Liferay DXP Multi-Level Workflow

5–7 Level Approval Workflow with Scripted Round-Robin Assignment

Liferay DXP Technical Notes

September 28, 2026

Abstract

This document explains how a multi-level approval workflow can be designed in Liferay DXP using Kaleo Workflow. The example uses seven approval levels and demonstrates how scripted assignments can be used to implement a round-robin assignment strategy. It also discusses workflow states, tasks, transitions, roles, Groovy scripts, persistence, concurrency, rejection paths, escalation, auditing, and production design considerations.

Contents

1 Introduction to Liferay Workflow **3**

2 Example Business Requirement **3**

3 Why Use Round-Robin Assignment? **4**

4 Workflow Building Blocks **4**
- 4.1 State . . . 4
- 4.2 Task . . . 5
- 4.3 Transition . . . 5
- 4.4 Assignment . . . 5
- 4.5 Condition . . . 5

5 Seven-Level Workflow Architecture **5**

6 Workflow with Rejection Paths **6**

7 Round-Robin Assignment Concept **7**

8 Why Scripted Assignment? **8**

9 Conceptual Groovy Round-Robin Script **8**

10 Avoid Hard-Coded Users **9**

11 Filtering Eligible Users **9**

12 Where Should the Counter Be Stored? **10**

13 Concurrency Problem **10**

14 Recommended Service Architecture **11**

15 Separate Round Robin Per Level **12**

16 Generic Assignment Function 12

17 Conceptual Kaleo Workflow Definition 12

18 Workflow Context 14

19 Context-Aware Round Robin 14

20 Example Seven-Level Design 15

21 Conditional Approval Levels 16

22 Fallback Strategy 16

23 Escalation 17

24 Notifications 17

25 Audit Trail 18

26 Round Robin vs Load Balancing 18

27 Weighted Round Robin 19

28 Transaction and Cluster Considerations 19

29 Security Considerations 19

30 Recommended Production Architecture 20

31 Pseudo Algorithm 21

32 Example End-to-End Execution 21

33 Failure Scenarios 22
- 33.1 No Users in Role . . . 22
- 33.2 Selected User Becomes Inactive . . . 22
- 33.3 Server Restart . . . 22
- 33.4 Multiple Liferay Nodes . . . 22
- 33.5 Concurrent Requests . . . 22
- 33.6 Workflow Script Error . . . 23

34 Testing Strategy 23

35 Interview Questions and Answers 23

36 Multiple Choice Questions 24

37 Quick Revision Table 26

38 Key Takeaway 26

39 Official Reference 27

1 Introduction to Liferay Workflow

Liferay DXP provides a workflow framework commonly known as **Kaleo Workflow**.

Workflow allows organizations to define controlled business processes around assets and records. Typical use cases include:

- Content publication approval
- Expense approval
- Purchase requests
- Employee onboarding
- Legal document review
- Contract approval
- Product approval
- Compliance review
- Customer onboarding
- Custom Liferay Object approval

A workflow describes:

Who performs what action, in what order, under what conditions

Core Idea

A Liferay workflow consists of states, tasks, transitions, assignments, conditions, notifications, actions, forks, joins, and other workflow elements that together define the lifecycle of an item.

2 Example Business Requirement

Assume an enterprise application requires a seven-level approval process.

The workflow is:

Submitter → Level 1 → Level 2 → Level 3 → Level 4 → Level 5 → Level 6 → Level 7 → Approved

Each level contains multiple eligible approvers.

Instead of assigning every request to the same person, requests should be distributed among eligible approvers using a **round-robin strategy**.

For example, if Level 1 contains:

$$U_1,\ U_2,\ U_3$$

then successive assignments should ideally be:

$$U_1 \to U_2 \to U_3 \to U_1 \to U_2 \to \cdots$$

3 Why Use Round-Robin Assignment?

Without an assignment strategy, one approver may receive disproportionately more work than others.

Round-robin assignment attempts to distribute tasks sequentially.

Suppose:

$$A = \{A_1, A_2, A_3, A_4\}$$

is the set of available approvers.

For workflow request number n, a basic assignment rule is:

$$i = n \bmod |A|$$

and:

$$\text{Assignee}(n) = A_i$$

For four approvers:

Request	Index	Assigned User
1	0	User A
2	1	User B
3	2	User C
4	3	User D
5	0	User A
6	1	User B

Important

A true enterprise round-robin implementation should not rely only on an in-memory script variable. The current assignment position normally needs durable storage if assignment order must survive restarts and remain correct across multiple application nodes.

4 Workflow Building Blocks

A Liferay workflow commonly uses the following elements.

4.1 State

A state represents a stage or status in the workflow.

Examples:

- Created
- Submitted
- Approved
- Rejected

4.2 Task

A task represents work that must be performed by a user.
Examples:

- Manager Approval
- Finance Approval
- Legal Review

4.3 Transition

A transition moves the workflow from one node to another.
Typical transitions include:

- Approve
- Reject
- Resubmit
- Escalate

4.4 Assignment

Assignment determines who receives responsibility for a task.
An assignment can conceptually target:

- User
- Role
- Organization Role
- Asset Creator
- Script-generated assignee

4.5 Condition

A condition determines the next workflow path using business logic.
For example:

$$\text{amount} > 1,000,000$$

could send the request to an additional executive approval level.

5 Seven-Level Workflow Architecture

Consider the following approval hierarchy.

Level	Task	Example Responsibility
1	Initial Review	Operational Reviewer
2	Supervisor Approval	Team Lead / Supervisor
3	Manager Approval	Department Manager
4	Finance Review	Finance Team

Level	Task	Example Responsibility
5	Compliance Review	Compliance Team
6	Director Approval	Business Director
7	Final Approval	Executive / Final Authority

The basic process is:

```
START
  |
  v
Level 1 Review
  |
  v
Level 2 Approval
  |
  v
Level 3 Approval
  |
  v
Level 4 Finance
  |
  v
Level 5 Compliance
  |
  v
Level 6 Director
  |
  v
Level 7 Final Approval
  |
  v
APPROVED
```

6 Workflow with Rejection Paths

A production workflow also needs rejection handling.

For example:

```
+----------------+
|    Submitted   |
+-------+--------+
        |
        v
    Level 1
        |
   Approve
        v
    Level 2
        |
   Approve
        v
    Level 3
```

```
                          |
                         ...
                          |
                          v
                      Level 7
                          |
                     Approve
                          v
                     APPROVED

Any Level
    |
    +------ Reject ------> REJECTED
```

Alternatively, rejection can return the item to the submitter:

```
Level N
   |
   | Reject
   v
Needs Revision
   |
   | Resubmit
   v
Level 1
```

7 Round-Robin Assignment Concept

Suppose Level 1 contains three users:

```
reviewer1
reviewer2
reviewer3
```

The assignment sequence should be:

```
Request 001 -> reviewer1
Request 002 -> reviewer2
Request 003 -> reviewer3
Request 004 -> reviewer1
Request 005 -> reviewer2
Request 006 -> reviewer3
```

Mathematically:

$$\text{nextIndex} = (\text{currentIndex} + 1) \bmod N$$

where:

$$N = \text{number of eligible approvers}$$

This ensures that after reaching the final user, assignment starts again from the first user.

8 Why Scripted Assignment?

Static assignment works well when every workflow task belongs to a fixed role.
For example:

`Finance Approval -> Finance Manager Role`

However, scripted assignment becomes useful when the assignee depends on runtime logic.
Examples include:

- Round-robin assignment
- Region-based assignment
- Department-based routing
- Amount-based assignment
- Customer category
- Product category
- Current workload
- Availability
- Shift
- Organization hierarchy

Static vs Dynamic Assignment

Static assignment answers:
"Which role owns this task?"
Scripted assignment can answer:
"Which specific eligible user should receive this particular workflow instance?"

9 Conceptual Groovy Round-Robin Script

The following example illustrates the core algorithm.

```
def approverUserIds = [
    20101L,
    20102L,
    20103L
]

def currentIndex = getCurrentRoundRobinIndex()

def selectedIndex =
    currentIndex % approverUserIds.size()

def selectedUserId =
    approverUserIds[selectedIndex]

setNextRoundRobinIndex(
    (selectedIndex + 1) % approverUserIds.size()
)
```

```
// Return or expose selectedUserId
// according to the assignment contract
// expected by the Liferay workflow definition.
```

The central algorithm is:

$$i_{\text{selected}} = i_{\text{current}} \bmod N$$

followed by:

$$i_{\text{next}} = (i_{\text{selected}} + 1) \bmod N$$

10 Avoid Hard-Coded Users

Hard-coding user IDs is usually unsuitable for production.

Instead of:

```
def users = [20101L, 20102L, 20103L]
```

eligible users should preferably be derived from a stable business concept such as a Liferay role.

For example:

```
def role = RoleLocalServiceUtil.getRole(
    companyId,
    "Level 1 Approver"
)

def users =
    UserLocalServiceUtil.getRoleUsers(
        role.getRoleId()
    )
```

Conceptually:

$$\text{Role} \rightarrow \text{Eligible Users} \rightarrow \text{Round Robin} \rightarrow \text{Selected User}$$

This makes administration significantly easier.

11 Filtering Eligible Users

A robust implementation may need to filter users.

For example:

```
def eligibleUsers = users.findAll { user ->

    user.isActive()

}
```

Additional enterprise filters could consider:

- Active/inactive status
- Organization membership
- Business unit
- Geographic region

- Leave status
- Shift
- Required permissions
- Approval limits

Thus the algorithm becomes:

$$\text{Role Members} \rightarrow \text{Eligibility Filter} \rightarrow \text{Round Robin} \rightarrow \text{Assignment}$$

12 Where Should the Counter Be Stored?

This is one of the most important architectural questions.

Suppose the script stores:

```
currentIndex = 2
```

only in memory.

After a server restart, that value may disappear.

In a cluster:

```
Node A -> counter = 3
Node B -> counter = 1
```

The nodes could then produce inconsistent assignments.

Therefore, persistent state should be considered.

Possible approaches include:

- Custom database table
- Liferay Object
- Dedicated OSGi service
- Counter/persistence service
- External coordination service

Production Recommendation

For a production round-robin workflow, keep assignment-state management outside the workflow script whenever the logic becomes complex.
A dedicated OSGi service can provide a cleaner architecture for persistence, transactions, locking, testing, monitoring, and reuse.

13 Concurrency Problem

Consider two workflow instances arriving simultaneously.

Both read:

$$\text{currentIndex} = 2$$

Both calculate:

$$2 \bmod 3 = 2$$

Therefore, both assign the same user.
This is a race condition.
The desired operation should conceptually be atomic:

$$\boxed{\text{Read} \rightarrow \text{Select} \rightarrow \text{Increment} \rightarrow \text{Persist}}$$

as one controlled operation.
Possible solutions include:

- Database locking
- Atomic database update
- Transactional service
- Distributed locking
- Optimistic locking with retry

14 Recommended Service Architecture

For complex applications, use:

```
Kaleo Workflow
      |
      v
Groovy Assignment Script
      |
      v
RoundRobinAssignmentService
      |
      +----> Find Eligible Users
      |
      +----> Read Assignment State
      |
      +----> Select Next User
      |
      +----> Update State
      |
      +----> Return Assignee
```

The script remains small:

```
def selectedUserId =
    roundRobinAssignmentService
        .getNextApprover(
            companyId,
            "LEVEL_1_APPROVER"
        )
```

The business logic lives in a testable Java/OSGi service.

15 Separate Round Robin Per Level

Each approval level should normally maintain its own assignment pool.

For example:

Level	Role	Counter
1	LEVEL_1_APPROVER	RR_L1
2	LEVEL_2_APPROVER	RR_L2
3	LEVEL_3_APPROVER	RR_L3
4	FINANCE_APPROVER	RR_L4
5	COMPLIANCE_APPROVER	RR_L5
6	DIRECTOR_APPROVER	RR_L6
7	FINAL_APPROVER	RR_L7

This prevents the activity at one approval level from affecting another level's rotation.

16 Generic Assignment Function

Instead of writing seven different algorithms, define one generic operation:

$$f(\text{companyId}, \text{roleName}, \text{queueKey}) \rightarrow \text{UserId}$$

For example:

```
long getNextApprover(
    long companyId,
    String roleName,
    String queueKey
)
```

Usage:

```
getNextApprover(
    companyId,
    "Level 1 Approver",
    "RR_LEVEL_1"
)

getNextApprover(
    companyId,
    "Finance Approver",
    "RR_LEVEL_4"
)

getNextApprover(
    companyId,
    "Final Approver",
    "RR_LEVEL_7"
)
```

This promotes reuse and maintainability.

17 Conceptual Kaleo Workflow Definition

The following is intentionally simplified to illustrate the structure rather than serve as a drop-in definition for every Liferay release.

```
<workflow-definition>

    <name>seven-level-round-robin</name>

    <state>
        <name>created</name>

        <initial>true</initial>

        <transitions>
            <transition>
                <name>submit</name>
                <target>level-1-review</target>
            </transition>
        </transitions>
    </state>

    <task>
        <name>level-1-review</name>

        <assignments>
            <!-- Scripted assignment logic -->
        </assignments>

        <transitions>

            <transition>
                <name>approve</name>
                <target>level-2-review</target>
            </transition>

            <transition>
                <name>reject</name>
                <target>rejected</target>
            </transition>

        </transitions>

    </task>

    <!-- Levels 2 through 7 -->

    <state>
        <name>approved</name>
    </state>

    <state>
        <name>rejected</name>
    </state>

</workflow-definition>
```

Version Awareness

Exact workflow-definition syntax, supported script configuration, and UI capabilities can differ by Liferay DXP release. Validate the XML/XSD and scripting features against the Liferay version used by the target environment.

18 Workflow Context

Workflow scripts can work with contextual information associated with the running workflow instance.

Useful values can include concepts such as:

- Company ID
- Group/Site ID
- User ID
- Entry class name
- Entry class primary key
- Workflow instance information
- Asset-related information

Conceptually:

```
def companyId = workflowContext.get("companyId")
def groupId = workflowContext.get("groupId")
def userId = workflowContext.get("userId")
```

These values can drive dynamic assignment.

19 Context-Aware Round Robin

A more advanced algorithm can maintain separate queues.

For example:

$$Q = (\text{Company}, \text{Site}, \text{Department}, \text{Level})$$

Therefore:

```
Delhi + Finance + Level 4
    -> Finance Queue A

Mumbai + Finance + Level 4
    -> Finance Queue B

Delhi + Compliance + Level 5
    -> Compliance Queue C
```

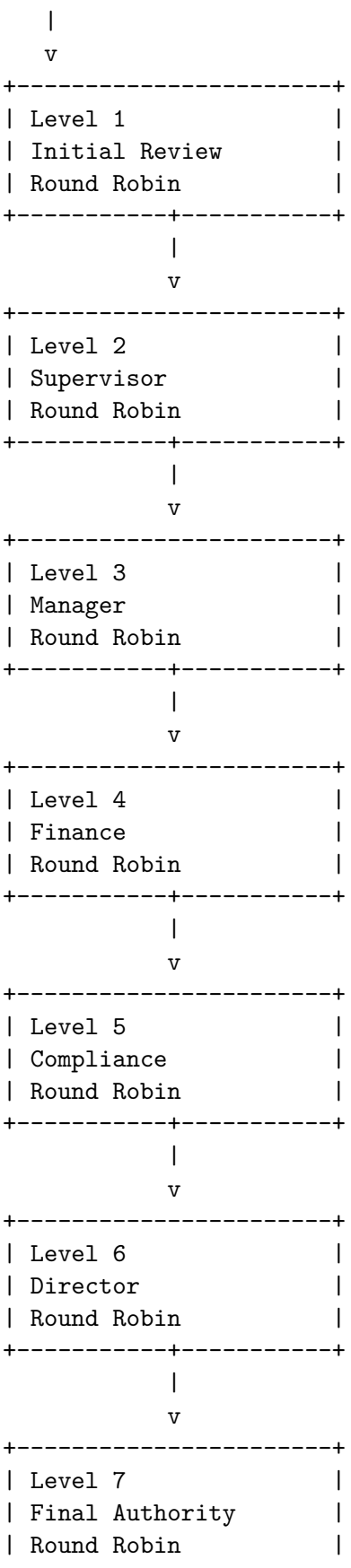
The round-robin key might therefore be:

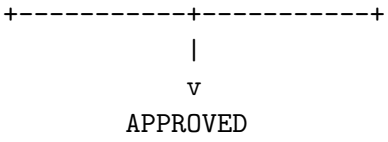
```
companyId + ":" +
groupId + ":" +
department + ":" +
approvalLevel
```

This allows the assignment engine to scale across multiple organizational units.

20 Example Seven-Level Design

A possible workflow is:





21 Conditional Approval Levels

Not every request necessarily needs all seven approvals.

For example:

$$\text{Amount} < 100,000$$

may require only three levels.

While:

$$100,000 \leq \text{Amount} < 1,000,000$$

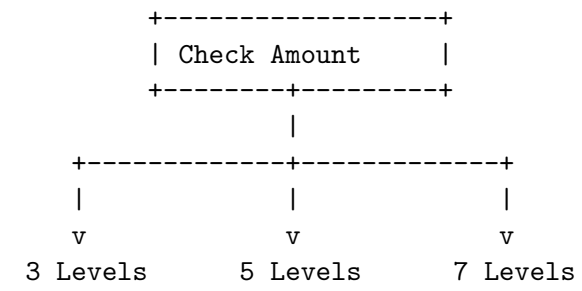
may require five levels.

And:

$$\text{Amount} \geq 1,000,000$$

may require all seven levels.

Conceptually:



This can be modeled with workflow conditions.

22 Fallback Strategy

What happens if a role contains no eligible users?

A robust workflow should define a fallback.

For example:

$$\text{Eligible Users} = \emptyset$$

Then:

$$\text{Assignment} \rightarrow \text{Fallback Role}$$

Possible fallback strategies include:

- Assign to Workflow Administrator
- Assign to Department Head
- Assign to a fallback role

- Pause workflow and notify administrators
- Escalate to the next authority

Never silently leave the task unassigned.

23 Escalation

Suppose an approver does not respond within a specified SLA.
Example:

$$T_{\text{waiting}} > 48 \text{ hours}$$

Then:

$$\text{Task} \rightarrow \text{Escalation}$$

Possible escalation actions include:

- Notify the current approver
- Notify the manager
- Reassign the task
- Move to an escalation role
- Increase task priority

Round robin can therefore be combined with SLA management.

24 Notifications

Notifications can be associated with workflow events.
For example:

```
Task Assigned
     |
     +--> Email Approver
     |
     +--> Portal Notification
     |
     +--> Optional external integration
```

Useful notification events include:

- Task assignment
- Approval
- Rejection
- Escalation
- Final approval
- Workflow completion

25 Audit Trail

A multi-level workflow should preserve an auditable history.

For each decision, useful information includes:

- Workflow instance
- Task
- Assignee
- Actual reviewer
- Assignment time
- Completion time
- Transition selected
- Comments
- Previous state
- Next state

Conceptually:

Level	Approver	Action	Time
1	User A	Approved	10:15
2	User D	Approved	11:32
3	User F	Approved	13:05
4	User H	Approved	14:45
5	User J	Approved	15:30

26 Round Robin vs Load Balancing

Round robin does not necessarily mean equal workload.

Suppose:

$$U_1 = 15 \text{ active tasks}$$

and:

$$U_2 = 2 \text{ active tasks}$$

A simple round-robin algorithm may still select U_1.

A workload-aware strategy could instead choose:

$$U^* = \arg\min_{u \in U} \text{ActiveTasks}(u)$$

This selects the eligible user with the lowest current workload.

Therefore:

Strategy	Basis	Benefit
Round Robin	Sequence	Simple/fair rotation
Least Loaded	Active tasks	Workload balancing
Weighted Round Robin	User weight	Capacity awareness
Skill Based	User capability	Expertise matching
Region Based	Geography	Local ownership

27 Weighted Round Robin

Suppose senior reviewers should receive twice as many requests as junior reviewers.

Weights could be:

$$w_A = 2, \quad w_B = 1, \quad w_C = 1$$

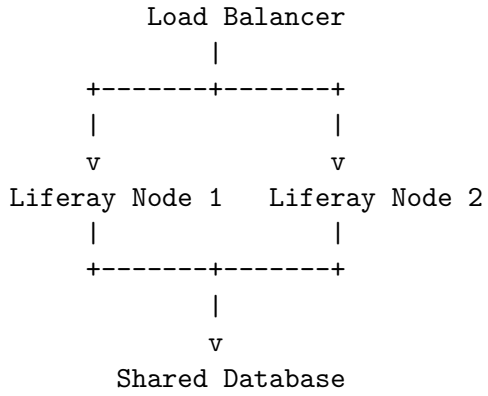
A conceptual sequence becomes:

$$A, A, B, C, A, A, B, C, \ldots$$

Weighted round robin can be useful when approvers have different capacities.

28 Transaction and Cluster Considerations

For a clustered Liferay environment:



The assignment state must remain consistent across nodes.

A purely local counter such as:

```
static int counter = 0;
```

is unsuitable.

Each JVM would maintain a different value.

The preferred design is:

Shared Persistent State + Atomic Update

29 Security Considerations

Scripted workflow logic should be treated as privileged server-side logic.

Important controls include:

- Restrict who can modify workflow definitions
- Avoid arbitrary user-controlled script execution
- Validate IDs and contextual inputs
- Use Liferay service APIs correctly
- Respect permissions
- Avoid exposing sensitive information in logs

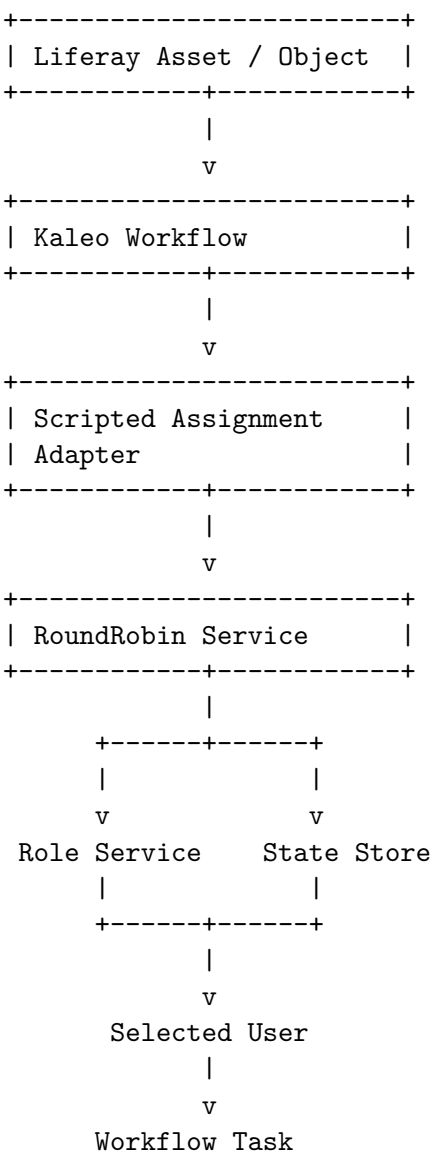
- Keep scripts small and maintainable

Security

Scripting capabilities can be restricted or disabled depending on Liferay configuration and product version. Production designs should not assume arbitrary Groovy execution is always available.

30 Recommended Production Architecture

For a seven-level enterprise workflow, a clean architecture is:



The responsibilities are separated as follows:

Kaleo Workflow Controls the business process.

Assignment Script Connects workflow execution to dynamic assignment logic.

Round-Robin Service Determines the next eligible approver.

Role Service Retrieves eligible users.

Persistent Store Maintains rotation state safely.

31 Pseudo Algorithm

A production-oriented algorithm can be expressed as:

```
function getNextApprover(level):

    role = getRoleForLevel(level)

    users = getEligibleUsers(role)

    if users is empty:
        return fallbackApprover(level)

    sort users deterministically

    lock roundRobinState(level)

    currentIndex = getCurrentIndex(level)

    selectedIndex =
        currentIndex mod users.size()

    selectedUser =
        users[selectedIndex]

    nextIndex =
        (selectedIndex + 1) mod users.size()

    saveCurrentIndex(level, nextIndex)

    unlock roundRobinState(level)

    return selectedUser
```

The deterministic sort is important because a round-robin algorithm requires a stable ordering of users.

32 Example End-to-End Execution

Assume a request named:

```
Purchase Request PR-10025
```

is submitted.
The execution could be:

```
PR-10025 submitted

Level 1
  Eligible: A, B, C
  Counter: 1
  Assigned: B
  Result: Approved

Level 2
  Eligible: D, E
  Counter: 0
```

```
  Assigned: D
  Result: Approved

Level 3
  Eligible: F, G, H
  Counter: 2
  Assigned: H
  Result: Approved

Level 4
  Eligible: Finance1, Finance2
  Assigned: Finance2
  Result: Approved

Level 5
  Eligible: Compliance1, Compliance2
  Assigned: Compliance1
  Result: Approved

Level 6
  Eligible: Director1, Director2
  Assigned: Director2
  Result: Approved

Level 7
  Eligible: Executive1, Executive2
  Assigned: Executive1
  Result: Approved

Final State: APPROVED
```

33 Failure Scenarios

A robust design should handle:

33.1 No Users in Role

Use a fallback or administrative queue.

33.2 Selected User Becomes Inactive

Re-evaluate eligibility before assignment.

33.3 Server Restart

Persistent counter/state prevents rotation loss.

33.4 Multiple Liferay Nodes

Use shared persistent state and concurrency controls.

33.5 Concurrent Requests

Use atomic state updates or locking.

33.6 Workflow Script Error

Log sufficient diagnostic information and ensure administrators can recover or reassign affected tasks.

34 Testing Strategy

Testing should include more than the normal approval path.

Test	Expected Result
Normal 7-level approval	Request reaches Approved state.
Reject at Level 3	Configured rejection path executes.
Round-robin rotation	Users receive tasks in expected sequence.
Inactive user	Inactive user is skipped.
Empty role	Fallback strategy executes.
Concurrent submissions	No unintended duplicate counter selection.
Server restart	Rotation state remains consistent.
Cluster test	All nodes observe consistent state.
Resubmission	Workflow returns through configured path.

35 Interview Questions and Answers

What is Kaleo Workflow?

Kaleo is Liferay's workflow framework for modeling approval and business processes using workflow nodes such as states, tasks, conditions, transitions, assignments, notifications, and actions.

What is scripted assignment?

Scripted assignment dynamically determines an assignee using runtime business logic instead of relying only on a statically configured user or role.

Why use round robin?

Round robin distributes assignments sequentially among a pool of eligible users instead of repeatedly assigning work to one user.

Why should the counter be persistent?

Persistence allows the assignment sequence to survive restarts and to be shared consistently by multiple Liferay nodes.

Why is concurrency important?

Two workflow instances may execute assignment simultaneously. Without an atomic update, both may read the same counter and assign the same user.

Round Robin vs Least Loaded

Round robin selects users sequentially. Least-loaded assignment selects the user with the smallest current workload.

36 Multiple Choice Questions

Question 1

Which Liferay framework is commonly used for workflow?

A. Service Builder

B. Kaleo

C. Clay

D. Elasticsearch

Answer: B

Question 2

What is the primary purpose of round-robin assignment?

A. Encrypt workflow data

B. Sequentially distribute tasks among eligible users

C. Delete completed tasks

D. Create Liferay sites

Answer: B

Question 3

For N users, which expression can be used for cyclic selection?

A. $i + N$

B. i/N

C. $i \bmod N$

D. $N - i^2$

Answer: C

Question 4

Why is an in-memory static counter problematic in a Liferay cluster?

A. It cannot store integers

B. Every node may have a different counter value

C. Groovy cannot read integers

D. Workflow tasks cannot use users

Answer: B

Question 5

Which approach is generally more suitable for complex production round-robin logic?

A. Large workflow scripts containing all business logic

B. Browser-side JavaScript

C. Dedicated service with persistent assignment state

D. JSP session variables

Answer: C

Question 6

What should happen if the eligible-user list is empty?

A. Divide by zero

B. Leave the task permanently lost

C. Execute a defined fallback strategy

D. Restart Liferay automatically

Answer: C

Question 7

What problem occurs if two requests read the same round-robin counter before either updates it?

A. Dead code

B. Race condition

C. Serialization error

D. Authentication failure

Answer: B

Question 8

Which strategy assigns work to the user with the fewest active tasks?

A. Pure round robin

B. Least-loaded assignment

C. Random assignment

D. Static role assignment

Answer: B

Question 9

Why should the eligible user list be deterministically ordered?

A. To reduce HTTP traffic

B. To provide a stable round-robin sequence

C. To encrypt user IDs

D. To create workflow states

Answer: B

Question 10

Which architecture is most appropriate for cluster-safe assignment?

A. JVM static variable

B. Browser localStorage

C. Shared persistent state with atomic updates

D. HTTP session only

Answer: C

37 Quick Revision Table

Concept	Meaning
Kaleo	Liferay workflow framework
Task	Human or system work step
Transition	Path from one workflow node to another
Assignment	Determines who owns a workflow task
Scripted Assignment	Dynamic runtime assignment logic
Round Robin	Sequential cyclic distribution
Persistent Counter	Stores rotation position across executions
Fallback	Alternative assignment when no normal assignee exists
Escalation	Action when a task exceeds its SLA or requires intervention
Audit Trail	Historical record of workflow decisions

38 Key Takeaway

A scalable Liferay multi-level approval workflow can be represented as:

$$\boxed{\text{Submission} \rightarrow \text{Workflow Level} \rightarrow \text{Eligible Role} \rightarrow \text{Round-Robin Engine} \rightarrow \text{Specific Approver}}$$

For seven levels:

$$L_1 \rightarrow L_2 \rightarrow L_3 \rightarrow L_4 \rightarrow L_5 \rightarrow L_6 \rightarrow L_7 \rightarrow \text{Approved}$$

The round-robin operation at each level can be modeled as:

$$\boxed{i_{\text{next}} = (i_{\text{current}} + 1) \bmod N}$$

For production deployments, however, the key architectural concern is not merely the modulo calculation. The system must also handle **persistent state, concurrency, clustering, inactive users, fallback assignment, auditing, rejection, resubmission, and administrative recovery**.

A strong design therefore keeps the Kaleo workflow focused on **process orchestration** while delegating complex assignment logic to a dedicated, testable service.

39 Official Reference

Liferay's official introduction to Workflow is available at:

https://learn.liferay.com/w/dxp/low-code/workflow/introduction-to-workflow

The exact workflow-definition schema, scripting availability, and supported workflow features should always be verified against the specific Liferay DXP version used by the target environment.