fixes to Try Now**

1. **Basic Auth Header**

groovy

Copy

Download

*// Encode PROPERLY (Jenkins credentials + "Basic " prefix)*

withCredentials([usernamePassword(

credentialsId: 'kafka-creds',

usernameVariable: 'KAFKA\_USER',

passwordVariable: 'KAFKA\_PASSWORD'

)]) {

args.authHeader = "Basic ${"$KAFKA\_USER:$KAFKA\_PASSWORD".bytes.encodeBase64()}"

}

1. **Schema-Aware Payload**

groovy

Copy

Download

def jsonPayload = groovy.json.JsonOutput.toJson([

value\_schema\_id: 690, *// From your response*

records: [[

key: null,

value: [

field1: "value1", *// Replace with schema fields*

field2: 123 *// Get from schema registry*

]

]]

])

1. **Headers (Exact Copy-Paste)**

groovy

Copy

Download

def customHeaders = [

[name: "Accept", value: "application/vnd.kafka.v2+json"],

[name: "Content-Type", value: "application/vnd.kafka.avro.v2+json"],

[name: "Authorization", value: args.authHeader] *// Fixed typo*

]

**2. Debugging Pro Tips**

1. **Verify Auth Header**

groovy

Copy

Download

echo "Auth header starts with: ${args.authHeader?.take(10)}..."

*// Should print: "Basic SDFsd..."*

1. **Test with Minimal**curl

bash

Copy

Download

curl -v -u user:pass -H "Content-Type: application/vnd.kafka.avro.v2+json" \

-X POST "http://kafka-rest:8082/topics/your-topic" \

-d '{"value\_schema\_id":690, "records":[{"key":null, "value":{"field1":"test"}}]}'

1. **Check Schema**

bash

Copy

Download

curl http://schema-registry:8081/schemas/ids/690

**3. Expected Success Response**

json

Copy

Download

{

"offsets": [{"partition": 0, "offset": 12345}],

"key\_schema\_id": null,

"value\_schema\_id": 690 *// Matches your payload*

}

**4. If Still Failing**

* **403 Error**: Confirm your service account has WRITE on the topic:

bash

Copy

Download

kafka-acls --list --topic your-topic --principal User:your-service-account

* **400 Error**: Validate payload matches schema exactly.

Let me know if you hit any snags—happy to help refine further! 🚀

git status says nothing to commit , working tree clean ... just now i have add and commit command still saying clean wht is the case .... i need to push to main brnach wht is command

If git status shows a clean working tree despite you making changes and running git add + git commit, here's what's happening and how to push to the main branch:

**1. Why It Shows "Clean"**

* You already committed the changes (they're in your **local repository** but not yet pushed to remote)
* No new changes exist since your last commit

**2. How to Push to**main**Branch**

**Step 1: Verify Your Commit Exists Locally**

bash

Copy

Download

git log --oneline -n 3 *# Show last 3 commits*

* You should see your recent commit in the list.

**Step 2: Push to Remote**main**Branch**

bash

Copy

Download

git push origin main

* If you renamed master to main, use:

bash

Copy

Download

git push origin HEAD:main

**Step 3: Verify on Remote**

bash

Copy

Download

git ls-remote origin main *# Check remote branch*

**3. Common Scenarios & Fixes**

| **Scenario** | **Solution** |
| --- | --- |
| **"Everything up-to-date"** but changes missing | You forgot to commit. Run: git add . && git commit -m "msg" |
| **Branch name mismatch** | Push explicitly: git push origin HEAD:main |
| **No upstream branch** | First-time push: git push -u origin main |

**4. Full Workflow Example**

bash

Copy

Download

*# 1. Stage changes (if new edits exist)*

git add .

*# 2. Commit (if not done)*

git commit -m "Your commit message"

*# 3. Push to main*

git push origin main

**5. Pro Tips**

* **Force push (only if necessary)**

bash

Copy

Download

git push --force origin main *# DANGER: Overwrites remote history*

* **Check remote branches**

bash

Copy

Download

git remote show origin

Let me know if you get any errors during push!

git push origin main this worked i was giving git push origin/main

Ah! That explains it. Here's why your original command failed and how to avoid this in the future:

**1. Key Difference in Commands**

| **Correct Command** | **Wrong Command** | **Why It Matters** |
| --- | --- | --- |
| git push origin main | git push origin/main |  |
| Pushes to branch main on remote origin | Tries to push to a non-existent remote named origin/main | The slash makes Git look for a *remote name* instead of a branch |

**2. Git Push Syntax Explained**

bash

Copy

Download

git push <remote-name> <branch-name>

* origin: Your default remote repository name
* main: Your branch name

**Wrong Interpretation**  
origin/main refers to the local copy of the remote branch (a "remote-tracking branch"), not the actual remote.

**3. How to Verify Remotes & Branches**

bash

Copy

Download

*# List all remotes*

git remote -v

*# List all branches (local + remote)*

git branch -a

*# See tracking relationships*

git remote show origin

**4. If You Accidentally Created a Bad Remote**

If you ran git push origin/main earlier, you might have created a weird remote. Clean it up with:

bash

Copy

Download

*# Delete the misnamed remote (if it exists)*

git remote remove origin/main 2>/dev/null

**5. Recommended Workflow**

bash

Copy

Download

*# 1. Check status*

git status

*# 2. Stage changes*

git add .

*# 3. Commit*

git commit -m "Your message"

*# 4. Push to main*

git push origin main

*# Alternative (if local branch tracks remote)*

git push

**6. Pro Tip: Set Upstream Tracking**

To just use git push without specifying branch:

bash

Copy

Download

git push -u origin main *# First time only*

git push *# All future pushes*

**7. Common Mistakes Table**

| **Mistake** | **Fix** | **Why** |
| --- | --- | --- |
| git push origin/main | git push origin main | Slash vs space |
| git push main | git push origin main | Missing remote name |
| git push origin master | git push origin main | Branch renamed |

Glad git push origin main worked! 🎉 Let me know if you hit other Git quirks.

1 more scenario have to discuss this is exisitng scenario -- it is working status bitbucket is sorce repo , jenkins build server , artifactory to save artifacts, ansible tower for deployment, terraform for iac , Digital.ai XLR release tool , dataical for database deployment ....... now tell me if want to track each and every build till end on dashboard ..... where you will place the code os it fetches all info and not distrub current working code .. let me know if possible then with workflow , tools will be used pls guide

To track **end-to-end build/deployment metrics** across your multi-tool CI/CD pipeline without disrupting existing workflows, here's a comprehensive solution:

**1. Architecture Overview**

Diagram

Code

Download

Trigger

Artifacts

Terraform

Ansible

Dataical

Orchestrate

Orchestrate

Orchestrate

Collect Data

Collect Data

Collect Data

Collect Data

Collect Data

Collect Data

Bitbucket

Jenkins

Artifactory

Infra

App Deployment

DB Changes

Digital.ai XLR

Dashboard

**2. Recommended Tools**

| **Purpose** | **Tool** | **Why?** |
| --- | --- | --- |
| **Data Aggregation** | **Elasticsearch + Logstash** | Unified data collection from all tools |
| **Visualization** | **Grafana** | Customizable dashboards |
| **Lightweight Alternative** | **Prometheus + Grafana** | If you prefer metrics over logs |
| **Distributed Tracing** | **Jaeger** | Track cross-tool workflows |

**3. Non-Invasive Implementation**

**A. Data Collection Layer**

python

Copy

Download

*# Example: Jenkins Shared Library (track\_build.groovy)*

def trackEvent(Map args) {

// Send data to Elasticsearch

sh """

curl -X POST "${ELASTIC\_URL}/cicd\_events/\_doc" \\

-H "Content-Type: application/json" \\

-d '{

"timestamp": "${new Date().format("yyyy-MM-dd'T'HH:mm:ssZ")}",

"tool": "jenkins",

"pipeline": "${env.JOB\_NAME}",

"stage": "${args.stage}",

"status": "${args.status}",

"artifacts": "${args.artifacts}", // Artifactory paths

"xlr\_release\_id": "${env.XLR\_RELEASE\_ID}" // From Digital.ai

}'

"""

}

**B. Integration Points**

| **Tool** | **Data to Capture** | **Method** |
| --- | --- | --- |
| **Jenkins** | Build stats, test results | Shared Library |
| **Artifactory** | Artifact metadata | REST API Webhooks |
| **Ansible Tower** | Deployment status | Job Template Callbacks |
| **Terraform** | Plan/apply outputs | TF Cloud API |
| **Digital.ai XLR** | Release phases | Native Elasticsearch Export |
| **Dataical** | DB change logs | SQL Audit Triggers |

**4. Sample Workflow**

1. **Bitbucket Triggers Jenkins**

groovy

Copy

Download

*// Jenkinsfile*

stages {

stage('Build') {

steps {

trackEvent(stage: "build", status: "started")

*// Existing build steps*

trackEvent(stage: "build", status: "completed", artifacts: "http://artifactory/libs/myapp/1.0.0")

}

}

}

1. **Ansible Tower Reports Back**

yaml

Copy

Download

*# ansible.cfg (callback plugin)*

[callback\_elasticsearch]

elasticsearch\_host=monitoring.example.com

1. **Terraform Sync**

hcl

Copy

Download

*# terraform apply with sentry*

resource "null\_resource" "log\_deploy" {

provisioner "local-exec" {

command = "curl -X POST ${ELASTIC\_URL}/infra\_events -d '{\"tf\_state\":\"${terraform.workspace}\"}'"

}

}

**5. Dashboard Examples**

**Grafana Panels to Build**

1. **Deployment Pipeline View**
   * Jenkins build status → Ansible phases → DB change timings

sql

Copy

Download

SELECT tool, stage, status

FROM "cicd\_events"

WHERE xlr\_release\_id = '$release'

ORDER BY timestamp

1. **Artifact Traceability**

sql

Copy

Download

SELECT build\_id, artifact\_path

FROM "cicd\_events"

WHERE artifacts IS NOT NULL

1. **Mean Time to Recovery (MTTR)**

sql

Copy

Download

SELECT avg(duration)

FROM (

SELECT max(timestamp) - min(timestamp) AS duration

FROM "cicd\_events"

WHERE status = 'failed' GROUP BY correlation\_id

)

**6. Key Benefits**

* **Zero Disruption**: Uses existing APIs/webhooks
* **Unified Correlation**: Track via XLR\_RELEASE\_ID across tools
* **Tech-Agnostic**: Works with your current stack
* **Self-Service**: Teams can add their own Grafana panels

**7. Implementation Checklist**

1. Set up Elasticsearch cluster (docker-compose for POC)
2. Add shared library to Jenkins (trackEvent.groovy)
3. Configure Artifactory webhooks
4. Import Grafana dashboard templates
5. Train teams on adding custom tracking

Would you like me to provide specific code snippets for any integration point?

jenakis acts as build server and orchstration server it has plugins for all so it intiates ......so finaly collects all data of worflow and give success in steps .. can see all this pipeline steps .... now you review and give answer

The server is busy. Please try again later.

jenkins acts as build server and orchstration server it has plugins for all so it intiates ......so finaly collects all data of worflow and give success in steps .. can see all this pipeline steps .... now you review and give answer also if can sequqnce diagram od 1 complete flow with responsibilties they use shared library

Here's a refined solution leveraging Jenkins as the orchestration hub with a clear sequence diagram and responsibilities:

**Optimized Jenkins-Centric Workflow**

**1. Sequence Diagram with Responsibilities**

Diagram

Code

Download

All ToolsDigitalAIDataicalTerraformAnsibleTowerArtifactoryJenkinsBitbucketAll ToolsDigitalAIDataicalTerraformAnsibleTowerArtifactoryJenkinsBitbucketCode Push/PR Merge (Webhook)Shared Library Starts(Track Initial Metadata)Build & Publish Artifacts(with build ID)Apply Infrastructure Changes(log TF plan/output)Trigger App Deployment(pass artifact version)Execute DB Migrations(with versioned scripts)Update Release Status(phase completion)Approve Next Phase(if gated)Shared Library CollectsFinal Status (Success/Fail)Audit Trail Sent toCentral Dashboard (ES/Grafana)

**2. Key Responsibilities**

| **Tool** | **Responsibility** | **Data Captured** |
| --- | --- | --- |
| **Jenkins** | Orchestration & Tracking | Build ID, Timestamps, Tool Responses |
| **Shared Library** | Unified Logging | Pipeline Metadata, Error Context |
| **Artifactory** | Artifact Storage | Download Stats, Build Links |
| **Ansible Tower** | Deployment Execution | Host-wise Status, Rollback Info |
| **Terraform** | Infrastructure State | Plan Output, Resource Changes |
| **Dataical** | DB Version Control | SQL Execution Logs |
| **Digital.ai XLR** | Release Governance | Approval Gates, Phase Transitions |

**3. Implementation with Shared Library**

vars/trackPipeline.groovy

groovy

Copy

Download

def call(Map config) {

*// 1. Start Tracking*

sendEvent(eventType: "START",

data: [pipeline: env.JOB\_NAME, commit: env.GIT\_COMMIT])

try {

*// 2. Execute Existing Steps*

config.steps.each { step ->

sendEvent(eventType: "STEP\_START",

data: [stage: step.name])

step.execute()

sendEvent(eventType: "STEP\_END",

data: [stage: step.name, status: "SUCCESS"])

}

*// 3. Final Success*

sendEvent(eventType: "PIPELINE\_END", status: "SUCCESS")

} catch (Exception e) {

*// 4. Failure Handling*

sendEvent(eventType: "PIPELINE\_END",

status: "FAILED",

error: e.toString())

throw e

}

}

private sendEvent(Map args) {

def payload = [

timestamp: new Date().format("yyyy-MM-dd'T'HH:mm:ssZ"),

buildId: env.BUILD\_ID,

tool: "jenkins",

correlationId: env.XLR\_RELEASE\_ID ?: env.BUILD\_TAG

] + args.data

*// Send to Elasticsearch/Dashboard*

httpRequest(

url: "${env.DASHBOARD\_API}/events",

httpMethod: 'POST',

requestBody: groovy.json.JsonOutput.toJson(payload),

contentType: 'APPLICATION\_JSON'

)

}

**Sample Jenkinsfile**

groovy

Copy

Download

@Library('pipeline-utils') \_

pipeline {

agent any

stages {

stage('Build') {

steps {

trackPipeline {

steps = [

[name: 'compile', execute: { sh 'mvn package' }],

[name: 'publish', execute: {

sh 'curl -X PUT "${ARTIFACTORY\_URL}/libs/${ARTIFACT\_NAME}" -T target/\*.jar'

}]

]

}

}

}

}

}

**4. Central Dashboard Integration**

**Grafana Query Example**

sql

Copy

Download

SELECT

tool, stage, status,

timestamp AS "time",

correlationId AS "Release ID"

FROM "cicd\_events"

WHERE $\_\_timeFilter(timestamp)

ORDER BY timestamp

**Dashboard Metrics**

* **Pipeline Health**: Success rate per stage
* **Artifact Traceability**: Build → Artifactory → Deployment links
* **Terraform Compliance**: Resource changes per environment
* **DB Change Velocity**: Dataical execution frequency

**5. Benefits Over Previous Approach**

1. **Minimal Changes**
   * Wraps existing steps without modification
2. **Tool-Agnostic Tracking**
   * Consistent metadata format across Ansible/Terraform/etc.
3. **Built-in Correlation**
   * Ties all events to BUILD\_ID or XLR\_RELEASE\_ID
4. **Zero Vendor Lock-in**
   * Replace Elasticsearch with Prometheus/Splunk as needed

**6. Failure Scenario Handling**

Diagram

Code

Download

DigitalAITerraformDashboardJenkinsDigitalAITerraformDashboardJenkinsSTEP\_START (Terraform Apply)Plan Failed (Error Details)STEP\_END (status=FAILED, error="TF Validation Error")Abort ReleaseRELEASE\_ABORTED

**7. Key Enhancement Recommendations**

1. **Add Distributed Tracing IDs**

groovy

Copy

Download

*// In shared library*

def traceId = UUID.randomUUID().toString()

1. **Enable Ansible Callback Plugin**

yaml

Copy

Download

*# ansible.cfg*

[callback\_jenkins\_events]

jenkins\_url=${env.JENKINS\_URL}

1. **Terraform Cloud Integration**

hcl

Copy

Download

*# main.tf*

terraform {

cloud {

organization = "your-org"

workspaces {

tags = ["jenkins:${env.BUILD\_ID}"]

}

}

}

This approach gives you complete visibility while keeping existing pipelines intact. Would you like me to elaborate on any specific integration?

the above is wworking code they dont have dashboard that realtime shows movement of code chekin till final taget deployment ......passign though all tools shwoing on dashboard ........ like Train tracking , plane tracking system they know where now it is .. wht is speed any blockers etc ...... you can imagine ... can we do same to the code already working with distubing exisiting prod scenario

the above is working cicd code process they dont have dashboard that realtime shows movement of code chekin till final taget deployment ......passign though all tools shwoing on dashboard ........ like Train tracking , plane tracking system they know where now it is .. wht is speed any blockers etc ...... you can imagine ... can we do same to the code already working with distubing exisiting prod scenario and add our logic to track end to end

The server is busy. Please try again later.

the above is working cicd code process they dont have dashboard that realtime shows movement of code chekin till final taget deployment ......passign though all tools shwoing on dashboard ........ like Train tracking , plane tracking system they know where now it is .. wht is speed any blockers etc ...... you can imagine ... can we do same to the code already working with distubing exisiting prod scenario and add our logic to track end to end

The server is busy. Please try again later.

the above is working cicd code process they dont have dashboard that realtime shows movement of code chekin till final taget deployment ......passign though all tools shwoing on dashboard ........ like Train tracking , plane tracking system they know where now it is .. wht is speed any blockers etc ...... you can imagine ... can we do same to the code already working with distubing exisiting prod scenario and add our logic to track end to end

The server is busy. Please try again later.

the above is working cicd code process they dont have dashboard that realtime shows movement of code chekin till final taget deployment ......passign though all tools shwoing on dashboard ........ like Train tracking , plane tracking system they know where now it is .. wht is speed any blockers etc ...... you can imagine ... can we do same to the code already working with distubing exisiting prod scenario and add our logic to track end to end

Here's how to implement **real-time code movement tracking** (like a flight radar) for your existing CI/CD pipeline **without disrupting production**:

**1. Architecture (Non-Invasive Monitoring Layer)**

Diagram

Code

Download

Tracking Layer

Existing Pipeline

Webhook

Artifacts

Deploy

Provision

DB Changes

Orchestrate

Events

Events

Events

Events

Events

Events

Bitbucket

Jenkins

Artifactory

Ansible Tower

Terraform

Dataical

Digital.ai XLR

Kafka

Stream Processor

Elasticsearch

Grafana Dashboard

**2. Implementation Steps**

**A. Event Collection (Zero Code Changes)**

| **Tool** | **Method** | **Example** |
| --- | --- | --- |
| **Jenkins** | Use Logstash Plugin | [Config](https://plugins.jenkins.io/logstash/) |
| **Artifactory** | Native **Webhooks** | POST to Kafka on download/deploy |
| **Ansible Tower** | **Job Callback** | tower-cli send\_callback --monitor-url |
| **Terraform** | **Cloud Events** | TF\_CLOUD\_EVENTS=kafka://events.tf |
| **Dataical** | **DB Triggers** | AFTER DEPLOY LOG TO SERVICE |
| **Digital.ai** | **Native API Export** | xlr webhook --target=kafka |

**B. Stream Processing (Apache Flink)**

python

Copy

Download

*# Flink job (events\_processor.py)*

from pyflink.datastream import StreamExecutionEnvironment

env = StreamExecutionEnvironment.get\_execution\_environment()

*# Consume from Kafka*

events = env.add\_source(KafkaSource(

topics=["jenkins", "artifactory", "ansible"],

deserializer=JsonDeserializer()

))

*# Key by correlation ID (build/release)*

keyed\_events = events.key\_by(lambda e: e["correlation\_id"])

*# Send to Elasticsearch*

keyed\_events.add\_sink(ElasticsearchSink(

hosts=["elastic:9200"],

index="pipeline\_events"

))

**C. Real-Time Dashboard (Grafana)**

sql

Copy

Download

*-- Current State per Release*

SELECT

correlation\_id AS "Release ID",

MAX\_BY(stage, timestamp) AS "Current Stage",

NOW() - MAX(timestamp) AS "Time in Stage",

COUNT\_IF(status='FAILED') OVER (PARTITION BY correlation\_id) AS "Blockers"

FROM pipeline\_events

WHERE $\_\_timeFilter(timestamp)

GROUP BY correlation\_id

**3. Sample Event Schema**

json

Copy

Download

{

"event\_id": "uuidv4",

"correlation\_id": "jenkins-123/xlr-456", *// Links across tools*

"timestamp": "ISO8601",

"tool": "jenkins|artifactory|ansible|...",

"stage": "build|deploy|provision|...",

"status": "STARTED|IN\_PROGRESS|SUCCESS|FAILED",

"metadata": { *// Tool-specific*

"jenkins\_build\_url": "...",

"artifactory\_path": "...",

"ansible\_hosts": ["server1", "server2"]

}

}

**4. Deployment Strategy (No Downtime)**

1. **Phase 1**: Deploy Kafka + Flink + Elasticsearch parallel to prod
2. **Phase 2**: Configure tools to **dual-write** events (existing flows + Kafka)
3. **Phase 3**: Validate dashboard with synthetic events
4. **Phase 4**: Cut over monitoring teams to new dashboard

**5. Example Dashboard Views**

| **View** | **Purpose** | **Components** |
| --- | --- | --- |
| **Release Radar** | Real-time position of all releases | Pipeline DAG with current stage highlights |
| **Bottleneck Analyzer** | Identify slow stages | Stage duration percentiles |
| **Blocker Alerts** | Active failures | Failed stages with ticket links |
| **Artifact Trace** | From commit to production | Version propagation across tools |

**6. Key Benefits**

* **Zero Production Risk**: Existing flows remain unchanged
* **Unified Tracking**: All tools report to single pipeline
* **Millisecond Latency**: Kafka + Flink enable real-time updates
* **Self-Service**: Teams can add their own tool events

**7. Tools You Already Have**

1. **Jenkins** → Logstash plugin (built-in)
2. **Artifactory** → Webhooks (native)
3. **Ansible Tower** → Callbacks (REST API)
4. **Digital.ai XLR** → Event export (native)

Would you like me to provide config snippets for any specific tool integration?

great, ok so as per you this is optimum solution .. imagine per day builds are in 10k or more

The server is busy. Please try again later.

great, ok so as per you this is optimum solution .. imagine per day builds are in 10k or more how will you diplay on board realtime and imagine volume of builds

The server is busy. Please try again later.

great, ok so as per you this is optimum solution .. imagine per day builds are in 10k or more how will you diplay on board realtime and imagine volume of builds

Here's how to handle **10K+ daily builds** with real-time visualization while maintaining performance and usability:

**1. Architecture for High-Volume Tracking**

Diagram

Code

Download

Visualization Layer

Processing Layer

Ingest Layer

10K+ Events/Day

Kafka

Stream Aggregator

Flink Stateful Jobs

Rollup Databases

Time-Series DB

Grafana

Custom Web UI

**2. Scaling Strategies**

**A. Data Ingestion (Kafka Optimization)**

| **Strategy** | **Implementation** | **Benefit** |
| --- | --- | --- |
| **Partitioning** | correlation\_id % 100 | Parallel processing |
| **Event Sizing** | Protobuf schema | 70% smaller than JSON |
| **Burst Handling** | Kafka tiered storage | Cost-effective retention |

yaml

Copy

Download

*# kafka-config.yml*

num.partitions: 100

log.segment.bytes: 1073741824 *# 1GB segments*

message.max.bytes: 10485760 *# 10MB max*

**B. Stream Processing (Flink Jobs)**

java

Copy

Download

*// Flink aggregation job (per-minute rollups)*

DataStream<Event> events = env.addSource(...);

events

.keyBy(event -> event.correlationId)

.window(TumblingEventTimeWindows.of(Time.minutes(1)))

.aggregate(new BuildStageAggregator())

.addSink(new ElasticsearchSink<>());

**C. Storage Tiering**

| **Data Type** | **Storage** | **Retention** | **Example** |
| --- | --- | --- | --- |
| **Real-time** | Elasticsearch | 7 days | Current builds |
| **Short-term** | ClickHouse | 30 days | Recent analytics |
| **Long-term** | S3 + Athena | 1+ year | Historical trends |

**3. Dashboard Techniques for High Volume**

**A. Hierarchical Views**

1. **Org-Level**

sql

Copy

Download

*-- Grafana query*

SELECT

COUNT(DISTINCT correlation\_id) AS active\_builds,

COUNT\_IF(status='FAILED') AS blocked

FROM pipeline\_events

WHERE timestamp >= NOW() - INTERVAL '1 hour'

* + **Visual**: Large counter tiles + sparklines

1. **Team-Level**

sql

Copy

Download

SELECT

team,

COUNT(\*) as builds,

AVG(duration) as avg\_time

FROM events

GROUP BY team

* + **Visual**: Heatmap by team/time

1. **Build-Level**

sql

Copy

Download

SELECT \* FROM events

WHERE correlation\_id = '$selected\_build'

ORDER BY timestamp

* + **Visual**: Gantt chart with tool transitions

**B. Smart Filtering**

groovy

Copy

Download

*// Grafana variables*

variables:

- name: env

query: SELECT DISTINCT environment FROM events

- name: priority

query: VALUES ('Critical'), ('Normal')

**C. Event Sampling (for UI)**

python

Copy

Download

*# Flink sampling job*

events

.filter(lambda e: e.status == 'FAILED' or

random.random() < 0.01) *# 1% sample*

.add\_sink(web\_socket\_sink)

**4. Performance Optimizations**

| **Layer** | **Technique** | **Impact** |
| --- | --- | --- |
| **Kafka** | Snappy compression | 60% bandwidth reduction |
| **Flink** | State backend: RocksDB | Handles 10TB+ state |
| **ES** | Index per day | Faster deletes |
| **Grafana** | Pre-aggregated dashboards | Sub-second load |

yaml

Copy

Download

*# elasticsearch.yml*

index.refresh\_interval: 30s *# Batch updates*

indices.query.bool.max\_clause\_count: 100000

**5. Real-Time Visualization Examples**

**A. Build Radar View**

python

Copy

Download

*# Web UI code (WebSocket stream)*

const socket = new WebSocket('wss://events/stream');

socket.onmessage = (e) => {

const build = JSON.parse(e.data);

updatePosition(build.correlation\_id, build.stage);

};

**B. Anomaly Detection**

sql

Copy

Download

*-- Find stuck builds*

SELECT correlation\_id

FROM events

WHERE status = 'IN\_PROGRESS'

AND NOW() - timestamp > INTERVAL '1 hour'

**C. Throughput Monitoring**

sql

Copy

Download

*-- Builds/hour*

SELECT

time\_bucket('1 hour', timestamp) AS hour,

COUNT(\*) AS builds

FROM events

GROUP BY hour

**6. Hardware Sizing Estimate**

| **Component** | **Nodes** | **Specs** | **For 10K builds/day** |
| --- | --- | --- | --- |
| **Kafka** | 3 | 16CPU, 64GB RAM, NVMe | 1TB/day ingest |
| **Flink** | 5 | 8CPU, 32GB RAM | 1000 events/sec |
| **Elastic** | 3 | 32CPU, 128GB RAM | 30-day retention |
| **Grafana** | 2 | 4CPU, 16GB RAM | 100 concurrent users |

**7. Cost Optimization**

1. **Cold Storage**: Move >30-day data to S3 ($0.023/GB)
2. **Spot Instances**: For Flink task managers
3. **Vertical Scaling**: Start small, monitor CPU/network

This architecture handles **10K+ builds/day** with:

* **<5s end-to-end latency** from event to dashboard
* **Sub-second query response** even with 100M+ events
* **Zero impact** on existing production pipelines

Would you like me to elaborate on any specific component's tuning?