Lesson 10 · Microsoft Semantic Kernel (C# / .NET)

PDF: this lesson · **Install (Linux · macOS · Windows):** guide · PDF

> **Part of local-ai-lab** - a hands-on course for building local AI.

> **Course home:** https://nikolareljin.github.io/local-ai-lab/ **Source:** https://github.com/nikolareljin/local-ai-lab

> **Lessons:** 1 · RAG → 2 · MCP → 3 · Hybrid retrieval → 4 · RAG safety → 5 · RAG evaluation → 6 · Repo assistant → 7 · LangChain → 8 · LangGraph → 9 · Ollama tools → **10 · Semantic Kernel (you are here)** → 11 · Bedrock Agents → 12 · Google ADK → ... → 15 · Docs from changes

> **Status: planned.** Outline below; full published slideshow lesson + step-by-step coming later. This is the course's **first C# / .NET lesson**, and it **runs locally** (against Ollama or a local OpenAI-compatible endpoint). Star the repo to follow along.

The idea

So far the course has been Python. **Microsoft Semantic Kernel (SK)** is the leading agent SDK for the **.NET / C#** world (with Python and Java ports). In this lesson you rebuild the document agent in C#, learning SK's model: a **Kernel**, **plugins** (your functions), and **automatic function calling**.

If you've done Lesson 9, you'll recognize the shape - SK's plugins are function calling with batteries included and strong typing.

What you'll learn

- **The Kernel** - SK's composition root; registering AI services and plugins
- **Connectors** - wiring SK to a **local** model (Ollama / OpenAI-compatible) so nothing leaves your machine
- **Native plugins** - exposing a C# method as a tool with `[KernelFunction]` (your `SearchDocs`)
- **Automatic function calling** - `FunctionChoiceBehavior.Auto()` so the model calls plugins on its own
- **Prompt + planning** - templated prompts and how SK orchestrates multi-step calls

The design we'll build

```
using Microsoft.SemanticKernel;
using System.ComponentModel;

// A native plugin: the C# equivalent of Lesson 1's retriever as a tool.
```

```
public class DocsPlugin
{
   [KernelFunction, Description("Search the user's local documents for relevant passages.")]
   public string SearchDocs([Description("What to search for")] string query)
       => DocsIndex.Search(query, k: 5);   // returns "[source:page] text" blocks
}

var builder = Kernel.CreateBuilder();
builder.AddOpenAIChatCompletion(            // point at a LOCAL endpoint
   modelId: "llama3.1",
   endpoint: new Uri("http://localhost:11434/v1"),
   apiKey: "ollama");
builder.Plugins.AddFromType<DocsPlugin>();
var kernel = builder.Build();

var settings = new OpenAIPromptExecutionSettings {
   FunctionChoiceBehavior = FunctionChoiceBehavior.Auto()
};
var answer = await kernel.InvokePromptAsync(
   "How do I reset the device? Cite sources.", new(settings));
```

> **The through-line:** `DocsPlugin.SearchDocs` is the same `search_docs` capability from Lessons 1, 2 and 9 - just expressed in idiomatic C#. The concept is portable; only the syntax changes.

Builds on

Concept	From
a search tool the model can call	Lesson 9
grounding + citations	Lesson 1
plugins, kernel, auto function calling in C#	**Lesson 10 (this one)**

Prerequisites

.NET 8 SDK; a local model endpoint (Ollama's OpenAI-compatible API, or LM Studio). Concepts from Lesson 9 help but aren't required.

Next lesson

Lesson 11 · AWS Bedrock Agents →
