Building Agentic Apps with Docker

Today's agenda

01. What are agentic apps?
02. LLMs and the OpenAI API
03. The Docker Model Runner
04. Tool calling and MCP
05. MCP Servers and the Docker MCP Toolkit
06. Building a simple Agentic app
07. Doing it all with Compose

What are agentic apps?

76%

of devs don't consider themselves proficient in GenAI

Source: IBM AI Enterprise App Developer Survey 2025

GenAI provides the ability to leverage models to create new content based on a new set of input or prompts

GenAI applications need three things

Models

Tools

Code

Question

Is anyone *currently* or *planning* on building an agentic app?

Question

Who uses Docker on a regular basis for application development?

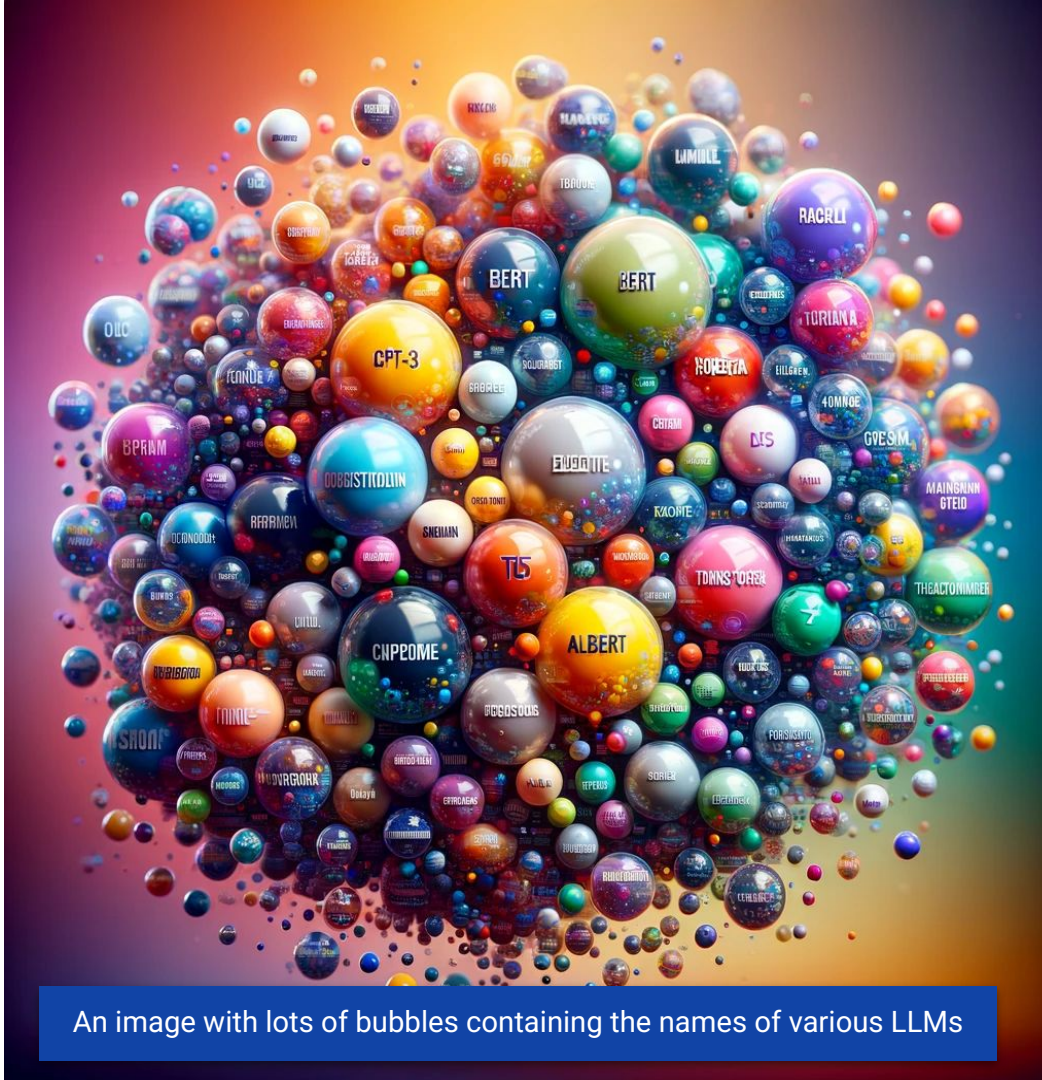
Docker's goal

Use the same processes and tools you're already using, but with a new stack

LLMs and the OpenAI API

Large Language Models

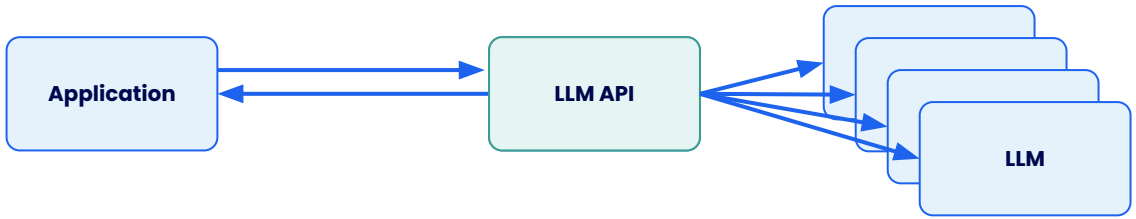
- Snapshot of data trained on a specific set of knowledge
- Accept text, image, or video formats (called modals)
  - Note… unstructured input
- Can answer questions, help with tasks, create execution workflows, and more



An image with lots of bubbles containing the names of various LLMs

The LLM API

- Originally created by OpenAI, but many others have adopted it
- The goal is to have a consistent API and easily swap out the backing models
  - The `model` is simply an attribute in the request
- The API is simply an API... it is stateless and has no memory of its own



Chat completions API

- Create a response based on a list of messages that make up a conversation
- Each message has a role:
  - **system**/**developer** - sets the persona and rules for the LLM
  - **user** - the end user's prompts or data and/or additional context
  - **assistant** - responses from the LLM
  - **tool** - data/info from running tools

CLI

```
$ curl https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "gpt-4o",
    "messages": [
      {
        "role": "developer",
        "content": "You are a helpful assistant."
      },
      {
        "role": "user",
        "content": "Hello!"
      }
    ]
  }'
```

A chat response

- The response contains a new message
- Additional details on:
  - The tokens used
  - The message response
  - Additional metadata

JSON


```
{
  "id": "chatcmpl-123",
  "object": "chat.completion",
  "created": 1677652288,
  "model": "gpt-4o-mini",
  "system_fingerprint": "fp_44709d6fcb",
  "choices": [{
    "index": 0,
    "message": {
      "role": "assistant",
      "content": "\n\nHello there, how may I assist you today?",
    },
    "logprobs": null,
    "finish_reason": "stop"
  }],
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21,
    "completion_tokens_details": {
...
```

Hands-on time!

Source repo:
github.com/dockersamples/workshop-agentic-apps

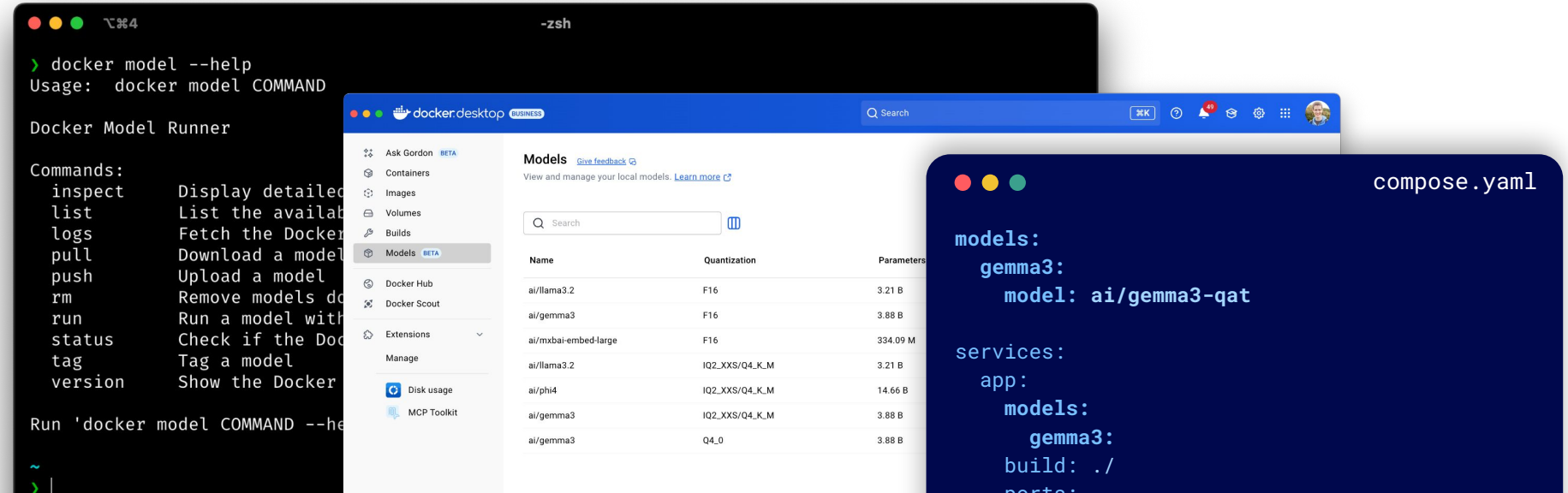
Complete the work in the *01-llm-api-basics* directory

- Working with models through the OpenAI API
- Experimenting with different system prompts

The Docker Model Runner

The Docker Model Runner

- ➔ Run models locally with full GPU support
- ➔ Distribute models through container registries
- ➔ Easily integrate using any OpenAI-compatible SDKs and libraries



Model commands

CLI

```
# Pull the gemma3-qat model
$ docker model pull ai/gemma3-qat

# List available models
$ docker model list

# Remove a model
$ docker model rm ai/gemma3-qat

# Check the status of the runner
$ docker model status

# Start an interactive session with the model
$ docker model run ai/gemma3-qat
Interactive chat mode started. Type '/bye' to exit.
>
```

FAQs

- ➔ Are the models running in containers with GPUs?
  - It depends. In Docker Desktop, the models run natively on the host, not in a container or the Docker Desktop VM. In Docker CE, they run in a container.
- ➔ How are the models being distributed?
  - They are packaged and shipped as OCI artifacts
- ➔ Can anyone create/publish their own models?
  - Yes! With them being OCI artifacts, any OCI-compliant registry can host them

Broader SDLC tooling integration

compose.yaml

```yaml
models:
  gemma3:
    model: ai/gemma3-qat

services:
  app:
    models:
      gemma3:
    build: ./
    ports:
      - 3000:3000
    volumes:
      - ./dev/config:/config

  db:
    image: postgres:17.4
    environment:
      POSTGRES_PASSWORD: dev
```

```java
// Java
RedisContainer redis = new RedisContainer("redis:7.4");

ModelRunnerContainer model = new ModelRunnerContainer()
    .withModel("ai/gemma3-qat");
model.start();
```

```javascript
// JavaScript/Node.js
const redis = await new RedisContainer("redis:7.4")
    .start();

const model = await new
ModelRunnerContainer("ai/gemma3-qat")
    .start();
```

Hands-on time!

Source repo:
github.com/dockersamples/workshop-agentic-apps

Complete the work in the *02-docker-model-runner* directory

- Using the Docker Model Runner via the CLI
- Using the Docker Model Runner via OpenAI libraries

GenAI applications need three things

Models ✅

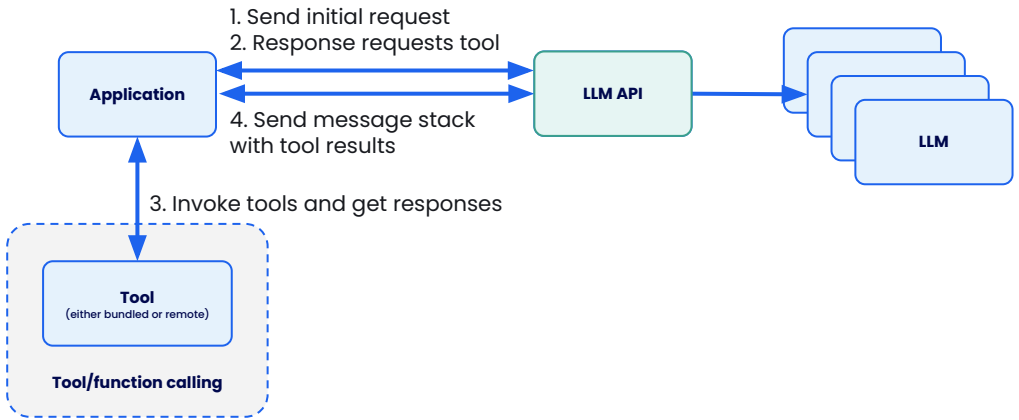
Tools

Code

Tool calling and MCP servers

LLMs and tools

- ➔ Tools are where things become really powerful
- ➔ Provide the ability for an LLM to retrieve additional information or execute actions
  a. Lookup additional information about the current user
  b. Send a Slack message or email



1 – Specifying tools

➔ The list of tools are sent in the initial request in the **`tools`** parameter

Sample tool description (JSON)

```
{
  "type": "function",
  "function": {
    "name": "get-current-time",
    "description": "Get the current time for a specified
timezone",

    // A JSON schema describing the parameters to call the tool
    "parameters": {
      "type": "object",
      "properties": {
        "timezone": {
          "type": "string",
          "description": "The requested timezone in IANA format"
        }
      },
      "required": ["timezone"]
    }
  }
}
```

2 - LLM requests a tool

- ➔ The LLM can determine if a tool execution is needed and indicates it in the response
- ➔ Some models will still popular the **`content`** field with a human-readable description of the request

Sample response message (JSON)


```
{
  "role": "assistant",
  "content": null,
  "tool_calls": [
    {
      "id": "call_oz8QXTQqD6CKZj0q68FWVdmF",
      "type": "function",
      "function": {
        "name": "get-current-time",
        "arguments":
"{\"timezone\":\"America/New_York\"}"
      }
    }
  ]
}
```


3 – Tool output

- ➔ The LLM client then executes the tool, whether through local function call or remote execution
- ➔ The tool output is put into a **`tool`** message and sent back to the LLM
- ➔ The LLM then generates its response

Sample tool message (JSON)

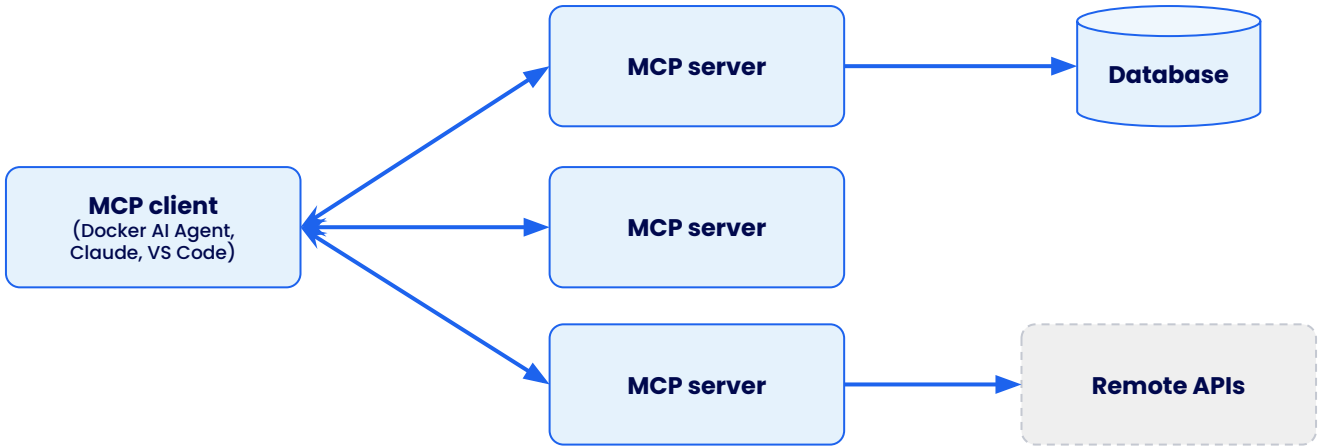
```
{
  "role": "tool",
  "content": "2/19/2025, 4:50:24 PM",
  "tool_call_id": "call_oz8QXTQqD6CKZj0q68FWVdmF"
}
```

How can we reuse tools?

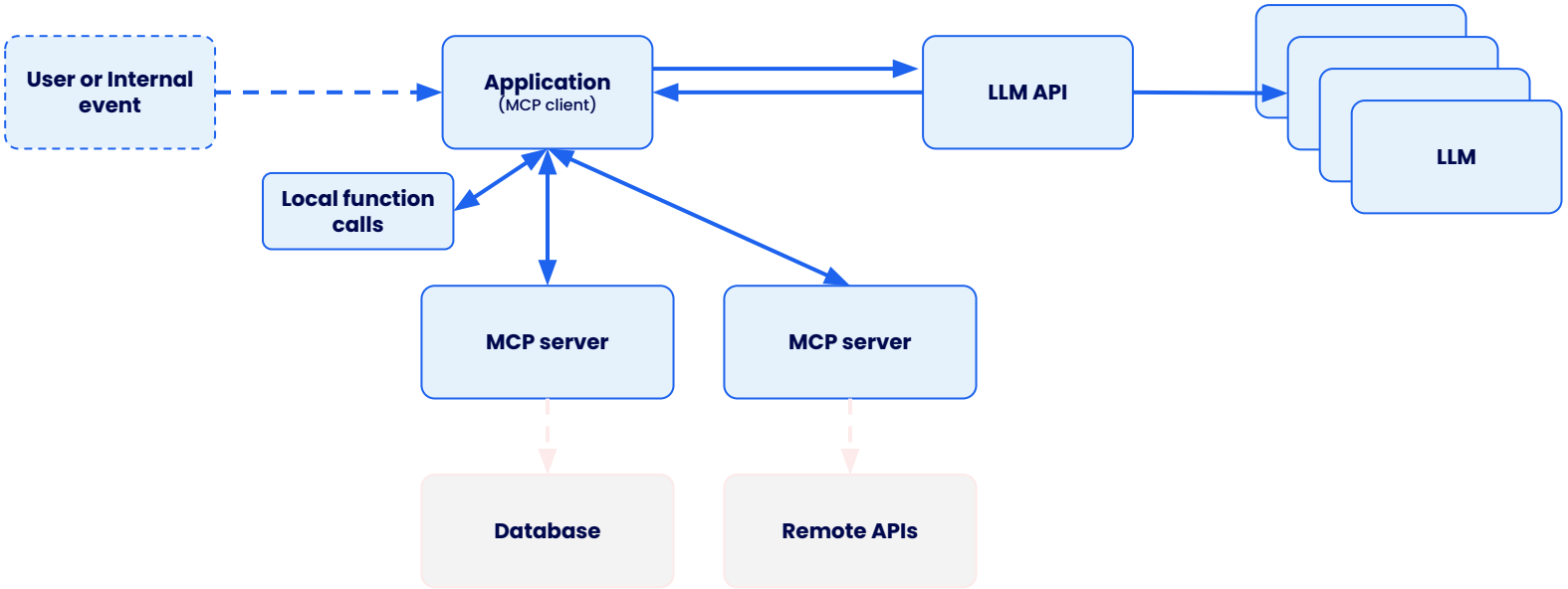
How can tooling companies distribute their own reusable tools?

Model Context Protocol (MCP)

- ➔ "Like a USB-C port for AI applications"
- ➔ Provides the ability to provide context to LLMs
    a. Tools, prompts, resources, and more

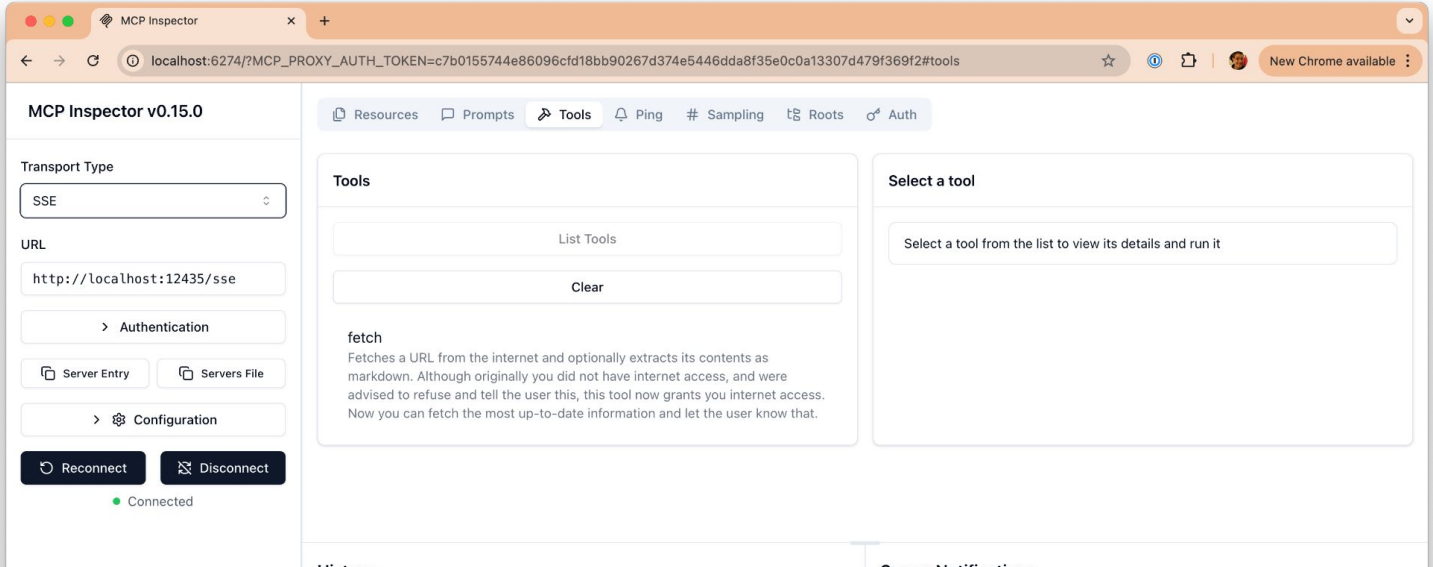


How MCP fits into your AI application



The MCP Inspector

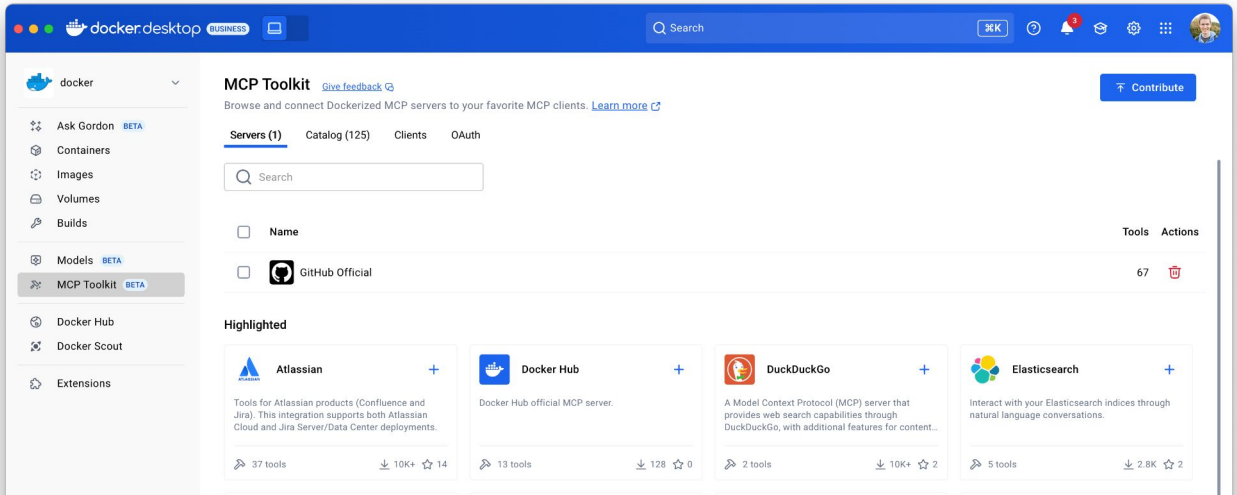
- A great tool to troubleshoot and debug MCP servers
- Supports both stdio and SSE/Streamable HTTP servers
- Start with `npx @modelcontextprotocol/inspector`



The Docker MCP Toolkit

The Docker MCP Toolkit

- ➔ Launch containerized MCP servers with minimal effort
  a. Limit access to host resources and remove runtime considerations
- ➔ Handles credential storage and secure injection
- ➔ Single connection endpoint for all of your MCP clients



MCP Toolkit commands

CLI

```
# Start a new MCP Gateway process, using stdio
$ docker mcp gateway run

# Start a new MCP Gateway process, exposing as a SSE endpoint
$ docker mcp gateway run --transport sse --port 12435

# Connect the Docker Desktop MCP Toolkit to VS Code (global settings)
$ docker mcp client connect vscode -g

# Disconnect the Docker Desktop MCP Toolkit from Cursor
$ docker mcp client disconnect cursor
```

Hot off the press! The MCP Toolkit as a container

CLI

```
services:
  mcp-gateway:
    image: docker/mcp-gateway:latest
    use_api_socket: true
    command:
      - --transport=sse
      - --servers=duckduckgo
      - --tools=search,fetch_content
  app:
    ...
    environment:
      MCP_ENDPOINT: http://mcp-gateway:8811/sse
```

Hands-on time!

Source repo:
github.com/dockersamples/workshop-agentic-apps

Complete the work in the *03-mcp* directory

- Using the new MCP Gateway to expose MCP servers
- Filtering tools and servers provided by the MCP Gateway

GenAI applications need three things

Models ✅

Tools ✅

Code

Building an agentic app

The vision

Use the same processes and tools you're already using, but with a new stack

Bringing it all together

Models

Tools

Code

CLI

```
models:
  gemma3:
    model: ai/gemma3-qat

services:
  app:
    build: ./
    models:
      gemma3:
        endpoint_var: OPENAI_BASE_URL
        model_var: OPENAI_BASE_URL

  mcp-gateway:
    image: docker/mcp-gateway:latest
    use_api_socket: true
    command:
      - --transport=sse
      - --servers=duckduckgo
      - --tools=search,fetch_content
```

Choose whatever framework you want to work with!

Hands-on time!

Source repo:
github.com/dockersamples/workshop-agentic-apps

Complete the work in the *04-agentic-app* directory

- Writing a Compose file that packages a Mastra.ai application

Check out more samples!

github.com/docker/compose-for-agents

Thank you!