Recovery and Prevention Plan: Exchange Online Outage, November 12-13, 2026

Deep Patel November 13, 2026

Incidents: MO1550821 (Microsoft 365 suite advisory) and EX1550543 (Exchange Online)

Why I wrote this

I watched this outage unfold on Microsoft's status posts, and I wanted to understand what was happening and what could make recovery faster. I have not seen Microsoft's internal systems, so everything here comes from public information. Where Microsoft has not said something, I mark it as unknown rather than guess.

How I collected the information

I followed the incident through Microsoft's @MSFT365Status posts and the incident records in the admin center. I also read reporting from the Associated Press, CNBC, CNN, Al Jazeera, BleepingComputer, and NBC News to check the timeline. I wrote down each statement from Microsoft with its timestamp, and I sorted each point into one of three groups:

- **Confirmed:** Microsoft said it.
- **Reported:** a news or secondary source said it, and I could not trace it to Microsoft.
- **Unknown:** nobody has said it publicly.

Microsoft's explanation changed during the incident. At 19:57 UTC it said it had not found a recent change that matched the start of the outage. Overnight it said it was reverting a network configuration change that did match the timing. I used the latest statement on each point and noted the earlier wording where it mattered.

What I could confirm

Confirmed

- Exchange Online mailbox access failed in the Americas from about 18:26 UTC on November 12, and then spread to other regions.
- Servers failed automated health checks and were taken out of service.
- A process on those servers that sets up secure connections became overwhelmed, and capacity dropped.
- The servers that were still running could not handle the traffic moved onto them.
- Servers that came back into service became unresponsive again once they got traffic.

- Microsoft paused deployments and configuration changes, redirected traffic, and started reverting a recent network change.
- Microsoft says there is no sign of malicious activity.

What I could not confirm

Unknown

- What triggered the first health-check failures.
- Whether the network change caused the outage or only lined up with its start.
- How many servers were affected, and whether they shared a configuration.
- Which resource the secure-connection process ran out of.
- Why the problem spread from the Americas to other regions.
- When full recovery will be complete.

What I think is happening

The removal rule creates a feedback loop. Health checks are supposed to protect the service by removing a broken server. But when a server is removed, its work goes to the others. If they are already busy, they get busier, and some of them fail their own health checks. Each removal makes the next one more likely.

The math. Say a pool of N servers runs at utilization u, which is the load divided by the total capacity. If k servers are removed, each survivor runs at u x N / (N - k). If u is 0.85 and 10 percent of the pool is removed, the survivors go to about 0.94. That is very close to the point where queues grow fast.

Queueing delay near saturation. For a single queue with random arrivals, the average wait is roughly u / (1 - u) times the service time. At 50 percent utilization the wait is about one service time. At 90 percent it is about nine, and at 95 percent it is about nineteen. A server can pass a simple liveness probe while its queues are already too long to be useful. That is why load shedding has to start well before the server reaches full utilization.

Connection setup and Little's Law. If setting up a connection takes longer when the server is busy, the number of connections being set up grows with load, and they all hold memory and CPU while they wait. A process like that will use more and more resources as traffic goes up. Microsoft's description of the secure-connection process fits this pattern, but that is a model, not something I can verify about their system.

Why recovery is slow. A server that comes back starts with empty caches and no open connections. Clients that were disconnected reconnect around the same time, so a lot of requests land on the server when it has the least capacity. Microsoft said returned servers became unresponsive under traffic, which fits

this. A server that is still warming up would show the same symptom, and the public information does not tell me which factor mattered most.

Failure domains. Servers that share a configuration, a health-check rule, or an upstream component will fail together if that shared thing breaks. The size of this outage suggests a wide failure domain, but Microsoft has not said how wide. The spread to other regions suggests either a shared dependency or redirected load, and the post-incident report should say which.

Redirects move load, they do not remove it. If the receiving region has less headroom than the sending one, a redirect can spread the overload. Microsoft redirected traffic early on and later spread connections across its wider network, but I cannot tell from public information whether that added to the overload.

Candidate causes of the resource use. For the resource use itself, I see four candidates that fit what Microsoft has said:

1. A memory leak, where memory grows under sustained traffic and is never released. A restart would clear it for a while, which would explain servers that recover and then fail again.
2. A resource that runs out under load, such as connections, threads, or file handles.
3. A lock or queue that backs up, so work piles up behind it.
4. A per-request cost that grows with load, such as work that scales with the number of open connections.

Profiling and heap data taken under load would separate these. I do not assume any of them without that evidence.

What I recommend

Recovery now

1. **Bring capacity back in groups.** Move to the next group only after the current one has stayed within latency and error limits under real traffic for a set period. This stops the first group from collapsing again while the second one comes up.
2. **Use slow start.** Give each returning server a small share of traffic and raise it step by step. If a step causes trouble, drop the server back one level instead of removing it.
3. **Rate-limit new connections on servers that are recovering.** A token bucket or similar limit on connection setup keeps reconnects from arriving all at once. Clients should retry with exponential backoff and jitter.
4. **Drain the mail backlog on its own budget**, at a fixed rate, so it does not compete with live mailbox access.
5. **Publish status for each region**, including how much capacity has come back. Customers need to know where recovery is still incomplete.

6. **Confirm the network rollback region by region**, and say whether it has held.
7. **Keep deployments and configuration changes frozen** until recovery is confirmed everywhere.

Next 30 days

1. **Cap automated removals.** A health check should be able to remove only a set share of a pool in one window. When the cap is reached, it should stop and escalate to an engineer. This limits the damage any one rule can do.
2. **Check remaining capacity before removing anything.** If the rest of the pool cannot carry current load with margin, degrade or quarantine the server instead.
3. **Separate overload from failure.** A busy server should get less work and stay in the pool. A broken server should be removed. Treating both the same turns a load problem into a removal event.
4. **Put a budget on automatic restarts**, per server and per pool per hour. When the budget runs out, escalate rather than restart again, because unlimited restarts can keep a pool stuck in a loop.
5. **Limit redirects** to regions with verified spare capacity, and cap how much can move in one step.
6. **Publish what triggered the first health-check failures** and what share of servers were affected, as far as that can be shared.

Longer term

1. **Separate liveness from readiness.** A process that answers a ping is alive, but it should only count as ready for full traffic once its caches, connections, and dependencies are in place.
2. **Pre-warm a server** before it joins the pool.
3. **Roll out network and health-check changes** to a small canary group first, with automatic rollback when error or capacity signals move.
4. **Run failure drills regularly.** Remove a server, a pool, and a region in a test environment, and confirm that the removal cap, redirect limit, and slow start all work.
5. **Size each region** so that losing its share of capacity can be absorbed without overloading the others.
6. **Profile the overloaded process under load**, and test each candidate above against diagnostic data before deciding on a fix.

Questions for the post-incident review

1. What triggered the first health-check failures, and was the network change the trigger?
2. How many servers were removed, and did one rule remove them together?

3. What resource did the secure-connection process run out of, and what drove its use under load?
4. Why did returned servers fail under traffic?
5. Did redirecting traffic add to the overload, and how much headroom did the receiving regions have?
6. Why did the impact spread across regions?

Limits of this work

I used only public information, so Microsoft may already have safeguards I cannot see. The queueing numbers come from simple models that show how this kind of failure can develop. They are not measurements from Microsoft's systems. The recommendations apply to any large service that uses health checks and load balancing, and how well they fit this incident depends on facts that have not been published yet.