Building Scalable, Edge Native Production Grade MCP Tools with RUST

Rajesh Sola

MCP Dev Summit Mumbai India

14–15 June 2026 • #MCPDevSummit

Agenda

01	AI Revolution, Scope of MCP
02	Why RUST based MCP
03	Writing MCP tools in RUST
04	Bring MCP in Agent Loop (RIG)
05	Common Pitfalls and Best Practices
06	Examples & Case Study

AI Revolution

Classic AI/ML

Gen AI

Agentic AI

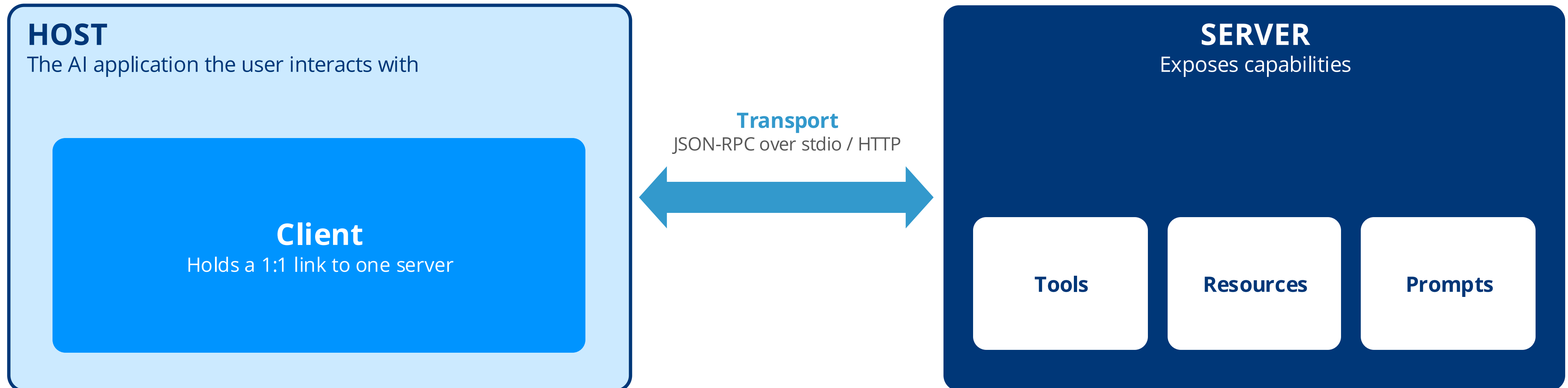
What is MCP

- **An open standard** for connecting AI applications to external tools and data.
- **Unified protocol for** several integrations.
- Built on a simple client-server model over JSON-RPC.
- Model-agnostic and language-agnostic by design.
- Empowering AI applications and AI Agents, with dynamic context

Why MCP Matters

- Eliminates brittle, one-off integrations
- **Reusable connectors** work across any MCP-compatible app.
- Keeps data access secure, scoped, and permissioned.
- **Lets models take real actions** — not just generate text.
- Backed by a fast-growing ecosystem of ready-made servers.

MCP Architecture



Capabilities are negotiated when the connection initializes.

MCP Servers

- **Expose capabilities** — tools, resources, and prompts a model can call.
- **Run anywhere** — local processes over stdio or remote services over Streamable HTTP.
- **Built in Rust** — the rmcp crate provides a fast, memory-safe server runtime.
- **Edge ready** — lightweight enough to run on constrained, embedded hardware.
- **Composable** — one server per domain, combined freely by any client.

MCP Clients

- **Live inside the host** — the AI application that initiates each connection.
- **Discover and invoke** — list the tools a server offers and call them on demand.
- **One client per server** — maintaining an isolated, stateful session.
- **Model-agnostic** — any LLM can drive the same client interface.
- **Handle transport** — negotiate stdio or Streamable HTTP automatically.

Core Building Blocks

- **Tools** — functions the model can call to take action.
- **Resources** — data and content the model can read.
- **Prompts** — reusable templates that guide interactions.
- **Sampling** — servers can request completions from the model.
- **Roots** — scope the files and data a server may access.

Tutorial

RUST MCP SDK

- Features
  - server
  - client
  - macros
  - schemars
  - auth
  - Elicitation
- Transports
  - STDIO
  - Streamable HTTP

https://docs.rs/rmcp/latest/rmcp/

Writing simple MCP Server

Add rmcp as a dependency to your RUST server project with following entry to **Cargo.toml**

```
[dependencies]
rmcp = { version = "0.16.0", features = ["server", "macros", "transport-io"] }
```

Add other dependencies like **tokio** for asynchronous runtime, **serde** for JSON, **schemars** for JSON schema

Writing simple MCP Server

Define parameters for the desired MCP tools in source code

```
#[derive(Debug, serde::Deserialize, schemars::JsonSchema)]
struct CalculatorParams{
    #[schemars(description = "First number")]
    firstNumber: f64,
    #[schemars(description = "Second number")]
    secondNumber: f64,
}
```

Writing simple MCP Server

```
#[derive(Clone)]
struct Calculator {
    tool_router: ToolRouter<Self>,
}
#[tool_router]
impl Calculator {
    fn new() -> Self {
        Self {
            tool_router: Self::tool_router(),
        }
    }

    #[tool(description = "Add two numbers")]
    async fn add(
        &self,
        Parameters(CalculatorParams{ a, b }): Parameters<CalculatorParams>,
    ) -> Result<CallToolResult, McpError> {
        Ok(CallToolResult::success(vec![Content::text(format!(
            "{} + {} = {}",
            a, b, a + b
        ))]))
    }
}
```

Writing simple MCP Server

```rust
#[tool_handler]
impl ServerHandler for Calculator {
    fn get_info(&self) -> ServerInfo {
        ServerInfo {
            instructions: Some("A small arithmetic MCP server for addition and multiplication".into()),
            capabilities: ServerCapabilities::builder().enable_tools().build(),
            ..Default::default()
        }
    }
}

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let service = Calculator::new().serve(stdio()).await?;
    service.waiting().await?;
    Ok(())
}
```

More MCP Tools/Examples

- SocketCAN tools
- Serial Bus tools
- MQTT Publish & Subscribe
- HTTP REST
- SQLite Database
- Kernel Driver Interfacing
- gRPC

MCP Inspector Tool
https://modelcontextprotocol.io/docs/tools/inspector

Testing the Server

- MCP Inspector Tool
- fastmcp CLI command
- RUST client code

Common Pit Falls

```
{
  "method": "tools/call",
  "params": {
    "name": "mqtt_publish",
    "arguments": {
      "topic": "weather/temperature",
      "payload": "24.2,78, 910" ,
      "broker": "hivemq.com",
      "port" : 1883
    },
  }
}
```

- Tool is hard coded with MQTT protocol specific parameters
- Data is not self-descriptive / LLM friendly

Common Pit Falls

```
{
  "method": "tools/call",
  "params": {
    "name": "publish_telemetry",
    "arguments": {
      "payload": "{
         speed:24.2,
         humidity:78,
         pressure:910"
      }
      "platform" : "hivemq"
    },
  }
}
```

- Avoid 1-1 mapping of embedded/internet protocols
- Tool parameters and responses should be LLM/Agent friendly and self-descriptive
- Design tools to be flexible, freedom to Agents to choose right tools and parameters

Further

- Integrations
  - Socket CAN + MQTT or HTTP REST
  - Sensor Data + MQTT or HTTP REST
- Agent in loop
- Observability & Tracing

Key Takeaways

- MCP standardizes how AI connects to the world.
- One protocol replaces countless custom integrations.
- Tools, resources, and prompts are the core primitives.
- The ecosystem is open and growing fast.
- Thank you — questions welcome. [contact / handle]

THANK YOU

Let’s build the agentic stack together

Rajesh Sola • #MCPDevSummit