MCP Just Dropped the Session. Here's What Changes for People Who Build Servers.

Unpublished spec sample by Alex Mercer. All protocol details are taken from the official MCP specification and its 2026-07-28 changelog at modelcontextprotocol.io.

For most of its short life, the Model Context Protocol worked a lot like a phone call. A client connected to a server, the two exchanged an `initialize` handshake, agreed on a protocol version and capabilities, and then kept that conversation open. Over HTTP, the server handed out an `Mcp-Session-Id` header, and everything that followed hung off that session.

The 2026-07-28 revision of the spec ends that model. MCP is now stateless: there is no handshake, no protocol-level session, and no session header. Every request carries what the server needs to know about the client, and the server answers it on its own terms. It's the biggest structural change to the protocol since it launched, and it touches nearly every part of a server implementation.

What actually changed

The changelog lists nine major changes. Three of them carry most of the weight.

The handshake is gone. The `initialize` request and the `notifications/initialized` reply have been removed. Instead, every request includes its protocol version and client capabilities in the `_meta` field, under keys like `io.modelcontextprotocol/protocolVersion` and `io.modelcontextprotocol/clientCapabilities`. Clients should identify themselves on each request, and servers should identify themselves in each result. If the versions don't match, the server returns an `UnsupportedProtocolVersionError`.

To keep up-front negotiation possible, the spec adds one new required method, `server/discover`. A client can call it first to learn which protocol versions, capabilities and identity a server supports, or skip it entirely and just start sending requests.

Sessions are gone. The Streamable HTTP transport no longer has `Mcp-Session-Id`, and list endpoints such as `tools/list`, `resources/list` and `prompts/list` are no longer allowed to vary by connection. If a server genuinely needs state across calls, the spec's answer is blunt: mint an explicit handle and pass it back and forth as an ordinary tool argument.

Servers no longer call clients. Earlier revisions let a server send its own requests to the client mid-operation, for example `elicitation/create` to ask the user a question or

`sampling/createMessage` to borrow the client's model. The new revision replaces that with a pattern called Multi Round-Trip Requests. When a server needs more information, it returns a result with `resultType: "input_required"` and a list of `inputRequests`. The client gathers the answers and retries the original request with `inputResponses` attached. Every result now carries a `resultType`, and clients must treat a missing value from an older server as `"complete"`.

Why make the protocol stateless

The changelog doesn't editorialize, but the direction is easy to read from the details. Stateless request-response is what the rest of the HTTP world already knows how to scale. A session pinned to one server process is awkward behind a load balancer, fragile across restarts, and hard to cache. A request that carries its own version, capabilities and identity can land on any replica.

Several smaller changes point the same way. `tools/list` and the other list and read results now must include `ttlMs` and `cacheScope` fields, so clients and shared intermediaries know how long a response stays fresh and whether it can be cached publicly. Servers should return tools in a deterministic order, which the spec notes helps client-side caching and LLM prompt-cache hit rates. Streamable HTTP POSTs must carry `Mcp-Method` and `Mcp-Name` headers, so a proxy can route a request without parsing the JSON body.

What it costs

Stateless designs move complexity rather than delete it, and a few trade-offs are written right into the spec.

Streams are now disposable. SSE resumability, the `Last-Event-ID` header and message redelivery have been removed from Streamable HTTP. If a response stream breaks, the in-flight request is lost and the client must re-issue it with a new request ID. Any tool that isn't safe to run twice now needs its own idempotency story.

Change notifications moved to a single channel. The old HTTP GET endpoint and `resources/subscribe` are replaced by `subscriptions/listen`, one long-lived POST response stream that a client opts into for specific notification types such as `toolsListChanged`. Progress and log messages tied to a specific request still flow on that request's own response stream.

Logging became per-request. `ping`, `logging/setLevel` and `notifications/roots/list_changed` are removed. A client now asks for logs by setting `io.modelcontextprotocol/logLevel` in a request's `_meta`, and servers must not emit `notifications/message` for requests that didn't ask.

What's deprecated, not deleted

Roots, Sampling and Logging are now formally deprecated. They still work during the deprecation window, but new implementations shouldn't add them. The spec's suggested replacements are practical: pass directories through tool parameters or configuration instead of Roots, call the LLM provider's API directly instead of Sampling, and log to `stderr` or OpenTelemetry instead of protocol-level Logging. The older HTTP+SSE transport, deprecated since 2025-03-26, is now officially on the removal path too, as is Dynamic Client Registration in favor of Client ID Metadata Documents.

The same revision also adopts a feature lifecycle policy with a minimum twelve-month deprecation window, so nothing on that list should vanish abruptly.

Experimental features become extensions

Tasks, the mechanism for long-running operations, moved out of the core protocol into an official extension, `io.modelcontextprotocol/tasks`. The redesign replaces the blocking `tasks/result` call with polling through `tasks/get`, adds `tasks/update` for sending input mid-flight, and drops `tasks/list`. Clients and servers declare extensions through a new `extensions` field in their capabilities, which gives the spec a clean place to grow without bloating the core.

What to do if you maintain a server

If your server is a simple stateless tool wrapper, the migration is mostly mechanical: implement `server/discover`, read version and capabilities from `_meta`, add `resultType`, `ttlMs` and `cacheScope` to results, and return tools in a stable order.

If your server leaned on sessions, the work is real. Anything you stored against a session ID has to become an explicit handle your tools hand back to the model. Anything that asked the user a question mid-call has to be rewritten as an `input_required` result plus a retry. And anything that depended on a stream surviving a network blip needs to tolerate being run again.

The spec includes a backward-compatibility section and a compatibility matrix for talking to older peers, so nothing forces a same-day cutover. But the direction is clear. MCP is betting that agents will talk to tools the way the rest of the web talks to services: one self-contained request at a time.