Python + MCP

🛠️ Dec 16: Building MCP servers with FastMCP
☁️ Dec 17: Deploying MCP servers to the cloud
👤 Dec 18: Authentication for MCP servers

🔗 Register at aka.ms/pythonmcp/series

Microsoft

Python + MCP

☁️ Deploying MCP servers to the cloud

aka.ms/pythonmcp/slides/deploying

Pamela Fox

Python Cloud Advocate

www.pamelafox.org

Today we'll cover...

- Options for cloud deployment
- Containerization of Fast MCP apps
- Deploying to Azure Container Apps
- Observability with Azure App Insights and Logfire
- Private networking with Azure virtual networks

Want to follow along?

1. Open this GitHub repository in GitHub Codespaces:
   aka.ms/python-mcp-demos

2. Follow instructions in README for "Getting started".

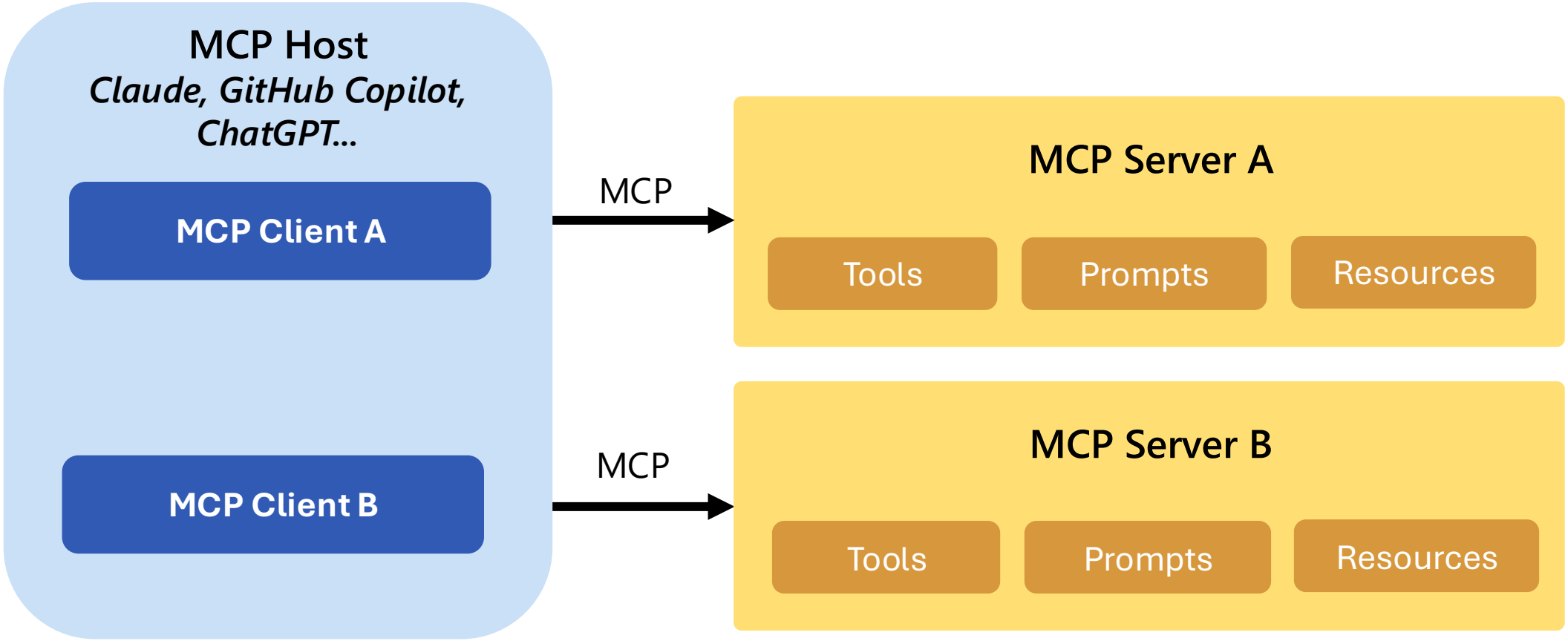
   Since today is about deploying, you'll need an Azure account to try deployment.

   If you're new to Azure, try the free trial: https://aka.ms/AzureFreeTrialDevRelAll

   Otherwise, note that some costs will be incurred for deployment. 💸

Cloud deployment

Recap: MCP architecture



https://modelcontextprotocol.io/docs/learn/architecture

MCP transports: STDIO vs. HTTP

	STDIO	HTTP (Streamable) 😍
Connection mode	Local communication via stdin/stdout	Network communication over HTTP
Server startup	Launched on demand by the client (e.g., VS Code)	Runs as a service accessible through a URL
Typical use	Local dev, CLI tools, single-user apps	Production, remote access, multiple simultaneous clients
Advantages	Simple, no network setup	Scalable, supports bidirectional streaming, easy web integration

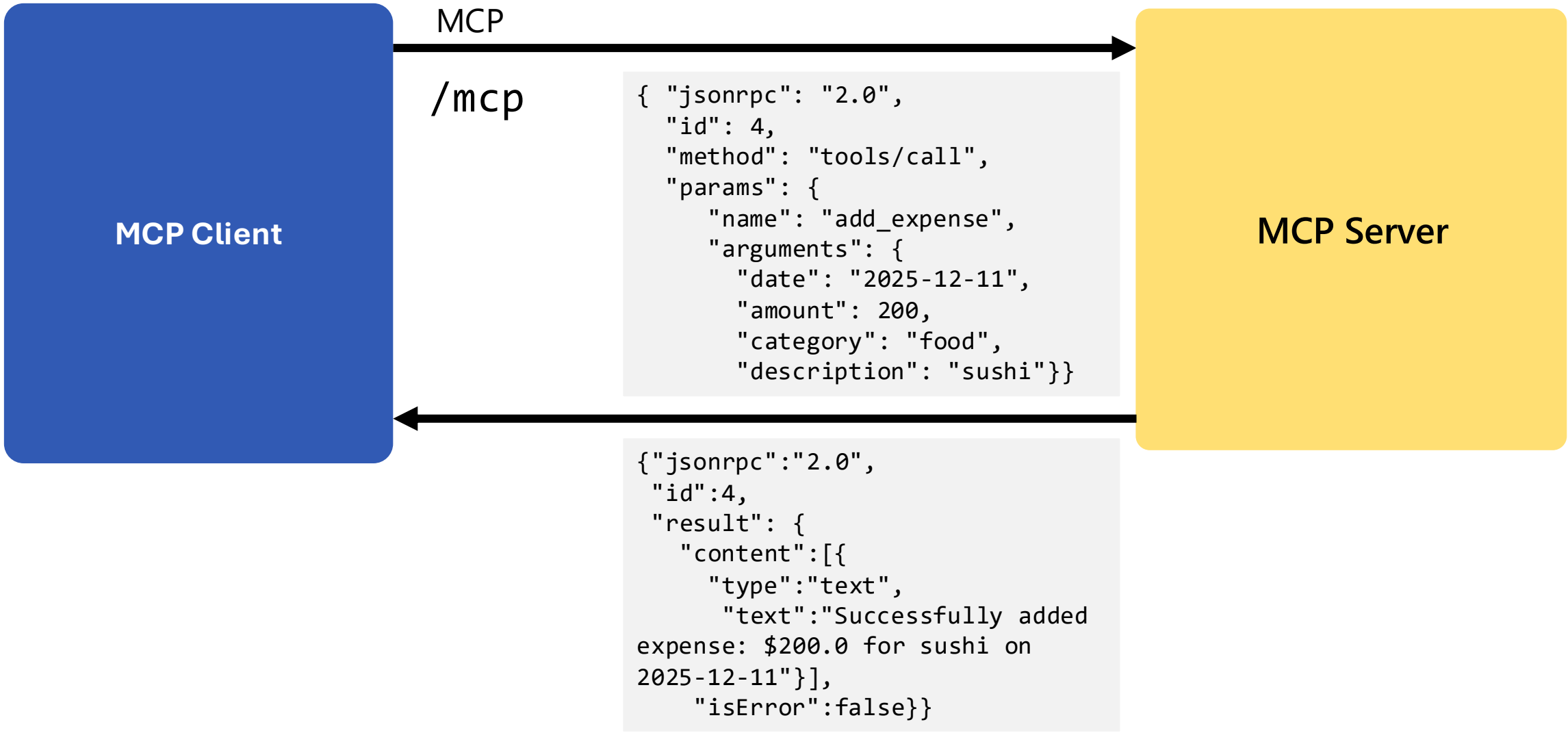
```
mcp.run(transport="stdio")
```

servers/basic_mcp_stdio.py

```
mcp.run(
  transport="streamable-http",
  host="0.0.0.0", port=8000)
```

servers/basic_mcp_http.py

MCP over HTTP



Running a HTTP server in production

Create an ASGI (Starlette) app based off the FastMCP instance:

```
mcp = FastMCP("Expenses Tracker", middleware=middleware)

# MCP tool/resource/prompt definitions

app = mcp.http_app()
```

🔗 aka.ms/python-mcp-demos: servers/deployed_mcp.py

Run the app with an ASGI-compatible server like Uvicorn:

```
uvicorn deployed_mcp:app --host 0.0.0.0 --port 8000
```

Deploying on Azure

😍 Pros: Managed features — Control & Flexibility

😢 Cons: Managed features — More DevOps

Functions

Requires:
pyproject.toml
server.py
host.json

App Service

pyproject.toml
server.py

Container Apps

pyproject.toml
server.py
Dockerfile

Kubernetes

pyproject.toml
server.py
Dockerfile
docker-compose.yaml

Deploying on Container Apps

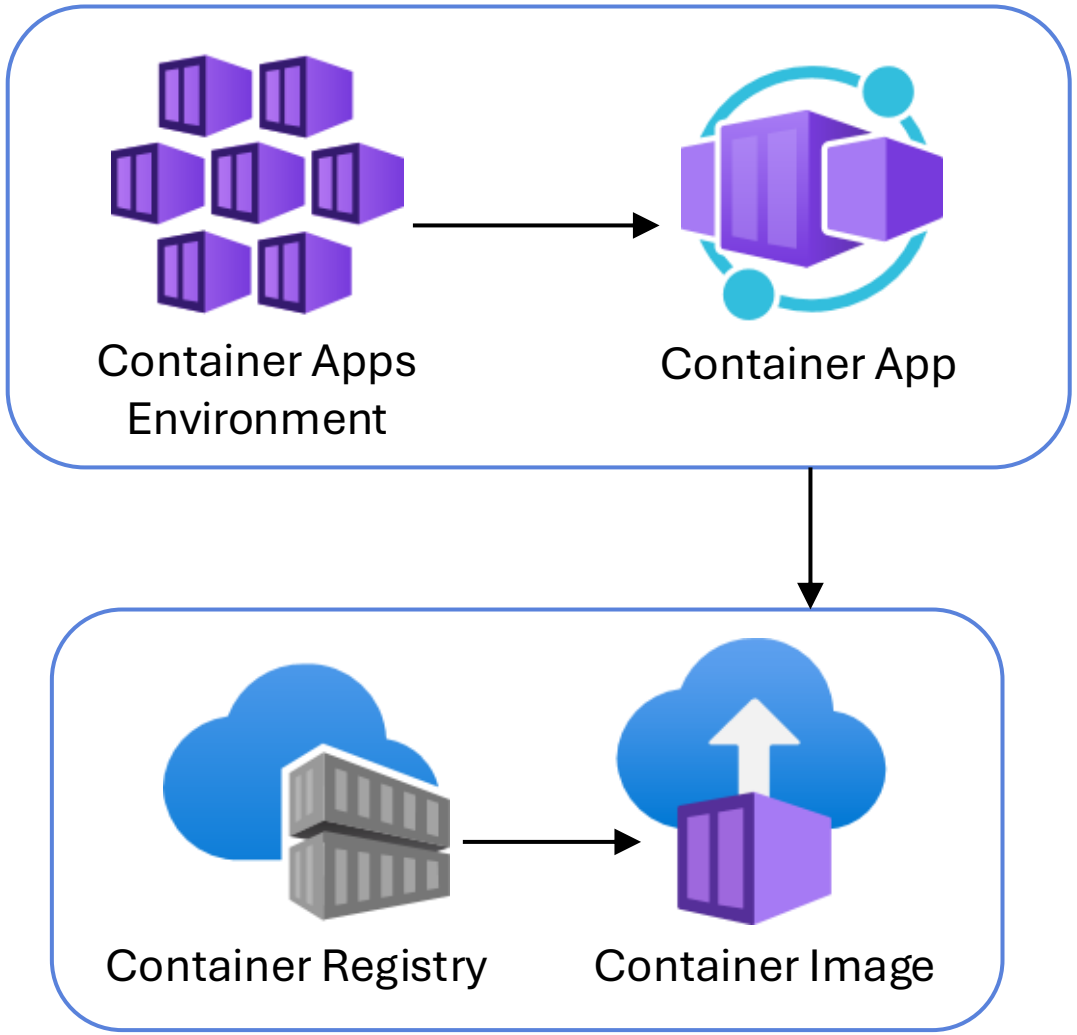
Containerizing a FastMCP server

Dockerfile:

```
FROM python:3.13-slim

WORKDIR /app

# Install the requirements

COPY pyproject.toml uv.lock ./
RUN pip install uv && uv sync --no-install-project

COPY . .
WORKDIR /app/servers

EXPOSE 8000

# Start the ASGI app
CMD ["uvicorn", "deployed_mcp:app", "--host", "0.0.0.0", "--port", "8000"]
```

🔗 aka.ms/python-mcp-demos: servers/Dockerfile

Architecture for FastMCP on Container Apps



Container Apps Environment is a virtual network boundary for one or more **Container Apps**.

The **Container App** runs an image from a registry, like Azure **Container Registry**.

Images can be pulled from any public or private repository.

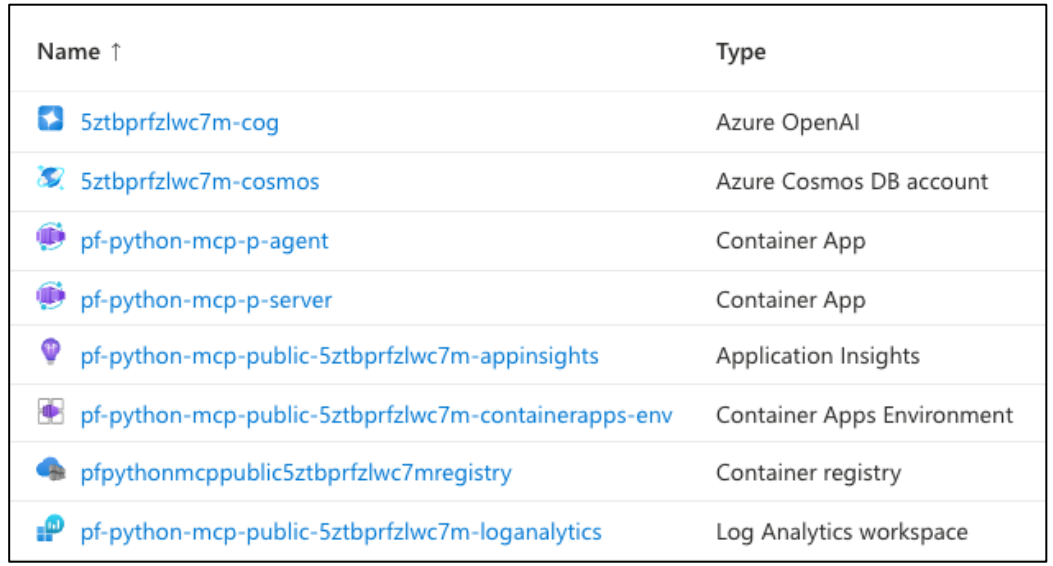
Deploying example server to Container Apps

1. Open this GitHub repository:

   aka.ms/python-mcp-demos

2. Follow instructions in README for "Deploying to Azure":

```
>> azd auth login
>> azd up
```

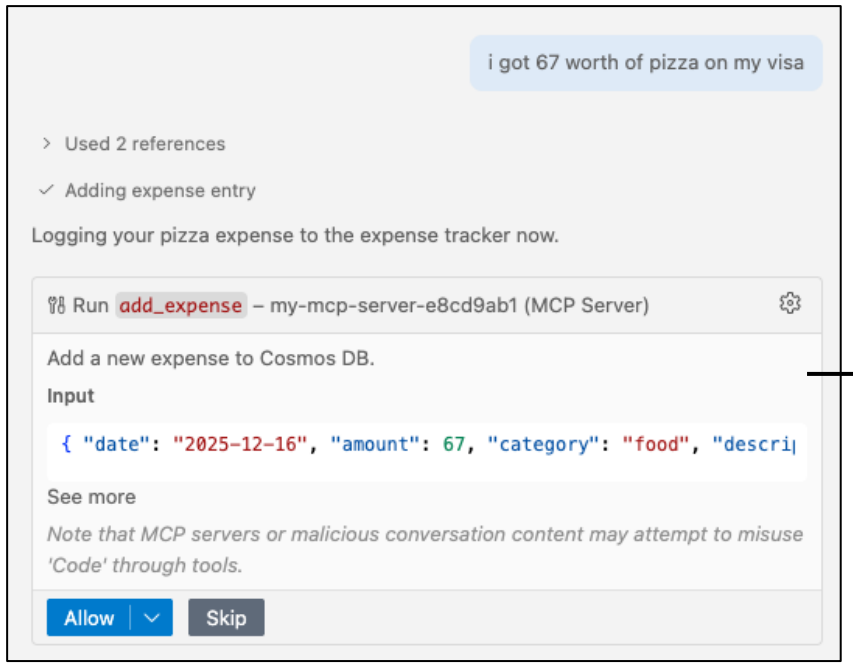


Name ↑	Type
5ztbprfzlwc7m-cog	Azure OpenAI
5ztbprfzlwc7m-cosmos	Azure Cosmos DB account
pf-python-mcp-p-agent	Container App
pf-python-mcp-p-server	Container App
pf-python-mcp-public-5ztbprfzlwc7m-appinsights	Application Insights
pf-python-mcp-public-5ztbprfzlwc7m-containerapps-env	Container Apps Environment
pfpythonmcppublic5ztbprfzlwc7mregistry	Container registry
pf-python-mcp-public-5ztbprfzlwc7m-loganalytics	Log Analytics workspace

Demo: Using the deployed server from VS Code

.vscode/mcp.json:

```
"expenses-mcp-http-deployed": {
  "type": "http",
  "url": "https://your-aca-subdomain.westus.azurecontainerapps.io/mcp"
}
```



Cosmos DB:


```
{
    "id": "26fe2c1d-299f-4c81-aa16-59fa9459c604",
    "date": "2025-12-16",
    "amount": 67,
    "category": "food",
    "description": "Pizza",
    "payment_method": "visa",
    "_rid": "kitdAP08zIcCAAAAAAAAAA==",
    "_self": "dbs/kitdAA==/colls/kitdAP08zIc=/docs/kitdAP08zIcCAAAAA
    "_etag": "\"69004ae2-0000-0700-0000-6941e2920000\"",
    "_attachments": "attachments/",
    "_ts": 1765925522
}
```

Demo: Using the deployed server from AI agent

Point agent at the deployed MCP server endpoint:

```
MCP_SERVER_URL = os.environ["MCP_SERVER_URL"]
async with (
  MCPStreamableHTTPTool(name="Expenses MCP", url=MCP_SERVER_URL) as mcp_server,
  ChatAgent(chat_client=client, name="Expenses Agent",
            instructions=f"You help users to log expenses.") as agent):
    user_query = "yesterday I bought a laptop for $1200 using my visa."
    result = await agent.run(user_query, tools=mcp_server)
```

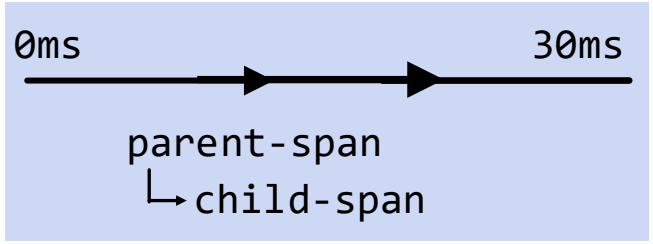
Then you can run agent locally *or* from a deployed container:

Observability in production 👀

Observability with OpenTelemetry (OTel)

OTel standardizes how applications emit traces, metrics, and logs — making debugging and performance analysis consistent across languages and vendors.

Traces



Operations composed of **spans** that show how a request moves through services, including timing, dependencies, and context propagation.”

Metrics

```
_/‾‾ cpu_usage: 30%
‾\_  latency_p95: 43ms
_/‾\_ errors: 0.2%
```

Numeric measurements such as CPU usage, request counts, latency, error rates, or any custom app metric.

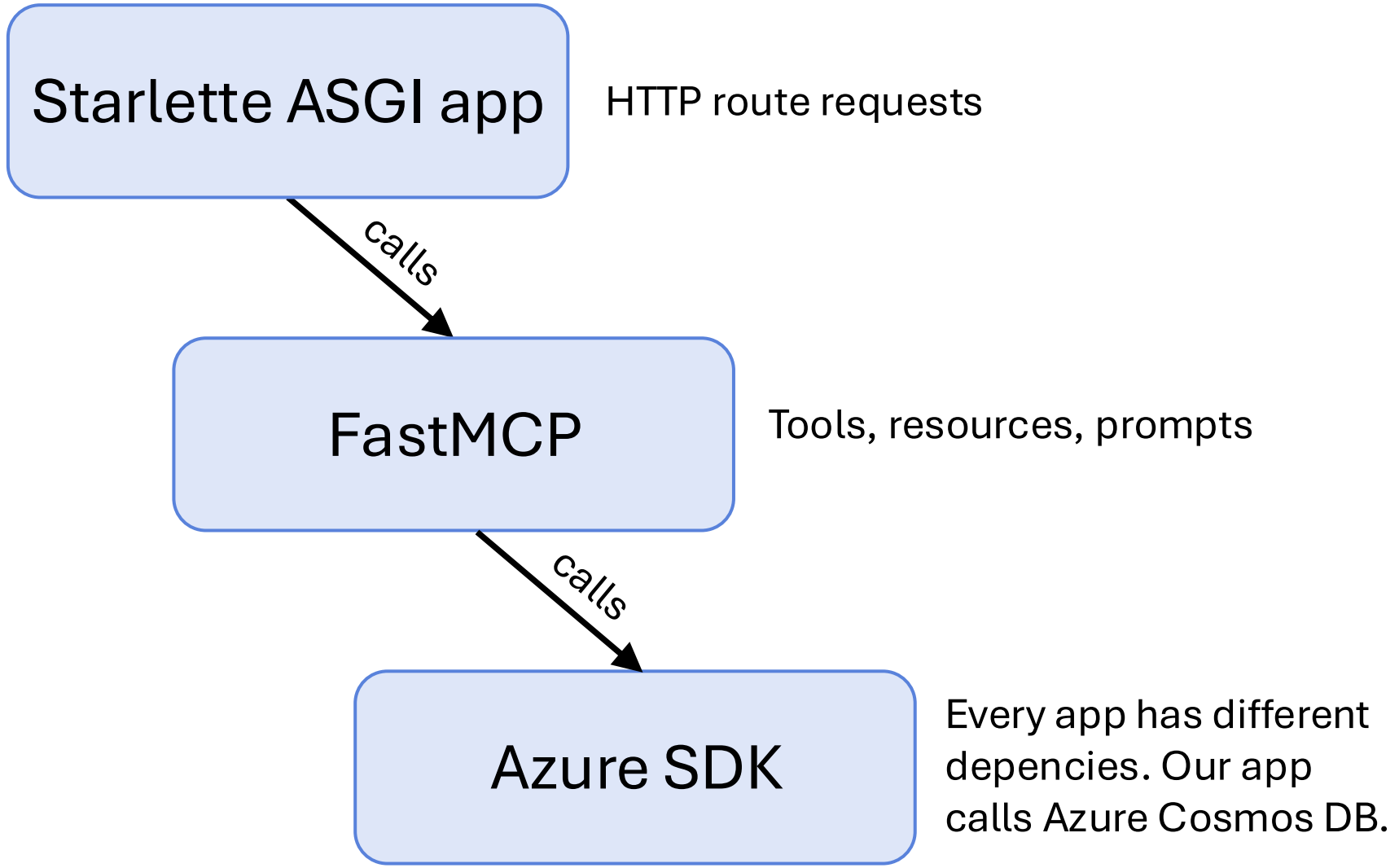
Logs

```
INFO: 2025-12-05: Server running
      {"region": "westus"}

ERROR: 2025-12-05: User not found
      {"user_id": 123}
```

Structured log records with message, severity, time stamp, and contextual attributes.

What do we want traces from?

For our sample app:



Storing traces from Starlette requests

Install in pyproject.toml:

```
opentelemetry-instrumentation-starlette
```

deployed_mcp.py:

```
app = mcp.http_app()
StarletteInstrumentor.instrument_app(app)
```

Span: `POST /mcp`

attribute	value
http.flavor	1.1
http.host	127.0.0.1:8000
http.method	POST
http.route	/mcp
http.scheme	https
http.status_code	200
http.user_agent	node
http.server_name	pf-mcp-deployed-server....
http.url	https://pf-mcp.deployed-server...

Storing traces from FastMCP tool calls

deployed_mcp.py:

```
mcp = FastMCP("Expenses", middleware=[OpenTelemetryMiddleware("ExpensesMCP")])
```

opentelemetry_middleware.py:

```
class OpenTelemetryMiddleware(Middleware):
  async def on_call_tool(self, context: MiddlewareContext, call_next):
    tool_name = context.message.name
    with self.tracer.start_as_current_span(f"tools/call {tool_name}",
      attributes={
       "gen_ai.tool.name": tool_name,
       "gen_ai.operation.name": "execute_tool",
       "gen_ai.tool.call.arguments": str(context.message.arguments),
       "mcp.method.name": context.method}) as span:
      try:
        result = await call_next(context)
        span.set_status(Status(StatusCode.OK))
        return result
      except Exception as e:
        span.set_status(Status(StatusCode.ERROR, str(e)))
        span.record_exception(e)
        raise
```

Span: `tools/call add_expense`

attribute	value
gen_ai.tool.name	add_expense
gen_ai.operation.name	execute_tool
gen_ai.tool.call.arguments	{'date': '2025-12-05', 'amount': 75, 'category': 'shopping', 'description': 'office supplies', 'payment_method': 'visa'}
mcp.method.name	tools/call

Storing traces from Azure SDK requests

Install in pyproject.toml:

```
azure-core-tracing-opentelemetry
```

Configure Azure SDK to always use opentelemetry as its observability format:

```
from azure.core.settings import settings

settings.tracing_implementation = "opentelemetry"
```

→ **Span:** `GET my-cosmos.documents.azure.com`

attribute	value
http.method	GET
http.status_code	200
http.target	/
http.url	https://my-cosmos.documents.azure.com...

OTel-compliant observability platforms

	Open source?	Managed version?
Azure Application Insights	❌	✅
Datadog	❌	✅
New relic	❌	✅
Grafana	✅	✅
Prometheus	✅	(via Grafana)
Jaeger	✅	❌
Logfire	✅	✅

...and many more!

Exporting OTel to Azure App Insights

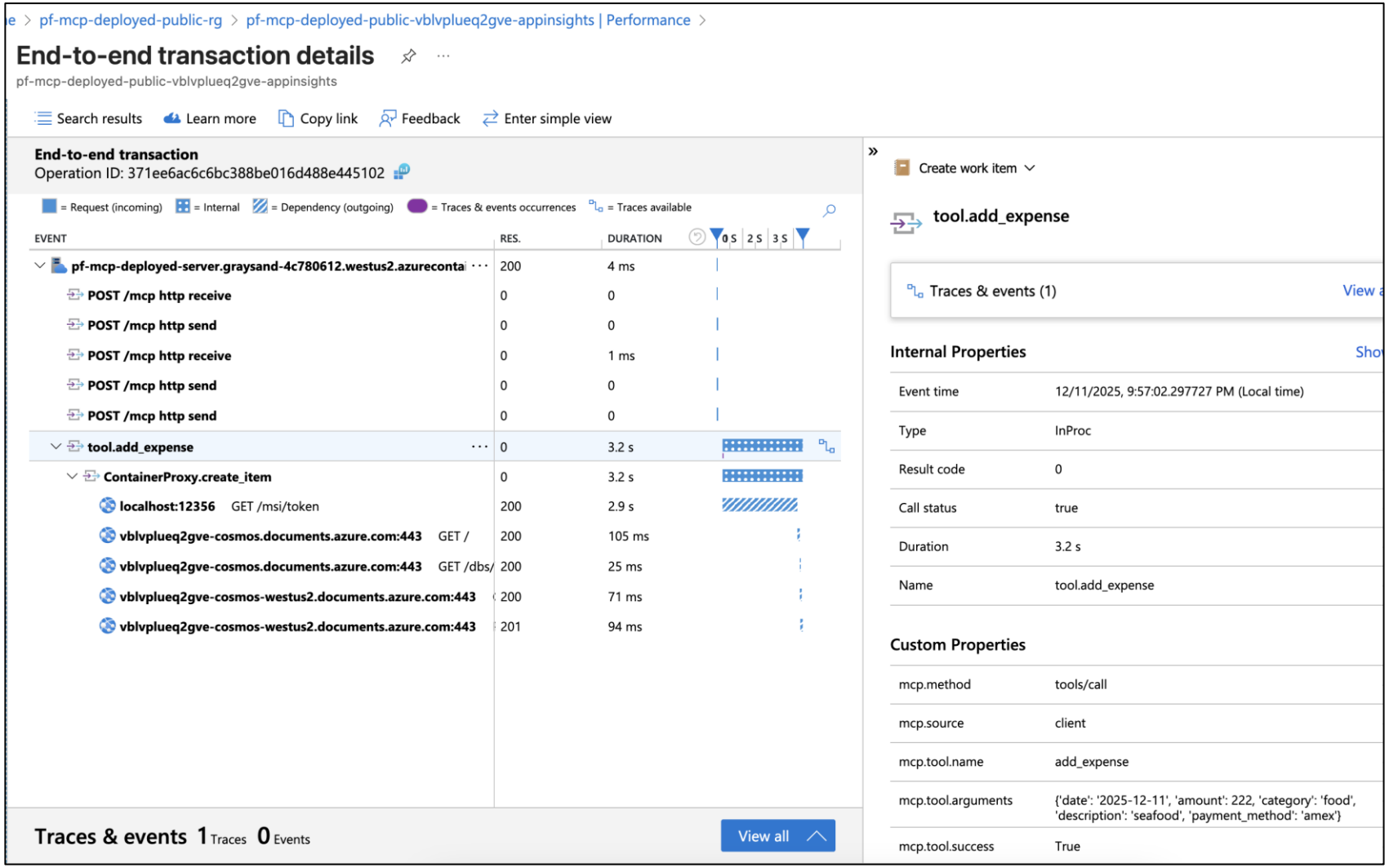
Install in pyproject.toml:

```
azure-monitor-opentelemetry
```

Use wrapper function `configure_azure_monitor` to configure global exporters based off the App Insights connection string:

```
from azure.monitor.opentelemetry import configure_azure_monitor

if os.getenv("APPLICATIONINSIGHTS_CONNECTION_STRING"):
    configure_azure_monitor()
```

Demo: Viewing traces in Azure App Insights



1. Navigate to App Insights in Portal

2. Find traces in Performances, Failures, or Logs tabs

3. Create dashboards of what matters to you

```
>> azd monitor
```

Exporting OTel to Logfire

Install in pyproject.toml:

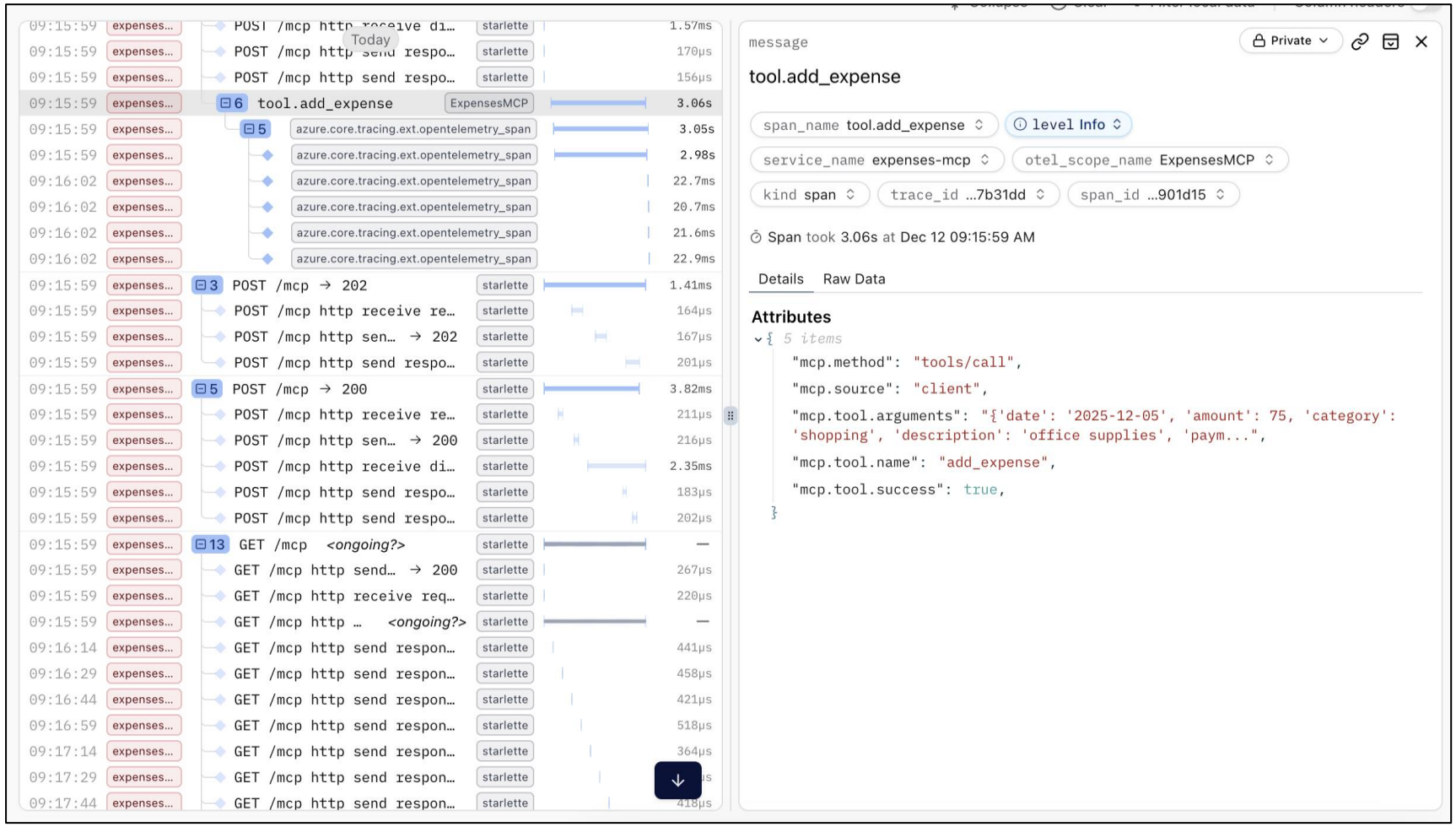
```
logfire
```

Use function `logfire.configure` to configure global exporters:

```python
import logfire

if os.getenv("LOGFIRE_TOKEN"):
  logfire.configure(service_name="expenses-mcp", send_to_logfire=True)
```

If you're not using App Insights, you can remove `OpenTelemetryMiddleware` and use `logfire.configure_mcp()` instead.

Demo: Viewing traces in Logfire



1. Enable Logfire in the deployed app

2. Open Logfire project

3. Watch traces in the live view

4. Create filters and dashboards of what matters to you

Private networking for MCP servers

The risks of public networks

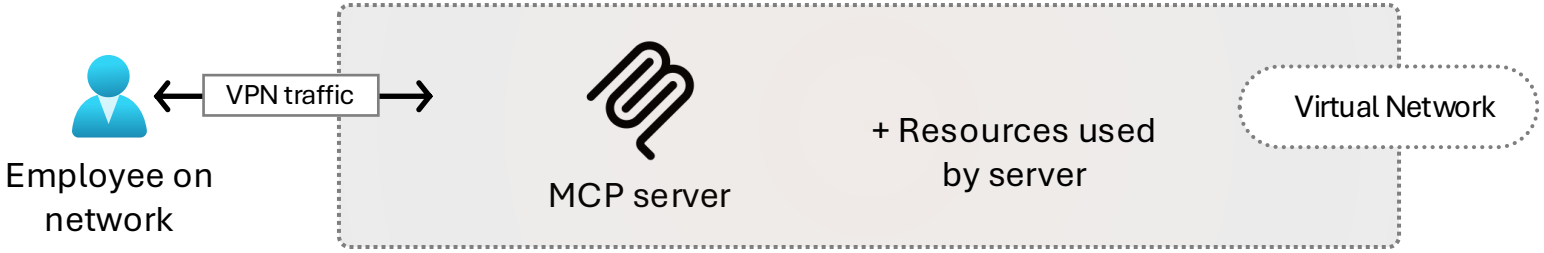
84% of attack paths involve Internet exposure

Source: 2024 State of Multicloud Security Risk, Microsoft

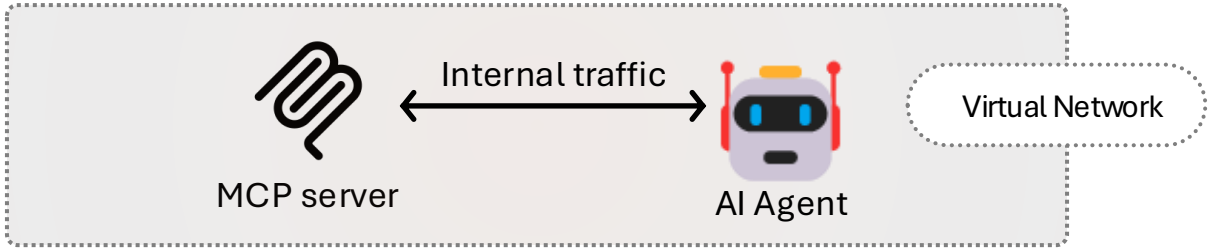
Is private networking right for you?

Private networking may be a good fit if...

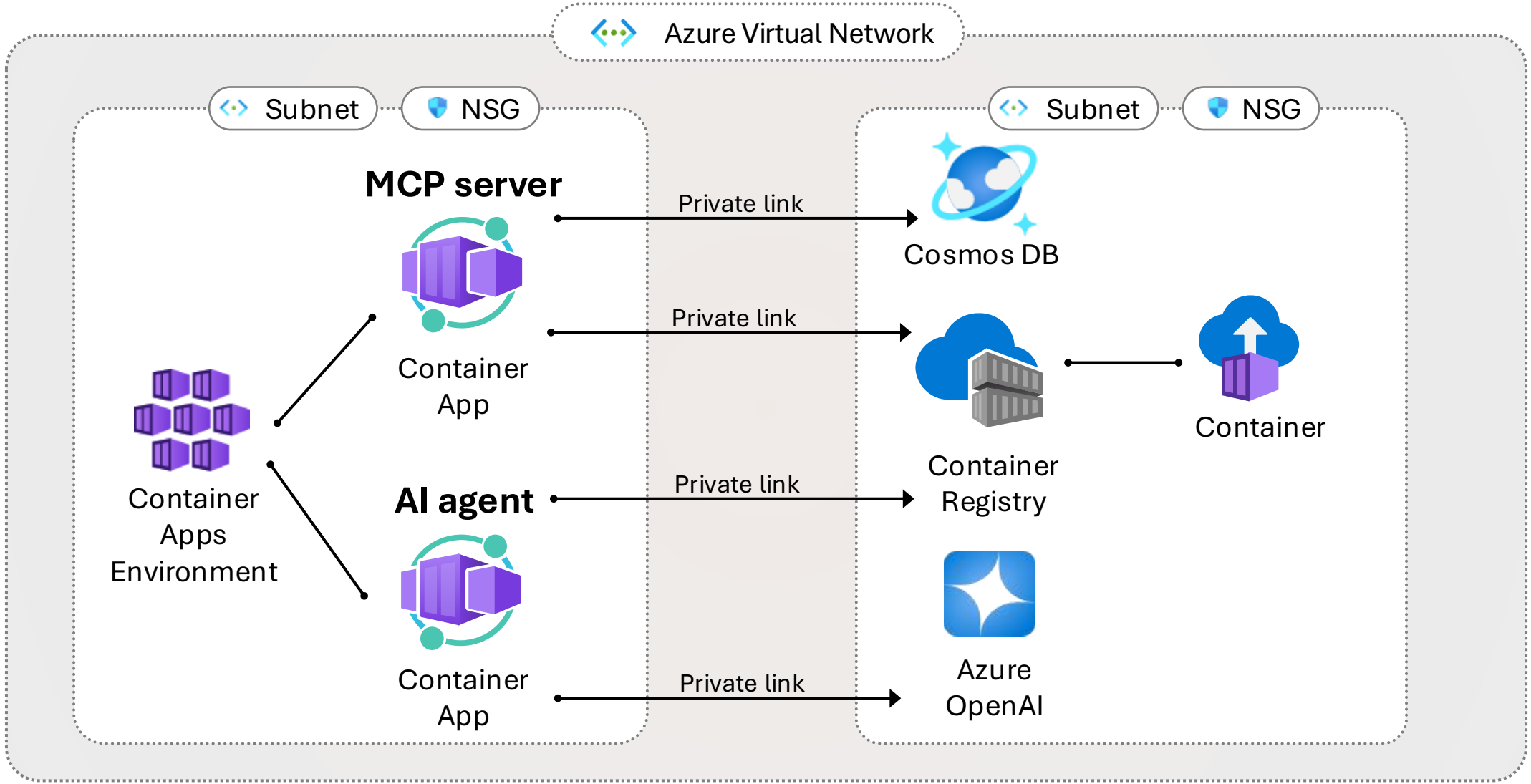
Your MCP servers only need to be accessed by internal employees who are working on a private network already or have VPN access.



Your MCP servers only need to be accessed by internal AI agents.



Architecture of private MCP + agent on Azure



Deploying example server with private network

1. Open this GitHub repository in GitHub Codespaces:

   aka.ms/python-mcp-demos

2. Follow instructions for "Deploy to Azure with private networking":

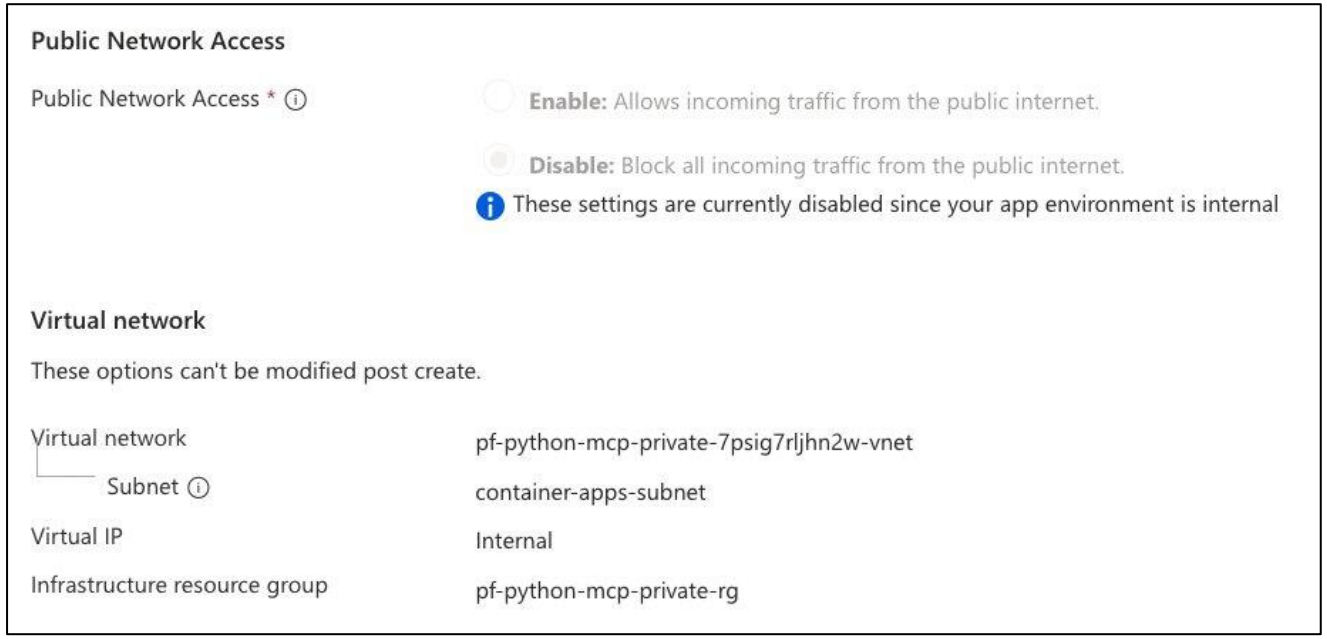
```
>> azd auth login
```

```
>> azd env set USE_VNET true
>> azd env set USE_PRIVATE_INGRESS true
```

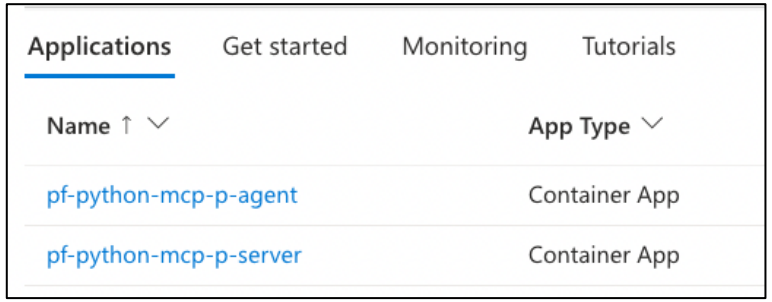
```
>> azd up
```

Demo: MCP server + agent on private network

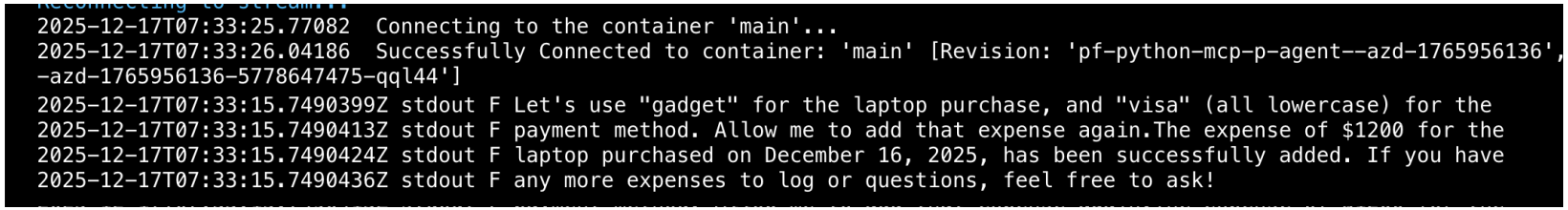
1 Verify that public network access is disabled:



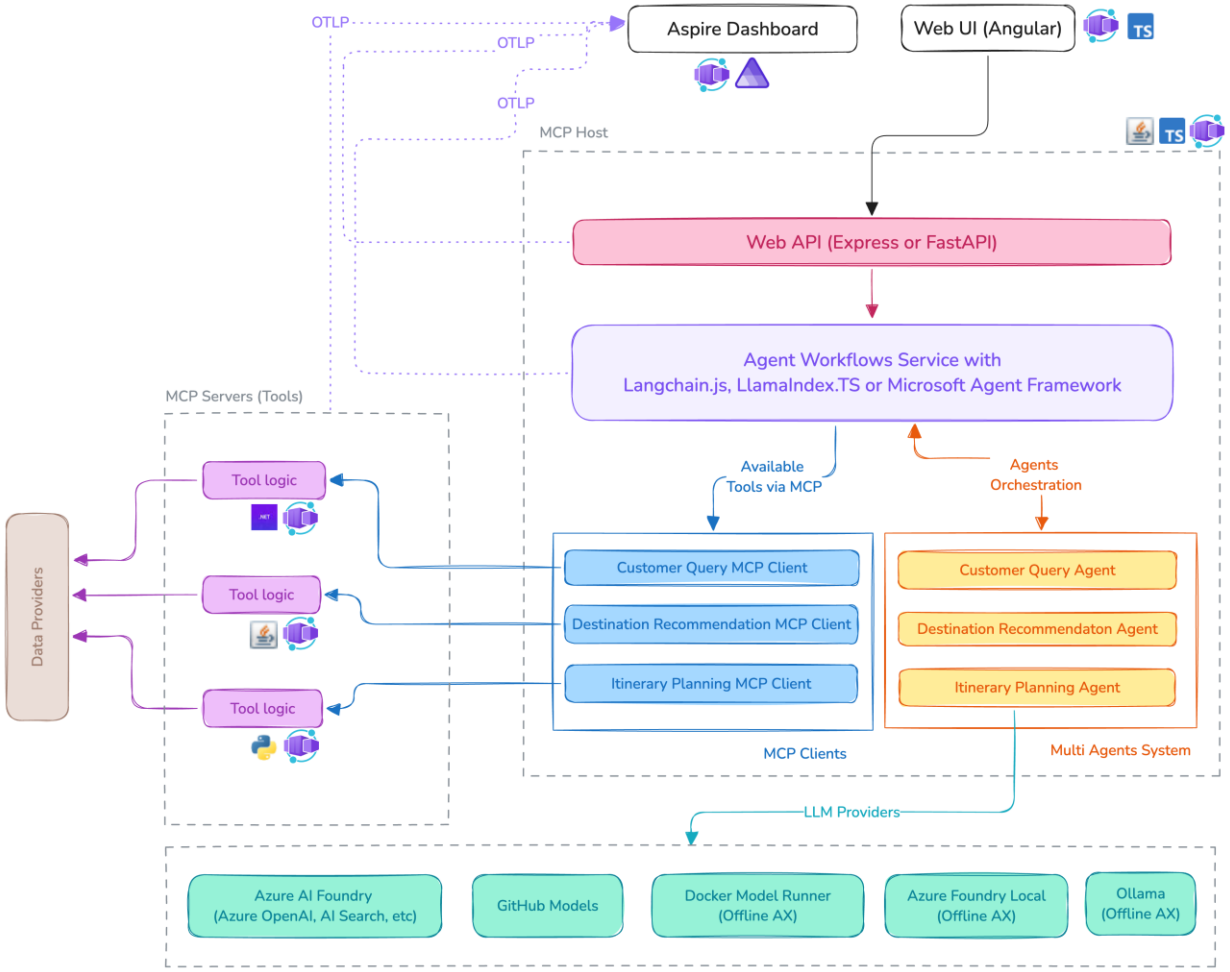
2 Select the agent app:



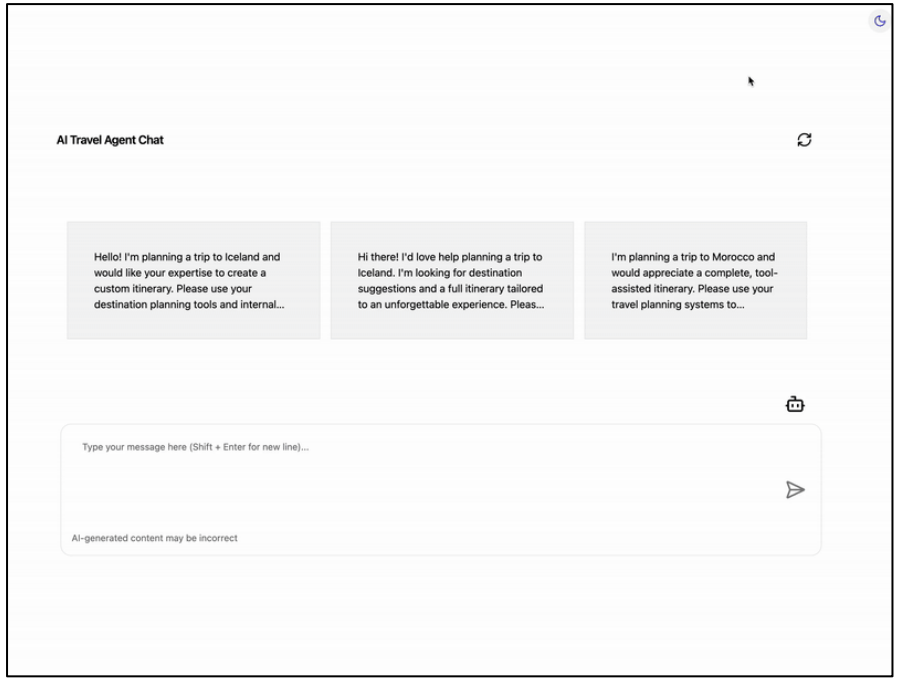
3 See agent call is successful:



Sample: AI travel agents



This project sets up multiple MCP servers in different languages, and orchestrates them from a Llamaindex agent.



https://github.com/Azure-Samples/azure-ai-travel-agents

More deployment options

Azure Functions

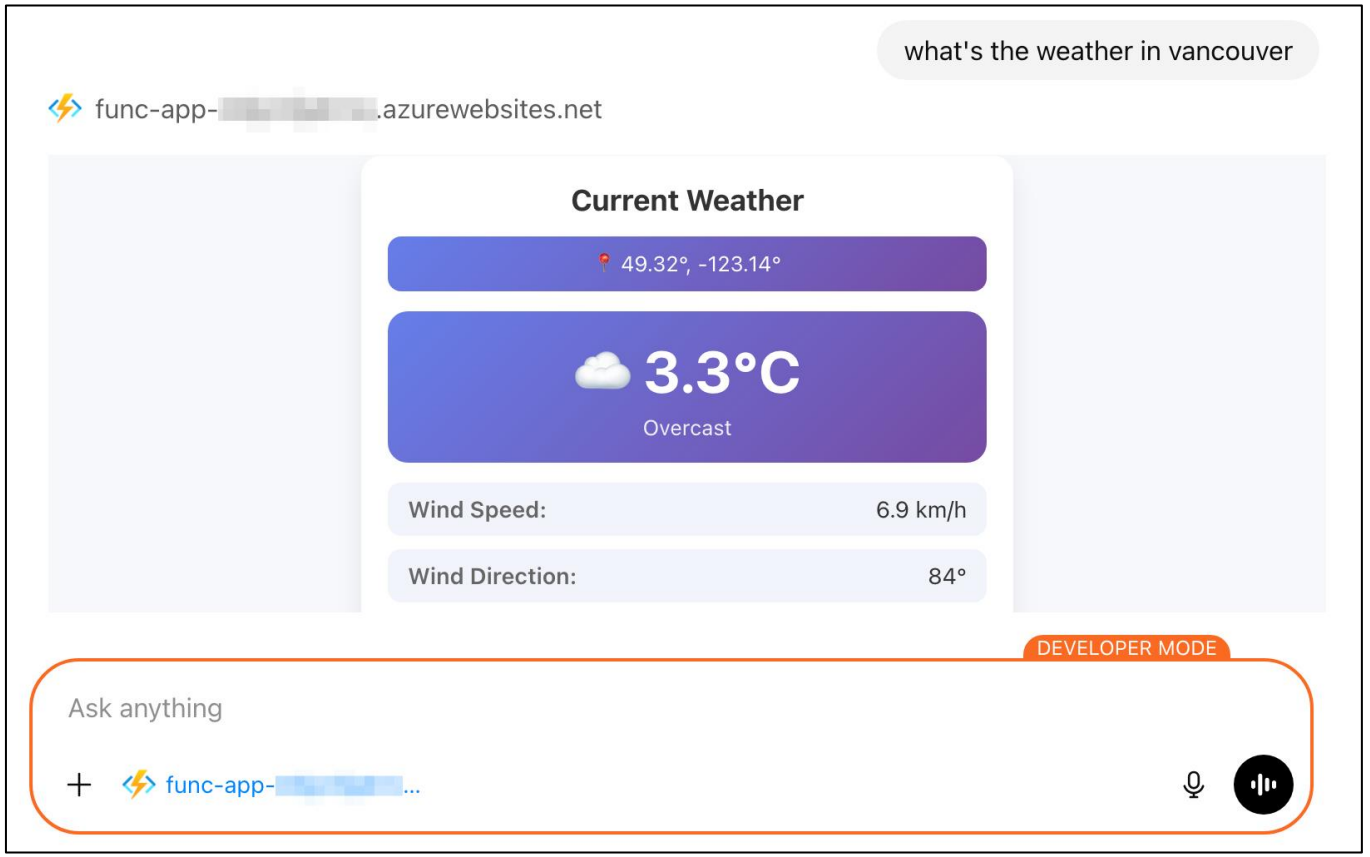
When deploying MCP servers on Functions, you can either:

- Use Azure function MCP bindings
  - https://learn.microsoft.com/azure/azure-functions/functions-bindings-mcp-trigger

```
@app.mcp_tool_trigger(
  arg_name="context",
  tool_name="add_expense",
  description="Save an expense to the database",
  tool_properties=tool_properties_add_expense_json,
)
def add_expense(context) -> str: ...
```

- Bring your own MCP server, like FastMCP servers
  - https://learn.microsoft.com/azure/azure-functions/self-hosted-mcp-servers
  - https://learn.microsoft.com/azure/azure-functions/scenario-host-mcp-server-sdks

Sample: Weather app for ChatGPT



This app builds a ChatGPT app using the FastMCP Python framework. It fetches real-time weather data and uses the OpenAI Apps SDK to display an interactive HTML widget directly within ChatGPT conversations.

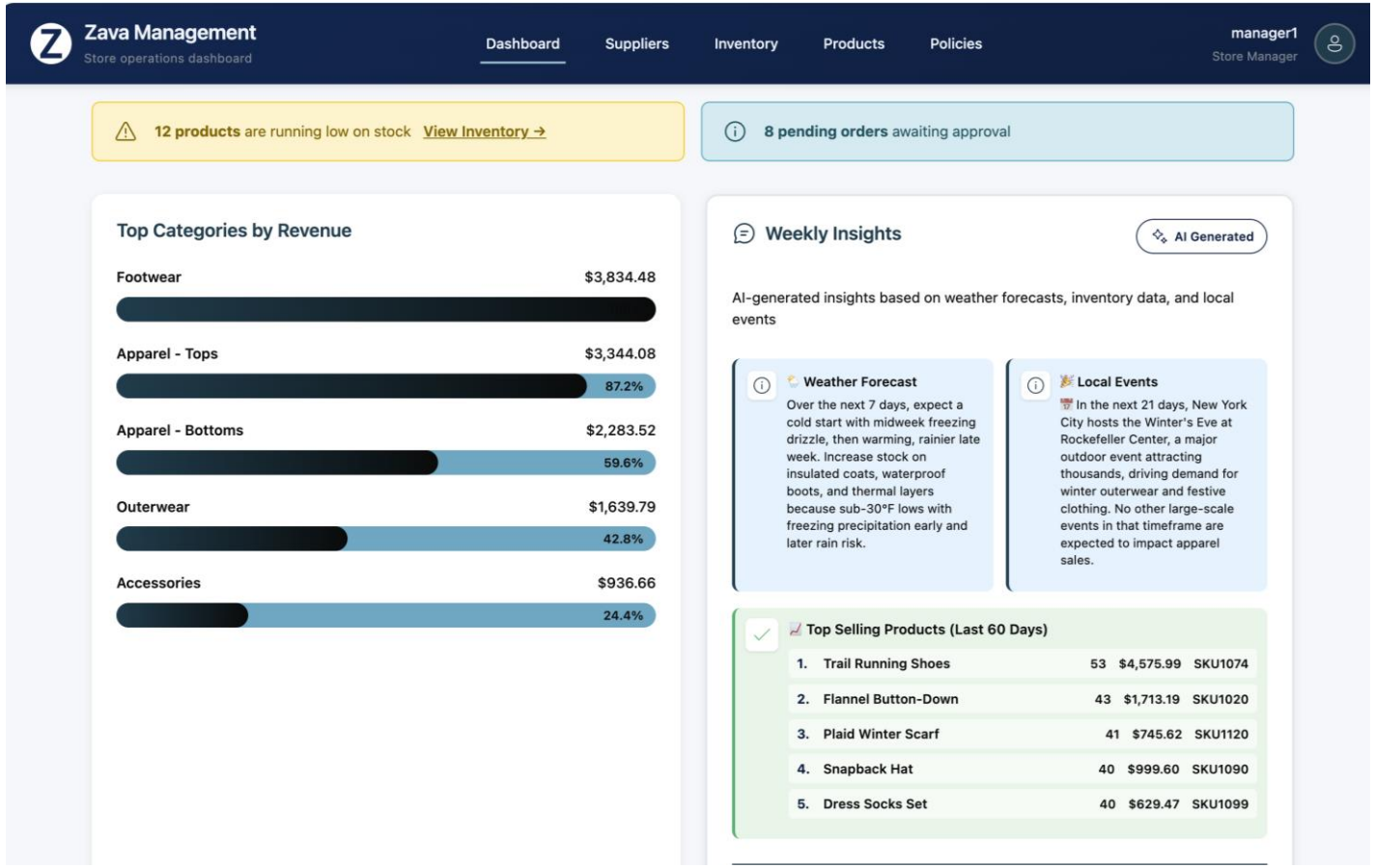
https://github.com/anthonychu/chatgpt-app-weather-functions-selfhosted

Azure Kubernetes Service

Use docker-compose.yaml to define the MCP and agent services:

```yaml
services:
  mcp:
    build:
      context: ./servers
      dockerfile: Dockerfile
    expose:
      - "8000"
  agent:
    build:
      context: ./agents
      dockerfile: Dockerfile
      environment:
        - MCP_HTTP=http://mcp:8000
    expose:
      - "8000"
    depends_on:
      - mcp
```

Sample: Agentic Shop



This app uses docker-compose to deploy 2 MCP servers, an API server (that runs agents), plus a frontend server.

```
agent = chat_client.create_agent(
  name="Retail Insights Analyzer",
  tools=[finance_mcp],
)
response = await agent.run(
  "Top 5 selling products for store 1",
  response_format=ProductsAgentResponse
)
products = response.value.products
```

https://github.com/tonybaloney/agentic-popup-shop

Next steps

Watch past recordings:

aka.ms/pythonmcp/resources

Come to office hours after each session in Discord:

aka.ms/pythonai/oh

Learn from MCP for Beginners:

aka.ms/mcp-for-beginners

🛠️ Dec 16:

Building MCP servers with FastMCP

☁️ Dec 17:

Deploying MCP servers to the cloud

👤 Dec 18:

Authentication for MCP servers