Build an MCP Server for Power BI: A Step-by-Step Guide

By Adrien Laurent, CEO at IntuitionLabs • 11/13/2025 • 45 min read

model context protocol power bi llms ai agents business intelligence natural language query azure data integration

Executive Summary

Recent advances in generative AI and large language models (LLMs) have created powerful new opportunities for business intelligence, enabling users to interact with data using natural language. To safely and effectively leverage these capabilities for enterprise analytics, it is crucial to bridge LLMs to organizational data stores. The *Model Context Protocol* (MCP) is a newly introduced open standard designed for just this purpose: to let AI agents query external data sources and tools in a controlled, structured way ([1] www.itpro.com) ([2] www.techradar.com). In particular, constructing an MCP server that provides access to Microsoft Power BI data can allow AI assistants (like Copilot or ChatGPT-enabled tools) to answer complex business questions using live Power BI dashboards and datasets.

This report presents a thorough, eight-step methodology for building an MCP server tailored to Microsoft Power BI. Each step is discussed in depth, covering both conceptual foundations and practical implementation details. We begin by reviewing the relevant background: the scale and importance of Power BI in enterprise analytics ([3] powerbi.microsoft.com), the emerging role of AI and natural language query in decision-making ([4] www.techradar.com) ([5] moldstud.com), and the fundamentals of the Model Context Protocol ([1] www.itpro.com) ([2] www.techradar.com). We then outline the step-by-step process, which includes defining requirements, setting up Azure Active Directory (AAD) authentication, configuring Power BI access, choosing development frameworks, coding the MCP server with appropriate tools, and finally testing and securing the deployment. Throughout, we cite industry data and expert sources to substantiate best practices and business drivers. For example, studies show that 75% of companies report *improved efficiency* when AI is integrated with data systems, and successful AI implementations are heavily tied to user training (67% success link) ([6] moldstud.com). We also draw on existing MCP implementations (such as the Microsoft 365 MCP server (www.mcp.pizza) and the community-built "powerbi-mcp" server ([7] github.com)) as case studies.

In summary, by following the eight-step blueprint detailed in this report, an organization can create a secure, robust MCP server that enables AI agents to query and interact with Power BI datasets in real time. This will empower business users to ask complex data questions in natural language, accelerating insights and driving data-driven decision-making. The implementation must also address security and governance carefully – MCP by itself has no built-in enterprise auth, so we rely on Azure AD and Role-Based Access Control ([8] www.itpro.com) ([9] www.techtarget.com). The final sections discuss implications and future directions, noting that analytical AI (data-driven decision-making) deserves attention alongside generative AI ([4] www.techradar.com), and that tools like MCP servers could become a cornerstone of modern BI platforms (much as Azure's own MCP server demonstrates ([10] learn.microsoft.com) ([11] www.techradar.com)).

Introduction

The growing integration of **artificial intelligence (AI)** and **business intelligence (BI)** is reshaping how organizations analyze data and make decisions. Traditionally, BI tools like Microsoft Power BI have enabled data analytics through dashboards and visual reports that users must interpret manually. However, with modern *large language models* (LLMs) such as GPT-4 and Claude, it has become possible to interact with data using natural language questions. For example, an executive could ask, "What were our top-selling products by region last quarter?" and receive an answer generated by querying the BI data. Early efforts at this – like Power BI's built-in Q&A feature – hint at the potential, but integrating LLMs with live enterprise data at scale is still an active technical challenge.

The **Model Context Protocol (MCP)** provides a promising solution to this challenge ([1] www.itpro.com) ([2] www.techradar.com). Introduced by Anthropic in late 2024, MCP is an open standard (akin to an interface specification) that defines how LLMs (the **hosts** or AI assistants) can securely query external **servers** (databases, APIs, or tools) via standardized **clients**. In practice, an MCP server is a program that implements a set of "tools" (actions) and "resources" (data queries) which an LLM can invoke. This decouples the LLM from direct database access – the LLM simply issues a high-level request, and the MCP server handles authentication, data retrieval, and response formatting ([1] www.itpro.com) ([12] www.techradar.com). The result is a modular, agent-like architecture where any LLM (e.g. ChatGPT with an MCP client) can plug into various data sources without bespoke integrations for each use case ([2] www.techradar.com) ([12] www.techradar.com).

Building an MCP server for Microsoft Power BI thus entails creating a program that exposes Power BI functionality through MCP. This means the LLM can ask to list datasets, query tables, refresh reports, etc., and the MCP server will perform those actions using Power BI's APIs under the hood. The server must handle authentication (using Azure AD), enforce permissions, and format responses for the LLM. By enabling this capability, organizations empower business users to "converse" with their BI data, lowering the barrier between raw data and insight ([4] www.techradar.com) ([5] moldstud.com).

Power BI itself is an **enterprise-scale analytics platform**. As Microsoft reports, its customers ingest *over 20 petabytes of data every month*, and the service handles *over 12 million queries per hour* ([3] powerbi.microsoft.com). </current_article_content>It hosts more than *30 million reports and dashboards* in use. Gartner recognizes Power BI as a leader in analytics platforms (Microsoft has held this position for 12 consecutive years) ([13] powerbi.microsoft.com). Given this scale, integrating AI directly into Power BI workflows can have huge impact. For instance, LLMs could scan across all dashboards to surface trends or anomalies, auto-generate natural-language summaries of the latest data, or answer ad-hoc business questions using underlying KPIs.

Empirical evidence underscores the value of AI in analytics. Recent industry analyses note that companies using AI-driven analytics report up to *30% faster operations* and *25-33% better predictive accuracy* compared to traditional methods ([14] moldstud.com) ([5] moldstud.com). In addition, automated reporting with AI can cut report delivery time by roughly *50%* ([15] moldstud.com). These gains come from automating tasks like data integration, forecasting, and report writing – precisely the kind of tasks an LLM + MCP server can facilitate. For example, Table 1 (below) summarizes key findings: 75% of surveyed firms saw boosted efficiency via data integration, and 67% linked project success to thorough training and adoption of AI tools ([6] moldstud.com). (All statistics in Table 1 are drawn from a July 2025 industry report on AI in business intelligence ([6] moldstud.com).) The implication is clear: organizations that streamline data access for AI see significant business benefits.

Area of Focus	Impact on Performance	Statistics
Data Integration	Streamlined operations	75% of companies report improved efficiency ([6] moldstud.com)
Predictive Analytics	Enhanced forecasting	33% improvement in accuracy ([6] moldstud.com)
User Training	Increased tool adoption	67% tied to enhanced success rates ([6] moldstud.com)

Table 1: Surveyed impacts of AI/analytics initiatives on business outcomes (data from Andersen et al., 2025 ([6] moldstud.com)). Emphasis on quality data integration and user training correlates with major efficiency and accuracy gains.

Despite the promise, building such a system requires careful engineering. MCP by itself does not magically solve security or governance; in fact, experts warn that MCP *lacks built-in enterprise-level authentication* and thus must be combined with robust identity controls ([8] www.itpro.com) ([9] www.techtarget.com). For this reason, our design uses Azure Active Directory (Entra ID) for authentication and Power BI's own RBAC (Role-Based Access Control) to restrict data access. We also leverage Microsoft's official support for MCP: for example, the **Azure MCP Server** is a Microsoft-provided reference implementation that shows how MCP can work with Azure resources ([10] learn.microsoft.com). In addition, the open-source "ms-365-mcp-server" demonstrates MCP with

Microsoft Graph and Office services (www.mcp.pizza). Finally, we note community projects like the Python-based *powerbi-mcp* server, which already enable natural language querying of Power BI datasets ([7] github.com).

In the following sections we will compare these perspectives, distill best practices, and then outline our eight-step process. We will cover technical specifics (APIs, code libraries, authentication flows), business rationale (why each step matters), and forward-looking insight (how such a project fits into broader AI/BI trends). Citations from academic, industry, and technical sources will be provided where appropriate to support each claim.

Background and Key Concepts

Microsoft Power BI in Enterprise Analytics

Power BI is a flagship BI platform combining self-service reporting, central governance, and data visualization. Its massive adoption underscores how critical BI is in modern enterprises. Microsoft reports that over **20 petabytes** of data flow into Power BI each month, powering more than **30 million active reports/dashboards** and serving **over 12 million queries per hour** ([3] powerbi.microsoft.com). Organizations rely on Power BI to aggregate diverse data sources (cloud services, databases, spreadsheets, IoT data, etc.) under a common semantic model. Features like shared datasets, report publishing, and paginated reports provide both flexibility and enterprise-grade governance ([16] powerbi.microsoft.com) ([17] learn.microsoft.com).

Given this ubiquity, enhancing Power BI with AI capabilities is a natural step. Users already perform many tasks via the Power BI service (administration, publishing, refreshing datasets). The Power BI REST APIs expose these capabilities programmatically ([17] learn.microsoft.com). For example, developers can use the APIs to list workspaces (groups), manage dataset refreshes, retrieve report metadata, and even embed reports in custom applications. While adept developers can write code against these APIs, non-technical users often struggle to extract insights quickly. Integrating natural-language AI can democratize access, letting employees with no coding skills ask questions of the data and get instant answers (sometimes in the form of auto-generated visuals or DAX queries). This convergence of AI and BI is part of a larger trend where “analytical AI” (data analysis, forecasting, decision support) supplements the celebrated “generative AI” (text/image generation) ([4] www.techradar.com).

Analytical AI is currently underutilized: as one TechRadar analysis notes, only *9% of UK firms* use AI for data management, even as generative AI adoption surges ([4] www.techradar.com). However, this is changing because investors and leaders now demand *data-driven reporting*. Private equity and C-suite executives increasingly expect AI-enhanced analysis to inform strategy ([18] www.techradar.com). Equipping Power BI with AI-driven query capability helps meet this demand by providing real-time insights grounded in the organization’s own metrics. For example, an AI agent could answer: “What is our projected inventory shortage based on current trends?” by pulling numbers from Power BI’s stored forecasts, rather than guessing from generic market data.

The Model Context Protocol (MCP)

Before diving into implementation, we must understand the Model Context Protocol (MCP) and its role in AI systems. MCP is a recent open standard (released by Anthropic in Nov 2024 ([1] www.itpro.com)) that formalizes how language models can securely use external tools and data. At a high level, MCP defines a *client-server architecture*:

- **Hosts** (applications or IDEs) run the LLM through an **MCP client**, which transforms user or model queries into the MCP format.
- **Servers** implement the MCP interface: they offer *tools* (actions the model can request) and *resources* (data or context that can be fetched).

When a user query involves external data, the MCP client sends a **JSON-RPC** request to the MCP server specifying which tool or resource is needed. The server executes the request (e.g. running a query), then returns results back through the client to the model ([1] www.itpro.com) ([12] www.techradar.com). This handshake allows LLMs to indirectly perform tasks (like database queries) without hardcoding such logic. Prominent AI development environments (VS Code with Copilot, Claude Desktop, etc.) are beginning to adopt MCP, letting them plug into any compliant server ([10] learn.microsoft.com) ([19] www.techtarget.com).

Key qualities of MCP include **standardization** and **modularity** ([2] www.techradar.com) ([12] www.techradar.com). By using a common protocol, MCP eliminates the "N×M problem" of connecting every LLM individually to every data source ([1] www.itpro.com) ([2] www.techradar.com). Instead, once a Power BI MCP server is built, *any* MCP-aware AI agent can reuse it. The "tools" exposed by the server are described with schemas (inputs/outputs) so that the LLM knows what arguments to provide and what results to expect. For instance, a tool named `queryDataset` might take parameters like workspace ID and SQL/DAX query text, and return a table of results. These tools are discovered dynamically at runtime, so AI agents can adapt on the fly ([20] www.techradar.com).

Another important feature is **contextual resources**. An MCP server can also publish *resources* (akin to data endpoints) that the model can fetch. For example, a Power BI server could offer a resource representing the latest values in a key metric; the LLM could request "fetch the contents of resource `powerbi://Sales/TotalRevenue`" to get up-to-date figures. This mechanism is powerful for providing the model with fresh context from the database ([21] github.com) ([12] www.techradar.com).

However, MCP as defined by Anthropic does not include built-in enterprise authentication or governance ([8] www.itpro.com) ([9] www.techtarget.com). Critically, the protocol assumes trust: once the AI agent is connected, the MCP server has whatever permissions it is granted. This raises concerns, as noted by experts: if misconfigured, an MCP server could expose sensitive data or be subject to injection attacks ([8] www.itpro.com) ([9] www.techtarget.com). Therefore, a secure implementation must explicitly handle authentication (e.g. require Azure AD tokens) and enforce fine-grained access control on top of MCP. We will incorporate these protections in our design (see Steps 2 and 2 below).

Modern implementations of MCP are emerging rapidly. For example, Microsoft has released an **Azure MCP Server** that implements a suite of tools for managing Azure cloud resources ([10] learn.microsoft.com). Similarly, community members have built servers that connect to popular services. The ms-365-mcp-server exposes Outlook, OneDrive, Excel, and Teams functionality via MCP (www.mcp.pizza). The existence of these examples shows that our goal – integrating Power BI into MCP – is not only feasible but already being explored by others. In particular, an open-source project "powerbi-mcp" (Python-based) already enables natural language queries over Power BI datasets ([7] github.com). We will draw lessons from these when designing our solution.

Power BI REST and Service Endpoints

To plan our MCP server, we should recall the APIs available for Power BI. Microsoft provides comprehensive REST APIs covering embedding, administration, and content management ([22] learn.microsoft.com). Table 2 (below) lists some of the main *operation groups* of the Power BI REST API, illustrating the scope of functionality ([23] learn.microsoft.com) ([24] learn.microsoft.com). These include administration interfaces (users, capacities, admin settings) and content interfaces (datasets, reports, dashboards). Our MCP server can leverage these APIs to perform actual data operations when an LLM requests them. For instance, to list all reports in a workspace, the server might call the `GET /v1.0/myorg/groups/{id}/reports` endpoint.

Operation group	Description
Admin	Administrative tasks (e.g. tenant settings, Power BI capacity Mgmt)
Groups (Workspaces)	Manage workspaces, including membership
Datasets	Manage datasets (refresh, delete, get schema, etc.)
Reports	Retrieve report definitions, metadata, export or create reports
Dashboards	Access dashboards and their tiles
Dataflows	Work with dataflow pipelines
Gateways	Manage on-premises data gateways
Users	Manage user accounts and permissions

Table 2: Sample Power BI REST API operation groups, from Microsoft documentation ([23] learn.microsoft.com) ([24] learn.microsoft.com). A Power BI MCP server can expose tools corresponding to selected operations (e.g. "list reports" or "refresh dataset").

In addition to REST, Power BI offers a **XMLA endpoint** for Premium workspaces, allowing SQL-based queries against datasets (using DAX or SQL) ([7] github.com). This means our MCP server could even support executing arbitrary data queries, not just metadata operations. We will explore both approaches: for many use cases the REST API suffices, but for real querying of large models, XMLA (Analysis Services) may be needed. The "powerbi-mcp" example uses XMLA to let GPT-4 generate DAX queries through the `pyadomd` library ([25] github.com). We will consider whether to incorporate similar capabilities.

Finally, authenticating to Power BI APIs requires Azure AD tokens. The APIs require an AAD-registered *service principal* or app, with the appropriate Power BI permissions granted in the tenant. Step 2 below will cover how to set this up. Once set up, our server code can use libraries (MSAL for Python or Node, or Microsoft.Graph SDK) to acquire tokens and call these APIs programmatically. All of this fits naturally into the MCP tool framework, as each API call can be wrapped into an MCP tool and invoked on demand by the AI agent.

Eight-Step Implementation Guide

The core of this report is a detailed, eight-step plan to build an MCP server for Microsoft Power BI. We enumerate the steps below as numbered subtopics, each with comprehensive discussion. In practice, these steps may overlap or iterate, but we present them linearly for clarity.

1. **Step 1: Define Objectives and Use Cases.** Clarify what queries and commands the MCP server should handle.
2. **Step 2: Set Up Azure AD and API Permissions.** Register an Azure AD application, configure scopes/permissions for Power BI.
3. **Step 3: Prepare Your Power BI Environment.** Ensure you have workspaces and datasets available, and enable necessary features (like XMLA endpoints).
4. **Step 4: Configure the Development Environment.** Choose a programming language/framework (e.g. Node.js or Python), install SDKs, and plan deployment (e.g. containerization).
5. **Step 5: Build the MCP Server Core.** Use the MCP SDK (or equivalent) to create a server that listens for JSON-RPC requests, and can call tools.
6. **Step 6: Implement Power BI Tools and Resources.** Add specific tools to the server (e.g. listReports, queryDataset) that use Power BI APIs under the hood.

7. **Step 7: Test and Iterate.** Validate the MCP server by simulating AI queries (using an MCP client like Claude or a test harness), refine the tool interfaces and prompts.
8. **Step 8: Secure and Deploy.** Containerize or host the server, enforce identity and access controls, and set up monitoring/logging for production use.

Each step will be elaborated with technical details, example code patterns, and citations to authoritative sources. We also highlight relevant considerations (e.g. performance, error handling, business impact) at each stage.

1. Define Objectives and Use Cases

The first step is requirements analysis. Engage stakeholders (data analysts, BI admins, business users) to identify concrete use cases for integrating LLM with Power BI. Possible objectives include:

- **Natural-language querying.** Users should be able to ask broad or specific questions (e.g. "What were last month's total sales by region?" or "Generate a trend analysis for revenue") and receive answers based on the live Power BI data.
- **Automated report generation.** The model could help auto-generate visuals or narrative summaries (e.g. "Create a bar chart of product category sales").
- **Dataset and report management.** Non-technical users might use natural language to request actions like "Refresh the quarterly budget dataset" or "Share Report X with my team".
- **Insight alerts.** The MCP server could support tools for anomaly detection queries (e.g. "Alert me if last week's sales deviated by more than 10% from forecast").
- **Data exploration aids.** Tools to list available tables, columns, or create schemas in response to prompts (e.g. "What tables exist in the Sales dataset?").

Document these use cases and the corresponding data and actions required. For example, if "list reports" is a use case, the server eventually needs a tool that returns report names/title by calling the Power BI API (e.g. GET `/reports` or `/groups/{id}/reports`). If "query the Sales dataset" is a use case, decide whether to use DAX\XMLA or a pre-built query endpoint.

From a business perspective, assess the potential impact of each use case. Analysts and executives often have actionable insights blocked by the need to manually find and combine data. Allowing them to freely query via AI can significantly speed decision-making and reduce reporting backlog. On the other hand, sensitive data (e.g. financials, HR data) may need access restrictions; these must be considered upfront. For example, we may decide that Q&A on restricted datasets requires additional approval or runs in a secure environment.

It is also critical to plan for success metrics: how will you measure the effectiveness of the MCP integration? Possible metrics include time saved per query, number of successful LLM queries answered correctly, user satisfaction scores, or ROI measures (e.g. if directors use this to make faster decisions). According to industry surveys, defining clear success metrics (aligned to business goals) is recommended when introducing AI/analytics ([26] www.techradar.com). This will guide design priorities: we might implement broad data-access tools first if the priority is user productivity, or focus on drill-down queries if technical flexibility is key.

Finally, produce a simple functional specification. This should list the planned MCP tools (such as *ListReports*, *GetDatasetSchema*, *RefreshReport*, *RunDAXQuery*, etc.) with a brief description of what each should do and what inputs/outputs it requires. Document expected user flows: e.g., "User asks AI: 'Show me top 5 products by revenue for Q1.' → LLM invokes *RunDAXQuery(dataset=Sales, dax='TOPN(5, Sales, Sales [Revenue], DESC)')* → server returns a table of results → LLM formats answer." This specification will drive the implementation.

2. Set Up Azure AD and API Permissions

Because Power BI is an Azure service, we use **Azure Active Directory (AAD)** to manage authentication. We must register an AAD application (service principal) that our MCP server will use to call Power BI APIs on behalf of users. Follow Microsoft's guidelines for registering an Azure AD app for Power BI ([27] learn.microsoft.com):

1. **Azure Portal Setup.** In the Azure portal, go to **Azure Active Directory → App registrations → New registration**. Give the app a name (e.g. "PowerBI-MCP-Server"), and set it to be single-tenant (if only for your organization). Note the *Application (client) ID* and *Tenant ID* once created.
2. **API Permissions.** Under the app's settings, go to **API Permissions** and **add** the *Power BI Service* API. You can choose *Delegated* permissions (for user-delegated flows) or *Application* permissions (for daemon/service flows). For a server intended to run without interactive login, a **service principal** with application permissions is typical. The needed permissions might include: `Tenant.Read.All`, `Workspace.Read.All`, `Report.ReadWrite.All`, etc., depending on use cases ([27] learn.microsoft.com). After adding, be sure to **Grant admin consent** for the tenant so the permissions are active.
3. **Create a Client Secret or Certificate.** Under "Certificates & secrets", generate a new client secret. Store the secret value securely (we will use it in our code). Alternatively, for higher security, bind a certificate to the app and use certificate-based auth.
4. **Enable Power BI Service Principal (for app perms).** If using application permissions, you must enable the AAD app as a Power BI service principal in the Power BI admin portal (under Tenant settings → Developer settings). This step allows your app to act within Power BI without user consent ([27] learn.microsoft.com).

After registration, our MCP server code can use the **Microsoft Authentication Library (MSAL)** to acquire OAuth 2.0 tokens with these credentials ([10] learn.microsoft.com). For example, in Node.js one could use `@azure/msal-node`, and in Python the `msal` package. The authentication flow will target either `https://login.microsoftonline.com/{tenantId}` with the client ID and secret, requesting scopes like `https://analysis.windows.net/powerbi/api/.default` (for Power BI service). Getting tokens is discussed in many Microsoft tutorials on using Power BI REST APIs.

It is critical to implement this authentication securely. For instance, ensure the client secret or certificate is **not** hard-coded but stored in environment variables or an Azure Key Vault. The server should also refresh tokens automatically as needed. The cached credentials should have limited scope and be revoked if leaked. By using AAD properly, we align with Microsoft's best practice: as the Azure documentation states, Azure MCP Server "uses Entra ID and Azure Identity library to follow Azure authentication best practices" ([10] learn.microsoft.com). We will mirror this approach (Step 5 will include using MSAL or similar libraries).

3. Prepare Your Power BI Environment

Next, ensure that your Power BI workspace(s) and datasets are configured for API access. This involves three main tasks:

- **Workspace and Dataset Readiness.** Identify the Power BI workspace (group ID) and dataset(s) to target. Ideally use a workspace that is on *Premium capacity* if you plan to leverage advanced features like the XMLA endpoint; otherwise, note that non-Premium workspaces have limitations on direct querying. For example, the open-source *powerbi-mcp* requires a Premium workspace with XMLA enabled ([28] github.com). If Premium is unavailable, the server can still use REST calls for metadata and refresh but may be unable to execute arbitrary SQL/DAX queries.
- **Enable XMLA Endpoints (Premium only).** In the Power BI admin portal (tenant or workspace settings), turn on the XMLA endpoint for your workspace. This will allow Drill-through queries. See Microsoft docs on

Power BI RPM for details. If enabled, we can later connect via `powerbi://api.powerbi.com/v1.0/myorg/{workspace}` using an Analysis Services driver (as *powerbi-mcp* demonstrates).

- **Dataset Access and Permissions.** If using **embed tokens** or user-delegated mode instead of a service principal, ensure the AAD user account running the queries has access to the workspace (viewer role at minimum on the workspace). With a service principal, ensure that the app has the necessary roles or is an admin on the workspace. You can grant a service principal user-level permissions via PowerShell or the admin portal.

Additionally, be mindful of **throttling and quotas**. Power BI REST enforces request limits per user and per dataset ([29] learn.microsoft.com). Design your tools to handle 429 "Too Many Requests" responses by backing off (exponential retry) and caching data where possible. Keeping usage patterns respectful will avoid disrupting normal dashboard refreshes or embedding. Finally, validate you can make a simple API call (for example, list datasets via `GET /v1.0/myorg/datasets`) with your new AAD app credentials before proceeding.

4. Configure the Development Environment

With design and access in place, set up the coding environment and dependencies. Two popular approaches are to build the MCP server in **TypeScript/Node.js** or **Python**. Microsoft's official MCP SDKs are available for both: the modelcontextprotocol/typescript-sdk provides an MCP server framework for Node.js ([21] github.com), while Anthropic and community tools (e.g. `big-whale-labs/mcp-python` on PyPI) support Python.

If choosing Node.js/TypeScript, install the MCP libraries (e.g. `npm i @modelcontextprotocol/sdk`) and use frameworks like Express for the HTTP endpoint as shown in the SDK README ([30] github.com) ([31] github.com). For Python, you may use `fastapi` or Flask to handle JSON-RPC posts. Note: *powerbi-mcp* was built in Python and uses `pythonnet` / `pyadomd` for XMLA and `aiohttp` for serving ([25] github.com) ([32] github.com). Either route is viable; consider the team's expertise and the available libraries.

The environment must include modules for Power BI access: for Node, use @microsoft/microsoft-graph-client or `@azure/msal-node` for AAD; for Python, use the `msal` library and the `requests` or `akita` libraries to call REST endpoints. If using XMLA, you will need ADOC.NET libraries: on Windows, this comes via `Microsoft.AnalysisServices.AdomdClient`; on Linux you might need Docker with the runtime (see *powerbi-mcp*'s Docker instructions ([33] github.com) ([34] github.com)).

Create a configuration file or environment variables file (.env) to store app settings: client ID, secret, workspace ID, dataset names, etc. For example, `CLIENT_ID`, `TENANT_ID`, `CLIENT_SECRET`, `WORKSPACE_ID`, etc. Use secure storage for secrets (e.g. Azure Key Vault). Also decide on logging: keep structured logs for each tool invocation for debugging and audit. (MCP doesn't specify logging, so you must build it – record request metadata, user identity if applicable, and the outcome).

It may help to containerize early. A Docker container ensures all dependencies (like ADOMD.NET) are packaged. The *powerbi-mcp* project provides a sample Dockerfile that includes .NET runtime for Linux ([34] github.com). If using Node, your container would just need Node 18+; if Python, a base image like `python:3.10-slim`. In either case, ensure HTTP ports (e.g. 5000 or 8000) are exposed.

Finally, create a basic "Hello World" MCP server to test the stack. For example, using the TypeScript SDK you might write a node script that registers a dummy tool (like addition) and listens on `/mcp` ([35] github.com) ([36] github.com). Start the server and try invoking the tool through the JSON-RPC interface (the SDK's README provides test examples). Verifying this skeleton works will build confidence before adding Power BI logic.

5. Build the MCP Server Core

With the environment ready, proceed to develop the MCP server itself. Using the chosen SDK or framework, implement the core server that will accept requests and route them to tools. The principal tasks are:

- **Initialize the server.** Create a new `McpServer` instance (or equivalent). In TypeScript, `new McpServer({ name: 'powerbi-mcp', version: '1.0.0' })` ([31] github.com). Provide descriptive metadata.
- **Configure the transport.** Decide how AI clients will connect. The simplest is HTTP (e.g., POST to `http://localhost:5000/mcp`). For example, with Express.js you can forward incoming requests to the MCP server's `handleRequest` method ([36] github.com). If multiple concurrent sessions are needed, ensure a thread-safe setup (e.g., one `McpServer` can support many MQTT-like sessions via unique IDs). Alternatively, other transports (std I/O, SSE) could be used, but HTTP covers most use cases.
- **Session management.** MCP supports stateful interactions: a given "session ID" allows the agent to maintain context across multiple tool calls. The server core should generate or track `sessionId` for each request stream. The TypeScript SDK example creates a new `StreamableHTTPServerTransport` with a unique session ID per HTTP request ([36] github.com). In Python, you might similarly tie session IDs to WebSocket connections or HTTP sessions. This allows, for example, the agent to open a conversation, call multiple tools in sequence, and finally receive a combined answer.
- **Define schemas and validation.** MCP encourages tools to specify input and output schemas for arguments (often with a library like Zod in TS or Pydantic in Python). This lets the system check LLM output consistency. You should design clear schemas for each tool (e.g. `GetReport(input: {reportName: string}) => { reportData: ... }`). The TypeScript example in the SDK demonstrates adding an "Addition Tool" with input schema `{a: number, b: number}` ([35] github.com). We will do similar, but with business-specific schemas.

At this core level, also incorporate **authentication middleware**. Since MCP itself is not authenticated, we must enforce our Azure AD auth on incoming requests. One approach is to require that each client call includes a bearer token or API key in headers. Your HTTP endpoint handler can validate this before passing to the MCP server. For example, you might require clients to first call a `/login` tool (as in some MCP servers) which returns an AAD authorization URL. However, a simpler approach is to handle auth outside of MCP: e.g. run the server behind Azure API Management or a reverse proxy that enforces AAD tokens. The ThriveTech's "ms-365-mcp-server" FAQ suggests calling a `login` tool to obtain a URL and code (www.mcp.pizza). For Power BI, because we likely use a service principal, our server can authenticate once on startup and simply use the token internally. We will clarify this in Step 6.

Finally, implement global error handling. Any exceptions in a tool should be caught and returned as a JSON-RPC error to the client, to prevent attacks (e.g. injection via bad inputs). Use logging and monitoring for unhandled exceptions. Remember that the MCP server is intended for trusted internal use (per Microsoft), so do not expose it publicly in unsecured fashion ([37] learn.microsoft.com).

By the end of Step 5, you should have a runnable MCP server that, for example, can accept a "ping" or sample tool call and respond. This serves as the framework into which we will plug the Power BI-specific logic.

6. Implement Power BI Tools and Resources

This is the core step: encoding Power BI functionality as MCP tools. Based on Step 1's requirements, choose a set of tools to register with the MCP server. Each tool typically has:

- **Name and description.** E.g. `listReports` with title "List all reports in a workspace" and a brief description.

- **Input schema.** What parameters it takes (workspace ID, report ID, query text, etc.).
- **Output schema.** The structured result it returns (e.g. an array of objects, or status).
- **Handler function.** The actual code that calls Power BI and returns results.

Below is an illustrative (not exhaustive) list of possible tools. These align with the Power BI REST API operations from Table 2:

Tool Name	Description	Inputs	Underlying API Call/Action
`listReports`	List all reports in a Power BI workspace.	`workspaceId: string`	Calls `GET /v1.0/myorg/groups/{workspaceId}/reports` to retrieve report metadata ([23] learn.microsoft.com).
`runDatasetQuery`	Execute a DAX or SQL query on a Power BI dataset.	`workspaceId`, `datasetId`, `query`	If using XMLA: connect to `powerbi://api.powerbi.com/v1.0/myorg/{workspace}` with ADOMD, execute query. Alternatively, use REST if available.
`refreshDataset`	Trigger a refresh of a dataset (e.g. nightly data load).	`workspaceId`, `datasetId`	Calls `POST /v1.0/myorg/groups/{workspaceId}/datasets/{datasetId}/refreshes` to queue a refresh.
`getTableSchema`	Return table/column schema for a dataset.	`workspaceId`, `datasetId`	Calls `GET /v1.0/myorg/groups/{workspaceId}/datasets/{datasetId}` and/or XMLA to retrieve table/column info.
`listWorkspaces`	List available workspaces accessible to the service principal.	(no inputs)	Calls `GET /v1.0/myorg/groups` to list groups the app has access to.
`createReport`	Export or create a report from JSON.	`reportName`, `workspaceId`	Example: use `ExportToFile` API or `POST /reports` to import a pbix. (optional advanced feature.)

(Table 3: Sample MCP tools for Power BI. The server developer defines the input/output schemas and implements each using the appropriate Power BI REST API or XMLA call.)

For each tool, write the handler code carefully. For example, in JavaScript/TypeScript using the modelcontextprotocol SDK, you might do (pseudocode):

```
server.registerTool(
 'listReports',
 {
 title: 'List Power BI Reports',
 description: 'Retrieve all report names in the workspace.',
 inputSchema: z.object({ workspaceId: z.string() }),
 outputSchema: z.object({ reports: z.array(z.string()) })
 },
 async ({ workspaceId }) => {
 // Acquire Power BI REST API token using MSAL
```

```
  const token = await acquireToken();
  // Call Power BI API
  const response = await axios.get(
  `https://api.powerbi.com/v1.0/myorg/groups/${workspaceId}/reports`,
  { headers: { Authorization: `Bearer ${token}` } }
  );
  const reportNames = response.data.value.map((r:any) => r.name);
  // Return content to MCP
  return {
  content: [{ type: 'text', text: `Reports: ${reportNames.join(', ')}` }],
  structuredContent: { reports: reportNames }
  };
  }
 );
```

In Python, the approach is similar: use `requests` or the official Power BI SDK to call the API. For `runDatasetQuery`, if you have XMLA, use `pyadomd` or `adodbapi` to connect to the dataset and execute the query text. Note that responses may be large; consider truncating or summarizing results before returning to the LLM.

Remember to integrate **error handling** in each tool. If the REST call returns an error (such as 401 Unauthorized or 429 Throttling), catch it and return an MCP error message. For example, if `refreshDataset` is called on a non-existent dataset, the server should send a helpful error like “Dataset not found or access denied.”

Since the tools need to feel “natural” to an AI, pay attention to naming and descriptions. The LLM will receive the title and description and may append them to the prompt to decide which tool to call ([20] www.techradar.com). Ensure descriptions are clear. For instance, rather than “getWSR” use “Get Workspace Reports” and a sentence like “Returns the list of report names in a given Power BI workspace”.

Additionally, MCP allows returning “structured content” alongside text. We should use that: return a JSON object (`structuredContent`) of the key data (e.g. `{ reports: [...] }`) so that the LLM can reason with the raw data and not just the text. This makes it easier for the LLM to generate narrative answers or to chain calls.

After defining tools, you may also define “resources.” For instance, if there’s a frequently-needed piece of data (like “Sales figures for current month”), you could set up a resource URI that the LLM can fetch via the `ResourceTemplate`. However, for a first implementation, focusing on tools is usually enough.

Finally, update the server to register all tools before starting to listen. Then, rebuild and restart the server. At this point, your MCP server is functionally ready: it knows how to do Power BI actions. Next, we must test it end-to-end.

7. Test and Iterate on AI Integration

With the MCP server running local or in a dev environment, we now test it by simulating AI client interactions. Use an MCP-compatible client – options include the **MCP Inspector** from modelcontextprotocol.io, Claude Desktop in agent mode, or even a simple script using the MSC (MCP Streamable HTTP) for testing. For example, [31] suggests using the Inspector CLI:

```
npx @modelcontextprotocol/inspector http://localhost:3000/mcp
```

This will prompt you to enter JSON-RPC commands manually. First, try the **login** tool (if you implemented one) or directly invoke a tool like `listWorkspaces`. For instance, send:

```
{ "id": 1, "jsonrpc": "2.0", "method": "tool:listWorkspaces", "params": {} }
```

and verify the response contains the expected list of workspaces. Then try `listReports` on one workspace ID returned. Use the JSON-RPC format carefully (strings vs types). Debug any issues: common problems are token expiration (solution: refresh token), permission denied (check AAD roles), or mis-typed resource IDs.

After basic tests, connect a real LLM client. For example, Claude Desktop supports adding custom MCP servers (www.mcp.pizza). Add your server with the command (as in the *ms-365-mcp-server* instructions) by specifying the command to run your server (or the URL if already running). Then use Claude: ask it a question like "List all reports in the Sales workspace." The agent should choose `listReports` if your tool descriptions were clear. Check that the answer is correct. Try multi-step conversations: e.g., "Which workspace has the Sales dataset?", "Then list reports in that workspace." This tests session persistence.

Iterate on prompts and output formatting. If the LLM's response is unclear, you may need to adjust the tool's output schema or how results are returned. For example, ensure that the tool's `content` text is well-phrased and that the structured data is accurate. If the LLM misunderstands, refine the **tool descriptions** or add example usages in the tool's prompt hint. Also, consider edge cases: what if the workspace has no reports? The tool should return an empty list gracefully, not an error.

Benchmark performance too. Trigger a heavy tool like a large dataset query and measure response time. If it's too slow (e.g. >10s), think about optimizations: perhaps limit the number of returned rows, or pre-aggregate data. Implement caching for repeated metadata calls (workspace list or table schemas) to avoid redundant API calls. The *powerbi-mcp* code, for example, uses asynchronous operations and caching for better performance ([25] github.com). If needed, add short-term in-memory caching to your server using a key (e.g. "workspace_{id}_reports").

Finally, gather user feedback from early testers. They might ask for additional tools (e.g. "Delete this report" or "Get dataset usage info") or features (like sending the result to email). Incorporate these as extra MCP tools if time permits. The iterative development resembles agile methodology: deliver minimal functionality first (list/query), then refine and expand.

8. Secure and Deploy the MCP Server

Once the MCP server is tested, productionalize it. Several best practices apply:

- **Containerization.** Package the server into a Docker container (or similar) for consistent deployment. The Dockerfile should set environment variables (client ID, secret, etc.) from secure sources and expose the configured port (e.g. 5000). The *powerbi-mcp* example demonstrates a container that automatically includes .NET on Linux ([34] github.com). If using Node, the Node image is simpler. Ensure your .env file is **excluded** from the image (dockerignore) to avoid leaking secrets ([32] github.com) ([38] github.com). Instead, pass secrets at runtime (`docker run -e CLIENT_SECRET=xxx ...`).
- **Authentication and Networking.** Do not expose the MCP server endpoint publicly without protection. It should live inside a secured network segment or behind a reverse proxy. Ideally, require inbound connections only from approved developer machines or agent-hosting services. Optionally, integrate with Azure AD for inbound auth: for example, set up Azure API Management or Azure App Service Authentication (EasyAuth) so that any request to `/mcp` must present a valid AAD token from a known client. This is in addition to the client identity we built in Step 2. Note that Microsoft recommends using Azure AD and RBAC with MCP for Azure resources ([10] learn.microsoft.com) – we should mirror that.

- **Logging and Monitoring.** Implement centralized logging. Log every MCP call, including the tool name, input parameters, who/what invoked it, and timestamp. Because MCP interactions treat the LLM as the "user", include any user context you have (perhaps require users to supply a username as part of the request, or run the MCP server under a specific service account). Store logs in a secure repository (Azure Monitor or an ELK stack). Also track metrics: e.g. requests per tool per hour, average latency, error rates. This will help you observe usage patterns and troubleshoot issues. Ensure logs do not inadvertently capture sensitive data (avoid logging full query results unless necessary).

- **Access Control.** Use Power BI's own RBAC to control data scope. For instance, if your AAD app has workspace-level access, it should only see those workspaces. Within each workspace, define roles (Admin, Member, Viewer) so that even if an LLM queries it, the underlying data privacy is upheld. If fine-grained row-level security (RLS) is set up on datasets, these filters will still apply to queries run through the server. In other words, respect the existing security model of Power BI in your server.

- **Refresh and Versioning.** Establish a process for updating the MCP server when needed (bug fixes or feature requests). Tag Docker images and maintain a private registry. Plan for continuous integration: each commit could trigger a build and run automated tests (e.g. verifying tool endpoints). Include version metadata in the MCP server (e.g. "version": "2025.11.13") so it's clear what release is running.

- **Approval and Documentation.** Before giving broad access, get sign-off from security and governance teams. Provide documentation for administrators: how to run and restart the server, set environment variables, adjust AAD settings, and monitor logs. Also prepare user documentation: how to invoke the AI agent, what commands to use, and what to do if something goes wrong (e.g. "please ask your IT support if the system fails"). Regularly review who has access to the MCP server and its logs.

Following these deployment practices ensures that the MCP server is not only functional but also robust, maintainable, and secure in a production environment. As Microsoft notes, a local MCP server is intended for developer use within the organization and should not be exposed outside the approved development environment ([37] learn.microsoft.com). By confining it to internal use and strict authentication, we align with that guidance.

Case Studies and Industry Examples

Several real-world examples demonstrate the concepts above. One illustrative case is the open-source **PowerBI-MCP Server** by Sulaiman Ali (2024) ([7] github.com). This Python-based server embodies many of the ideas we discuss: it uses a service principal with Azure AD (for `msal`) ([7] github.com) ([28] github.com) and connects to Power BI via the XMLA endpoint. Its features include automatic DAX generation (using GPT-4) and table discovery, which are advanced capabilities. Importantly, it shows how an MCP server can be packaged in Docker (with .NET runtime for Linux) and how tools can be defined for interactive querying ([25] github.com) ([32] github.com). Studying such a project provides practical insights: for example, it uses endpoint `/sse` and JSON-RPC `/messages/` to communicate ([34] github.com), which our server could emulate if needed.

An organizational case: consider a retail firm with thousands of daily sales records. The BI team has built Power BI dashboards for store managers. By deploying an MCP server, the company integrated ChatGPT for Business (hypothetical) so managers could type "Which store had the highest sales last month, and why?" The AI agent used the MCP server's `runDatasetQuery` tool to fetch sales data and an analysis tool to compute the answer – all in seconds. This replaced a manual "export to spreadsheet" process that took hours each month. While this is a fictionalized account, it aligns with industry trends: a report by TechRadar indicates that *97% of data analysts already use AI* (broadly defined) to streamline analytics tasks ([26] www.techradar.com). Those tools often reduce errors and reveal insights missed by humans.

Another perspective comes from enterprise IT: the Azure MCP Server documentation itself anticipates scenarios like "list storage accounts" or "run Kusto queries" via natural language ([39] learn.microsoft.com). Extending that idea to business data is analogous. Just as DevOps teams might ask their code editor's Copilot to "How do I scale this database?", business users could ask the data: "How do I scale sales in Region X?" via our MCP server-backed AI. The core is the same: *providing live context and actions to the model*.

Finally, surveys suggest that firms integrating AI with analytics need a change in culture: in our Step 1 we emphasized objectives and training. The MoldStud research (Table 1) highlighted that **training** is a major success factor ([6] moldstud.com). In practice, the "user" of an MCP server includes non-technical business users, so change management is key. For example, companies that succeed often hold workshops on how to phrase questions to an AI assistant and how to interpret its answers. This underscores the need in Step 1 to align the project with organizational learning strategies.

Implications, Benefits, and Future Directions

Building an MCP server for Power BI is not just a technical exercise; it has strategic implications for how enterprises harness AI and data. We summarize key observations:

- **Bridging Generative and Analytical AI.** Analytical AI (transforming data into insights) is proven to drive business value, but has lagged adoption ([4] www.techradar.com). By connecting an LLM with live BI data, we blend generative (the LLM's language abilities) with analytical power, giving leaders a "best of both worlds" tool. Microsoft itself is moving in this direction (e.g. "Power BI GPT" features announced in 2023), so our MCP approach is aligned with industry direction. Ultimately, users can ask more sophisticated questions than canned Q&A forms, effectively making every dashboard "conversational".

- **Enhanced Productivity and ROI.** As shown in Table 1 and elsewhere ([14] moldstud.com) ([26] www.techradar.com), AI can significantly increase efficiency. Analysts spend many hours preparing data – MCP-powered queries reduce that time. An eight-step implementation is an upfront investment; future calculations should quantify its ROI. One can measure metrics like reductions in report turnaround time, user adoption of the chat interface versus old methods, and business metrics impacting decisions. For example, if a CFO takes weekly strategy calls based on AI answers instead of waiting on analyst reports, that speed has real value.

- **Security and Governance Challenges.** MCP creates a powerful new attack surface: an internal server that can query data indiscriminately. We must therefore emphasize security. This report's steps incorporate strict AAD authentication and following Microsoft's guidelines to keep the server internal ([37] learn.microsoft.com). Looking ahead, organizations may push for enhancements in MCP itself — for instance, adding formal support for fine-grained permissions or built-in user identity propagation. Indeed, experts have noted these gaps (MCP-UPD and related risks ([8] www.itpro.com) ([9] www.techtarget.com)). Enterprises building such solutions should engage with the MCP standard community (Anthropic, AI Alliance) to shape secure protocols.

- **Standardization and Tooling Evolution.** MCP is still nascent (launched Nov 2024), but momentum is growing. Major AI tools (GitHub Copilot, Claude, Microsoft's Copilot) are adopting it ([10] learn.microsoft.com) ([12] www.techradar.com). This suggests that investing in an MCP server today will pay dividends as tools mature. In the future, we may see an ecosystem of MCP "plugins" for various services, much like web plugins. For example, a marketplace of MCP servers could allow quickly adding new data sources. For Power BI specifically, one could imagine Microsoft releasing first-party MCP tools for common BI tasks, which we would then simply adopt.

- **Impact on Data Culture.** The adoption of an MCP server also encourages a *data-centric culture*. By making data interrogable in plain language, we reduce siloes between IT and business teams. Staff who were hesitant to learn SQL or DAX can now participate in analysis. This democratization supports a "data culture" — a concept even Microsoft underscores in its Power BI messaging ([3] powerbi.microsoft.com). However, it requires careful governance: companies must update policies on data access, AI usage, and audit trails. Over time, such systems might even enforce ethical or regulatory constraints (e.g. not answering certain queries), a direction worth planning for.

- **Future Feature Possibilities.** Looking forward, this architecture could be extended. For example, one might add real-time context by feeding streaming data into the MCP server as resources. Or integrate with other Azure services (e.g. let the LLM call an Azure Function through MCP to run custom analytics). The combination of MCP and Power BI could form the backbone for more advanced "AI analytics agents." The use of *agentic AI* (multiple AI agents calling each other's tools) is already emerging ([40] www.techtarget.com). In such a world, our Power BI MCP server could be one module in a network of AI agents, collaborating to solve larger business problems.

In summary, the architecture we propose is both timely and forward-looking. It addresses current business needs for faster data insights while laying groundwork for future AI-augmented analytics. Organizations undertaking this project should remember that the technical steps (1–8 above) are one dimension; equal attention must go to people, process, and policy. As the TechTarget analysis warns, a true standard like MCP only succeeds if the industry collaborates ([41] www.techtarget.com). By following a structured approach and by sharing lessons learned (as in this report), developers contribute to that broader ecosystem.

Conclusion

This report has presented an in-depth blueprint for building an MCP server to integrate Microsoft Power BI with AI models in eight steps. We began by motivating the endeavor: the convergence of AI and BI offers enormous potential for smarter decision-making ([3] powerbi.microsoft.com) ([4] www.techradar.com). We then reviewed the technical foundations – the scale of Power BI's usage, the capabilities of its REST APIs ([17] learn.microsoft.com), and the new Model Context Protocol that enables tool use by LLMs ([1] www.itpro.com) ([2] www.techradar.com).

The core of the report outlined the eight steps in detail. First, we stressed defining clear requirements and use cases, aligning them to business goals and training needs (echoing the 67% success rate tied to training ([6] moldstud.com)). We then covered the Azure AD setup and Power BI preparation needed to securely connect to data sources. The subsequent steps walked through environment setup, coding the MCP server core, fleshing out Power BI-specific tools, and rigorous testing. At each point we cited documentation and examples: for instance, leveraging the Azure MCP Server for guidance ([10] learn.microsoft.com) and studying existing implementations like the "powerbi-mcp" project ([7] github.com). The final step emphasized secure deployment, mirroring Microsoft's caution that MCP should remain internal to the organization ([37] learn.microsoft.com) and be protected by proper auth and RBAC ([8] www.itpro.com) ([9] www.techtarget.com).

In analyzing outcomes, we drew on industry research to demonstrate value. Automated analytics processes can reduce reporting time by ~50% and improve accuracy by ~25–33% in many companies ([14] moldstud.com). With an MCP server, each user query becomes an opportunity to realize such gains. Nevertheless, we also acknowledged emerging caveats. Surveys indicate some enterprises are slowing AI adoption ([42] www.tomshardware.com), underscoring the need to prove ROI and handle data responsibly. Our approach emphasizes trust: using Azure AD and logging to meet auditors' concerns.

Looking ahead, the implications are significant. MCP servers could become standard components of enterprise analytics toolkits, enabling new forms of interactive reporting. Our eight-step guide should serve as a template not only for Power BI, but for any data service one wants to make "AI-ready." As the analytical AI landscape evolves, solutions like this will help organizations stay competitive by turning data into actionable intelligence. Finally, we reiterate that thorough documentation and citation of best practices (as done here) are key: enterprises should apply evidence-based methods, grounded in both technical references (Microsoft docs, SDKs) and empirical studies ([10] learn.microsoft.com) ([6] moldstud.com). By doing so, they will harness the full potential of AI in business intelligence while managing its risks, thus preparing for a future where intelligent data agents are integral to decision-making.

References: This report has drawn on a broad range of sources: Microsoft official documentation on Power BI and MCP ([10] learn.microsoft.com) ([17] learn.microsoft.com), tech media analysis of the Model Context Protocol ([20] www.techradar.com) ([12] www.techradar.com), industry research on AI in analytics ([6] moldstud.com) ([4]

www.techradar.com), and relevant open-source projects and case studies (www.mcp.pizza) ([7] github.com). Each factual claim above is backed by the cited literature.

External Sources

[1] https://www.itpro.com/technology/artificial-intelligence/what-is-model-context-protocol-mcp#:~:Model...

[2] https://www.techradar.com/pro/the-4-most-critical-aspects-of-model-context-protocol-mcp-for-developers-building-ai-native-architectures#:~:1.%20...

[3] https://powerbi.microsoft.com/fi-fi/blog/announcing-new-ai-and-enterprise-features-for-power-bi/#:~:In%20...

[4] https://www.techradar.com/pro/why-analytical-ai-deserves-equal-attention-in-the-age-of-generative-ai#:~:The%2...

[5] https://moldstud.com/articles/p-the-impact-of-ai-on-decision-making-in-business-intelligence-strategies#:~:Integ...

[6] https://moldstud.com/articles/p-the-impact-of-ai-on-decision-making-in-business-intelligence-strategies#:~:Area%...

[7] https://github.com/sulaiman013/powerbi-mcp#:~:A%20M...

[8] https://www.itpro.com/technology/artificial-intelligence/what-is-model-context-protocol-mcp#:~:Howev...

[9] https://www.techtarget.com/searchenterpriseai/news/366616516/Anthropics-new-standard-raises-AI-privacy-other-concerns#:~:While...

[10] https://learn.microsoft.com/en-us/azure/developer/azure-mcp-server/overview#:~:The%2...

[11] https://www.techradar.com/pro/the-future-of-ai-applications-mcp-servers#:~:The%2...

[12] https://www.techradar.com/pro/the-future-of-ai-applications-mcp-servers#:~:MCP%2...

[13] https://powerbi.microsoft.com/fi-fi/blog/announcing-new-ai-and-enterprise-features-for-power-bi/#:~:But%2...

[14] https://moldstud.com/articles/p-the-impact-of-ai-on-decision-making-in-business-intelligence-strategies#:~:Utili...

[15] https://moldstud.com/articles/p-the-impact-of-ai-on-decision-making-in-business-intelligence-strategies#:~:Moreo...

[16] https://powerbi.microsoft.com/fi-fi/blog/announcing-new-ai-and-enterprise-features-for-power-bi/#:~:%E2%9...

[17] https://learn.microsoft.com/en-us/rest/api/power-bi/#:~:...

[18] https://www.techradar.com/pro/why-analytical-ai-deserves-equal-attention-in-the-age-of-generative-ai#:~:The%2...

[19] https://www.techtarget.com/searchenterpriseai/news/366616516/Anthropics-new-standard-raises-AI-privacy-other-concerns#:~:MCP%2...

[20] https://www.techradar.com/pro/the-4-most-critical-aspects-of-model-context-protocol-mcp-for-developers-building-ai-native-architectures#:~:1.%20...

[21] https://github.com/modelcontextprotocol/typescript-sdk#:~:The%2...

[22] https://learn.microsoft.com/en-us/rest/api/power-bi/#:~:The%2...

[23] https://learn.microsoft.com/en-us/rest/api/power-bi/#:~:Opera...

[24] https://learn.microsoft.com/en-us/rest/api/power-bi/#:~:Pipel...

[25] https://github.com/sulaiman013/powerbi-mcp#:~:,Prin...

[26] https://www.techradar.com/pro/ai-and-automation-are-here-to-settle-roi-questions-of-analytics#:~:workf...

[27] https://learn.microsoft.com/en-us/rest/api/power-bi/#:~:To%20...

[28] https://github.com/sulaiman013/powerbi-mcp#:~:,opti...

[29] https://learn.microsoft.com/en-us/rest/api/power-bi/#:~:Throt...

[30] https://github.com/modelcontextprotocol/typescript-sdk#:~:Let%2...

[31] https://github.com/modelcontextprotocol/typescript-sdk#:~:%2F%2...

[32] https://github.com/sulaiman013/powerbi-mcp#:~:%E2%9...

[33] https://github.com/sulaiman013/powerbi-mcp#:~:Syste...

[34] https://github.com/sulaiman013/powerbi-mcp#:~:The%2...

[35] https://github.com/modelcontextprotocol/typescript-sdk#:~:%2F%2...

[36] https://github.com/modelcontextprotocol/typescript-sdk#:~:app.p...

[37] https://learn.microsoft.com/en-us/azure/developer/azure-mcp-server/overview#:~:The%2...

[38] https://github.com/sulaiman013/powerbi-mcp#:~:Impor...

[39] https://learn.microsoft.com/en-us/azure/developer/azure-mcp-server/overview#:~:Scena...

[40] https://www.techtarget.com/searchenterpriseai/news/366616516/Anthropics-new-standard-raises-AI-privacy-other-concerns#:~:Anthr...

[41] https://www.techtarget.com/searchenterpriseai/news/366616516/Anthropics-new-standard-raises-AI-privacy-other-concerns#:~:Howev...

[42] https://www.tomshardware.com/tech-industry/artificial-intelligence/ai-adoption-rate-is-declining-among-large-companies-us-census-bureau-claims-fewer-businesses-are-using-ai-tools#:~:A%20r...

IntuitionLabs - Industry Leadership & Services

North America's #1 AI Software Development Firm for Pharmaceutical & Biotech: IntuitionLabs leads the US market in custom AI software development and pharma implementations with proven results across public biotech and pharmaceutical companies.

Elite Client Portfolio: Trusted by NASDAQ-listed pharmaceutical companies including Scilex Holding Company (SCLX) and leading CROs across North America.

Regulatory Excellence: Only US AI consultancy with comprehensive FDA, EMA, and 21 CFR Part 11 compliance expertise for pharmaceutical drug development and commercialization.

Founder Excellence: Led by Adrien Laurent, San Francisco Bay Area-based AI expert with 20+ years in software development, multiple successful exits, and patent holder. Recognized as one of the top AI experts in the USA.

Custom AI Software Development: Build tailored pharmaceutical AI applications, custom CRMs, chatbots, and ERP systems with advanced analytics and regulatory compliance capabilities.

Private AI Infrastructure: Secure air-gapped AI deployments, on-premise LLM hosting, and private cloud AI infrastructure for pharmaceutical companies requiring data isolation and compliance.

Document Processing Systems: Advanced PDF parsing, unstructured to structured data conversion, automated document analysis, and intelligent data extraction from clinical and regulatory documents.

Custom CRM Development: Build tailored pharmaceutical CRM solutions, Veeva integrations, and custom field force applications with advanced analytics and reporting capabilities.

AI Chatbot Development: Create intelligent medical information chatbots, GenAI sales assistants, and automated customer service solutions for pharma companies.

Custom ERP Development: Design and develop pharmaceutical-specific ERP systems, inventory management solutions, and regulatory compliance platforms.

Big Data & Analytics: Large-scale data processing, predictive modeling, clinical trial analytics, and real-time pharmaceutical market intelligence systems.

Dashboard & Visualization: Interactive business intelligence dashboards, real-time KPI monitoring, and custom data visualization solutions for pharmaceutical insights.

AI Consulting & Training: Comprehensive AI strategy development, team training programs, and implementation guidance for pharmaceutical organizations adopting AI technologies.

Contact founder Adrien Laurent and team at https://intuitionlabs.ai/contact for a consultation.