Certainly, you can implement a multi-level approval and certification process to ensure that components are thoroughly reviewed and certified before being shared with the community. Here's a breakdown of how you can classify each step into different approval levels:

**Level 1 Approval: Submission and Initial Review**

1. **Component Submission:** Engineers submit their components with relevant documentation via the forum platform.
2. **Initial Review:** The tech committee conducts an initial review of the component to ensure it meets basic criteria like completeness, relevance, and adherence to submission guidelines.

**Level 2 Approval: Documentation and Quality Review**

1. **Documentation:** If the component passes the initial review, the submitter is required to provide comprehensive documentation including usage instructions, code examples, and diagrams.
2. **Quality Review:** The tech committee conducts a more detailed review of the component's documentation and code quality. They assess whether it adheres to best practices, is well-documented, and can be easily understood by other engineers.

**Level 3 Approval: Peer Review**

1. **Peer Review:** The component is made available for peer review to the broader engineering community. Other engineers can test the component, review its documentation, and provide feedback on its usability and quality.

**Level 4 Approval: Certification and Final Review**

1. **Certification:** If the component receives positive feedback during peer review, it is submitted for formal certification. The tech committee performs a final review to ensure that the component is of high quality, well-documented, and aligned with organizational standards.

**Level 5 Approval: Publication**

1. **Publication:** Once the component successfully completes the certification process, it is published on the forum for other engineers to access and use.

**Level 6 Approval: Ongoing Maintenance and Updates**

1. **Maintenance and Updates:** Engineers are encouraged to maintain and update their components over time. Each update goes through a simplified version of the approval process to ensure that changes are well-documented and do not introduce issues.

You can assign each approval level a specific set of criteria that need to be met before progressing to the next level. Additionally, you can set up milestones or checkpoints for each level, requiring certain tasks to be completed before moving forward. This helps ensure that the components are thoroughly reviewed, documented, and validated before being shared with the community.

Keep in mind that the above structure is just a suggestion, and you can tailor it to the specific needs and complexity of your organization and the components being shared. The key is to maintain a balance between thorough evaluation and avo

Start

|

v

[Level 1 Approval]

|

v

Is Component Submission Complete?

|

v

[Initial Review]

| |

v v

No Yes

| |

v v

[Rejection] --> End

| |

v v

v |

[Level 2 Approval]

| |

v v

Is Documentation Complete and High-Quality?

|

v

[Documentation and Quality Review]

| |

v v

No Yes

| |

v v

[Rejection] --> End

| |

v v

v |

[Level 3 Approval]

| |

v v

Is Peer Review Favorable?

|

v

[Peer Review]

| |

v v

No Yes

| |

v v

[Rejection] --> End

| |

v v

v |

[Level 4 Approval]

| |

v v

Is Component Certified?

|

v

[Certification and Final Review]

| |

v v

No Yes

| |

v v

[Rejection] --> End

| |

v v

v |

[Publication]

| |

v v

|

End

* Each box represents a step or approval level.
* Arrows indicate the flow of the process.
* Diamond-shaped boxes represent decision points (yes/no questions).
* If a component doesn't meet the criteria at any level, it gets rejected and the process ends.
* If the component successfully passes through all levels, it gets published.

**2nd Level:**

Certainly, let's map the described process onto a multi-level approval structure for better clarity. I'll break down the process into different approval levels:

**Level 1 Approval: Engineer Submits Problem**

1. **Engineer Submits Problem:** An engineer submits a problem along with a detailed description of the issue.

**Level 2 Approval: Initial Problem Review**

1. **Initial Problem Review:** The initial review team assesses whether the problem description is complete, well-defined, and meets the submission guidelines.

**Level 3 Approval: CCB Review**

1. **CCB Review:** The problem description that passed the initial review is submitted to the Change Control Board (CCB) for further evaluation.

**Level 4 Approval: CCB Evaluation**

1. **CCB Evaluation:** The CCB evaluates the problem description to determine its potential value to the organization. They assess whether the problem is common and useful.

**Level 5 Approval: Developer Assignment**

1. **Developer Assignment:** If the problem passes through the previous approval levels, developers are assigned to work on creating a solution for the identified problem.

Here's the flowchart with the approval levels integrated:

Start

|

v

[Level 1 Approval: Engineer Submits Problem]

|

v

Is Problem Description Complete?

|

v

[Level 2 Approval: Initial Problem Review]

| |

v v

No Yes

| |

v v

[Problem Rejected] --> End

| |

v v

v |

[Level 3 Approval: CCB Review]

| |

v v

|

[Level 4 Approval: CCB Evaluation]

| |

v v

Is Problem Approved as Common and Useful?

|

v

[Level 5 Approval: Developer Assignment]

| |

v v

|

End

* Each level represents a distinct stage of the process.
* Approval levels are integrated into the process stages to indicate where approvals occur.
* If a problem submission doesn't meet the criteria at any level, it gets rejected, and the process ends.
* If the problem submission successfully passes through all levels, developers are assigned to develop a solution.

If you're using AWS Lake Formation, you can centralize and simplify the management of data access and permissions for your data lakes, including S3 buckets. AWS Lake Formation provides a way to define and enforce fine-grained access controls for your data, making it easier to manage data access across multiple accounts and services. Here's how you can leverage AWS Lake Formation for cross-account access to your S3 buckets:

**1. Set Up AWS Lake Formation:**

1. **Create a Data Lake:** Set up a data lake using AWS Lake Formation in the producer account. This involves defining the data catalog, databases, and tables that will be accessible to other accounts.
2. **Create Lake Formation Database/Table:** Within the producer account, create the necessary databases and tables that correspond to the S3 buckets you want to share with consumer accounts.

**2. Share Data with Consumer Accounts:**

1. **Share Databases/Tables:** Use AWS Lake Formation to share the created databases and tables with the consumer accounts. This allows you to control who has access to specific data within your data lake.

**3. Configure Permissions for Consumer Accounts:**

For each consumer account and service, configure permissions and access:

**In Consumer Account 1 and Consumer Account 2:**

1. **Create Cross-Account IAM Roles:**
   * Create IAM roles for EMR, Glue, and Athena in both consumer accounts. These roles will assume permissions granted by AWS Lake Formation in the producer account.
2. **Attach Trust Policies to Cross-Account Roles:**
   * Attach trust policies to the IAM roles in the consumer accounts, allowing EMR, Glue, and Athena services to assume these roles.
3. **Create and Attach Access Policies:**
   * For each IAM role, attach policies that grant access to AWS Lake Formation resources, specifically the databases and tables shared from the producer account.

**For Amazon EMR:**

1. When creating the EMR cluster in both consumer accounts, specify the IAM role you created for EMR that allows access to AWS Lake Formation resources.

**For AWS Glue:**

1. When setting up Glue jobs or crawlers in both consumer accounts, specify the IAM role you created for Glue that allows access to AWS Lake Formation resources.

**For Amazon Athena:**

1. Configure Athena in both consumer accounts to use the IAM role you created for Athena that allows access to AWS Lake Formation resources.

**Copying S3 Files from Producer to Consumer Accounts:**

If you need to copy S3 files from the producer account to consumer accounts, you can leverage AWS DataSync or custom Lambda functions as discussed earlier.

**Summary:**

By using AWS Lake Formation, you can simplify the process of granting cross-account access to data lakes and associated resources. Lake Formation provides centralized access control and permissions management, making it easier to enforce fine-grained access policies and share data across multiple AWS accounts while maintaining security and control.

Certainly, I'll walk you through the steps to achieve this using AWS Lake Formation, along with sample IAM roles, Lake Formation grants using AWS CLI commands, and a Lambda function for copying S3 files. Note that this is a simplified example, and you'll need to adapt it to your specific use case and account setup.

**Step 1: Set Up AWS Lake Formation in Producer Account**

1. Create a Data Lake using the AWS Management Console or AWS CLI.
2. Define Databases and Tables within your data lake for the data you want to share.

**Step 2: Share Data with Consumer Accounts**

Use the following AWS CLI commands to share data with consumer accounts using Lake Formation:

1. Grant permissions for Consumer Account 1:

aws lakeformation grant-permissions --principal DataLakePrincipalIdentifier=<ARN of Consumer Account 1's IAM role> --resource Database=<DatabaseName> --permissions "ALTER" "CREATE\_TABLE" "DROP" "SELECT"

aws lakeformation grant-permissions --principal DataLakePrincipalIdentifier=<ARN of Consumer Account 2's IAM role> --resource Database=<DatabaseName> --permissions "ALTER" "CREATE\_TABLE" "DROP" "SELECT"

**Step 3: Configure IAM Roles for Consumer Accounts**

**In Consumer Account 1 and Consumer Account 2:**

1. Create an IAM role for EMR, Glue, and Athena services.
2. Attach trust policies to these roles.

Sample IAM trust policy for EMR role:

json

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": {

"Service": "elasticmapreduce.amazonaws.com"

},

"Action": "sts:AssumeRole"

}

]

}

**Step 4: Use AWS Services in Consumer Accounts**

1. For Amazon EMR:
   * When creating the EMR cluster in the consumer accounts, specify the IAM role you created for EMR.
2. For AWS Glue:
   * When setting up Glue jobs or crawlers in the consumer accounts, specify the IAM role you created for Glue.
3. For Amazon Athena:
   * Configure Athena in the consumer accounts to use the IAM role you created for Athena.

**Step 5: Copy S3 Files Using Lambda Function**

You can create a Lambda function to copy S3 files from the producer account to consumer accounts. Here's a high-level example of the Lambda function:

1. **Set Up Lake Formation Permissions:**

In both the source (producer) and destination (consumer) accounts, use AWS Lake Formation to manage permissions for accessing data.

* + In the source account:
    - Create a database and table that correspond to the S3 bucket you want to copy from.
    - Use Lake Formation to grant necessary permissions to the consumer accounts' IAM roles for the source data (database/table).
  + In the destination account:
    - Create a database and table that correspond to the S3 bucket you want to copy to.
    - Use Lake Formation to grant necessary permissions to the consumer accounts' IAM roles for the destination data (database/table).

1. **Configure IAM Roles:**
   * In the source account, ensure that your Lambda execution role has permissions to assume a role in the destination account. Attach the necessary trust policy to allow the Lambda role to assume roles in the destination account.
   * In the destination account, create an IAM role that grants access to the destination S3 bucket. Attach a trust policy that allows the role in the source account (Lambda execution role) to assume this role.
2. **Lambda Function Code:**

Update the Lambda function code to perform the following steps:

* + Assume the IAM role in the destination account using the **sts** client to obtain temporary credentials.
  + Use the obtained temporary credentials to interact with AWS services in the destination account.
  + Use the Lake Formation-controlled IAM role in the consumer account to access the source data.
  + Copy the S3 objects from the source S3 bucket (using Lake Formation-controlled permissions) to the destination S3 bucket (using assumed role permissions).

Here's a simplified example of the Lambda function code:

import boto3

def lambda\_handler(event, context):

source\_bucket = 'source-bucket'

destination\_bucket = 'destination-bucket'

sts\_client = boto3.client('sts')

assumed\_role = sts\_client.assume\_role(

RoleArn='arn:aws:iam::DESTINATION\_ACCOUNT\_ID:role/DestinationRole',

RoleSessionName='AssumeRoleSessionName'

)

# Use assumed\_role credentials to interact with destination AWS services

s3\_dest = boto3.client('s3', aws\_access\_key\_id=assumed\_role['Credentials']['AccessKeyId'],

aws\_secret\_access\_key=assumed\_role['Credentials']['SecretAccessKey'],

aws\_session\_token=assumed\_role['Credentials']['SessionToken'])

# Use source role in consumer account to access source data using Lake Formation-controlled permissions

s3\_source = boto3.client('s3')

# List objects in source bucket

objects = s3\_source.list\_objects(Bucket=source\_bucket)['Contents']

for obj in objects:

# Copy each object to destination bucket

copy\_source = {'Bucket': source\_bucket, 'Key': obj['Key']}

destination\_key = 'copied/' + obj['Key'] # Change destination path as needed

s3\_dest.copy\_object(CopySource=copy\_source, Bucket=destination\_bucket, Key=destination\_key)

return {

'statusCode': 200,

'body': 'Files copied successfully!'

}