

# REL10-BP04 Use bulkhead architectures to limit scope of impact

[PDF \(/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#rel\\_fault\\_isolation\\_use\\_bulkhead\)](#)

[RSS \(wellarchitected-reliability-pillar.rss\)](#)

Implement bulkhead architectures (also known as cell-based architectures) to restrict the effect of failure within a workload to a limited number of components.

**Desired outcome:** A cell-based architecture uses multiple isolated instances of a workload, where each instance is known as a cell. Each cell is independent, does not share state with other cells, and handles a subset of the overall workload requests. This reduces the potential impact of a failure, such as a bad software update, to an individual cell and the requests it is processing. If a workload uses 10 cells to service 100 requests, when a failure occurs, 90% of the overall requests would be unaffected by the failure.

## Common anti-patterns:

- Allowing cells to grow without bounds.
- Applying code updates or deployments to all cells at the same time.
- Sharing state or components between cells (with the exception of the router layer).
- Adding complex business or routing logic to the router layer.
- Not minimizing cross-cell interactions.

**Benefits of establishing this best practice:** With cell-based architectures, many common types of failure are contained within the cell itself, providing additional fault isolation. These fault boundaries can provide resilience against failure types that otherwise are hard to contain, such as unsuccessful code deployments or requests that are corrupted or invoke a specific failure mode (also known as *poison pill requests*).

---

## Implementation guidance

On a ship, bulkheads ensure that a hull breach is contained within one section of the hull. In complex systems, this pattern is often replicated to allow fault isolation. Fault isolated boundaries restrict the effect of a failure within a workload to a limited number of components. Components

outside of the boundary are unaffected by the failure. Using multiple fault isolated boundaries, you can limit the impact on your workload. On AWS, customers can use multiple Availability Zones and Regions to provide fault isolation, but the concept of fault isolation can be extended to your workload's architecture as well.

The overall workload is partitioned cells by a partition key. This key needs to align with the *grain* of the service, or the natural way that a service's workload can be subdivided with minimal cross-cell interactions. Examples of partition keys are customer ID, resource ID, or any other parameter easily accessible in most API calls. A cell routing layer distributes requests to individual cells based on the partition key and presents a single endpoint to clients.

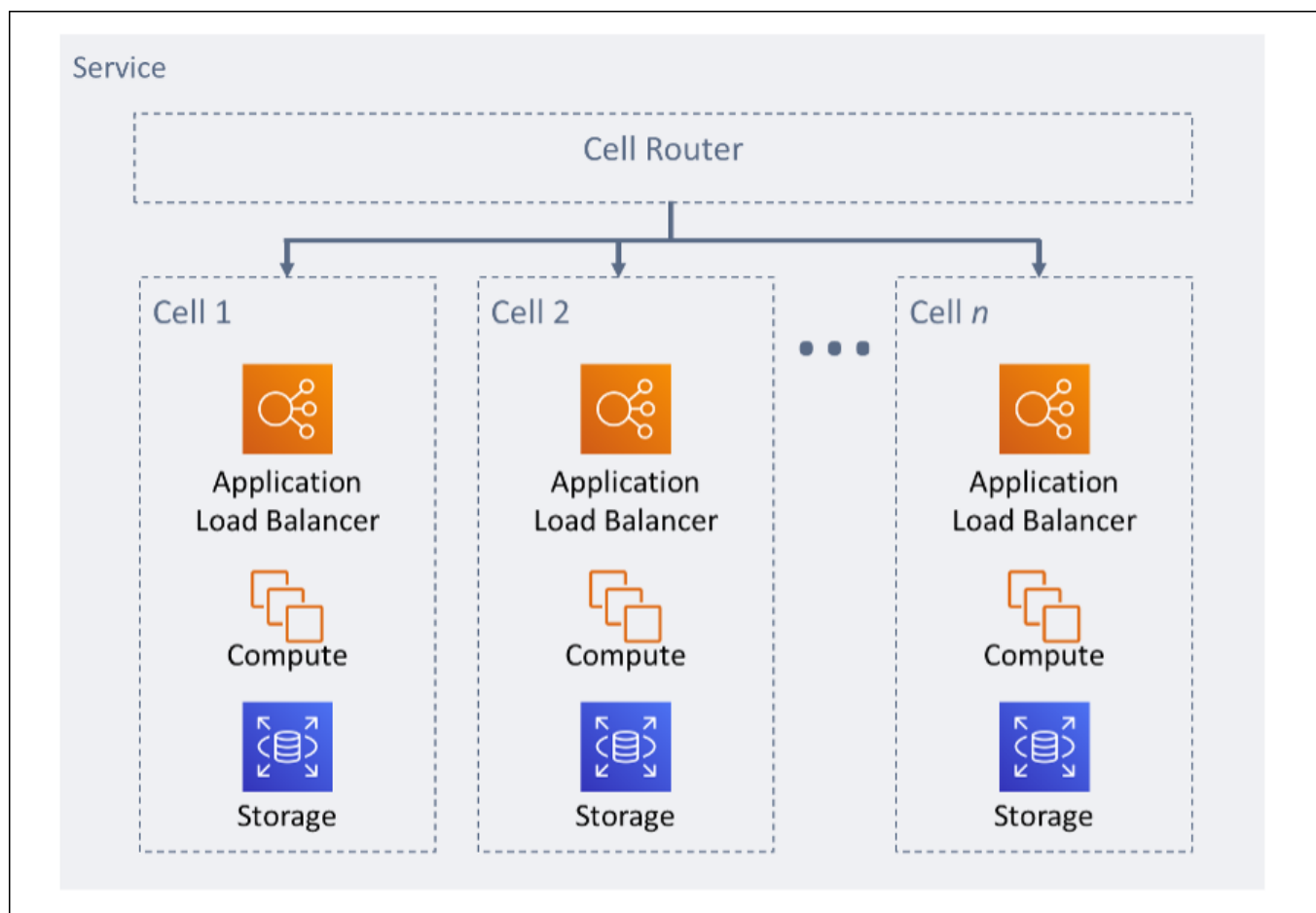


Figure 11: Cell-based architecture

## Implementation steps

When designing a cell-based architecture, there are several design considerations to consider:

1. **Partition key:** Special consideration should be taken while choosing the partition key.
  - It should align with the grain of the service, or the natural way that a service's workload can be subdivided with minimal cross-cell interactions. Examples are customer ID or

resource ID.

- The partition key must be available in all requests, either directly or in a way that could be easily inferred deterministically by other parameters.

2. **Persistent cell mapping:** Upstream services should only interact with a single cell for the lifecycle of their resources.

- Depending on the workload, a cell migration strategy may be needed to migrate data from one cell to another. A possible scenario when a cell migration may be needed is if a particular user or resource in your workload becomes too big and requires it to have a dedicated cell.
- Cells should not share state or components between cells.
- Consequently, cross-cell interactions should be avoided or kept to a minimum, as those interactions create dependencies between cells and therefore diminish the fault isolation improvements.

3. **Router layer:** The router layer is a shared component between cells, and therefore cannot follow the same compartmentalization strategy as with cells.

- It is recommended for the router layer to distribute requests to individual cells using a partition mapping algorithm in a computationally efficient manner, such as combining cryptographic hash functions and modular arithmetic to map partition keys to cells.
- To avoid multi-cell impacts, the routing layer must remain as simple and horizontally scalable as possible, which necessitates avoiding complex business logic within this layer. This has the added benefit of making it easy to understand its expected behavior at all times, allowing for thorough testability. As explained by Colm MacCárthaigh in [Reliability, constant work, and a good cup of coffee](http://aws.amazon.com/builders-library/reliability-and-constant-work/) (<http://aws.amazon.com/builders-library/reliability-and-constant-work/>), simple designs and constant work patterns produce reliable systems and reduce anti-fragility.

4. **Cell size:** Cells should have a maximum size and should not be allowed to grow beyond it.

- The maximum size should be identified by performing thorough testing, until breaking points are reached and safe operating margins are established. For more detail on how to implement testing practices, see [REL07-BP04 Load test your workload](#) ([./rel\\_adapt\\_to\\_changes\\_load\\_tested\\_adapt.html](#))
- The overall workload should grow by adding additional cells, allowing the workload to scale with increases in demand.

5. **Multi-AZ or Multi-Region strategies:** Multiple layers of resilience should be leveraged to protect against different failure domains.

- For resilience, you should use an approach that builds layers of defense. One layer protects against smaller, more common disruptions by building a highly available architecture using multiple AZs. Another layer of defense is meant to protect against rare

events like widespread natural disasters and Region-level disruptions. This second layer involves architecting your application to span multiple AWS Regions. Implementing a multi-Region strategy for your workload helps protect it against widespread natural disasters that affect a large geographic region of a country, or technical failures of Region-wide scope. Be aware that implementing a multi-Region architecture can be significantly complex, and is usually not required for most workloads. For more detail, see [REL10-BP02 Select the appropriate locations for your multi-location deployment](#) ([./rel\\_fault\\_isolation\\_select\\_location.html](#)) .

6. **Code deployment:** A staggered code deployment strategy should be preferred over deploying code changes to all cells at the same time.
- This will help minimize potential failure to multiple cells due to a bad deployment or human error. For more detail, see [Automating safe, hands-off deployment](#) (<http://aws.amazon.com/builders-library/automating-safe-hands-off-deployments/>) .

**Level of risk exposed if this best practice is not established:** High

---

## Resources

### Related best practices:




- [REL07-BP04 Load test your workload](#) ([./rel\\_adapt\\_to\\_changes\\_load\\_tested\\_adapt.html](#))
- [REL10-BP02 Select the appropriate locations for your multi-location deployment](#) ([./rel\\_fault\\_isolation\\_select\\_location.html](#))

### Related documents:


- [Reliability, constant work, and a good cup of coffee](#) (<http://aws.amazon.com/builders-library/reliability-and-constant-work/>)
- [AWS and Compartmentalization](#) (<http://aws.amazon.com/blogs/architecture/aws-and-compartmentalization/>)
- [Workload isolation using shuffle-sharding](#) (<http://aws.amazon.com/builders-library/workload-isolation-using-shuffle-sharding/>)
- [Automating safe, hands-off deployment](#) (<http://aws.amazon.com/builders-library/automating-safe-hands-off-deployments/>)

### Related videos:

- [AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small](#) (<https://www.youtube.com/watch?v=O8xLxNje30M>)

- [AWS re:Invent 2018: How AWS Minimizes the Blast Radius of Failures \(ARC338\)](https://youtu.be/swQbA4zub20)    
(<https://youtu.be/swQbA4zub20>)
- [Shuffle-sharding: AWS re:Invent 2019: Introducing The Amazon Builders' Library \(DOP328\)](https://youtu.be/sKRdemSirDM?t=1373)    
(<https://youtu.be/sKRdemSirDM?t=1373>)
- [AWS Summit ANZ 2021 - Everything fails, all the time: Designing for resilience](https://www.youtube.com/watch?v=wUzSeSfu1XA)    
(<https://www.youtube.com/watch?v=wUzSeSfu1XA>)

**Related examples:**

- [Well-Architected Lab - Fault isolation with shuffle sharding](https://wellarchitectedlabs.com/reliability/300_labs/300_fault_isolation_with_shuffle_sharding/)    
([https://wellarchitectedlabs.com/reliability/300\\_labs/300\\_fault\\_isolation\\_with\\_shuffle\\_sharding/](https://wellarchitectedlabs.com/reliability/300_labs/300_fault_isolation_with_shuffle_sharding/))

---

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.