how MCP (Model Context Protocol) works in an MCP dashboard in a clear, step-by-step way:

✅ 1. What is MCP?

MCP (Model Context Protocol) is a protocol designed to connect **AI models** with **external tools, APIs, and data sources** in a structured way. It acts as a **bridge** between your AI agent and the services you want to integrate (e.g., Bitrise API, GitHub, CI/CD pipelines).

✅ 2. Core Components of MCP

- **Server**: Runs the MCP implementation and exposes endpoints for communication.
- **Transport Layer**: Handles the connection between the dashboard and MCP (usually via HTTP or SSE).
- **Tools & Resources**: These are the actual integrations (e.g., Bitrise API, database, file system).
- **Client (Dashboard)**: Your React-based UI that interacts with MCP to fetch and display data.

✅ 3. How MCP Works in a Dashboard

Here's the flow:

1. **Dashboard Initialization**
    - Your React app starts and connects to the MCP server using a transport (e.g., StreamableHTTPServerTransport).
    - It establishes a **session** for communication.
2. **Tool Registration**
    - MCP server registers tools like:
        - bitrise.getApps
        - bitrise.getBuildStatus
    - These tools define what actions the dashboard can perform.
3. **Request-Response Cycle**
    - The dashboard sends a **request** to MCP (e.g., getApps).
    - MCP server calls the corresponding tool, fetches data from Bitrise API, and returns the result.
4. **Streaming Updates**
    - For real-time updates (e.g., build status), MCP uses **Server-Sent Events (SSE)** or WebSockets.
    - The dashboard listens for these events and updates the UI dynamically.

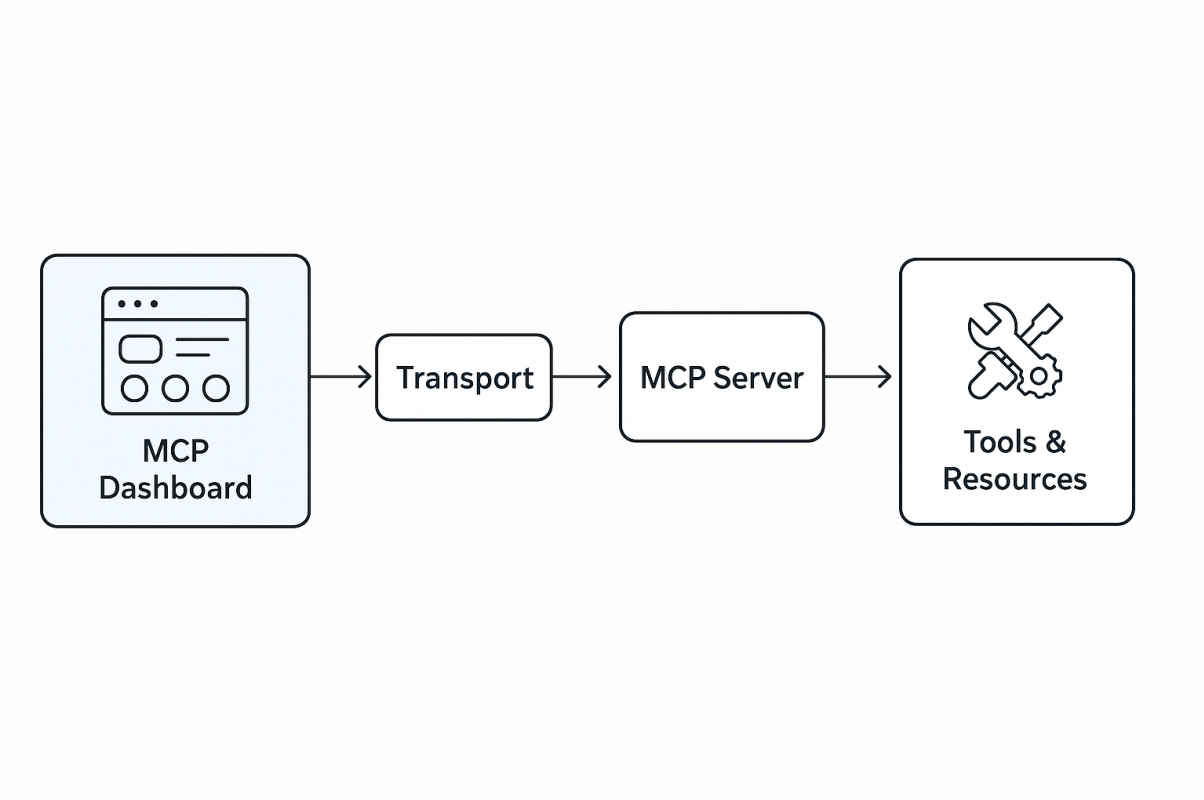
5. **Context Management**
    - MCP maintains **context** for the session, so the dashboard can make multiple related requests without re-authentication.

✅ 4. Example Workflow in Your Dashboard

- **User opens dashboard → React app connects to MCP server**
- **Dashboard calls MCP tool → MCP fetches data from Bitrise API**
- **MCP returns JSON → Dashboard renders build statuses (Success, Failed, In Progress)**

✅ 5. Why MCP is Useful

- **Standardized Protocol**: No need to hardcode API logic in the dashboard.
- **Scalable**: Easily add new tools (e.g., GitHub, Jira) without changing the UI logic.
- **Real-Time Updates**: SSE/WebSocket support for live pipeline status.



Here's a clean **architecture diagram** of an MCP-enabled dashboard.

How the pieces interact (typical flow)

1. **Dashboard boots** → initializes the MCP client and connects over **Transport** (HTTP/SSE).
2. **Session established** → auth headers (e.g., Authorization: Bearer <token>) are passed securely.
3. **UI requests data** → e.g., call bitrise.getApps / bitrise.getBuildStatus.
4. **MCP Server resolves tool** → invokes the registered Bitrise tool; handles rate limits, retries, and error mapping.
5. **Results stream back** → **SSE/WebSocket** events update the dashboard in real time (Running ▸ Success ▸ Failed).
6. **State updates** → UI components rerender (cards, tables, badges); context maintained for subsequent calls.

Key responsibilities

- **Dashboard (Client)**
  - Renders UI, manages app state, and calls MCP tools.
  - Subscribes to **live streams** for build updates.
- **Transport**
  - Keeps a reliable channel (SSE/WebSocket) and propagates session/auth.
- **MCP Server**
  - **Registers tools**, validates inputs, enforces **permissions**, and returns structured results.
  - Emits **events** for live status (build started/finished).
- **Tools & Resources**
  - Actual integrations (Bitrise, GitHub, Jira, DBs, files).
  - Use service accounts/API keys stored in a **secure vault**.

Optional enhancements

- **Caching layer**: memoize frequent reads (e.g., app list) to reduce API calls.
- **RBAC**: role-based access (Support Analyst vs Admin).
- **Observability**: logs + metrics per tool (latency, errors).
- **Backoff & retries**: especially for CI/CD APIs with quotas.

While both **API Gateway** and **MCP Server** seem to expose APIs to a dashboard, they serve **different purposes and operate at different layers**. Here's the detailed comparison:

✅ 1. API Gateway

- **Purpose**:
  Acts as a **reverse proxy** and entry point for multiple backend services. It routes, secures, and manages API traffic.
- **Key Features**:
  - **Routing**: Directs requests to appropriate microservices.
  - **Security**: Handles authentication, authorization, rate limiting.
  - **Transformation**: Converts request/response formats (e.g., JSON ↔ XML).
  - **Aggregation**: Combines data from multiple services into one response.
  - **Monitoring**: Tracks API usage, logs, and metrics.
- **Use Case**:
  - If your dashboard needs to call multiple APIs (Bitrise, GitHub, Jira), an API Gateway can unify them under one endpoint like /api/*.

✅ 2. MCP Server

- **Purpose**:
  Implements **Model Context Protocol**, which is not just routing but **context-aware orchestration for AI agents**. It provides structured access to tools and resources for AI workflows.
- **Key Features**:
  - **Tool Registration**: Defines actions like getBuildStatus, getApps.
  - **Context Management**: Maintains session state for AI reasoning.
  - **Streaming**: Supports SSE/WebSocket for real-time updates.
  - **Structured Protocol**: Uses MCP schema for requests/responses (not raw REST).
  - **AI Integration**: Designed for LLMs or agentic workflows, not just dashboards.
- **Use Case**:
  - If your dashboard interacts with an **AI agent** or needs **contextual tool execution** (e.g., "Fetch Bitrise builds and summarize failures"), MCP is the right choice.

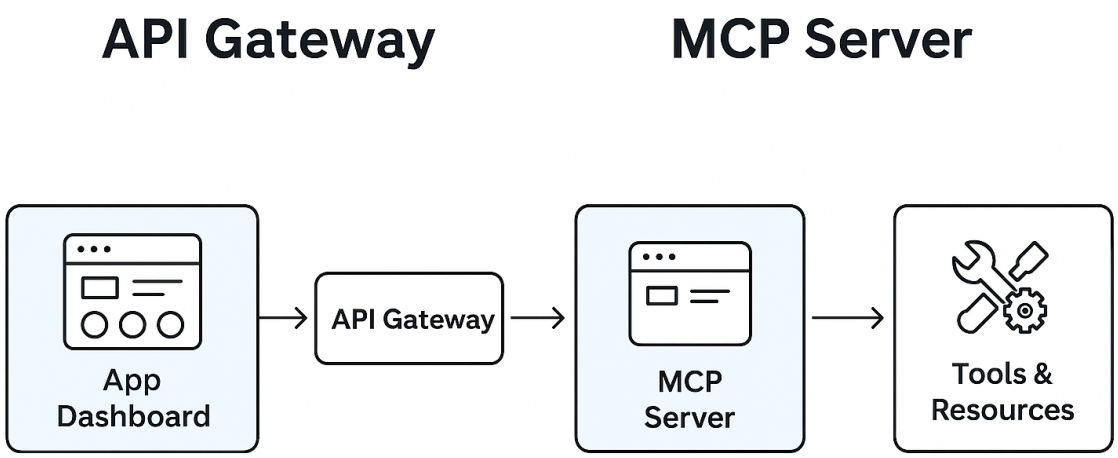
✅ Key Differences

Feature	API Gateway	MCP Server
Primary Role	API traffic management	AI tool orchestration & context handling
Protocol	REST, GraphQL, gRPC	MCP (structured JSON schema)
Focus	Routing & security	Context-aware tool execution
Real-time Updates	Optional (via WebSocket)	Built-in SSE/WebSocket for streaming
AI Integration	No	Yes

✅ Analogy

- **API Gateway** = Traffic cop for APIs (routes cars to correct roads).
- **MCP Server** = AI assistant's toolbox manager (knows which tool to use and why).

Here's a side-by-side comparison diagram of **API Gateway vs MCP Server** in the context of a dashboard.



API Gateway focus: routing, aggregation, auth, throttling, observability.
MCP Server focus : tool orchestration, context-aware requests, SSE/WebSocket events, AI/agent workflows.

Add Bitrise API calls to diagram