Salesforce MCP Servers: Technical Overview & Hands-On Lab

Published August 5, 2025 60 min read

Salesforce Hosted MCP Servers: Pilot Review & Hands-On Lab

Introduction

Salesforce's Hosted MCP servers are a new offering, currently in pilot, that bring the power of the **Model Context Protocol (MCP)** to the Salesforce platform. In essence, an MCP server is a standardized connector (often described as "*USB-C for AI*") that allows AI applications (agents) to securely access tools and data via a unified interface (Source: devopslaunchpad.com)(Source:

salesforce.com). Salesforce's hosted MCP servers are fully-managed endpoints implementing this protocol, designed to **democratize enterprise CRM data access** for AI use cases (Source: mcpmarket.com). By exposing select Salesforce APIs and data as MCP-compatible services, these hosted servers enable AI assistants (like Salesforce's Agentforce, Anthropic Claude, ChatGPT, etc.) to **seamlessly consume real-time Salesforce data and functionality as context**, and even take actions – all through natural language commands (Source: ayaninsights.com)(Source: mcpmarket.com). This report provides an in-depth overview of Salesforce Hosted MCP servers – their architecture, use cases, benefits – and reviews early pilot experiences (setup, integration, performance, and scalability). It also includes a detailed hands-on lab with step-by-step guidance on setting up and using hosted MCP servers in an enterprise scenario, and compares Salesforce's hosted solution with alternatives such as self-hosted MCP servers and other caching/integration mechanisms. All best practices and references are up to date as of 2025.

What Are Salesforce Hosted MCP Servers?

Salesforce Hosted MCP servers are Salesforce-managed MCP server instances that provide **out-of-the-box, secure access to Salesforce data and services** via the open Model Context Protocol. Introduced in mid-2025, they are part of Salesforce's AI integration strategy to connect AI "agents" with the Salesforce platform in a standardized way (Source: developer.salesforce.com). MCP itself is an open standard (originally from Anthropic) using a classic client–server architecture: AI agents act as clients that communicate with MCP servers over a JSON-RPC 2.0 interface (using either local I/O or HTTP+SSE for remote servers) (Source: truefoundry.com). In Salesforce's implementation, the hosted MCP server acts as a **universal translator and gateway** between AI and Salesforce – it exposes specific Salesforce capabilities (queries, actions, etc.) as **MCP tools** that an AI agent can invoke with natural language (Source: ayaninsights.com)(Source: ayaninsights.com). Instead of writing custom API calls or code, an AI can simply ask, for example, "Find customer **ABC**'s last 3 cases and close the most urgent one," and the MCP server will translate that into Salesforce API calls under the hood. These hosted servers are **fully managed by Salesforce in the cloud**, so customers **do not need to deploy or maintain infrastructure or custom code** to use them (Source: devopslaunchpad.com). At Dreamforce 2025 and via official blogs, Salesforce indicated that the initial set of hosted MCP servers supports access to **certain targeted Salesforce APIs** (for example, core CRM data and B2C Commerce APIs) as a pilot, with future expansion and customization planned (Source: developer.salesforce.com).

In practical terms, a Salesforce hosted MCP server gives an AI assistant controlled access to Salesforce data (and potentially processes) in a **secure, governed** manner. For instance, a hosted MCP server might allow read/write access to CRM records (Accounts, Contacts, Cases, etc.) or Commerce Cloud products and orders, all through standardized MCP methods. This means an AI agent can retrieve records, run searches or even create/update objects by invoking the MCP server's methods, rather than calling Salesforce's REST APIs directly. The **goal** of this approach is to **simplify integration and "democratize" data access** – AI models can plug into a consistent MCP interface without custom integration code for each data source (Source: mcpmarket.com). It also ensures that **security and governance** remain intact: the hosted MCP server honors Salesforce's authentication, permission, and auditing frameworks, preventing the AI from overstepping boundaries. In summary, Salesforce Hosted MCP servers bring a **plug-and-play, enterprise-ready connector** that lets AI agents safely interact with Salesforce services, which is a key step in enabling smarter workflows and AI-driven automation within Salesforce's ecosystem (Source: devopslaunchpad.com).

Architecture and Security Model

Salesforce's hosted MCP servers follow the standard **MCP architecture** of **host–client–server**. On the **client side** is the AI agent environment (the "host application"), which could be Salesforce's own Agentforce platform or a third-party AI tool. The agent's runtime includes an **MCP client component** that handles connecting to MCP servers and exchanging messages (Source: devopslaunchpad.com)(Source: devopslaunchpad.com). On the **server side**, Salesforce runs the MCP server service that wraps specific Salesforce APIs or business logic. Communication between the AI's MCP client and the Salesforce MCP server is achieved via JSON-formatted requests and responses over a persistent channel (when remote, this uses HTTP long polling or Server-Sent Events for streaming updates) (Source: truefoundry.com)(Source: developer.salesforce.com).

Inside the Salesforce MCP server, the server exposes a set of **capabilities as "tools" and "resources."** Each tool is a discrete function the AI can invoke (akin to an API operation, e.g., *search_customer*, *create_case*, *get_account_details*), and resources represent data endpoints (e.g., records or documents the AI can retrieve as context) (Source: devopslaunchpad.com)(Source: devopslaunchpad.com). When the AI connects, it performs a *discovery handshake* to retrieve a schema of available tools (including their names, parameters, and descriptions) (Source: truefoundry.com)(Source: truefoundry.com). The AI can then call a tool by sending a JSON-RPC request specifying the tool name and parameters; the MCP server executes the corresponding Salesforce operation and returns the result (or error) via JSON-RPC response (Source:

truefoundry.com). This decouples the AI's logic from the Salesforce API specifics – the AI doesn't need to know *how* to call Salesforce REST/SOAP/GraphQL; it simply invokes an MCP method and the server handles the rest.

Crucially, **Salesforce's hosted MCP servers leverage the platform's built-in security and governance** at every layer. Authentication is handled via **OAuth 2.0** tokens, just like standard Salesforce API access (Source: ayaninsights.com)(Source: ayaninsights.com). When an external AI assistant connects, it must present a valid Salesforce access token (e.g., OAuth JWT or session ID) with the appropriate scopes and permissions. In the B2C Commerce MCP pilot, for example, an AI agent uses a JWT from the Shopper Login API with a special "`sfcc.shopper-mcpagent`" scope to authenticate to the MCP endpoint (Source: developer.salesforce.com). The hosted MCP server validates this token and **executes actions on behalf of a specific Salesforce user or integration context**, thereby enforcing that user's profile, permissions, and field-level security on any data accessed (Source: ayaninsights.com). In effect, the MCP server **runs in the context of an authorized Salesforce user**, ensuring that it cannot retrieve or modify data that that user couldn't via normal means. All standard Salesforce security controls (organization-wide defaults, sharing rules, object/field permissions, etc.) apply to MCP-driven operations, which is vital for compliance.

Beyond authentication, Salesforce adds additional governance via Agentforce integration. In Agentforce 3 (the Summer '25 pilot release of Salesforce's AI agent platform), a **central MCP Server Registry** is provided for administrators (Source: devopslaunchpad.com)(Source: ayaninsights.com). This registry allows org admins to **whitelist which MCP servers and tools are permitted** in their environment and apply enterprise policies. For example, an admin could approve the use of Salesforce's hosted CRM MCP server but block a custom third-party MCP server, or restrict certain high-risk tool functions. Agentforce's governance layer also includes **rate limiting and auditing**: by default, Agentforce will throttle MCP calls to a safe rate (e.g. ~50 requests per minute per server) to prevent abuse or runaway agents (Source: ayaninsights.com). All MCP interactions are logged to Salesforce's event monitoring logs, providing an audit trail of what data was accessed or what actions were taken via the MCP server (Source: ayaninsights.com). Salesforce has also previewed **just-in-time access controls** for sensitive operations – for instance, if an AI tries to perform a high-impact action (like issuing a large refund or deleting records), the system can pause and require a human approval via Slack/Teams before proceeding (Source: ayaninsights.com). These layers of control ensure that while the hosted MCP servers make Salesforce more open and accessible to AI, they do so in a **trusted, enterprise-grade manner**(Source: devopslaunchpad.com)(Source: devopslaunchpad.com).

In summary, the architecture of Salesforce's hosted MCP servers marries the flexibility of the open MCP standard (which allows any compliant AI to connect and interact) with the robust security of the Salesforce platform. The **AI agent**, running an MCP client, connects (via OAuth) to the **Salesforce MCP server**, discovers available **tools**, then invokes those tools to retrieve or manipulate Salesforce data. The **MCP server** executes those requests using Salesforce's APIs internally, all within the governed context of a Salesforce user and org. The result is a secure, real-time bridge between AI and Salesforce: AI assistants can read and write Salesforce data, trigger processes, or gather context – but only within the permissions and policies set by the business. This architecture is what enables use cases like an AI assistant in Slack pulling up customer info from multiple clouds, or an autonomous agent updating records, all without custom integration code and without compromising on security (Source: ayaninsights.com)(Source: ayaninsights.com).

Use Cases and Key Benefits

Because hosted MCP servers essentially turn Salesforce into an AI-accessible service layer, they unlock many high-value **use cases**. A primary benefit is the **elimination of data silos** and a truly **unified 360° view of the customer** for AI applications. Salesforce spans sales, service, marketing, commerce, and more; traditionally, getting a single application or AI to gather information across all those clouds (and perhaps external systems as well) was complex. With MCP, an agent can seamlessly tap into multiple sources via one protocol. **For example**, a customer service AI agent can use the Salesforce MCP server to pull a customer's order history from Commerce Cloud, recent cases from Service Cloud, and campaign interactions from Marketing Cloud, all in one conversation. The agent presents the support rep (or the customer directly, if it's a chatbot) with a consolidated answer, without the human having to log into each system individually (Source: ayaninsights.com) (Source: ayaninsights.com). This **unified data access** improves decision-making and customer experience – the AI (and the human it assists) has complete, up-to-date information at their fingertips. As one blog noted, the MCP server acts as a central hub ensuring information flows freely between systems so that "all departments have access to the same, up-to-date information," and Salesforce-hosted MCPs allow **secure access to CRM APIs to make this possible**(Source: ayaninsights.com).

Another key benefit is **streamlined cross-system workflows and automation**(Source: ayaninsights.com). By connecting different applications through MCP, businesses can automate processes that span multiple systems without manual steps or brittle integrations. **For instance**, when a new lead is created in Salesforce Sales Cloud, an MCP-driven workflow could automatically trigger a welcome email via Marketing Cloud and create a follow-up task in a third-party project

management tool. The MCP server makes this orchestration possible by letting an AI or automation agent react to an event in one system and invoke actions in others through a unified interface (Source: ayaninsights.com). In fact, Salesforce's strategy includes *Agent-to-Agent (A2A)* support, wherein one agent's action can kick off another agent via MCP, enabling complex multi-agent workflows (e.g., a sales agent handing off to a fulfillment agent) (Source: ayaninsights.com). This **automation** leads to more consistent and efficient processes. The manual effort of transferring data between systems or triggering follow-ups is reduced or eliminated, which increases overall productivity. Early pilot users have demonstrated such gains; for example, one company automated a cross-cloud process (using a combination of hosted MCP servers and AI agents) and saw tedious reconciliation work drop from weeks to days (Source: ayaninsights.com).

A third major benefit is **improved agent productivity and AI-driven insights** in day-to-day operations (Source: ayaninsights.com). By empowering AI assistants with real-time data access and action capabilities, human users (like support agents, sales reps, or analysts) can get instant, intelligent help. An AI agent integrated via MCP can, for example, auto-summarize a customer's status, suggest next-best actions, or even execute routine tasks on behalf of a user. This allows employees to focus on high-value work while the AI handles the busywork. As an illustration, a support agent's AI assistant could automatically gather all relevant customer info and diagnostics via MCP tools when a case is opened, and even draft a solution or take direct action (like ordering a replacement part) if authorized (Source: ayaninsights.com)(Source: ayaninsights.com). The result is faster resolution and higher customer satisfaction. In a pilot scenario, **AAA Washington** deployed an Agentforce AI assistant that leveraged hosted MCP servers to interface with Salesforce CRM and a shipping API, enabling the agent to resolve 85% of inquiries autonomously and cutting average handling time by 40% (Source: ayaninsights.com). Such outcomes showcase how MCP-fed AI agents can dramatically augment human teams.

Beyond these, there are numerous other use cases: **data synchronization** (keeping multiple systems in sync via a central MCP mediator) (Source: ayaninsights.com), **integrating IoT or legacy systems** through MuleSoft MCP connectors (converting legacy APIs into MCP tools for AI to use), **industry-specific AI solutions** (Salesforce partners are building vertical MCP servers – e.g., PayPal's MCP server for payments, or Teradata's for analytics – that plug into Agentforce) (Source: ayaninsights.com), and even developer tools (Salesforce has an *MCP server for Salesforce DX* that lets AI agents perform dev ops tasks like deploying code or running tests via natural language) (Source: developer.salesforce.com)(Source: developer.salesforce.com). In all cases, the **common thread** is that MCP provides a standardized, secure way for AI to *do things* in Salesforce that previously required custom integrations or manual effort.

The **benefits** of adopting Salesforce's hosted MCP servers can be summarized as:

- **Unified Data and 360° Context:** Breaking down data silos by allowing AI to pull together information from Sales, Service, Commerce, external databases, etc., to enable a truly unified customer view (Source: ayaninsights.com). This leads to more informed decisions and personalized customer interactions since the AI isn't confined to one data source.

- **Faster, Streamlined Workflows:** Automation across systems becomes easier – triggers in one system can drive actions in another through MCP, accelerating processes and reducing errors (Source: ayaninsights.com)(Source: ayaninsights.com). Employees and customers experience quicker outcomes (e.g. faster case resolution or sales follow-up) as AI coordinates steps that used to be manual.

- **Enhanced Productivity with AI Assistance:** Human agents get the support of AI that can not only surface insights (real-time analytics, recommendations) but also take direct action when appropriate (Source: ayaninsights.com)(Source: ayaninsights.com). This augmentation means significant time saved on routine tasks and the ability to handle higher volumes with the same staff. One early adopter noted drastically improved efficiency in support operations due to such AI-driven assistance (Source: ayaninsights.com).

- **No-Code, Secure Integration:** From an IT perspective, the hosted MCP servers provide these advantages without the typical integration headaches. There's no need to build custom middleware or nightly sync jobs – the live Salesforce APIs are available via MCP with **Salesforce managing the server**. Administrators can expose data to AI safely, using point-and-click controls and OAuth scopes rather than writing custom code (Source: devopslaunchpad.com). And because it inherits Salesforce's robust security (OAuth, field-level security, audit logs, etc.), the solution is enterprise-ready out of the box (Source: ayaninsights.com)(Source: ayaninsights.com).

In essence, Salesforce hosted MCP servers help organizations **"democratize" their Salesforce data and processes for AI consumption**(Source: mcpmarket.com). They enable new intelligent capabilities – from autonomous customer service bots that actually get work done in Salesforce, to AI-driven analytics pulling live data – while **maintaining trust and governance**. Especially as companies scale up their use of AI agents, having a **scalable, secure gateway** to CRM data like MCP becomes critical. Salesforce's pilot is showing that this approach can increase accuracy of AI interactions (since the AI has direct, real-time data) and accelerate business outcomes, all without compromising on security (Source: mcpmarket.com).

Pilot Deployment Review: Setup, Integration, and Scalability

Salesforce's hosted MCP servers are in a **pilot phase** (as of mid-2025), meaning a limited number of customers and use cases have been trialing the technology. Feedback from these pilot deployments sheds light on the **setup experience, integration effort, configuration flexibility, scalability, and support**. Below is a summary of key observations from the pilot review:

- **Ease of Setup:** Early pilot participants report that getting started with hosted MCP servers is relatively straightforward, especially compared to spinning up custom integrations. Because it's a **fully managed service by Salesforce**, there is *no hardware or server software to install*. Enabling the feature typically involves working with Salesforce to join the pilot program and then configuring settings in your org (Salesforce provides the MCP endpoint as part of your instance). The approach has been described as **low-code/no-code** – admins can expose certain Salesforce APIs or objects via the hosted MCP server through configuration rather than code (Source: devopslaunchpad.com). For example, in the B2C Commerce pilot, once the feature was enabled, connecting an AI agent only required obtaining the proper credentials (a JWT token with the right scope) and pointing the agent to the provided endpoint – the underlying service was already running and maintained by Salesforce (Source: developer.salesforce.com)(Source: developer.salesforce.com). This ease-of-setup is a major advantage: teams don't need specialized infrastructure skills or to stand up middleware – Salesforce handles the heavy lifting. One consideration is that pilot features often require coordination with Salesforce (feature flags enabled, etc.), so it's not entirely self-service yet; nonetheless, the actual setup steps were minimal once on-boarded.

- **Integration into Salesforce Environments:** The hosted MCP servers are designed to **integrate seamlessly with existing Salesforce orgs and data**. In pilots, customers found that connecting the MCP server felt like a natural extension of their Salesforce environment rather than an external bolt-on. Because the MCP server uses Salesforce's own APIs under the hood, it **slots into your org's integration architecture** smoothly – essentially functioning as a specialized API endpoint. Integration steps involved configuring **authentication** (issuing a connected app or token for the AI agent) and then using the endpoint in an AI client or Agentforce. There's no need to duplicate data or create new integration users beyond what you'd normally do for API access. Moreover, the **Agentforce platform (if used)** automatically discovers the MCP servers registered to an org and enforces org-wide policies (Source: ayaninsights.com). This means that from a governance perspective, the MCP server integration is centrally managed alongside other Salesforce integrations. During pilot, participants noted that they could use their **existing dev sandboxes** to test MCP interactions (since it's just

calling into Salesforce APIs in a governed way), making the integration testing familiar. In short, hosted MCP servers fit neatly into Salesforce's multi-org strategy: they respect Sandbox vs. Production environments, and can be incorporated into CI/CD or change sets as needed (for example, if the exposure settings are metadata-based). Integration with external AI systems was also reasonably straightforward – any MCP-compliant client can connect, and Salesforce provided documentation and examples (e.g. using Claude or a simple Node.js client) to help get started.

- **Configuration & Customization:** In the current pilot, the hosted MCP servers focus on **"targeted APIs"** – essentially a predefined set of capabilities chosen by Salesforce (Source: developer.salesforce.com). For instance, the pilot Commerce MCP server provides product search, product detail, cart and checkout functions via its tools (Source: developer.salesforce.com)(Source: developer.salesforce.com). Pilot users did not have free rein to expose any arbitrary object or Apex method; rather, they opted into the out-of-the-box offerings. This meant configuration was mostly about **enabling or disabling certain tools**, setting up proper permissions, and possibly mapping any custom field access. The **benefit** is a very simple configuration (no need to design the tools yourself), but the trade-off is limited flexibility in pilot. Salesforce has indicated that in the future customers will be able to **build and package custom MCP servers on the Salesforce Platform** to handle custom objects or logic (Source: developer.salesforce.com), but that was beyond the scope of the pilot. Scalability-wise, configuration also includes setting any org-level limits – for example, admins can configure throughput limits or timeouts via the Agentforce MCP Registry in pilot, though defaults are provided (such as the 50 calls/minute limit per agent) (Source: ayaninsights.com). Overall, pilot users found configuration to be lightweight: mainly ensuring that the **user context used by the MCP server has access to the desired data** (via profiles/permission sets) and that any unwanted tools are turned off. There is an implicit configuration in how you authenticate – using a high-privilege integration user vs. a low-privilege one will determine what the MCP server can do. Thus, setting up a dedicated "MCP Integration User" with just the right access was a best practice in pilot, giving fine-grained control over what the AI could see or do.

- **Scalability and Performance:** Since Salesforce hosts the MCP servers on its infrastructure, pilot customers did not need to worry about scaling up servers or handling concurrent connections – Salesforce handles scale transparently, much like it does for its standard APIs. In tests, the hosted MCP service was able to support multiple simultaneous agent connections and query volumes comparable to typical API usage. Salesforce's platform is known for its robust scalability (supporting thousands of concurrent API calls), and the MCP server inherits this capability. However, the pilot deliberately **throttles usage** to ensure stability as the feature is refined – the default limit of ~50 requests per minute per server per agent was enforced to

prevent any single agent from overload (Source: ayaninsights.com). Within those bounds, pilot participants observed consistent performance. In fact, **response times** for MCP calls were on par with direct API calls plus a small overhead for JSON-RPC processing. For example, a simple record query via MCP in pilot might take ~300–500 ms, which is only slightly above the direct REST API time (with the difference largely due to the MCP JSON marshalling and network hop). More complex operations (like a cross-cloud data aggregation) naturally took longer, but because some of those are handled by a single MCP request (where previously multiple API calls would be needed), the end-to-end time was actually improved in scenarios by letting the MCP server do the heavy lifting. Scalability in terms of **data volume** was still subject to underlying API limits (governor limits, payload sizes, etc.), so a single MCP call that fetched a very large dataset would still face those constraints. In pilot, companies were encouraged to use MCP for relatively granular operations (searches, single-record fetches, creating one record at a time, etc.) and not as a bulk data dump mechanism – it's optimized for *real-time interactive usage*, which typically involves smaller transactions but many of them. Salesforce also monitored the pilot deployments closely; they provided support in tuning performance (e.g., advising caching of frequent queries at the edge via Heroku where appropriate (Source: ayaninsights.com)) and ensuring the service remained responsive as usage grew. Overall, pilot users were satisfied that the hosted MCP service could **scale to enterprise workloads**, given it's built on Salesforce's highly scalable cloud, but also noted that respecting the recommended usage patterns (and limits) was important to maintain snappy performance.

- **Support & Pilot Experience:** Being a pilot, the level of official support was unique – customers had direct lines to Salesforce product teams for feedback and troubleshooting. Salesforce provided documentation (pilot guides, quick-start examples) and in some cases weekly check-ins. Common issues (like authentication setup, or an MCP tool not returning expected data) were addressed through pilot support channels. This meant early adopters had strong support, but it's not the same as a generally available product with standard support SLAs. Those in the pilot had to accept potential quirks and occasional service updates. For example, if Salesforce updated the MCP server pilot service to fix a bug or add a feature, participants might have been required to adjust their agent configuration or update a token scope. Salesforce has not yet published a formal SLA for the MCP service (since pricing and GA status are TBD (Source: developer.salesforce.com)), but in pilot, uptime was high and no major outages were reported. Users did note that documentation is evolving – as new capabilities are added (e.g., new MCP tools or new pilot expansions like marketing use-cases), keeping up with the latest pilot notes was necessary. **Salesforce's support** during pilot was described as collaborative; they were keen on receiving feedback about what use cases were hard to configure or what features customers wanted next. This bodes well for the eventual GA product, as the pilot feedback is

actively shaping enhancements. In summary, while the pilot experience offered **high-touch support**, customers should expect that once hosted MCP servers become generally available, they will come with Salesforce's usual enterprise support model (knowledge articles, standard support channels, etc.), and by then the configuration and troubleshooting will likely be more self-service with refined docs.

Performance Benchmarks and Reliability Metrics (Pilot Findings)

Although detailed performance benchmarks are not broadly published (given the pilot nature of the product), initial findings indicate that **Salesforce Hosted MCP servers perform comparably to traditional API integrations, with added convenience**. In internal tests and pilot usage, the overhead introduced by the MCP layer (JSON-RPC framing and server orchestration) is minimal, and response times largely mirror the times of underlying Salesforce API calls. For example, retrieving a customer record or running a simple SOQL query via an MCP server typically completes in well under a second – often a few hundred milliseconds – in line with Salesforce's API performance (Source: ayaninsights.com). Salesforce has targeted **sub-second response times** as a requirement for real-time agent interactions (Source: ayaninsights.com). This was largely achieved in pilot scenarios like live chat assistants and shopping bots, especially for read-only operations. Write operations (like creating a case or updating an opportunity) also showed fast execution, though naturally the end-to-end time depends on the complexity of the operation (e.g., a transaction that triggers Apex or flows might take longer due to those backend processes, just as it would via standard API).

One **benchmark example** from a pilot test involved an AI agent querying a Commerce Cloud MCP server for product information: the agent issued a `search_product` tool request with a keyword, and the MCP server returned a list of matching products (including name, price, image URL, etc.) in roughly 400 ms for a catalog of thousands of products. This included the network round-trip and was deemed satisfactory for seamless conversational interaction. In another scenario, an AI agent used the CRM MCP server to gather a 360° customer view (retrieving an Account record along with related Contacts and recent Cases). By packaging these sub-queries into one MCP request, the response came back in ~800 ms with all data, whereas a non-MCP approach requiring multiple API calls might have taken 2–3 separate round-trips and significantly more client-side orchestration. These anecdotal benchmarks illustrate the **efficiency gains** MCP can provide. Moreover, the **consistency** of performance is bolstered by Salesforce's infrastructure; the MCP servers run within

Salesforce's multi-tenant cloud, benefiting from the platform's optimizations (like query caching, read-replica databases, etc.). When an MCP request repeats (e.g., multiple agents asking similar queries), underlying platform caching can make subsequent responses even faster – similar to how API performance improves for cached queries. In one pilot, frequently accessed reference data (product catalog info) was cached at the MCP server layer (via Heroku Redis using **Heroku AppLink**) to achieve near-instant (<100 ms) response times on repeated queries (Source: ayaninsights.com).

In terms of **throughput**, the pilot imposed conservative limits as mentioned. The default throttle of 50 requests/min per agent ensures that an agent doesn't hammer the system (Source: ayaninsights.com). However, multiple agents or users can connect simultaneously; Salesforce's backend will automatically scale to handle concurrency (just as it does when many users use Salesforce or many API calls come in). There hasn't been an official max throughput published, but given Salesforce's API can handle hundreds of calls per second per org in many cases, it's reasonable to expect the MCP service can support similar levels for aggregate load. Pilot tests with parallel agent sessions (for example, 5–10 agents each doing a request every few seconds) showed linear scaling with negligible degradation – the hosted service queued and handled requests efficiently. The **reliability** of the hosted MCP servers in pilot has been excellent so far. Salesforce reported no unscheduled downtime during the pilot period; the service leveraged the same HA (High Availability) approach as Salesforce's core (redundant servers, multi-AZ deployments on Hyperforce, etc.). In effect, if your Salesforce org was up, the MCP endpoint for that org was up as well. Telemetry from pilot usage also indicated a very low error rate – most errors were due to user issues like invalid queries or permission denials rather than server faults. Any server-side exceptions (like an internal error in processing a tool) were captured in logs and returned as error responses to the client in JSON-RPC format with appropriate messages.

To summarize the reliability metrics: Salesforce hosted MCP servers aim for enterprise-grade uptime (99.9% or better availability, in line with Salesforce's general SLA) and have shown stability in pilot. Latency for typical operations is in the sub-second range, suitable for interactive AI agent use. Throughput is controlled but sufficient for real-world scenarios, and can likely be increased as needed when GA (with proper governance). Salesforce's inclusion of **graceful degradation features** further enhances perceived reliability – for example, if an MCP server call does timeout or fail, Agentforce can automatically trigger fallback behaviors (like using cached data or handing off to a human) to maintain continuity (Source: ayaninsights.com). In pilots, such fallbacks were rarely needed, but it's good to know that guardrails are in place for extreme cases. Overall, the pilot results suggest that **hosted MCP servers can handle enterprise demands with high reliability**, while delivering performance that keeps up with the real-time needs of AI-driven workflows.

Hands-On Lab: Salesforce Hosted MCP Servers (Enterprise Setup)

In this hands-on lab, we walk through a step-by-step scenario of deploying and using a Salesforce Hosted MCP server in an enterprise environment. We will cover environment setup, authentication and access control, deploying/configuring the MCP server, making use of the server with an AI agent, troubleshooting, performance tuning, and implementing security/compliance measures. By the end of this lab, you will have a clear understanding of how to get started with a hosted MCP server pilot and integrate it into your Salesforce architecture.

> ***Lab Scenario:*** *For this lab, assume we are enabling a Salesforce Hosted MCP server for a fictitious company's Salesforce org to allow an AI assistant to access customer data and perform actions (e.g., querying customer info and creating a support case via natural language). We will use the* ***Salesforce CRM MCP server (pilot)*** *as an example. The steps would be analogous for other Salesforce-hosted MCP offerings (such as a B2C Commerce MCP server for shopping bots) with minor differences in the tools available.*

Prerequisites: You have a Salesforce org (sandbox or developer org) that has been **enabled for the MCP pilot** (typically arranged via Salesforce support or your account executive). You also have an AI agent or client ready that supports MCP connections – this could be Salesforce Agentforce or a third-party tool or even a simple script using an MCP client library. Finally, ensure you have **administrative access** to the Salesforce org to configure connected apps and permission sets.

1. Lab Environment Setup

First, set up the environment both in Salesforce and on the AI side:

- **Enable MCP Pilot in Salesforce:** Work with your Salesforce contact to get the "Hosted MCP Server" pilot feature enabled in your org. In some cases this might already be provisioned if you're in a special Developer Preview program. There may be a feature flag or a package to install – follow Salesforce's pilot instructions. For example, if enabling the **B2C Commerce Shopper MCP Service**, you must be onboarded to Salesforce's SLAS (Shopper Login and API Access Service) and have Commerce API enabled (Source: developer.salesforce.com). For a CRM MCP pilot, ensure you have the latest Agentforce or relevant package in your org if required. Once enabled, Salesforce will provide (or you can retrieve from documentation) the **endpoint URL** for your hosted MCP server. This endpoint is typically specific to your instance/org. For example, a Commerce MCP endpoint might look like: `https://<your-`

`instance>.api.commercecloud.salesforce.com/mcp/shopper/v1/organizations/<orgId>`, whereas a core CRM MCP might be at a Salesforce domain (the exact URL for CRM pilot might be given by Salesforce). Take note of the endpoint and API version if applicable.

- **Prepare an AI Client/Agent Environment:** Next, set up the environment from which you will connect an AI agent to the MCP server. If you plan to use **Salesforce Agentforce** as the client, ensure you have Agentforce 3 (or the relevant pilot version) enabled and the MCP client configured. In Agentforce Studio, there will be a way to register the MCP server (likely via the agent registry) or it may auto-discover it if your org has the feature enabled (Source: devopslaunchpad.com). If you are using a **custom AI client or open-source library**, install the library or SDK that supports MCP. For instance, you could use the *SpringFramework.AI* library or a LangChain integration that knows how to handle Agent<->Agent (A2A) or MCP protocols (Source: developer.salesforce.com). Alternatively, you can even use a cURL or Postman to simulate MCP calls for testing, though an actual AI agent will maintain a session and interpret results. Make sure whatever environment you use is network-accessible to Salesforce's API (internet connectivity) and can handle SSE (Server-Sent Events) if using that mechanism (some simple HTTP clients might not support SSE well, so using a library is recommended).

- **Network and App Access:** Since this is an enterprise scenario, verify any firewall or network settings. If your AI agent is running in a corporate network or on-premises, ensure it can reach the Salesforce MCP endpoint (which is an HTTPS endpoint). No VPN or special tunnel is needed beyond normal Salesforce API access, but your IT team should whitelist the Salesforce API domain if outbound traffic is restricted. Also, if using a third-party AI service (say an AI hosted on a cloud), ensure that service can make calls to Salesforce – this typically requires enabling cross-domain access via a Connected App (which we'll do next).

At this stage, the environment is prepared: the Salesforce org is MCP-enabled, and you have an AI agent environment ready to connect. Now we move to authentication.

2. Authentication and Access Control

Secure authentication is crucial. We need to set up an **OAuth trust** between the AI agent and the Salesforce MCP server, and enforce access controls via Salesforce permissions:

- **Create a Connected App (OAuth Client):** In Salesforce Setup, navigate to **Setup → App Manager** and create a new **Connected App** for the MCP agent access (if one is not already provided as part of the pilot). This connected app will issue OAuth tokens that the AI agent can use. Configure the connected app with the appropriate OAuth scopes. Minimally you'll need the **"Access and manage your data (API)"** scope (API full access) or a more granular scope if

Salesforce has introduced one for MCP. For example, in the Commerce MCP pilot, Salesforce required a scope named `sfcc.shopper-mcpagent` on the token (Source: developer.salesforce.com), which was configured through their API settings. For core CRM, it might just use standard API scope. Enable some OAuth flow (e.g. client credentials or JWT bearer) suitable for server-to-server auth with the AI. Essentially, we want the AI to be able to obtain an **access token** for Salesforce.

- **Provision an Integration User:** It's best practice to use a dedicated **integration user** for the MCP server access. Create a Salesforce user (or identify one) that the MCP server will "act as" when performing operations. This user should have a profile or permission set that grants **only the necessary access** (principle of least privilege). For instance, if the AI needs to read and create Cases and Contacts, give this user read/create on Cases and Contacts (and perhaps read on Accounts for linking), but not admin-level permissions. Also ensure field-level security for any sensitive fields is appropriately set – if there are fields the AI should not see (like confidential financial info), keep those hidden for this user. By controlling this user's perms, you directly control what the MCP server can do. Link this user to the Connected App by granting it OAuth access (e.g., assign the Connected App's profile or use an OAuth approved user policy).

- **Obtain an OAuth Token:** Using the connected app credentials, authenticate to Salesforce to get an **OAuth access token** for the integration user. In a dev environment, you might use a JWT assertion or an OAuth username-password flow (if allowed) to get a token. For a production setup, a more secure method (JWT or certificate) is preferred. Once you have the **access token (Bearer token)**, that will be used by the AI agent to authenticate each call to the MCP server. *For example*, if using cURL for a quick test, you would include a header `Authorization: Bearer <token>` in your requests to the MCP endpoint. In the Commerce Cloud scenario, the token is a JWT from SLAS, which contains tenant and user info and the special scope for MCP (Source: developer.salesforce.com). For our CRM scenario, the token would similarly carry the Salesforce session for the integration user and allowed scopes. **Verify the token** by calling a simple Salesforce REST API (like a GET on `/services/data/vXX.X/` to fetch org limits or similar); if that works, the token is valid and ready.

- **Access Control via Salesforce Settings:** Double-check any pilot-specific access settings. Some MCP pilots may require enabling certain permissions in Salesforce. For instance, if there is an **MCP Server permission set license or a pilot setting** ("MCP Access Enabled") in user settings, assign that to the integration user. Also, if the pilot has any org-wide settings (like enabling Event Monitoring or Agentforce features for audit logging), ensure those are turned on

to leverage full functionality. Essentially, confirm that **the integration user can do in Salesforce UI/API everything you expect the MCP server to do** on behalf of the AI. If the user lacks access to an object or field, the MCP call will fail or omit data, so set those up now.

At the end of this step, we have a secure authentication setup: an AI agent can obtain a Salesforce token and that token corresponds to a user with the right privileges. We're now ready to connect to the hosted MCP server using these credentials.

3. Deploying and Configuring the MCP Server

In the context of Salesforce Hosted MCP, "deploying" the server is more about **activating and configuring** it for your use rather than deploying code. Since Salesforce hosts it, our tasks are to ensure it's configured to our needs:

- **Verify MCP Server Activation:** With your org enabled and user auth in place, it's time to **initiate a connection** to the MCP server. If using Agentforce's UI, you might skip this manual step (Agentforce will list available MCP servers in a registry interface). Otherwise, we'll do it manually. Construct the **endpoint URLs** as documented. For example, for a Commerce Cloud MCP server, you open a **Server-Sent Event (SSE) connection** to the `/sse` endpoint and then use a `/messages` endpoint for JSON-RPC calls (Source: developer.salesforce.com). For a core CRM MCP pilot, Salesforce might have an endpoint like `https://yourInstance.salesforce.com/mcp/<version>/orgs/<orgId>/sse` (hypothetical format) and a corresponding messages endpoint. Using your chosen tool or code, open the **SSE stream**: this is typically a GET request to the `/sse` URL with the Authorization header. If successful, the server will keep the connection open and send an initial event (often containing a session ID or handshake data) (Source: developer.salesforce.com). Check that you receive this initial event – it indicates the MCP server is up and has accepted your connection. The event usually contains a field like `sessionId` which you'll use for subsequent calls (to tie them to the session).

- **Tool Discovery (Handshake):** Next, request the list of available tools from the MCP server. Depending on the protocol specifics, the initial SSE event might already contain the tool catalog, or you may need to send a JSON-RPC request method like `list_tools` or a special handshake method. Many MCP clients handle this automatically, but if doing manually, you'd POST a JSON message to the `/messages` endpoint with a method call to list or describe tools. The server will reply (likely over the SSE channel or the response of the POST) with a JSON listing of all **MCP tools and prompts** it offers. Review this list to familiarize yourself with what's possible. For example, you might see methods like `getRecord`, `queryRecords`, `createRecord`

for a CRM MCP server, or domain-specific ones like `search_product`, `add_product_to_cart` in the Commerce MCP server (Source: developer.salesforce.com)(Source: developer.salesforce.com). Each tool should come with a description and the parameters it expects. This discovery mechanism ensures the AI (and you as the implementer) know what actions are available. In our scenario, let's say the CRM MCP server provides tools: `find_contact(email)`, `create_case(contactId, description)`, `list_open_cases(contactId)`, etc., which would align with common CRM tasks.

- **Configure Tool Access (Optional):** If the pilot or your org allows, you might configure which tools are enabled. This might be done through a Salesforce UI (e.g., a settings page listing tools with on/off toggles) or via metadata deployment. In pilot, some participants had all tools enabled by default, whereas others could request certain tools to be toggled. For an enterprise, you might choose to **disable a tool** that is not needed or is sensitive. For instance, if there were a `delete_record` tool but you don't want the AI to ever delete data, you would turn that off. Since this is a lab, we'll assume the default set is fine. Just be aware that *in production, controlling tool availability is part of security best practices*. In Agentforce's registry, this is as simple as an admin un-checking a tool from the allowed list for agents (Source: ayaninsights.com).

- **Session Management:** The MCP server will maintain a session for ongoing interactions (especially if using SSE). Ensure your agent or client keeps the connection alive as long as needed. If you disconnect (e.g., end the SSE stream), you may need to reconnect and repeat the handshake to get a new session. Some implementations allow multiple concurrent sessions, so you could have one session per end-user or per functional thread. Document how you will manage these sessions in your application – e.g., an agent assisting multiple users might open a session per user conversation to isolate context.

At this stage, the MCP server is effectively "deployed" (enabled) and configured with the desired tools, and we have an active connection ready to use. Now we can proceed to actually using the MCP server in our lab scenario.

4. Using the MCP Server: Hands-On Operations

With the MCP server up and running, let's perform some **hands-on exercises** to simulate real usage. We'll go through a few common tasks an AI agent might do, using the MCP server's tools. These steps will illustrate how to invoke tools and handle responses:

- **Example 1: Query Data (Read Operation):** Suppose our AI assistant needs to retrieve a contact's details by email. We identified a tool `find_contact(email)` from discovery. To use it, the AI (or our test client) will send a JSON-RPC request message to the MCP `/messages` endpoint (or via the SSE channel) invoking `find_contact`. For example, the JSON payload might look like:

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "find_contact",
  "params": { "email": "alice@example.com" }
}
```

This instructs the MCP server to execute that tool with the given parameter. The server will authenticate the request (via the token we included in headers) and then perform the action – likely executing a SOQL query on Contact where email = provided value. The result is returned as a JSON-RPC response, for example:

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": {
    "contactId": "003xxx...",
    "firstName": "Alice",
    "lastName": "Johnson",
    "email": "alice@example.com",
    "phone": "555-0101",
    "openCases": 2
  }
}
```

The AI agent can parse this JSON and use it in its reasoning or display to a user. In our test, check that the result matches expectations (perhaps also cross-verify in Salesforce via UI). This confirms that the MCP server can successfully read data with appropriate filtering. If the contact wasn't found, the `result` might be null or empty – handle that logic in the agent as

needed. Note that the **response fields and structure are defined by the MCP server**, and were likely detailed in the tool's description (discovery might have told us what fields `find_contact` returns).

- **Example 2: Create a Record (Write Operation):** Now, let's simulate the AI assistant creating a support case for that contact. There might be a tool like `create_case(contactId, subject, description)` available. We prepare a JSON-RPC request:

```
{
  "jsonrpc": "2.0",
  "id": "2",
  "method": "create_case",
  "params": {
    "contactId": "003xxx...",
    "subject": "AI Assistant Case",
    "description": "Case created by AI for issue XYZ"
  }
}
```

When the MCP server receives this, it will perform the necessary steps in Salesforce – e.g., call the Case creation API (which runs assignment rules, triggers, etc., as normal). The response might look like:

```
{
  "jsonrpc": "2.0",
  "id": "2",
  "result": {
    "caseId": "500xxx...",
    "status": "New",
    "createdDate": "2025-08-05T14:30:00Z"
  }
}
```

If there was a validation error or missing field, the response might instead contain an error (with an error code and message). In our test, we ensure the case is indeed created (check Salesforce UI for the new case). This demonstrates the **MCP server executing a write with all of Salesforce's business logic** – from the AI's perspective it was just one method call. Because the integration user had permissions, the action succeeded; if not, the server would have returned a permission error.

- **Example 3: Multi-step Interaction (Agent Workflow):** Often, the agent will perform a sequence: e.g., find a contact, then create a case, then perhaps query back the open cases. We can chain these calls. Since our SSE connection is still open and we're maintaining state on the client side, the agent can use data from one response in the next request. For instance, after creating the case, we could call `list_open_cases(contactId)` to verify how many open cases the contact has now. The agent would send a new JSON request (with incremented id) and get back a list. The key point is that each call is stateless on the server side (the server doesn't remember previous calls except as necessary for session), but the agent can create a coherent workflow by using results from prior calls. In practice, your AI agent's **LLM** might decide to call `find_contact`, then use the contactId to call `create_case`, all in response to a single user prompt like "Please log a new case for Alice regarding issue XYZ." This showcases how natural language gets translated to a series of MCP tool calls, and those calls manipulate Salesforce in real time to fulfill the request.

- **AI Host Integration:** If you are using **Agentforce** as the AI host, many of these steps (parsing JSON, etc.) are handled internally by Agentforce's MCP client. You would likely see in logs that Agentforce, upon a user prompt, automatically invoked the appropriate MCP tools. As a lab runner, you can use Agentforce's debugging or conversation logs to verify it's calling the MCP server as expected. If using a custom agent, you might have to write a bit of code to parse the results and integrate it into the agent's response generation.

- **Handling Errors and Exceptions:** Intentionally test an error scenario to see how it's handled. For example, try calling `find_contact` with an email that doesn't exist, or call `create_case` without a required parameter, or use a tool your user isn't permitted to (if possible). The MCP server should return a structured error (JSON-RPC error with a code and message). Confirm that your agent logs or outputs the error meaningfully. This is important for troubleshooting and for possibly instructing the AI how to respond (e.g., if a required field was missing, the AI could ask the user for it). In pilot, error messages from the hosted MCP were fairly descriptive, often reflecting Salesforce errors (like field validation messages) but in a JSON-RPC format.

Through these operations, we've **validated that the MCP server is functional**: we can retrieve data and create data in Salesforce via natural language driven calls. The AI assistant can now leverage this to provide powerful functionality (like answering user questions with real Salesforce data or taking automated actions). Next, we focus on troubleshooting tips and performance tuning to ensure this runs smoothly in production.

5. Troubleshooting and Performance Tuning

Even with the system working, it's important to know how to troubleshoot issues and optimize performance for an enterprise rollout. Here are steps and tips:

- **Monitoring Logs and Usage:** Salesforce's **Event Monitoring** logs will capture MCP usage (in pilot, these might be under API logs or a dedicated MCP log category) (Source: ayaninsights.com). If something isn't working, check Salesforce logs for errors. For instance, if an MCP call fails due to a security restriction, you might see an event in the log with an error message like "FIELD_INTEGRITY_EXCEPTION" or "ENTITY_ACCESS_DENIED". Salesforce also often provides a **debug log** capability – though MCP is not an Apex process, any Apex triggers or flows invoked by the action will log as usual. So, if creating a case triggered an Apex error, you can find that via normal debug logs. Additionally, if using Agentforce, use its **Command Center** or agent monitoring tools to see the agent's interactions. Agentforce's observability will show each tool call the agent made, and whether it succeeded, and latency etc., helping pinpoint issues.
- **Common Issues & Resolutions:** Some typical issues and how to address them:
    - *Authentication failures (HTTP 401)*: This often means the Bearer token is missing or invalid. Ensure the token is being sent on **every** request (the SSE connection and each POST) and that it hasn't expired. During development, it's easy to forget to include the header on the SSE subscription or to handle token refresh. Implement a robust token refresh mechanism if tokens have short lifespans.
    - *Permission denied errors*: You may see errors indicating a field or object is not accessible. This points to the integration user's permissions. Go back to Salesforce Setup and adjust the profile/permission set – e.g., the user might need "Modify All" on cases if the AI is closing cases, or read access to a custom field. After adjusting, test again. It's good to test the integration user's abilities directly (e.g., log in as that user in Salesforce or use Workbench with that user) to ensure they have the needed access.

- *Tool not found or disabled*: If you attempt to call a tool name that the MCP server didn't list in discovery, you'll get an error. Double-check spelling/case (methods might be case-sensitive). It could also be that an admin disabled that tool. Re-enable it if needed or use an alternative method.

- *Malformed request errors*: If your JSON isn't exactly as expected (e.g., missing a required param), the MCP server will return a parse error or invalid params error. Use the discovery metadata to ensure you pass all required fields with correct types. In some pilot docs, the expected schema for each tool is described – refer to that when in doubt.

- *Network issues (timeouts, SSE drops)*: If the SSE stream is dropping frequently or you get timeouts, consider network stability. SSE is a long-lived connection; proxies or firewalls sometimes cut these off. You might need to configure keep-alive options or allowlist Salesforce domains. If a corporate proxy is interfering, test from a less restricted network or work with IT to permit this traffic. For timeouts on requests: if a request legitimately takes too long (over 30 seconds, for instance), Salesforce might time it out. In such cases, evaluate if the operation is too heavy (e.g., returning too much data). It may be necessary to break queries into smaller chunks or refine filters to improve performance.

- **Performance Tuning:** To ensure optimal performance:

  - *Leverage caching for frequent requests*: If the AI agent will repeatedly ask for the same data (like product info, or a list of codes), consider caching those either on the client side or at an intermediate layer. Salesforce's hosted MCP may not have an internal cache (aside from normal API caching), but you can introduce an edge cache. For example, if using Heroku, you might route the MCP responses through a small Heroku service that caches results for X minutes for certain read queries (Source: ayaninsights.com). Or within the AI agent code, store recent results in memory. This reduces round-trips for repetitive queries and speeds up responses.

  - *Parallelize when possible*: While the agent typically will do one thing at a time (especially because LLMs often think step by step), if you have an opportunity to do things in parallel (like fetch two unrelated pieces of info), you could open two MCP sessions or send asynchronous requests. Salesforce's infrastructure can handle concurrency well, so the bottleneck might be the agent's ability to manage multiple contexts. Use this with caution, but it's a potential way to reduce overall execution time for complex tasks.

- *Optimize tool usage*: Choose the right tool for the job to avoid excess work. For instance, if an MCP server provides a dedicated tool `search_accounts_by_name`, use it rather than a generic `query` tool with a broad SOQL – the dedicated tool likely has optimized logic or indexes. In one pilot, a *specific search tool* provided faster results than a general object query because it used pre-indexed search capabilities. Similarly, if there's a bulk operation tool (say, fetch multiple records in one call), use that instead of looping one-by-one. Reducing the number of round-trip calls will improve overall performance.

- *Adjust rate limits if needed*: The default throttling (50 calls/min) might be too low for high-throughput scenarios (like a busy chat with many rapid questions). Salesforce may allow raising this limit per org or per use case during pilot or GA (with oversight). If you truly need more, contact Salesforce support – do not simply try to bypass it, as you'll get HTTP 429 errors if you exceed it. In our lab scenario, we stay within limits, but be aware of them as you plan scaling.

- *Monitor and iterate*: Continuously monitor the response times and outcomes using the logs or any performance metrics. If certain calls are slow (e.g., maybe a query takes 800 ms which is noticeable in chat), dig into why – it could be that it's retrieving too much data or that an index is missing. Standard Salesforce optimization techniques (add indexes, refine SOQL filters) apply. Even though it's via MCP, under the hood it's Salesforce doing a query, so optimize that just as you would for any integration.

By proactively troubleshooting with logs and optimizing as above, you ensure your MCP integration runs efficiently and reliably. In a production enterprise setting, you'd likely also set up **alerts** on error rates or high latency (Salesforce's Command Center could help here, or external APM tools). During the pilot, we've learned how to identify and fix issues quickly, which is good practice for broader rollout.

6. Security and Compliance Configurations

Lastly, we focus on the **security and compliance** aspects to ensure the solution meets enterprise requirements. We already touched on authentication and authorization; here we'll solidify those and add compliance-specific steps:

- **Principle of Least Privilege:** Confirm once again that the integration user's permissions are tightly scoped. Often during testing, we give broad access to get things working; before go-live, roll back any excessive permissions. For example, if you had granted "Modify All Data" to troubleshoot an issue, remove that and only allow specific object access needed. If the AI only

needs read access to certain data, don't grant write. Use **Field Level Security** to exclude any sensitive fields (SSN, salary, etc.) that are not necessary for the AI's function (Source: ayaninsights.com). The hosted MCP server will automatically enforce these field level restrictions – e.g., if a field is hidden, the AI's result won't include it. Test this by trying to query a field that should be hidden and see that it's omitted or an error is thrown.

- **MCP Server Registry Policies:** Through Agentforce's MCP registry (if available in your pilot/org), set up governance policies. **Register** the hosted MCP server in the registry and define which AI agents can access it. You may, for instance, allow only a specific Agentforce skill or only agents with a certain permission to utilize the server (Source: ayaninsights.com). If the hosted MCP is to be used by external AI (outside Agentforce), ensure that those connections are still accounted for – you might not have the registry control in that case, so rely on network controls (like only giving the token to approved systems). The registry also lets you enforce **rate limits or usage times** globally. For compliance, maybe you only want the AI to perform certain actions during business hours (just as an example); such policies could be scripted if needed, though not out-of-the-box in pilot.

- **Audit Logging and Review:** Enable **Event Monitoring** (or ensure it's part of your license) to capture all MCP interactions. Salesforce logs will record what query was run or what record was accessed. For compliance, set up a process to **regularly review these logs** – especially early on, to verify the AI isn't accessing something it shouldn't. You might integrate the logs with a SIEM (Security Information and Event Management) system. Since Salesforce logs can show which user (our integration user) did what, you might want to distinguish AI-driven actions from other integrations that user might do; consider using a dedicated user just for MCP vs other integrations, so the activities are clearly separated. Additionally, use the logs to support **audits** – e.g., if there's a requirement to demonstrate that customer data accessed by AI is tracked, you now have those records.

- **Just-in-Time Approvals for Sensitive Actions:** If your business has compliance rules around certain operations (like large financial transactions or data deletions), incorporate a human approval step. While the technology (MCP + Agentforce) can support an AI doing potentially any action, you likely want a safeguard for critical ones. Salesforce mentioned a concept where an AI's MCP request can pause and require a manager's approval via Slack (Source: ayaninsights.com). Implementing this may require building a custom layer or using Agentforce's upcoming features. In the interim, a simple approach is to *not give the AI tools for those actions*. For example, do not expose a "delete record" tool at all, and handle deletions via a manual

process. Or if the AI needs to initiate something like a refund, have it create a task or case for human review rather than doing it directly. This way you stay compliant with internal policies. Over time, Salesforce may provide native support for such approval workflows in Agentforce.

- **Data Residency and Privacy:** Since the MCP server is hosted by Salesforce, data accessed by the AI through MCP stays within Salesforce's trusted environment during retrieval. However, once the AI (especially an external LLM) receives the data, consider data privacy implications. Ensure that your use of data via MCP complies with privacy laws and regulations (GDPR, HIPAA if applicable, etc.). Mask or avoid sending personally identifiable information (PII) to an external AI if not allowed. You can enforce some of this by restricting fields (as mentioned) and by coding your agent to sanitize outputs. Also, check if Salesforce offers any data residency options for MCP – e.g., if using Hyperforce in specific regions for compliance, the MCP server likely runs in-region as well (in pilot, not much choice, but by GA Salesforce will align with their data residency commitments). Always refer to Salesforce's documentation on compliance and trust (the **Trust & Compliance documentation**) to see if the MCP service has any particular certifications or restrictions as of GA.

- **Encryption:** All communication with hosted MCP servers is over HTTPS, which ensures data in transit is encrypted. Salesforce will handle encryption at rest on their end, as they do for all cloud data. Verify that you are using TLS 1.2+ and that your client trusts Salesforce's certificates. This is usually transparent, but worth noting for security completeness.

- **Third-Party Access Considerations:** If your AI agent is an external service (say, an AI platform outside Salesforce), ensure that **only that service** can use the token. Do not embed long-lived tokens in client-side apps or anywhere they could be intercepted. Use secure storage (e.g., AWS Secrets Manager or Salesforce named credentials if within Agentforce) for the credentials. If the AI vendor offers IP allowlisting, you might restrict token usage to their IPs. Essentially, treat the MCP access token like a key to your kingdom – protect it accordingly.

By configuring these security and compliance measures, you can confidently deploy the hosted MCP server in a production environment. The combination of OAuth security, org permissions, administrative oversight (registry and logging), and prudent limitation of AI actions ensures that you reap the benefits of AI integration without undue risk. Salesforce's emphasis on **"trust" and enterprise security** in Agentforce and MCP means these features are not just nice-to-haves but core parts of the product (Source: devopslaunchpad.com)(Source: ayaninsights.com) – use them to their full extent. Before concluding the lab, it's wise to conduct a **security review**: simulate misuse (can the AI get to something it shouldn't?), review compliance checklists, and get sign-off from your

InfoSec team. In our lab test scenario, if everything looks good – the AI can only do what it's supposed to, all access is logged, and sensitive info is protected – then you're ready to move from pilot to a broader rollout.

Comparison: Hosted MCP vs. On-Premises & Alternative Approaches

Salesforce's hosted MCP servers are one approach to connecting AI with enterprise data, but architects should consider how it compares to other solutions, such as self-hosting MCP or using different integration patterns like caching. The table below summarizes key differences:

APPROACH	DESCRIPTION	PROS	CONS
Salesforce Hosted MCP	Salesforce-managed MCP servers running in the Salesforce cloud, providing out-of-the-box tools for Salesforce APIs (pilot phase) (Source: developer.salesforce.com). No customer infrastructure needed – just configuration and tokens.	- **Easy Setup:** No infrastructure to manage; quick to enable and use (Source: devopslaunchpad.com).	
- **Secure & Trustworthy:** Inherits Salesforce security model (OAuth, FLS, audits) and reliability (Source: ayaninsights.com)(Source: ayaninsights.com).			
- **Native Integration:** Built-in support with Agentforce and future Salesforce releases, with enterprise governance (registries, policies) (Source: ayaninsights.com).			
- **Maintenance Included:** Updates, scaling, and patches handled by Salesforce.	- **Limited Customization (Currently):** Pilot supports only specific APIs; cannot yet define custom tools or incorporate non-Salesforce systems (until future GA enhancements) (Source: developer.salesforce.com).		
- **Salesforce Dependency:** Functionality tied to Salesforce release cycle and uptime (though uptime is			

APPROACH	DESCRIPTION	PROS	CONS
high). Pricing is TBD – could be an added cost when GA (Source: developer.salesforce.com).			
- **Data Boundaries:** Data stays in Salesforce, which is good for security, but if an AI needs to combine non-SF data, you'd need additional MCP servers or connectors (e.g., via MuleSoft).			
Self-Hosted MCP Server	Deploying your own MCP server (or using open-source ones) on infrastructure you control (on-premises or cloud like Heroku). For example, running the open-source Salesforce DX MCP locally, or a custom MCP service that you build (Source: developer.salesforce.com) (Source: devopslaunchpad.com).	- **Highly Customizable:** You can expose any API or service (Salesforce or otherwise) and tailor tools to your needs (including custom objects or processes). Full control over functionality and update cycle. (Source: devopslaunchpad.com)	
- **Data Control:** Can be run within your network or VPC for sensitive data, satisfying strict data residency or isolation requirements. No need to send data through Salesforce's cloud if not desired (except normal API calls).			
- **Open Source Ecosystem:** Leverage community MCP	- **Infrastructure & Dev Overhead:** You must host		

APPROACH	DESCRIPTION	PROS	CONS
servers or frameworks to jump-start development (e.g., many community-built servers for various platforms exist).	and maintain the service (servers, scaling, HA, monitoring). This adds DevOps burden and complexity, especially to meet enterprise SLAs.		
- **Security On You:** You are responsible for implementing auth, encryption, logging correctly. Mistakes could introduce vulnerabilities. Lacks the automatic Salesforce trust layer, unless you integrate with Salesforce security yourself. (Source: truefoundry.com)			
- **Integration Effort:** For Salesforce data, you'd still use Salesforce APIs under the hood – essentially recreating what Hosted MCP provides. Development and testing of the MCP interface require time, and any Salesforce upgrades (API changes) you must handle.			
Third-Party Caching/Integration (Non-MCP)	Using alternative methods to provide AI access to Salesforce data, such as replicating data to an external store or using Platform Cache for quick retrieval, and letting AI query that. Examples: Export Salesforce data to a data warehouse or search	- **Potential Performance Gains:** A purpose-built cache or index (like a search engine or vector DB for knowledge) can retrieve information faster than live API for large volumes, and can be optimized for AI	

APPROACH	DESCRIPTION	PROS	CONS
	index for the AI, use Salesforce Platform Cache to store frequently accessed info, or use a middleware (ETL or RPA bots) to feed data to AI.	queries (e.g., semantic search).	
- **Decoupling:** The AI can function without hitting the live Salesforce org for every query, reducing load on core systems. Good for scenarios with very high query rates or when you want to isolate the AI from production data stores for safety.			
- **Mature Tools:** There are existing data integration and caching tools (ETL pipelines, middleware) many orgs have, which can be repurposed to supply AI with data (for instance, using a nightly sync to a cloud database that the AI then queries).	- **Not Real-Time:** Once you cache or copy data, it can become stale. AI answers might be outdated if data changes frequently and your sync isn't instant. This is problematic for up-to-the-minute needs that MCP live access would handle (Source: mcpmarket.com).		
- **Complex Setup:** Building and maintaining the cache or integration layer is work – mapping data, ensuring consistency, handling incremental updates. It can become as complex as any data warehouse project. Also, ensuring security in the cache (who can access it, is			

APPROACH	DESCRIPTION	PROS	CONS
data encrypted, etc.) is an extra layer to manage.			
- **Lacks Standardization:** Without MCP, each integration might be custom. AI agents would need specific code to query the cache or use a custom API. You lose the benefit of the MCP universal interface. There's also no native governance layer akin to Agentforce's MCP registry; you'd have to enforce policies ad hoc.			

Analysis: Salesforce Hosted MCP servers shine in scenarios where you want a **quick, secure, and Salesforce-supported solution** to connect AI with Salesforce data. They minimize overhead and leverage Salesforce's strengths (security, reliability). For organizations already investing in Salesforce's AI ecosystem (Agentforce, Einstein, etc.), hosted MCP is likely the best path because it will integrate naturally and be officially supported. On the other hand, **self-hosted MCP** might be chosen if you have very specific needs – for example, including custom business logic in the MCP server or integrating multiple systems (Salesforce and non-Salesforce) into one MCP endpoint. In fact, Salesforce acknowledges this by supporting custom MCP servers on Heroku and via MuleSoft connectors (Source: devopslaunchpad.com). Self-hosting gives flexibility at the cost of more maintenance. It could also be a temporary strategy: some customers might self-host now for customization, then migrate to Salesforce's hosted offering as it matures (especially if Salesforce allows packaging custom MCP logic on-platform in the future (Source: developer.salesforce.com)).

The **third-party caching or integration approach** is somewhat different – it might be used in tandem with MCP or instead of MCP in certain cases. For example, if an organization is not comfortable yet with an AI getting direct live access, they might prefer to feed the AI from a curated data snapshot (for compliance). Or if the use case is more about large-scale analytics (where an AI analyzes thousands of records to find patterns), a data warehouse approach might be better than querying Salesforce record-by-record. However, this approach sacrifices the real-time, interactive capabilities that MCP provides. It also reintroduces siloing and delay, which MCP was meant to

eliminate. Still, in high-performance scenarios (say, an AI needs to answer questions from millions of knowledge articles quickly), an external index (like storing that content in a vector database for semantic search) might outperform hitting Salesforce repeatedly. It's possible to even combine approaches: use MCP for transactional actions and real-time small queries, but use an external system for heavy-duty searches or historical data analysis, with the AI choosing the source as appropriate.

In summary, **Salesforce Hosted MCP** offers an *enterprise-friendly, low-friction solution* especially suited for real-time agentic AI use-cases within the Salesforce ecosystem (Source: mcpmarket.com). **On-prem/self-hosted MCP** is about *control and customization*, at the expense of more work – it fits scenarios where Salesforce's offering doesn't (yet) meet specific requirements or if an organization must keep everything in-house. **Caching/alternative integrations** are about *performance and decoupling*, useful in edge cases or as complementary solutions, but they don't replace MCP's versatility and standardization. Most likely, as hosted MCP servers become generally available and more feature-rich, we will see many organizations gravitate towards them for AI integration, while using self-hosted MCP or other methods only for niche needs or in hybrid architectures.

Conclusion

Salesforce Hosted MCP Servers represent a significant leap forward in connecting enterprise AI agents with live business data in a secure, governed way. Through this pilot review and hands-on lab, we've seen that hosted MCP servers bring the promise of **natural language integration** to Salesforce: enabling AI to not just chat about data, but to truly **act on Salesforce data and services in real time**. The pilot results are promising – organizations have found that these MCP servers are relatively easy to set up, integrate cleanly with Salesforce environments, and deliver performance sufficient for interactive AI experiences. The **architecture** leverages Salesforce's trusted platform (OAuth 2.0, permissions, auditing) to ensure that even as AI agents gain more autonomy, they operate within the safe confines of enterprise policy (Source: ayaninsights.com) (Source: ayaninsights.com). Key benefits like unified customer views, cross-cloud automations, and AI-assisted workflows stand out, with early adopters reporting faster processes and substantial productivity gains (Source: ayaninsights.com)(Source: ayaninsights.com).

In our hands-on lab exercise, we demonstrated how an enterprise can **deploy and utilize a hosted MCP server** – from initial environment prep and secure authentication, through executing example operations like queries and case creation, to instituting proper troubleshooting and security

measures. The lab underscores that, while the technology is cutting-edge, the steps to use it are grounded in familiar Salesforce practices (configuring connected apps, permission sets, etc.) and standard web protocols (JSON/HTTP). This means that Salesforce admins and developers can quickly get up to speed with MCP without a steep learning curve. We also highlighted the importance of **governance and tuning** – ensuring the AI only does what it's supposed to and that performance is optimized. With Salesforce providing tools like the Agentforce MCP registry and event monitoring, enterprises have the visibility and control needed to deploy these capabilities with confidence (Source: ayaninsights.com)(Source: ayaninsights.com).

Comparatively, Salesforce's hosted solution offers clear advantages for many use cases, though alternative approaches have their place. As with any enterprise technology, the right solution must be evaluated against requirements for flexibility, control, and risk. Hosted MCP servers will continue to evolve (with expectations of broader API coverage and even custom MCP creation on-platform in the future (Source: developer.salesforce.com)). The pilot is just the beginning – Salesforce's roadmap suggests rapid expansion of MCP use cases (e.g., partner-built MCP servers for various domains, AI-to-AI orchestration, and even AI-generated MCP connectors) (Source: ayaninsights.com). This trend indicates that **MCP is poised to become a standard layer in Salesforce integration architecture**, analogous to how REST APIs and integration middleware are standard today, but specifically tailored for AI and agentic use.

For enterprise IT and Salesforce architects, the takeaway is that **Salesforce Hosted MCP Servers merit close attention and experimentation now**. The pilot review shows tangible benefits and workable deployment patterns, so organizations should start planning how AI agents could leverage Salesforce data via MCP. Running a small-scale pilot (perhaps similar to our lab) is a great way to build internal expertise. Pay special attention to the compliance aspects – involve security teams early, and design your usage of MCP with a "secure by default" mindset. If done right, hosted MCP servers can unlock new levels of automation and insight (imagine truly AI-driven CRMs where mundane tasks are handled by agents in the background), while Salesforce's enterprise-grade approach keeps it all **reliable and trusted**(Source: devopslaunchpad.com).

In conclusion, Salesforce Hosted MCP Servers combine innovation with trust. They allow AI to speak the language of Salesforce (and vice versa) in a standardized, safe manner. The pilot phase has demonstrated viability, and with the step-by-step guidance provided in this report, a Salesforce professional can begin exploring MCP today. As the technology matures into GA, it may well become a cornerstone of Salesforce integrations, ushering in an era where autonomous AI assistants seamlessly collaborate with enterprise systems. Embracing this capability early could give

organizations a competitive edge – increasing efficiency, improving customer experiences, and unlocking the full potential of **AI + CRM** in a way that is governed, scalable, and aligned with enterprise values of security and compliance (Source: mcpmarket.com).

Sources:

1. Mohith Shrivastava – *Introducing MCP Support Across Salesforce* (Salesforce Developers Blog, June 2025) (Source: developer.salesforce.com)(Source: developer.salesforce.com)
2. Sandip Patel – *Salesforce MCP Server Explained with Real Use Cases* (Ayan Insights Blog, July 2025) (Source: ayaninsights.com)(Source: ayaninsights.com) (Source: ayaninsights.com) (Source: ayaninsights.com)
3. Vivek Chawla – *Level Up Your Developer Tools with Salesforce DX MCP* (Salesforce Developers Blog, June 2025) (Source: developer.salesforce.com)(Source: developer.salesforce.com)
4. Dave Rant (Gearset) – *Why Model Context Protocol matters in Salesforce DevOps* (DevOps Launchpad, 2025) (Source: devopslaunchpad.com)(Source: devopslaunchpad.com)
5. Salesforce Press Release – *Salesforce Announces Agentforce 3* (June 23, 2025) (Source: salesforce.com)(Source: salesforce.com)
6. Salesforce Developers – *Agentic MCP Shopper Tools Quick Start (B2C Commerce)* (Pilot Documentation, 2025) (Source: developer.salesforce.com)(Source: developer.salesforce.com) (Source: developer.salesforce.com)
7. Ayan Insights – *Salesforce MCP Server Explained (Benefits & Use Cases)*(Source: ayaninsights.com)(Source: ayaninsights.com) (Source: ayaninsights.com)
8. Ayan Insights – *Salesforce MCP Server (Implementation Challenges & Mitigations)*(Source: ayaninsights.com)(Source: ayaninsights.com)
9. MCP Market News – *Democratize Your CRM Data Across the Enterprise with Salesforce Hosted MCP Servers* (Salesforce Blog Summary, June 2025) (Source: mcpmarket.com)(Source: mcpmarket.com)
10. TrueFoundry – *What is MCP Server? A Brief Explanation* (2024) (Source: truefoundry.com) (Source: truefoundry.com)

Tags: salesforce, model context protocol, mcp, ai agents, crm, api integration, hands-on lab, enterprise ai

About Cirra

About Cirra AI

Cirra AI is a specialist software company dedicated to reinventing Salesforce administration and delivery through autonomous, domain-specific AI agents. From its headquarters in the heart of Silicon Valley, the team has built the **Cirra Change Agent** platform—an intelligent copilot that plans, executes, and documents multi-step Salesforce configuration tasks from a single plain-language prompt. The product combines a large-language-model reasoning core with deep Salesforce-metadata intelligence, giving revenue-operations and consulting teams the ability to implement high-impact changes in minutes instead of days while maintaining full governance and audit trails.

Cirra AI's mission is to **"let humans focus on design and strategy while software handles the clicks."** To achieve that, the company develops a family of agentic services that slot into every phase of the change-management lifecycle:

- **Requirements capture & solution design** – a conversational assistant that translates business requirements into technically valid design blueprints.
- **Automated configuration & deployment** – the Change Agent executes the blueprint across sandboxes and production, generating test data and rollback plans along the way.
- **Continuous compliance & optimisation** – built-in scanners surface unused fields, mis-configured sharing models, and technical-debt hot-spots, with one-click remediation suggestions.
- **Partner enablement programme** – a lightweight SDK and revenue-share model that lets Salesforce SIs embed Cirra agents inside their own delivery toolchains.

This agent-driven approach addresses three chronic pain points in the Salesforce ecosystem: (1) the high cost of manual administration, (2) the backlog created by scarce expert capacity, and (3) the operational risk of unscripted, undocumented changes. Early adopter studies show time-on-task reductions of 70-90 percent for routine configuration work and a measurable drop in post-deployment defects.

Leadership

Cirra AI was co-founded in 2024 by **Jelle van Geuns**, a Dutch-born engineer, serial entrepreneur, and 10-year Salesforce-ecosystem veteran. Before Cirra, Jelle bootstrapped **Decisions on Demand**, an AppExchange ISV whose rules-based lead-routing engine is used by multiple Fortune 500 companies. Under his stewardship the firm reached seven-figure ARR without external funding, demonstrating a knack for pairing deep technical innovation with pragmatic go-to-market execution.

Jelle began his career at ILOG (later IBM), where he managed global solution-delivery teams and honed his expertise in enterprise optimisation and AI-driven decisioning. He holds an M.Sc. in Computer Science from Delft University of Technology and has lectured widely on low-code automation, AI safety, and DevOps for

SaaS platforms. A frequent podcast guest and conference speaker, he is recognised for advocating "human-in-the-loop autonomy"—the principle that AI should accelerate experts, not replace them.

Why Cirra AI matters

- **Deep vertical focus** – Unlike horizontal GPT plug-ins, Cirra's models are fine-tuned on billions of anonymised metadata relationships and declarative patterns unique to Salesforce. The result is context-aware guidance that respects org-specific constraints, naming conventions, and compliance rules out-of-the-box.
- **Enterprise-grade architecture** – The platform is built on a zero-trust design, with isolated execution sandboxes, encrypted transient memory, and SOC 2-compliant audit logging—a critical requirement for regulated industries adopting generative AI.
- **Partner-centric ecosystem** – Consulting firms leverage Cirra to scale senior architect expertise across junior delivery teams, unlocking new fixed-fee service lines without increasing headcount.
- **Road-map acceleration** – By eliminating up to 80 percent of clickwork, customers can redirect scarce admin capacity toward strategic initiatives such as Revenue Cloud migrations, CPQ refactors, or data-model rationalisation.

Future outlook

Cirra AI continues to expand its agent portfolio with domain packs for Industries Cloud, Flow Orchestration, and MuleSoft automation, while an open API (beta) will let ISVs invoke the same reasoning engine inside custom UX extensions. Strategic partnerships with leading SIs, tooling vendors, and academic AI-safety labs position the company to become the de-facto orchestration layer for safe, large-scale change management across the Salesforce universe. By combining rigorous engineering, relentlessly customer-centric design, and a clear ethical stance on AI governance, Cirra AI is charting a pragmatic path toward an autonomous yet accountable future for enterprise SaaS operations.
