Liferay DXP 7.4 Clustering

High Availability, Load Balancing, Shared Resources, Search, Sessions, Caching, Failover, and Production Architecture

Liferay DXP Technical Notes

September 28, 2026

Abstract

This document explains Liferay DXP clustering for high availability. It covers multi-node Liferay architectures, load balancing, shared databases, document storage, Elasticsearch, cluster communication, session strategies, sticky sessions, cache considerations, deployment consistency, failover, monitoring, troubleshooting, and production design principles.

The objective is to understand not merely how to run multiple Liferay servers, but how to ensure that those servers behave as a coordinated high-availability platform.

Contents

1 Introduction **4**

2 What is High Availability? **4**

3 Why Cluster Liferay? **5**

4 Basic Two-Node Architecture **5**

5 Main Components **6**

6 Load Balancer **7**

7 Load Balancing Algorithms **7**
- 7.1 Round Robin . . . 7
- 7.2 Least Connections . . . 8
- 7.3 Weighted Distribution . . . 8

8 Health Checks **8**

9 Liferay Application Nodes **8**

10 Shared Database **9**

11 Database High Availability **9**

12 Connection Pool Considerations **10**

13 Document Library Storage **10**

14 Shared Document Store **10**

15 Search Infrastructure **11**

16 Search as an Independent Layer 11

17 Cluster Communication 12

18 Cluster Link 12

19 Caching in a Cluster 12

20 Do Not Use JVM Memory as Shared State 13

21 Sessions 13

22 Sticky Sessions 13

23 Session Replication 14

24 Stateless Design 14

25 Sticky Sessions vs Stateless Design 15

26 Consistent Configuration 15

27 Consistent OSGi Deployments 15

28 Client Extensions and Clustering 15

29 Microservices and HA 16

30 Failure Scenario: Liferay Node Failure 16

31 Failure Scenario: Database Failure 17

32 Failure Scenario: Search Failure 17

33 Failure Scenario: Document Store Failure 17

34 Single Points of Failure 18

35 Redundant Load Balancer 18

36 Production Architecture 19

37 Network Considerations 19

38 Reverse Proxy Configuration 20

39 Clustering and OpenID Connect 20

40 Clustering and Workflow 21

41 Clustering and Liferay Objects 21

42 Clustering and Headless APIs 21

43 Rolling Deployments 22

44 Blue-Green Deployment 22

45 Backup is Not High Availability **23**

46 High Availability vs Disaster Recovery **23**

47 RPO and RTO **23**
47.1 Recovery Point Objective 23
47.2 Recovery Time Objective 23

48 Monitoring **24**

49 Important JVM Metrics **24**

50 Database Monitoring **25**

51 Troubleshooting: Node Not Joining Cluster **25**

52 Troubleshooting: Random User Logout **25**

53 Troubleshooting: File Exists on One Node Only **25**

54 Troubleshooting: Different Behavior Per Request **26**

55 Troubleshooting: Stale Data **26**

56 Common Anti-Patterns **26**

57 Recommended Checklist **27**

58 Interview Questions **27**

59 Multiple Choice Questions **28**

60 Quick Revision Table **32**

61 Architecture Summary **32**

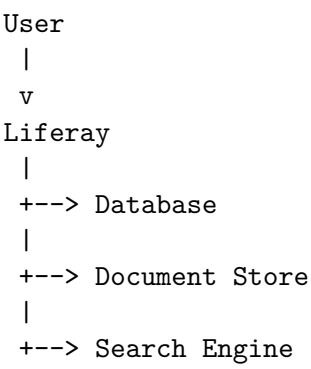
62 Key Takeaway **33**

63 Version Considerations **33**

64 Official Reference **33**

1 Introduction

A basic Liferay installation commonly contains one application node:



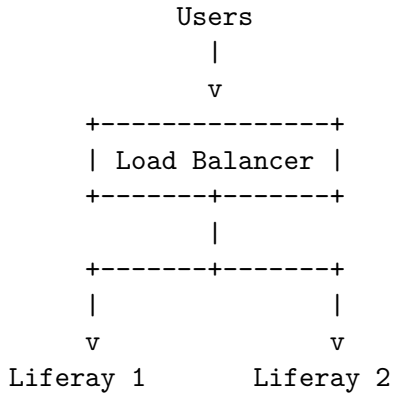
This architecture is straightforward, but the Liferay application server itself can become a single point of failure.

If the server fails:

$$\text{Liferay Node Failure} \Rightarrow \text{Portal Unavailable}$$

For business-critical applications, this may be unacceptable.

A cluster introduces multiple Liferay application nodes.



If one node becomes unavailable, traffic can be directed to another.

Core Idea

Liferay clustering is not simply "run two Liferay servers."
A production cluster requires multiple Liferay nodes to operate against appropriately shared or coordinated infrastructure, including database, document storage, search, configuration, deployment artifacts, and other stateful services.

2 What is High Availability?

High Availability (HA) means designing a system to remain available despite component failures.

Availability can be expressed as:

$$A = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}}$$

For example:

$$A = 99.9\%$$

still permits approximately:

$$8.76 \text{ hours/year}$$

of downtime.
Higher availability targets require progressively stronger redundancy and operational practices.

3 Why Cluster Liferay?

Common objectives include:

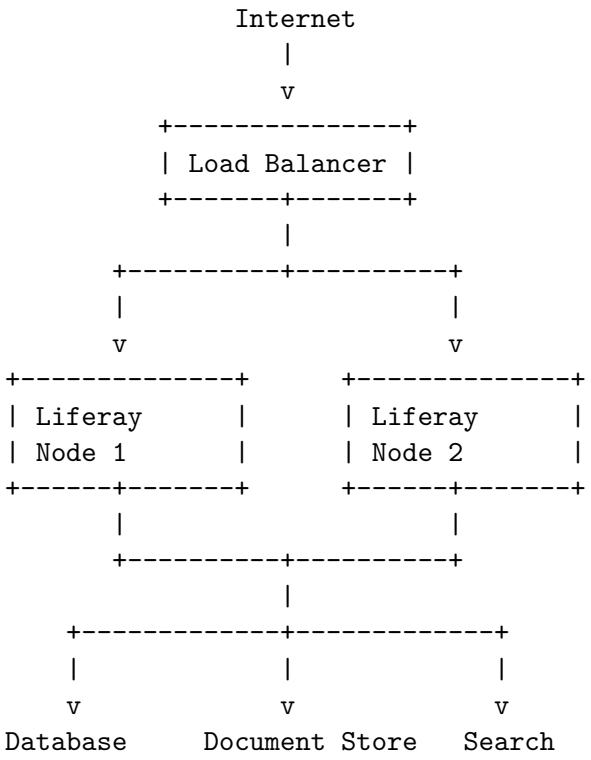
- High availability
- Fault tolerance
- Horizontal application scaling
- Maintenance with reduced downtime
- Distribution of user traffic
- Better resilience to node failures

A cluster can therefore support:

$$\boxed{\text{Availability} + \text{Resilience} + \text{Scalability}}$$

4 Basic Two-Node Architecture

A simplified two-node topology is:



The nodes should see the same logical application data.

5 Main Components

A Liferay cluster generally involves several layers:

1. Load Balancer
2. Multiple Liferay Application Nodes
3. Shared/Highly Available Database
4. Shared Document Storage
5. Search Infrastructure
6. Cluster Communication
7. Session Strategy
8. Cache Strategy
9. Consistent Deployments
10. Monitoring and Health Checks

A useful formula is:

$$\boxed{HA = LB + N + DB + DS + SE + CC + SS + MON}$$

where:

LB = Load Balancer

N = Liferay Nodes

DB = Database

DS = Document Storage

SE = Search Engine

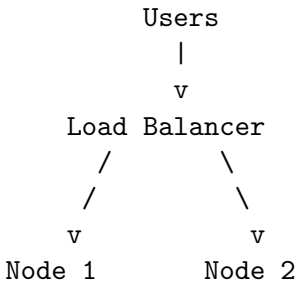
CC = Cluster Communication

SS = Session Strategy

MON = Monitoring

6 Load Balancer

The load balancer is the entry point for incoming requests.



Its responsibilities can include:

- Traffic distribution
- Health checks
- Removing failed nodes
- Reintroducing recovered nodes
- TLS termination
- Sticky-session handling

Common technologies include:

- NGINX
- HAProxy
- Apache HTTP Server
- Cloud load balancers
- Hardware load balancers

7 Load Balancing Algorithms

Common algorithms include:

7.1 Round Robin

$$R_1 \rightarrow N_1$$

$$R_2 \rightarrow N_2$$

$$R_3 \rightarrow N_1$$

$$R_4 \rightarrow N_2$$

7.2 Least Connections

Send new traffic to the node with the fewest active connections.

$$N^* = \arg\min_N Connections(N)$$

7.3 Weighted Distribution

If Node 1 is more powerful:

$$W_1 = 2$$

$$W_2 = 1$$

then Node 1 can receive approximately twice as much traffic.

8 Health Checks

A load balancer should not blindly send traffic to every configured node.

Instead:

```
Load Balancer
     |
     +--> Check Node 1 -> Healthy
     |
     +--> Check Node 2 -> Failed
                          |
                          v
                  Remove from Pool
```

When Node 2 recovers:

```
Node 2 Healthy
      |
      v
Return to Pool
```

High Availability Principle

A redundant node is useful only when the routing layer can detect failures and stop directing user requests to the unhealthy node.

9 Liferay Application Nodes

All application nodes should normally run compatible application versions and deployments.

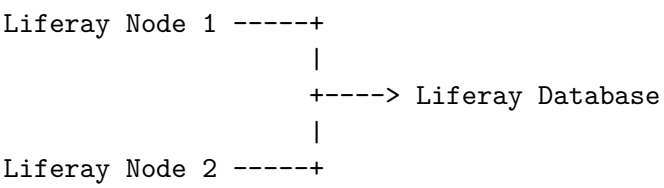
For example:

Node	Liferay Version	Application Version
Node 1	7.4.x	Release 25
Node 2	7.4.x	Release 25
Node 3	7.4.x	Release 25

Running inconsistent application artifacts can lead to unpredictable behavior.

10 Shared Database

All Liferay nodes normally access the same logical database.



This means:

$$DB_1 = DB_2 = DB_{shared}$$

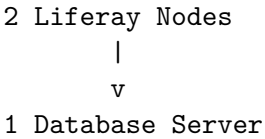
From the application's perspective, portal entities such as:

- Users
- Sites
- Roles
- Objects
- Web Content
- Workflow data
- Permissions

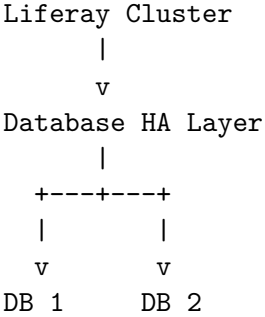
must represent the same shared portal state.

11 Database High Availability

A common mistake is:



The Liferay tier is redundant, but the database remains a single point of failure.
A stronger architecture is:



The exact implementation depends on the database platform.

12 Connection Pool Considerations

Suppose each Liferay node has a maximum database connection pool of:

$$P = 100$$

For:

$$N = 4$$

nodes, the potential application connection demand is:

$$P_{total} = N \times P$$

$$P_{total} = 4 \times 100 = 400$$

Therefore, scaling application nodes without considering database capacity can overload the database.

Capacity Planning

Horizontal application scaling does not automatically create end-to-end scalability.
The database, search engine, document store, network, and external systems must also support the resulting load.

13 Document Library Storage

Documents must also be available consistently to all Liferay nodes.

Bad design:

```
Node 1
 |
 +--> Local Documents A

Node 2
 |
 +--> Local Documents B
```

Suppose a user uploads a file through Node 1.
Later:

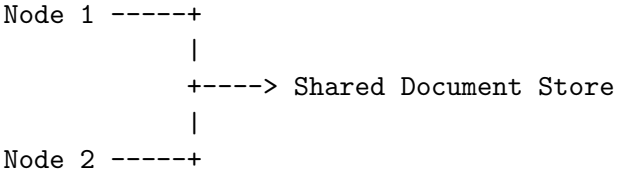
$$Request \rightarrow Node2$$

If Node 2 cannot access the same document repository, the file may appear unavailable.
Therefore:

$$\boxed{\text{All Nodes} \rightarrow \text{Same Logical Document Store}}$$

14 Shared Document Store

Conceptually:



The actual implementation depends on the supported document library store and deployment architecture.

The critical requirement is consistent access.

15 Search Infrastructure

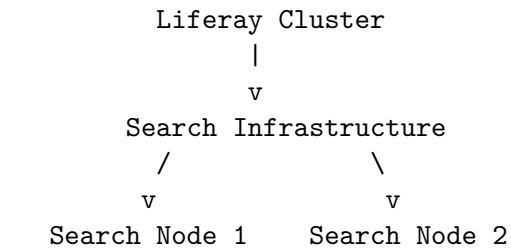
Liferay relies heavily on search infrastructure.

Typical search responsibilities include:

- Content indexing
- Search
- Asset discovery
- User search
- Object search
- Faceting

A production architecture should avoid treating an embedded or single-node search engine as a complete HA strategy.

Conceptually:



16 Search as an Independent Layer

It is useful to think of the system as:

$$\text{Portal Tier} \neq \text{Search Tier}$$

The Liferay application tier can scale independently from the search tier.

For example:

3 Liferay Nodes

does not imply:

3 Search Nodes

Search topology should be designed according to search workload, resilience requirements, and the supported search platform.

17 Cluster Communication

Liferay nodes may need to communicate information associated with clustered operation.

Conceptually:

```
Node 1 <-------> Node 2
  ^                ^
  |                |
  +-------> Node 3
```

Cluster communication can be relevant for areas such as distributed coordination and cache-related notifications.

18 Cluster Link

Liferay provides cluster-link capabilities for communication among cluster nodes where required by the platform.

Conceptually:

$$N_1 \leftrightarrow N_2 \leftrightarrow N_3$$

The exact configuration depends on:

- Liferay version
- Network topology
- Deployment environment
- Multicast/unicast availability
- Firewall rules

Do Not Copy Legacy Configuration Blindly

Cluster configuration has evolved across Liferay releases. Properties and architecture examples from older Liferay versions should not be copied into Liferay 7.4 without verifying the current documentation.

19 Caching in a Cluster

Caching improves performance but introduces consistency concerns.

Suppose:

$$Node_1.Cache(User101) = OldValue$$

while:

$$Node_2$$

updates User 101.

Node 1 must not indefinitely continue serving stale data.

Therefore, clustered applications require appropriate cache invalidation/coordination behavior.

Conceptually:

```
Node 2
 |
 | Update Entity
 v
Database
 |
 +--> Cache Invalidation / Coordination
            |
            v
          Node 1
```

20 Do Not Use JVM Memory as Shared State

Consider:

```
public static Map<String, Object> data =
    new HashMap<>();
```

In a cluster:

$$Node_1.Map \neq Node_2.Map$$

Each JVM has its own memory.
Therefore, JVM-local static variables should not be treated as shared cluster state.
This is especially important for:

- Counters
- User state
- Workflow assignment state
- Locks
- Business caches
- Temporary application state

21 Sessions

HTTP sessions require careful consideration in a cluster.
Suppose:

```
Request 1 -> Node 1
Request 2 -> Node 2
```

If the application depends on Node 1's local session state, Request 2 may not see that state.
There are several strategies.

22 Sticky Sessions

Sticky sessions attempt to keep a user's requests on the same node.

```
User A
 |
 +--> Node 1
 +--> Node 1
 +--> Node 1

User B
 |
 +--> Node 2
 +--> Node 2
 +--> Node 2
```

This is also called:

Session Affinity

Advantages:

- Simple
- Reduces cross-node session complexity
- Can improve performance

Limitation:
If Node 1 fails, its JVM-local session state may be lost unless another session persistence/replication strategy exists.

23 Session Replication

Application servers can support session replication.
Conceptually:

```
Node 1
 |
 | Session Update
 v
Node 2
```

This can improve failover behavior, but replication has a cost:

More Session Data ⇒ More Network/Serialization Overhead

Therefore, applications should avoid storing unnecessarily large objects in HTTP sessions.

24 Stateless Design

Where practical, stateless application design simplifies horizontal scaling.
Instead of:

State → Node Memory

prefer appropriate external/shared state mechanisms.
For example:

State → Database / Shared Service

Statelessness reduces dependency on a particular application node.

25 Sticky Sessions vs Stateless Design

	Sticky Session	**Stateless**
Node affinity	Yes	No
Failover complexity	Higher	Lower
Application design	Simpler initially	More scalable
Horizontal scaling	Good	Excellent
Node independence	Lower	Higher

26 Consistent Configuration

All nodes should operate with compatible Liferay configuration.

For example:

```
Node 1
portal-ext.properties
OSGi configuration
Client Extensions

Node 2
portal-ext.properties
OSGi configuration
Client Extensions
```

Configuration drift can cause difficult production failures.

A preferred model is:

Configuration as Code

with version-controlled deployment.

27 Consistent OSGi Deployments

Suppose:

```
Node 1 -> qap-service 1.4
Node 2 -> qap-service 1.3
```

The load balancer may send requests to either node.

The user may therefore observe different behavior depending on which node receives the request.

This should be avoided.

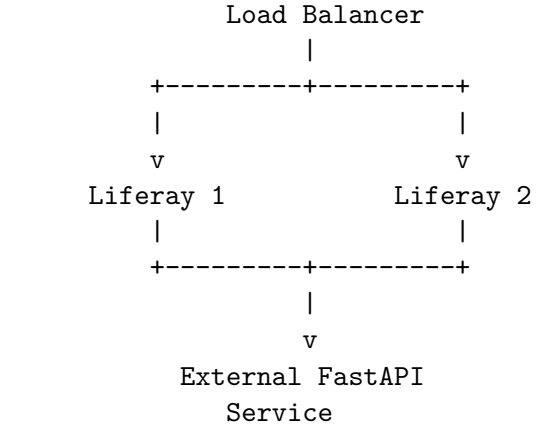
A cluster should normally satisfy:

$$AppVersion(N_1) = AppVersion(N_2) = \cdots = AppVersion(N_n)$$

28 Client Extensions and Clustering

Client Extensions can reduce the amount of custom code deployed inside each Liferay node.

For example:

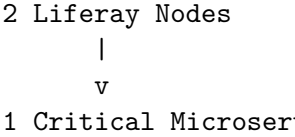


Instead of deploying specialized Python/AI/quantum logic to each Liferay node, it can run as an independently scalable service.

29 Microservices and HA

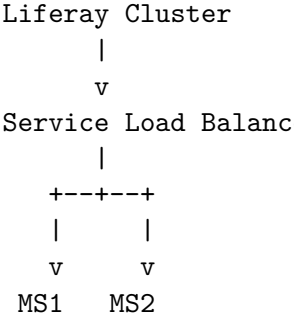
External microservices themselves must also be highly available if they are critical.

Bad architecture:



The microservice becomes the new single point of failure.

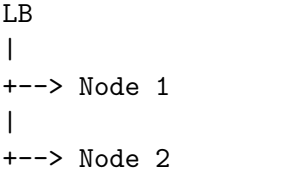
Better:



HA must be considered end-to-end.

30 Failure Scenario: Liferay Node Failure

Initial state:



Node 1 fails:

```
LB
|
+--> Node 1 [DOWN]
|
+--> Node 2 [UP]
```

The load balancer detects the failure and removes Node 1:

```
LB
|
+--> Node 2
```

Users continue through Node 2.

31 Failure Scenario: Database Failure

Suppose:

```
Node 1 ---+
          |
          +--> DB [DOWN]
          |
Node 2 ---+
```

Both nodes may become effectively unavailable.
This demonstrates:

$$\boxed{\text{Application Redundancy} \neq \text{System Redundancy}}$$

The database must have an appropriate HA strategy as well.

32 Failure Scenario: Search Failure

If the search service becomes unavailable, search-dependent functionality can fail or degrade even if Liferay application nodes remain alive.
Therefore, monitor:

- Search cluster health
- Index status
- Search latency
- Disk usage
- JVM memory

33 Failure Scenario: Document Store Failure

If the document repository is unavailable:

$$\text{Liferay UI} = \text{Available}$$

but:

$$\text{Document Operations} = \text{Unavailable}$$

High availability must therefore include storage services.

34 Single Points of Failure

For every architecture component, ask:

What happens if this component disappears?

Potential single points include:

- Load balancer
- Database
- Search engine
- Document storage
- DNS
- Network gateway
- Authentication provider
- Critical microservice
- Shared filesystem

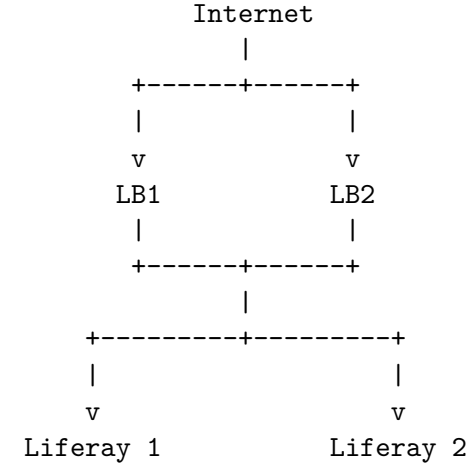
35 Redundant Load Balancer

Even the load balancer can become a single point of failure.

Instead of:

```
Internet
   |
   v
LB 1
   |
Liferay Cluster
```

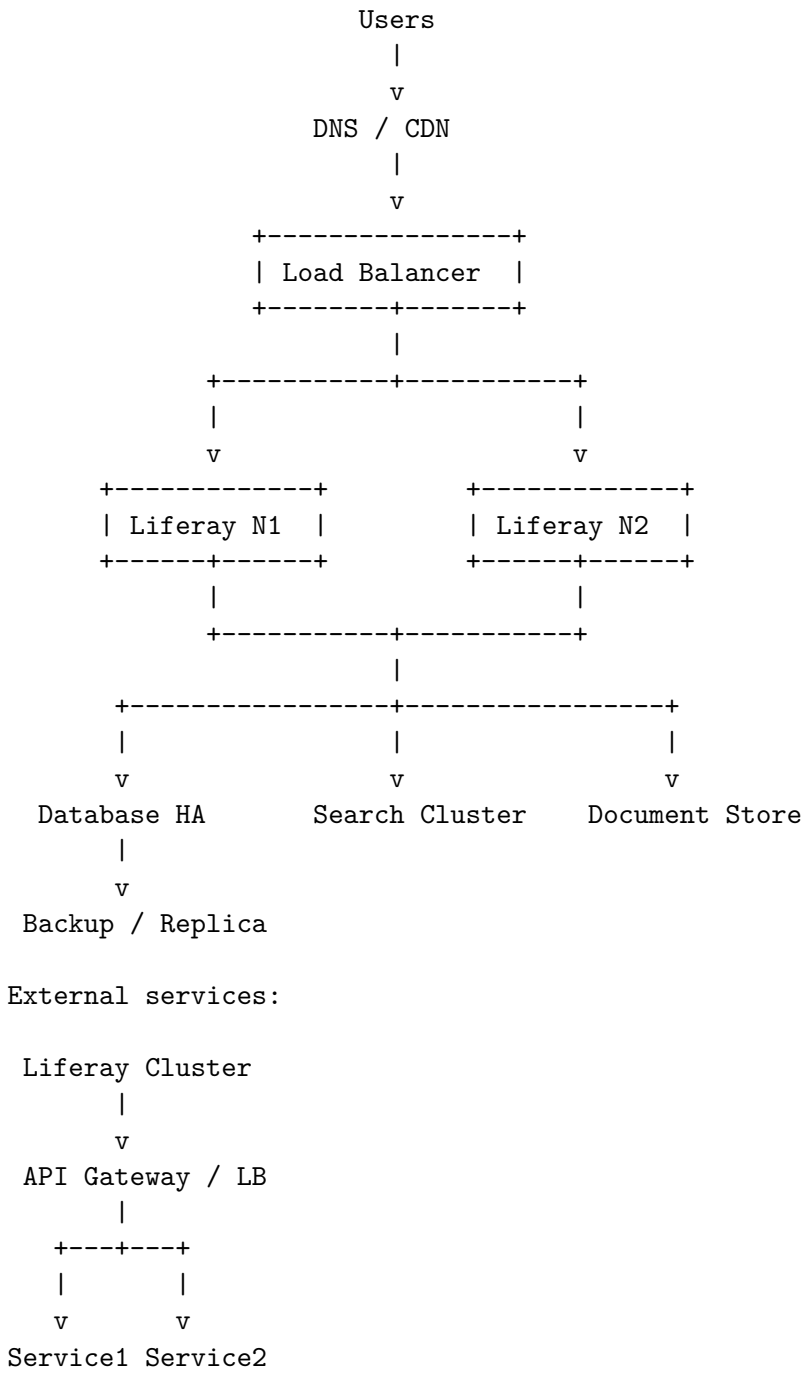
a stronger design is:



The exact mechanism can involve cloud-native or network-level high-availability services.

36 Production Architecture

A more complete architecture is:



37 Network Considerations

Cluster nodes need reliable network communication.

Check:

- Firewall rules
- Port availability
- DNS resolution

- Network latency
- Packet loss
- Load balancer timeouts
- Proxy headers
- TLS configuration

A cluster can fail even when every server is individually healthy if network communication is unreliable.

38 Reverse Proxy Configuration

When a load balancer terminates HTTPS:

```
Browser
 |
 | HTTPS
 v
Load Balancer
 |
 | HTTP / HTTPS
 v
Liferay
```

Liferay must correctly understand the original external protocol, host, and related proxy information.

Otherwise issues can occur with:

- Redirect URLs
- OAuth callbacks
- OpenID Connect
- Generated links
- Secure cookies

39 Clustering and OpenID Connect

Suppose the portal uses an external OpenID Provider.

Architecture:

```
                  OpenID Provider
                        ^
                        |
                        | OIDC
                        |
User --> Load Balancer --> Liferay Cluster
```

The identity provider should normally interact with the stable public portal URL:

```
https://portal.example.com
```

rather than individual node URLs.

Thus:

Identity Provider → Public Cluster Endpoint

40 Clustering and Workflow

Workflow processing may execute on any eligible application node.

Therefore, custom workflow logic should not rely on JVM-local state.

Bad example:

```
static int roundRobinCounter = 0;
```

because:

$$Counter_{Node1} \neq Counter_{Node2}$$

For round-robin workflow assignment, use shared persistent state and appropriate concurrency control.

41 Clustering and Liferay Objects

Liferay Object data is stored in the shared application data layer.

Therefore:

```
Request
   |
   +--> Node 1 --> Object Entry
   |
   +--> Node 2 --> Same Logical Object Data
```

External consumers should normally call the public load-balanced Liferay endpoint:

```
https://portal.example.com/o/c/orders/
```

rather than:

```
http://node1:8080/o/c/orders/
```

This preserves failover and load balancing.

42 Clustering and Headless APIs

Headless API architecture:

```
React / FastAPI / Mobile
          |
          | HTTPS
          v
     Load Balancer
       /       \
      v         v
 Liferay 1   Liferay 2
      \         /
       \       /
      Shared Data
```

Clients should remain unaware of which application node handles each request.

43 Rolling Deployments

A cluster can support reduced-downtime deployment patterns.

Conceptually:

1. Remove Node 1 from the load balancer.
2. Deploy/test Node 1.
3. Return Node 1 to the pool.
4. Remove Node 2.
5. Deploy/test Node 2.
6. Return Node 2.

Architecture:

```
Step 1:
LB -> Node 2
     Node 1 upgrading

Step 2:
LB -> Node 1
     Node 2 upgrading

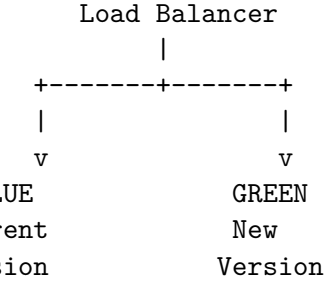
Step 3:
LB -> Node 1 + Node 2
```

Rolling Upgrade Warning

Do not assume every Liferay upgrade or application/database change is safe for rolling deployment.
Database schema changes and incompatible application versions can make simultaneous mixed-version operation unsafe.

44 Blue-Green Deployment

Another strategy is blue-green deployment.



After validating Green:

$$Traffic : Blue \rightarrow Green$$

This can simplify rollback for compatible deployment scenarios.

45 Backup is Not High Availability

A backup provides recovery after failure.

High availability aims to continue service during failure.

Therefore:

$$\boxed{Backup \neq HA}$$

but both are necessary.

A good architecture needs:

$$\boxed{HA + Backup + Disaster\ Recovery}$$

46 High Availability vs Disaster Recovery

High availability usually protects against local component failures.

Disaster Recovery (DR) addresses larger events such as:

- Data center failure
- Region outage
- Major corruption
- Catastrophic infrastructure loss

Conceptually:

```
Primary Region
    |
    | Replication / Backup
    v
DR Region
```

47 RPO and RTO

Two important DR metrics are:

47.1 Recovery Point Objective

RPO asks:

$$\boxed{\text{How much data can we afford to lose?}}$$

Example:

$$RPO = 5 \text{ minutes}$$

47.2 Recovery Time Objective

RTO asks:

$$\boxed{\text{How long can recovery take?}}$$

Example:

$$RTO = 30 \text{ minutes}$$

48 Monitoring

Monitor the complete system rather than only Liferay.

Important metrics include:

- Node availability
- CPU
- JVM heap
- Garbage collection
- Thread pools
- Database connections
- Request latency
- HTTP error rate
- Search health
- Search latency
- Document storage availability
- Disk space
- Network latency

49 Important JVM Metrics

Monitor:

$$HeapUsed$$

$$HeapMax$$

$$GC_{pause}$$

$$ThreadCount$$

For example:

$$HeapUsage = \frac{HeapUsed}{HeapMax} \times 100$$

Sustained high heap utilization may indicate:

- Insufficient heap
- Memory leak
- Excessive caching
- Large sessions
- Application defects

50 Database Monitoring

Important database metrics include:

- Active connections
- Connection wait time
- Slow queries
- CPU
- Memory
- Disk I/O
- Lock contention
- Replication health

If the database is overloaded, adding more Liferay nodes can actually make the problem worse.

51 Troubleshooting: Node Not Joining Cluster

Check:

- Cluster configuration
- Network connectivity
- Firewall
- Node addressing
- Logs
- Liferay version consistency

52 Troubleshooting: Random User Logout

Possible causes include:

- Sticky-session misconfiguration
- Session replication issues
- Cookie configuration
- Proxy configuration
- Node restart

53 Troubleshooting: File Exists on One Node Only

Likely cause:

Local Document Storage

instead of a shared storage architecture.
Check the Document Library storage configuration.

54 Troubleshooting: Different Behavior Per Request

If behavior appears random:

```
Request 1 -> Correct
Request 2 -> Old behavior
Request 3 -> Correct
Request 4 -> Old behavior
```

suspect node inconsistency.
Check:

- Deployed modules
- Configuration
- Client Extensions
- Liferay versions
- Environment variables

55 Troubleshooting: Stale Data

Possible areas include:

- Cache invalidation
- Search indexing
- Custom JVM-local caches
- External caches
- Replication delay

Do not immediately assume the database itself contains stale data.

56 Common Anti-Patterns

Avoid:

1. Different application versions on different nodes.
2. Local-only document storage.
3. JVM static variables as shared state.
4. Hard-coding individual node URLs.
5. Single-node search for critical HA systems.
6. Single database without considering DB availability.
7. Unmonitored load balancers.
8. Large HTTP sessions.
9. Manual configuration drift.
10. Assuming two Liferay nodes automatically equal HA.

57 Recommended Checklist

Before production:

- ☐ Multiple Liferay nodes
- ☐ Load balancer health checks
- ☐ Stable public portal URL
- ☐ Shared/HA database
- ☐ Shared document storage
- ☐ Production search infrastructure
- ☐ Cluster communication verified
- ☐ Session strategy defined
- ☐ Identical application deployments
- ☐ Consistent configuration
- ☐ Monitoring configured
- ☐ Backup tested
- ☐ DR strategy documented
- ☐ Node failure tested
- ☐ Database failure considered
- ☐ Search failure considered

58 Interview Questions

What is Liferay clustering?

Liferay clustering is an architecture where multiple Liferay application nodes operate as part of the same logical platform, typically behind a load balancer and using coordinated/shared infrastructure.

Why is a load balancer required?

It distributes incoming requests among healthy Liferay nodes and can stop routing requests to failed nodes.

Should cluster nodes use the same database?

The nodes normally operate against the same logical Liferay database so that they see consistent portal data.

Why must document storage be shared?

A document uploaded through one node must also be accessible when another node handles the subsequent request.

What is a sticky session?

A sticky session, or session affinity, causes the load balancer to prefer the same backend node for requests belonging to a particular user/session.

Can static Java variables store cluster state?

No. Each Liferay node runs in its own JVM, so static variables are local to that JVM.

Does two Liferay nodes guarantee high availability?

No. The database, search engine, document store, load balancer, network, authentication systems, and critical integrations can still be single points of failure.

What is configuration drift?

Configuration drift occurs when nodes that should be equivalent have different application artifacts or configuration values.

59 Multiple Choice Questions

Question 1

What is the main purpose of Liferay clustering?

A. CSS compilation

B. High availability and scalability

C. Creating JSP files

D. Replacing the database

Answer: B

Question 2

What typically distributes traffic among Liferay nodes?

A. Elasticsearch

B. Load Balancer

C. Service Builder

D. Picklist

Answer: B

Question 3

What should cluster nodes normally share logically?

A. JVM heap

B. Liferay database

C. Java static variables

D. Browser cache

Answer: B

Question 4

Why should Document Library storage be shared?

A. To compile themes

B. So every node can access the same documents

C. To replace OAuth

D. To create users

Answer: B

Question 5

What is another name for a sticky session?

A. Session affinity

B. Object relationship

C. Search replica

D. Workflow transition

Answer: A

Question 6

What happens to a Java static variable in a two-node cluster?

A. It is automatically shared

B. Each JVM has its own copy

C. It moves into Elasticsearch

D. It becomes a database column

Answer: B

Question 7

If the only database fails, what is likely to happen?

A. Both Liferay nodes can continue normally

B. The application tier may become unavailable

C. CSS stops compiling only

D. Nothing happens

Answer: B

Question 8

Which component should detect failed Liferay nodes?

A. Load balancer health checking

B. JSP

C. Picklist

D. Object Layout

Answer: A

Question 9

What is configuration drift?

A. Nodes having inconsistent configuration

B. Database backup

C. OAuth token generation

D. Creating Objects

Answer: A

Question 10

Which design is generally preferable for horizontal scaling?

A. Heavy JVM-local state

B. Stateless application design where practical

C. Hard-coded node URLs

D. Local-only documents

Answer: B

Question 11

Which statement is correct?

A. Backup and HA mean exactly the same thing

B. HA eliminates the need for backup

C. Backup and HA solve different problems

D. Clustering eliminates DR requirements

Answer: C

Question 12

What does RTO describe?

A. Acceptable recovery duration

B. Number of Liferay Objects

C. OAuth token size

D. Number of search results

Answer: A

Question 13

What does RPO describe?

A. Acceptable data-loss window

B. JVM heap size

C. Number of load balancers

D. HTTP session size

Answer: A

Question 14

What should external Headless API consumers normally call?

A. An individual internal node URL

B. The stable load-balanced portal endpoint

C. The database directly

D. A random JVM port

Answer: B

Question 15

What is a major danger of adding many application nodes without capacity planning?

A. The database or other shared infrastructure may become overloaded

B. Java stops supporting HTTP

C. Objects cannot have fields

D. Liferay stops supporting users

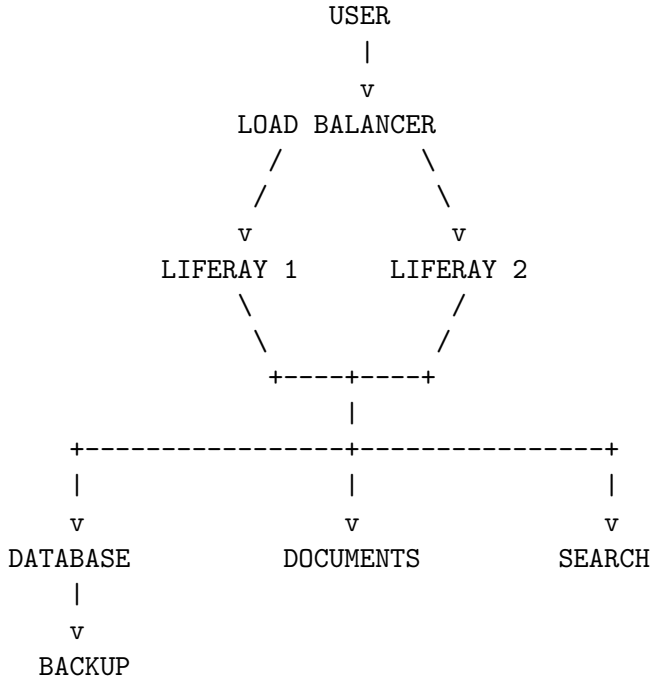
Answer: A

60 Quick Revision Table

Concept	Meaning
Cluster	Multiple coordinated Liferay nodes
High Availability	Design for continued service during failures
Load Balancer	Distributes requests across healthy nodes
Health Check	Determines whether a node should receive traffic
Sticky Session	Keeps a user/session associated with a node
Shared Database	Common logical Liferay data source
Shared Document Store	Documents accessible to all nodes
Search Cluster	External scalable/resilient search infrastructure
Cluster Communication	Communication/coordination between Liferay nodes
Configuration Drift	Differences between supposedly equivalent nodes
Stateless Design	Minimizes dependence on node-local state
Rolling Deployment	Upgrade nodes sequentially where compatible
Blue-Green	Switch traffic between old and new environments
RPO	Maximum acceptable data-loss interval
RTO	Maximum acceptable recovery duration

61 Architecture Summary

The simplest way to remember Liferay clustering is:



But each infrastructure component must itself be evaluated for availability.
Therefore:

$$\boxed{\text{Liferay HA} \neq \text{Multiple Liferay Nodes Only}}$$

Instead:

$$\boxed{\text{Liferay HA} = \text{Redundant App Tier} + \text{Reliable Shared State} + \text{Failure Detection} + \text{Redundant Dependencies}}$$

62 Key Takeaway

A robust Liferay clustering strategy should answer five questions:

1. Where does traffic go if a Liferay node fails?
2. Where is shared application state stored?
3. Can every node access the same documents and data?
4. What happens when the database, search engine, or another dependency fails?
5. Can a node be replaced without changing the application's externally visible identity?

The ideal application-node principle is:

$$\boxed{\text{Node}_1 \approx \text{Node}_2 \approx \cdots \approx \text{Node}_N}$$

in terms of version, configuration, and deployed application artifacts.
From the user's perspective:

$$\boxed{\text{Cluster} \rightarrow \text{One Logical Liferay Platform}}$$

The user should not need to know which individual Liferay node is serving the request.

63 Version Considerations

Liferay 7.4 spans multiple GA and quarterly releases, and clustering configuration has evolved over time.

Before applying any cluster configuration, verify:

- Exact Liferay DXP quarterly release or Portal GA
- Supported search engine/version
- Application server behavior
- Document Library store
- Cluster communication requirements
- Session strategy
- Deployment architecture
- Database platform

Avoid copying clustering properties from older Liferay versions without validating them against the documentation for the actual target release.

64 Official Reference

For current configuration details, consult Liferay's official documentation on clustering for high availability.

The production topology should always be adapted to the exact Liferay release, application server, database, search platform, storage architecture, and infrastructure environment being used.