Microsoft

# Hands-On Demo
# Azure PaaS App

**App Url:** https://salespoc.azurewebsites.net

**GitHub Url**: https://github.com/csdmichael/SalesPOC.UI

**Read Me**: Frontend UI for Sales POC App

**Prepared by**:
Michael Yaacoub
Sr. Solution Engineer – AI Applications
Microsoft

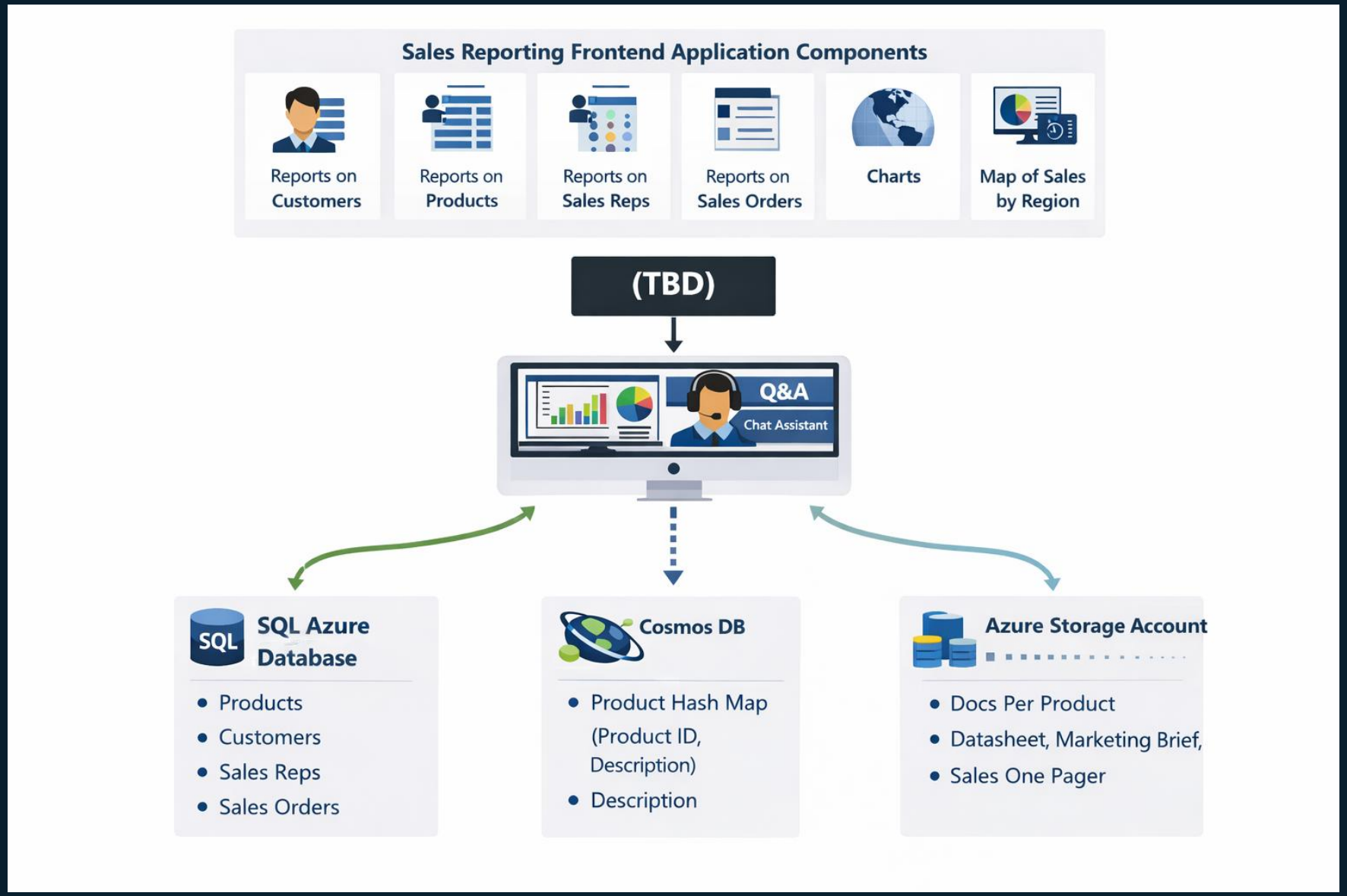
# Steps for building Sales POC

Microsoft

0. High Level Requirements & UX Mockups
1. Identify functional Requirements and non-functional Requirements
2. Generate Architecture Diagram to satisfy requirements
3. Build APIs and publish to Azure Container App
4. Create new API Management Instance in Azure and Import Sales-API
5. Create MCP Server in APIM
6. Go to API Center -> Register APIM Instances for Dev, Testing, Prod for API discovery & Linting
7. Go to AI Foundry -> Build an agent that uses MCP Server (exposed in APIM) as a tool
8. Build Frontend Application and Publish to Azure App Service
9. Test the Application & Sales Chat AI Assistant

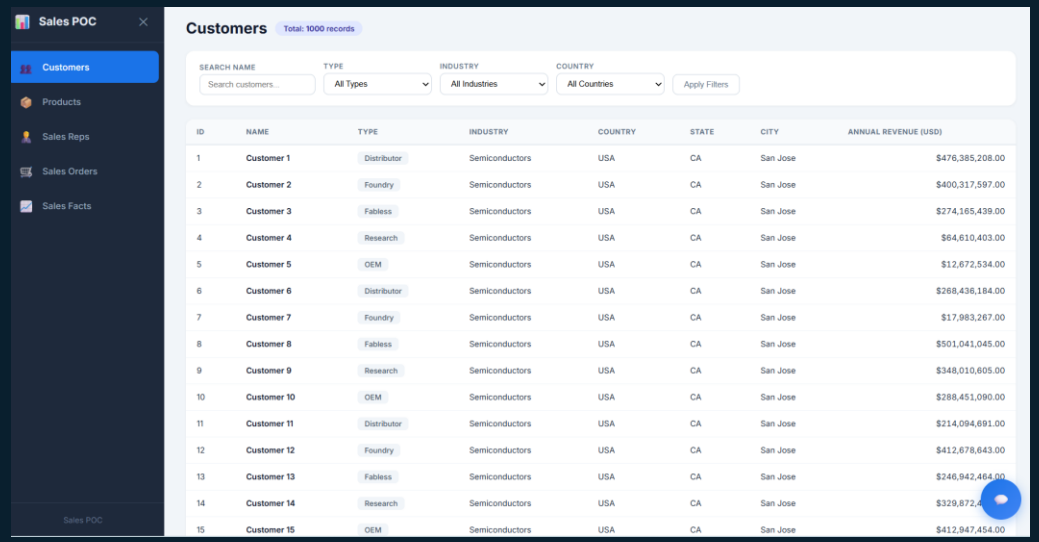
# 0. High Level Requirements

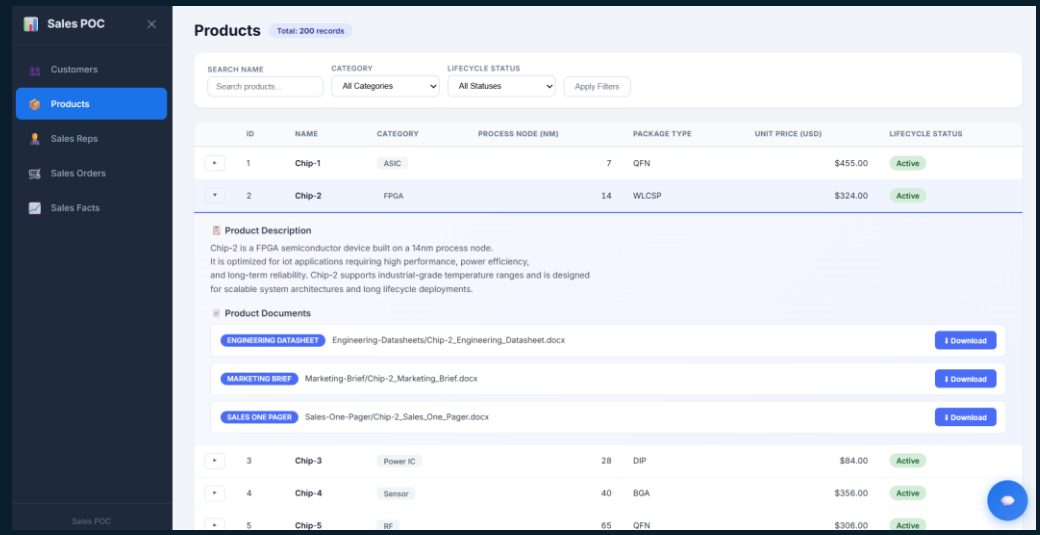
Microsoft

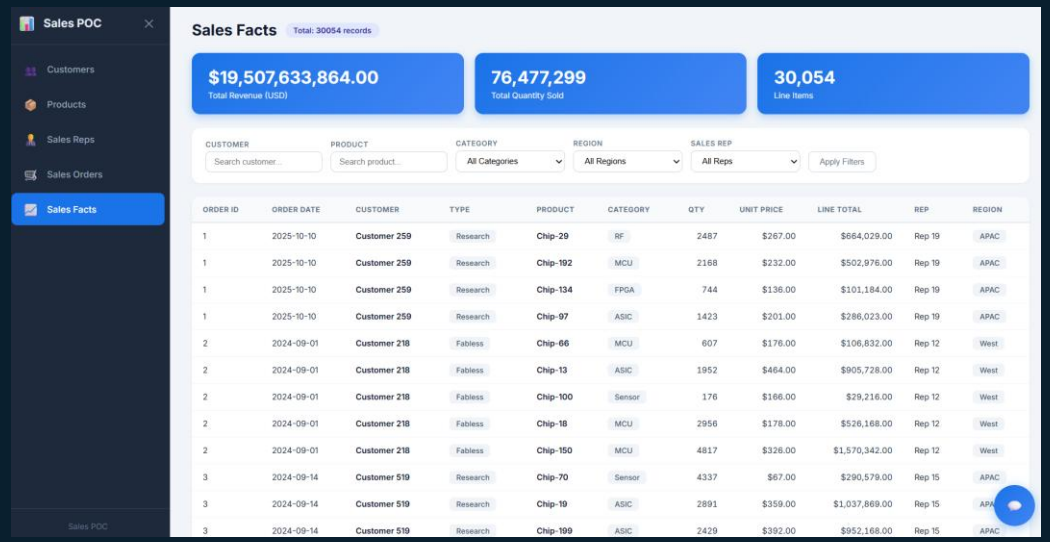


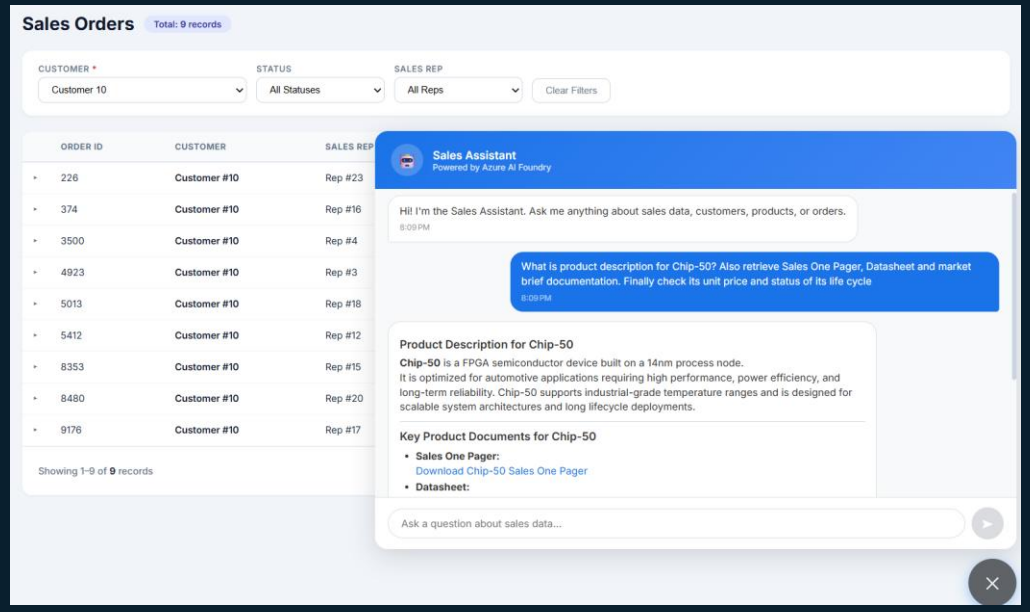
# 0. UX Mockups

Microsoft



# 1. Identify functional Requirements and non-functional Requirements

Microsoft

## Functional Requirements *(What the system does for the user)*

### 1. Reports to display and filter data

Provide interactive reports and dashboards for
**Customers** (segmentation), **Products** (performance), and **Sales Reps** (quota attainment)

### 2. Real-Time Order Tracking

Real-time visibility into order status, fulfillment pipelines, and historical transaction logs

### 3. AI-Powered Natural Language Query (NLQ)

An integrated chatbot that allows users to ask questions like *"What are the top 10 products by price?*" and receive instant data visualizations.

## Next Phase

### 1. Role-Based Access Control (RBAC)



## Non-Functional Requirements

*How the system operates and its quality constraints.*

### 1. Intuitive UX/UI

A "zero-training" interface designed for mobile and desktop, ensuring high adoption rates among non-technical sales staff.

### 2. Low Latency Performance

Chatbot responses and dashboard renders must occur in **under 5 seconds** to maintain user engagement.

### 3- Reusability of APIs across organization

Reuse same APIs in Sales Transactional System (Consumer Application) and make them available for internal teams

## Next Phase

### 1. Scalability – Low priority as it is internal App

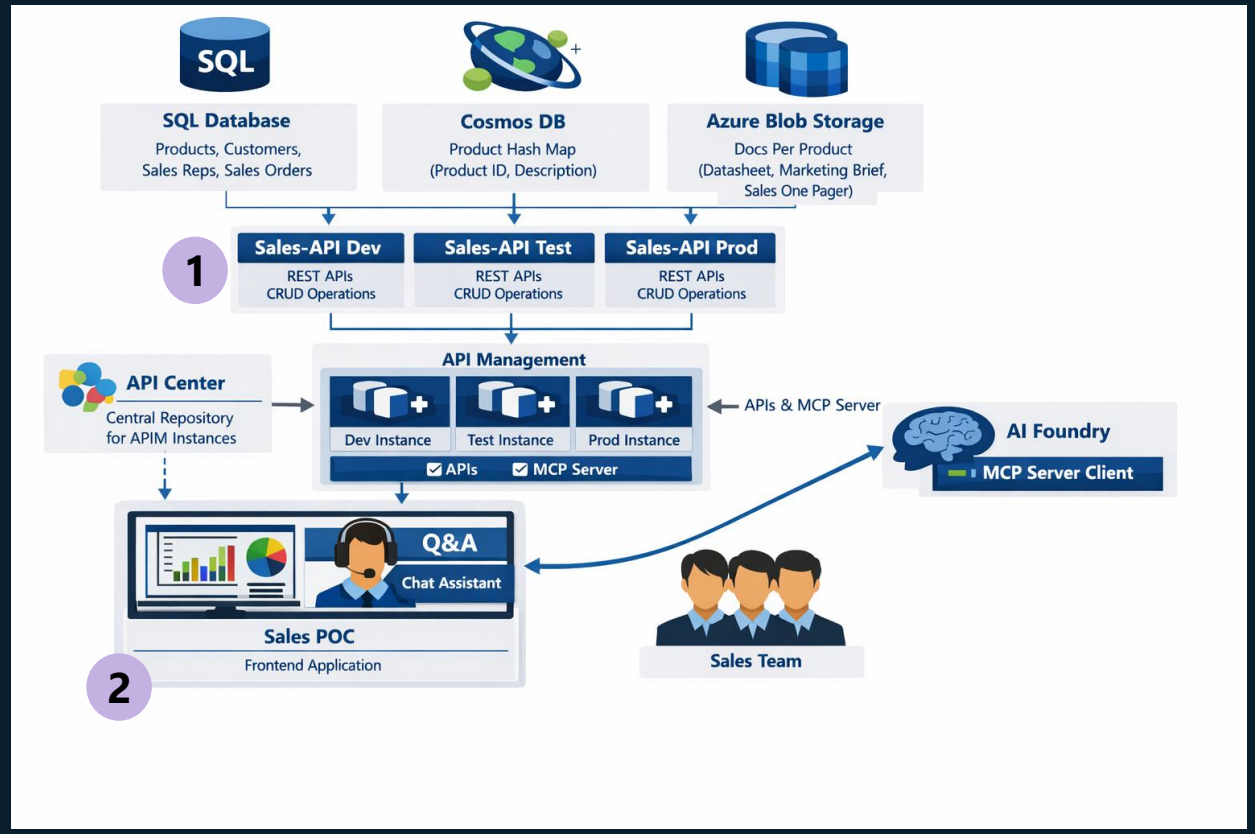
### 2. High Availability – Low priority as it is internal App

### 2. Data Security & Compliance

# 2. Generate Architecture Diagram to satisfy requirements

Microsoft

- Account for big Picture & Integrations & Reusability in other systems
- Estimate the system capacity in short and long term to determine the proper Services, Service Tiers, SKUs
- Choose relevant Azure Services to implement a reliable / scalable / cost effective Architecture




## Decisions to make

1. **Which Service to use for Sales-API?**
   - ➢ We need to review big picture
2. **Which Service to use for Frontend Internal Reporting Web App?**
   - ➢ Azure App Service (Internal Sales Team – Small number of users)

# 2. Account for big picture - Sales Transactional System

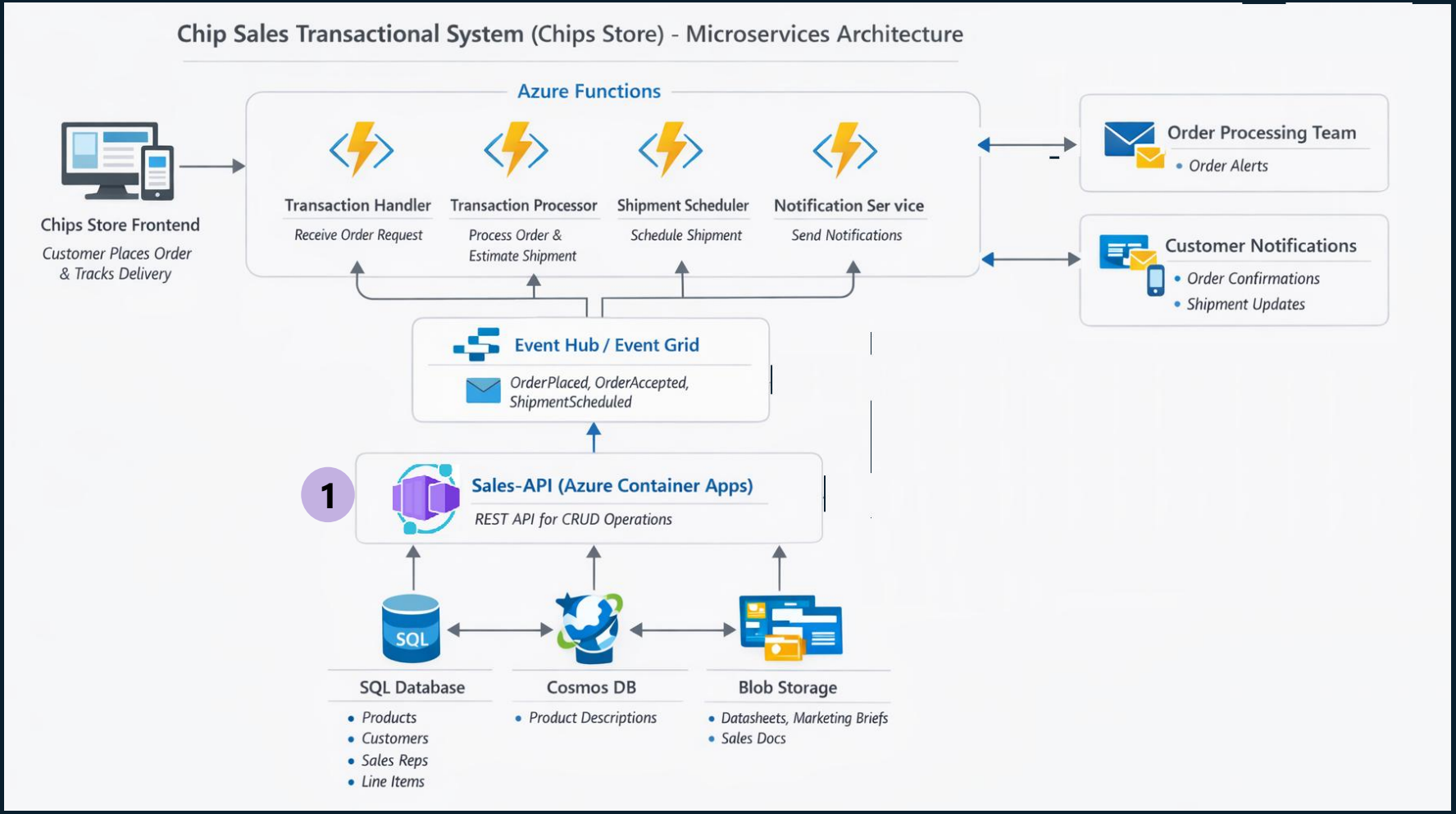
Microsoft

**Big Picture helps understand scale**

**Sales-API** can use **"Azure Container Apps"**

**Why?**

**Shared service in internal Report System and Sales Transactional System used by external Consumers**



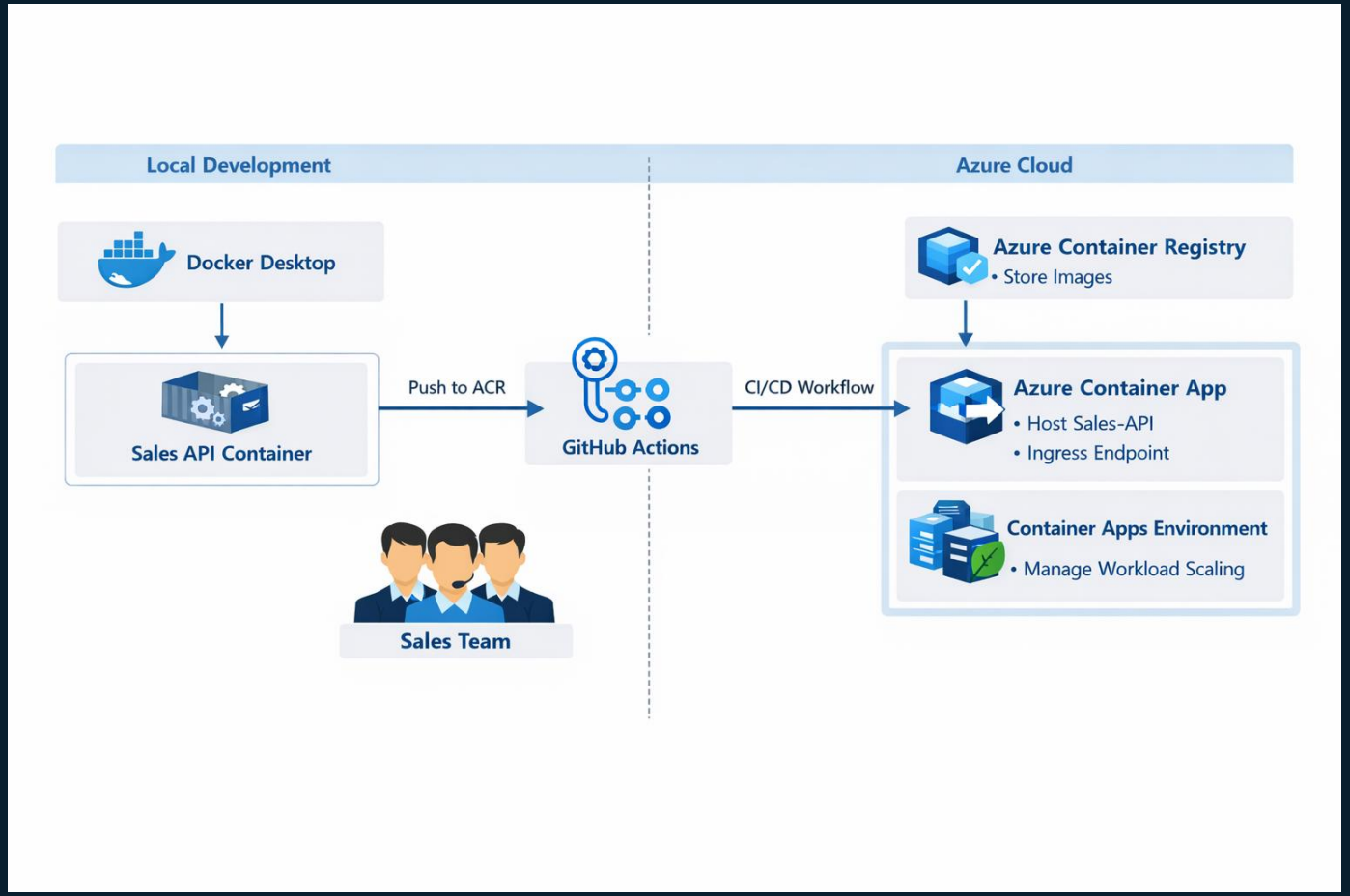
# 3. Build APIs – Azure Container Apps

Microsoft

## Components Needed

- **Docker Desktop**
- **GitHub Action** to push docker container (e.g. Sales-API) to **ACR**
- **ACR** stores Container Images
- Deploy Sales-API from **ACR** to **ACA** (Azure Container App)
- **ACA** hosts Sales-API
- **Container Apps Environment** manages Workload Scaling

## Flow to create Azure Container App




- **ACR**: Azure Container Registry
- **ACA**: Azure Container Apps

# 3. Build APIs – Azure Container Apps

Microsoft

1. Go to Azure Portal
   a) Create "**Azure Container Registry**" and fetch Uri (Identifier)
   b) Create "**Azure Container App**" and fetch Uri (Identifier)

2. Fetch Connection Strings for all Data Sources that you need to connect to in your App Service

3. Create a GitHub Repository (e.g. SalesPOC.API) and clone it to your computer

4. Open Visual Studio Code (or Visual Studio) and leverage "Copilot Coding Agent" to generate the source code.
   **Example Prompt**: "Generate a .NET 10 REST APIs to implement CRUD operations for the following data sources: {Paste list of data source names and connection strings / access keys} and generate GitHub actions workflow file to deploy the code to Azure Container Registry and Azure Container App {Paste Uri / identifier for ACR, ACA}. Add an ingress https endpoint for Sales-API in Azure Container App".

5. Run Code locally in browser to ensure that it works as expected or re-prompt Copilot to fix issues

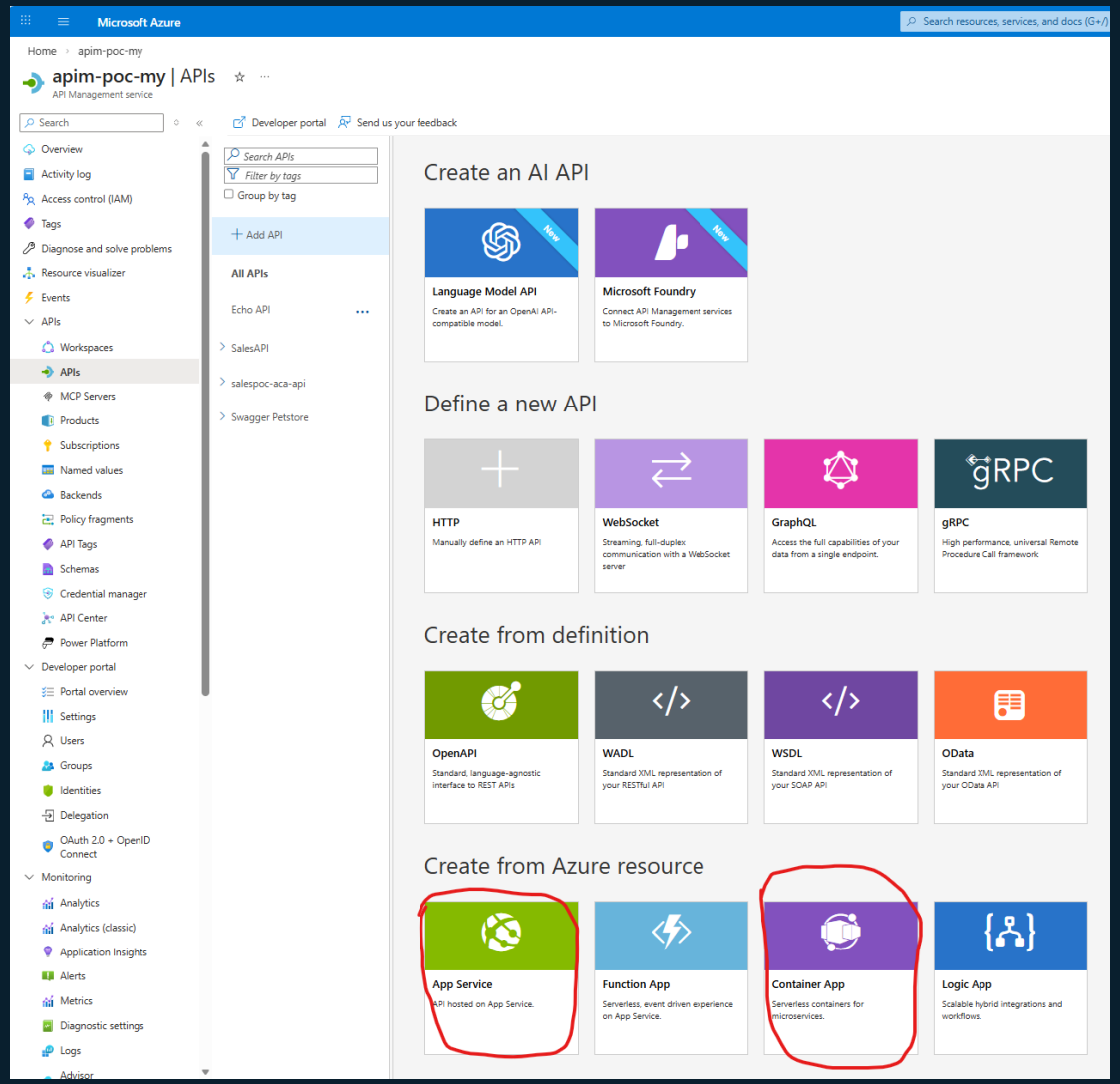
6. Ask Copilot Coding Agent to generate a swagger documentation and / or OpenAI Specs to import in APIM later

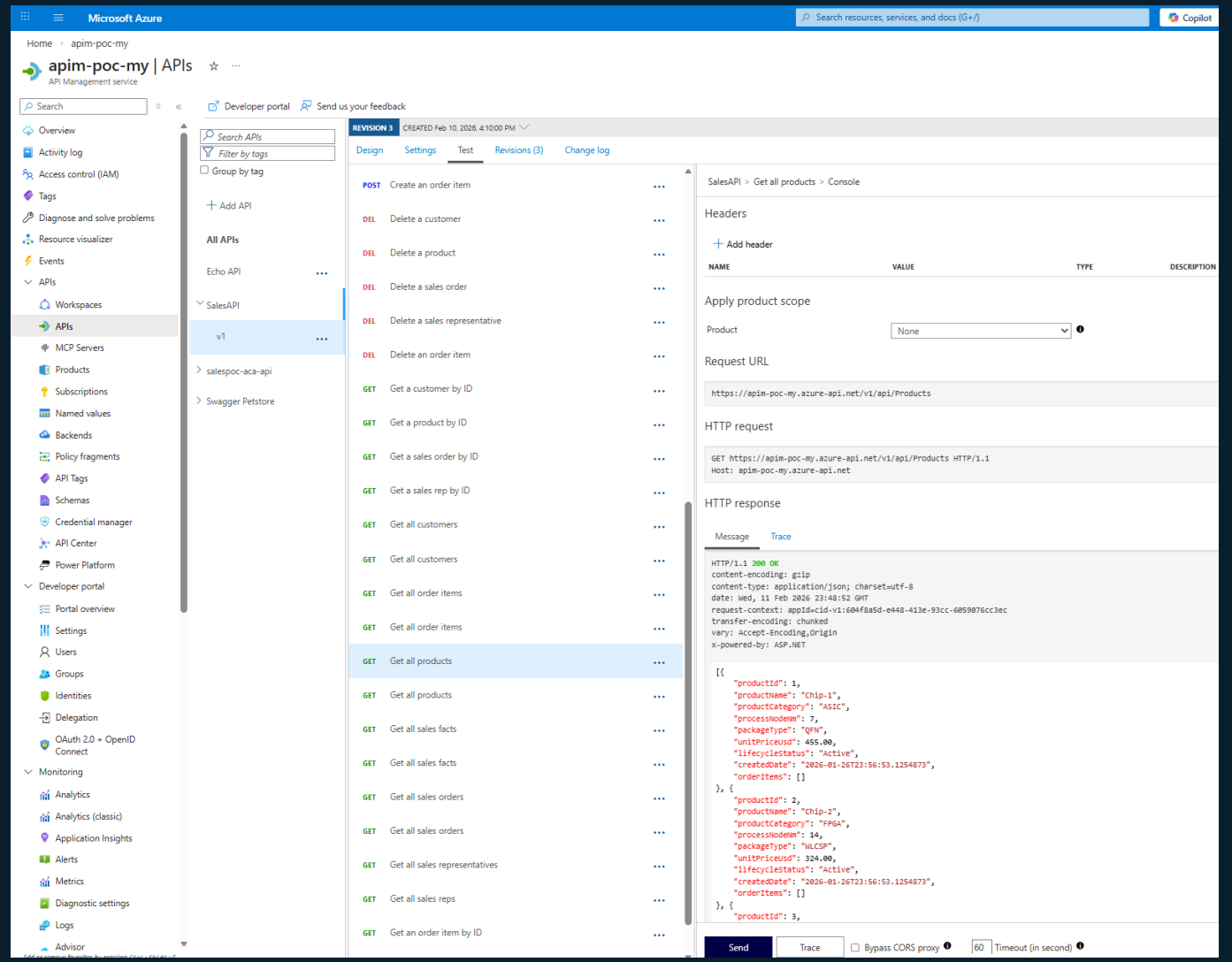
7. Go to GitHub Repo online and ensure that GitHub Actions ran successfully and deployed App

8. Test APIs

# 4. Create new API Management Instance in Azure and Import Sales-API

Microsoft

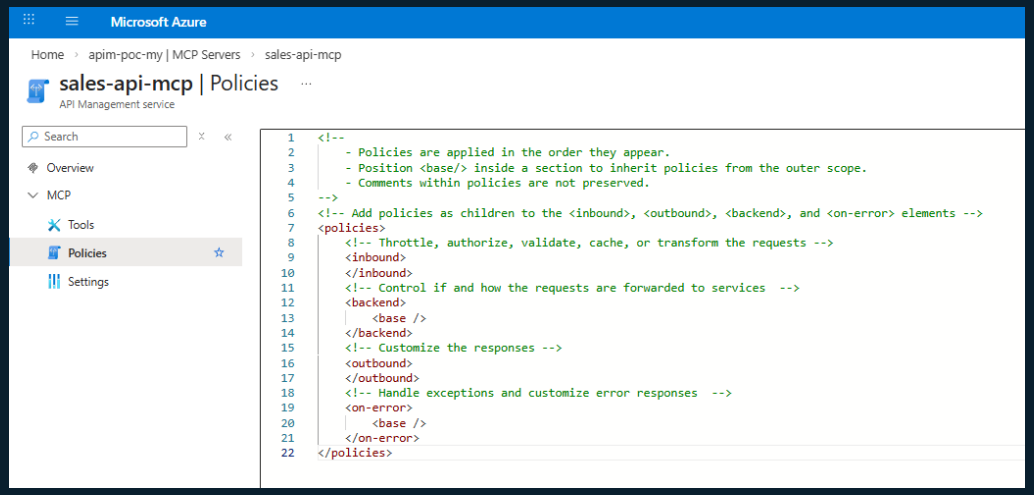


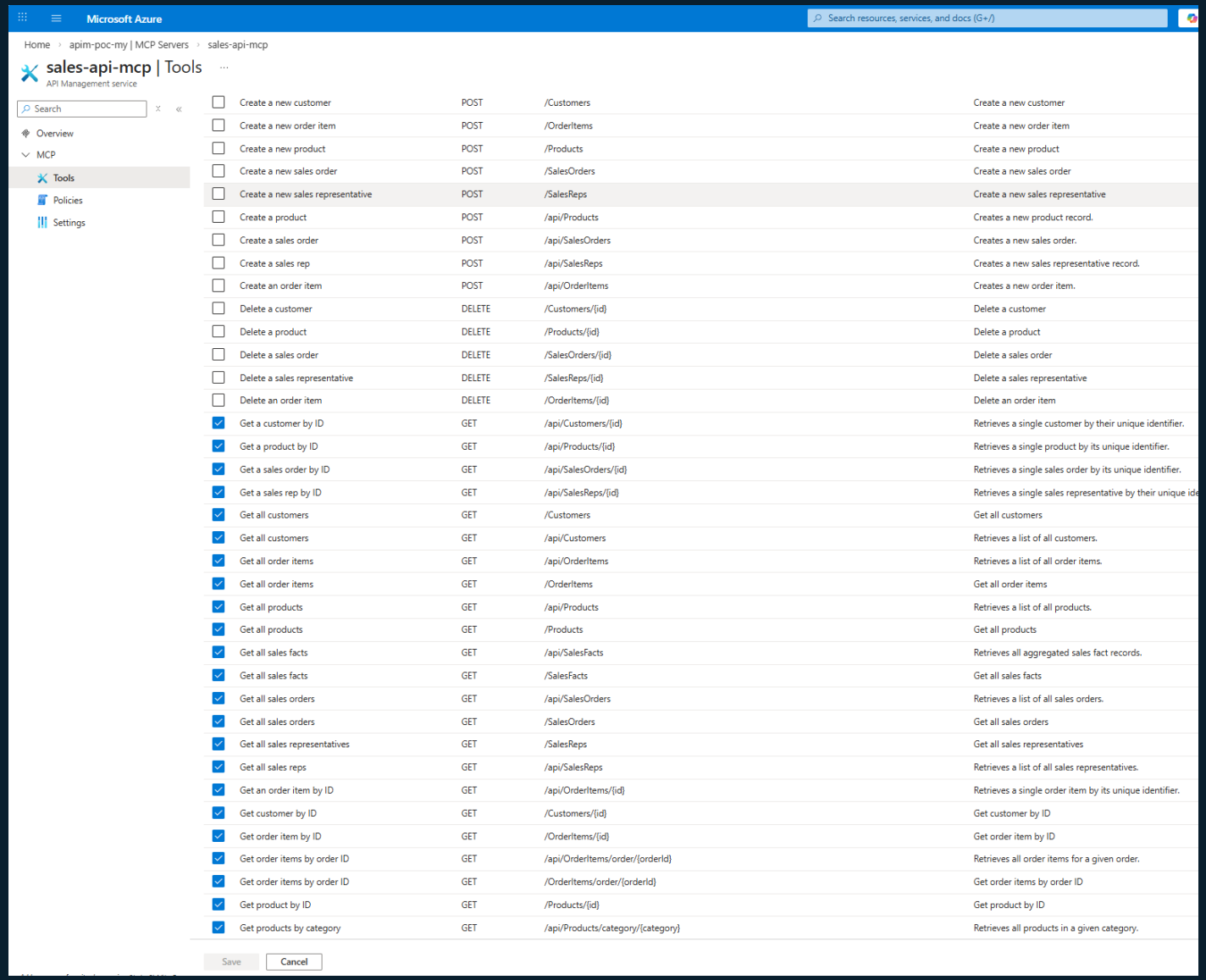



# 5. Create MCP Server in APIM

Microsoft

- Expose Tools (API operations)
- Define policies for AI Agents to consume MCP Server

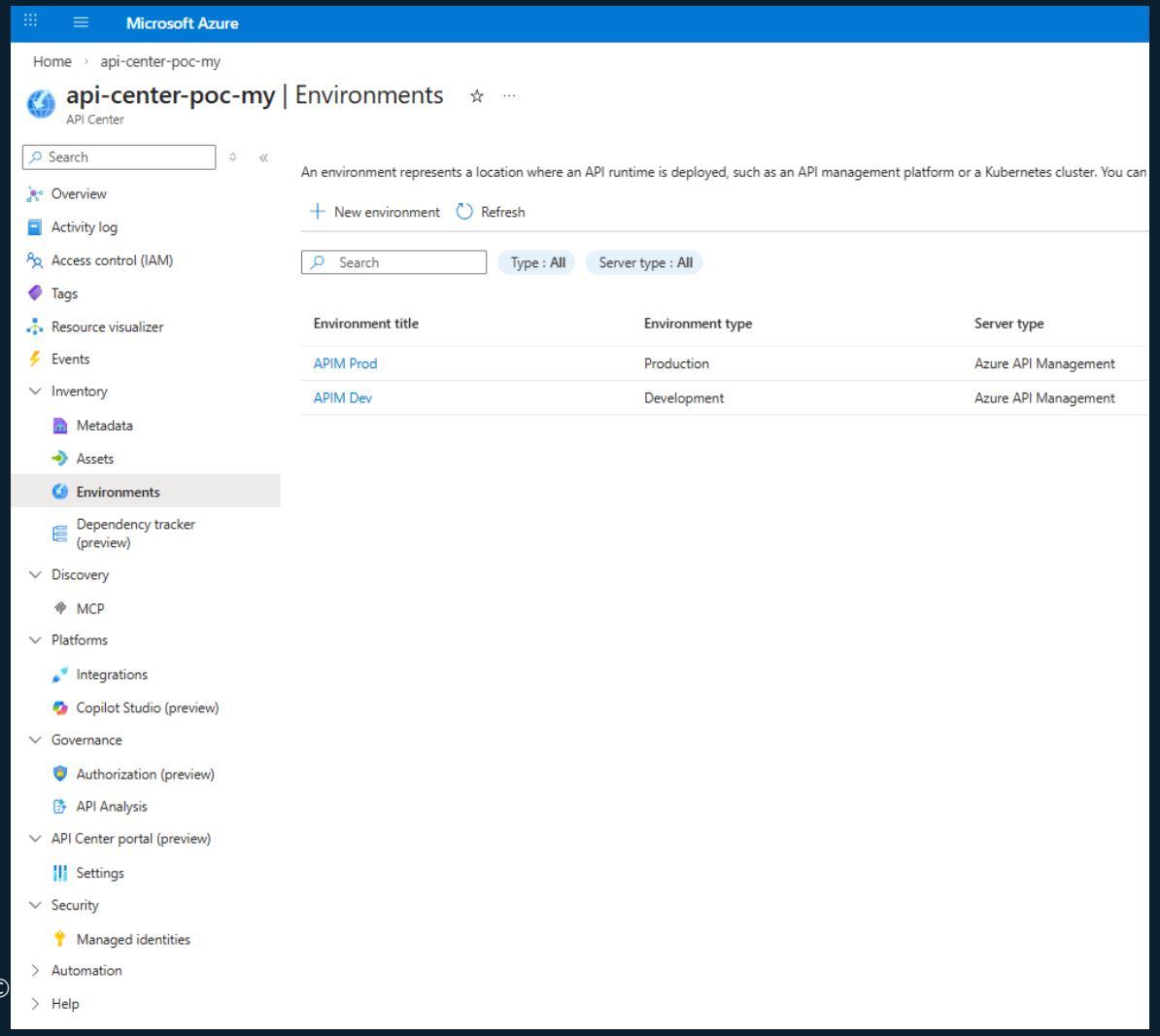


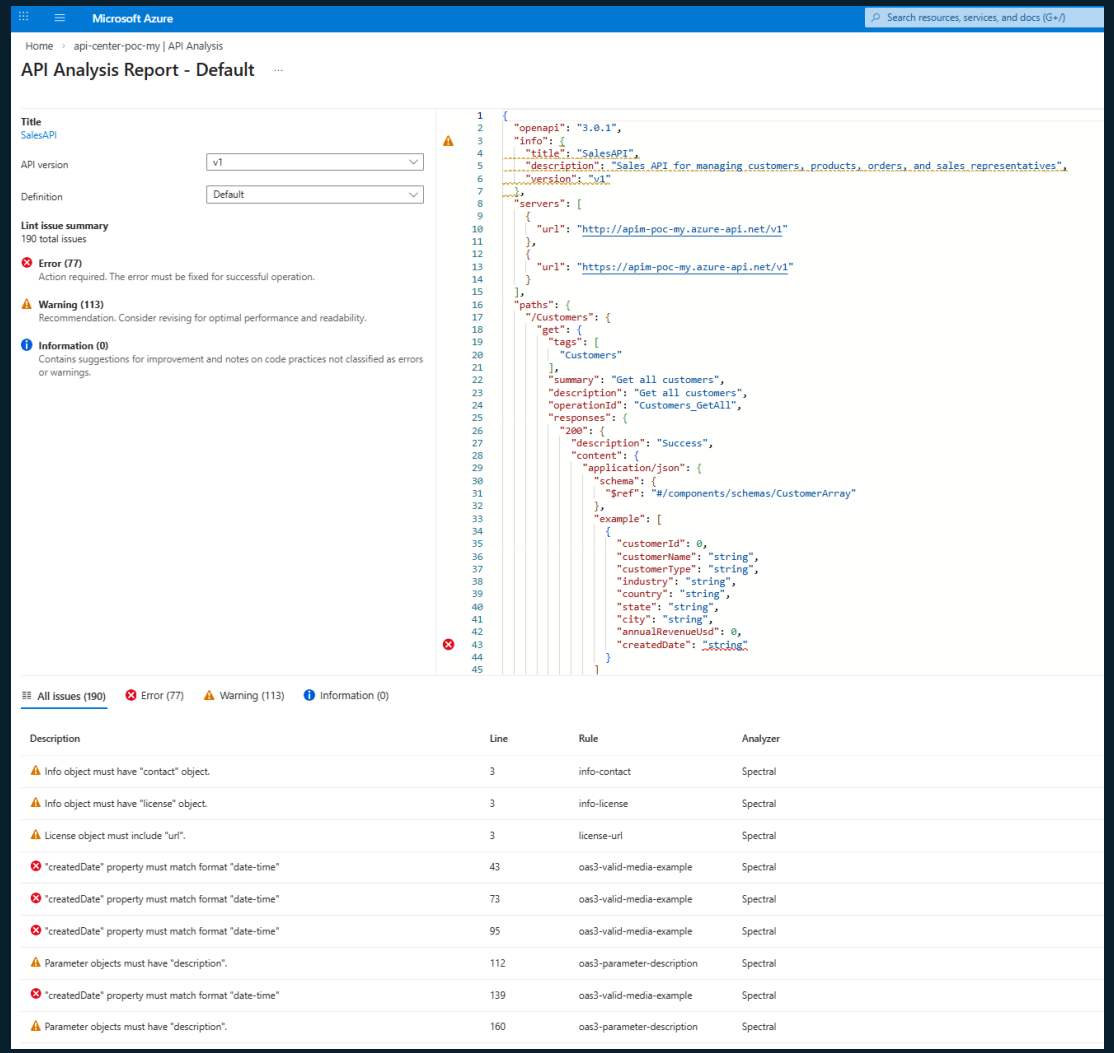
# 6. API Center – Register APIM Instance

Microsoft

- Register APIM Instances for Dev, Test, Prod
- Review Portal to discover APIs & MCP Servers



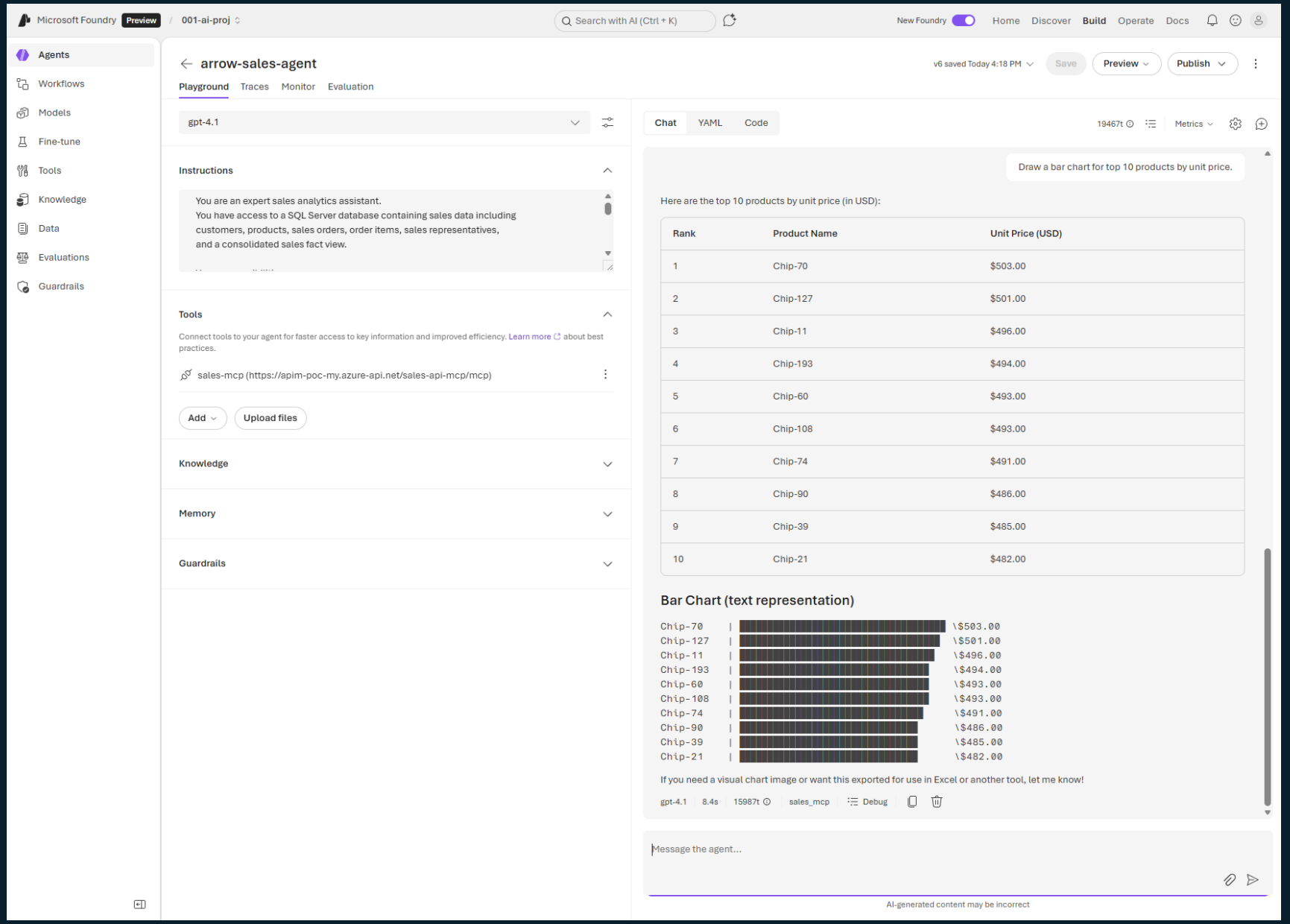

- API Analysis / Linting



# 7. AI Foundry – Build Sales Agent using MCP Server

Microsoft

- Create a new agent
- Add MCP Server as a Tool under Tools
- Provide clear instructions
- Add Guardrails (Responsible AI)



# 8. Build Frontend web App – Azure App Service

Microsoft

1. Go to Azure Portal -> Create “Web App” Service and fetch Uri (Identifier)

2. Fetch API URL from APIM that will be used in the Frontend Web App

3. Create a GitHub Repository (e.g. SalesPOC.UI) and clone it to your computer

4. Open Visual Studio Code (or Visual Studio) and leverage “Copilot Coding Agent” to generate the source code.

   **Example Prompt**:
   “Generate a web application using Angular and Typescript to display and filter data using this API End-Point {Paste Sales-API URL from APIM}.
   Create a separate screen for Customers, Products, Sales Reps, Sales Orders, Sales Facts.
   For each product in Products screen, add a button to show the Product description and Product documentation which includes 3 documents per product (Datasheet, Marketing Brief, Sales One Pager).
   Finally, add a Sales Chat Assistant that uses sales-agent from AI Foundry.
   AI Foundry Project Endpoint is {Paste Foundry Project Endpoint} and API Key is {Paste API Key}”

5. Run Code locally in browser to ensure that it works as expected or re-prompt Copilot to fix issues

6. Go to GitHub Repo online and ensure that GitHub Actions ran successfully and deployed App

7. Test Application

# Sales POC Alternative Architecture – Data Platform

Microsoft