

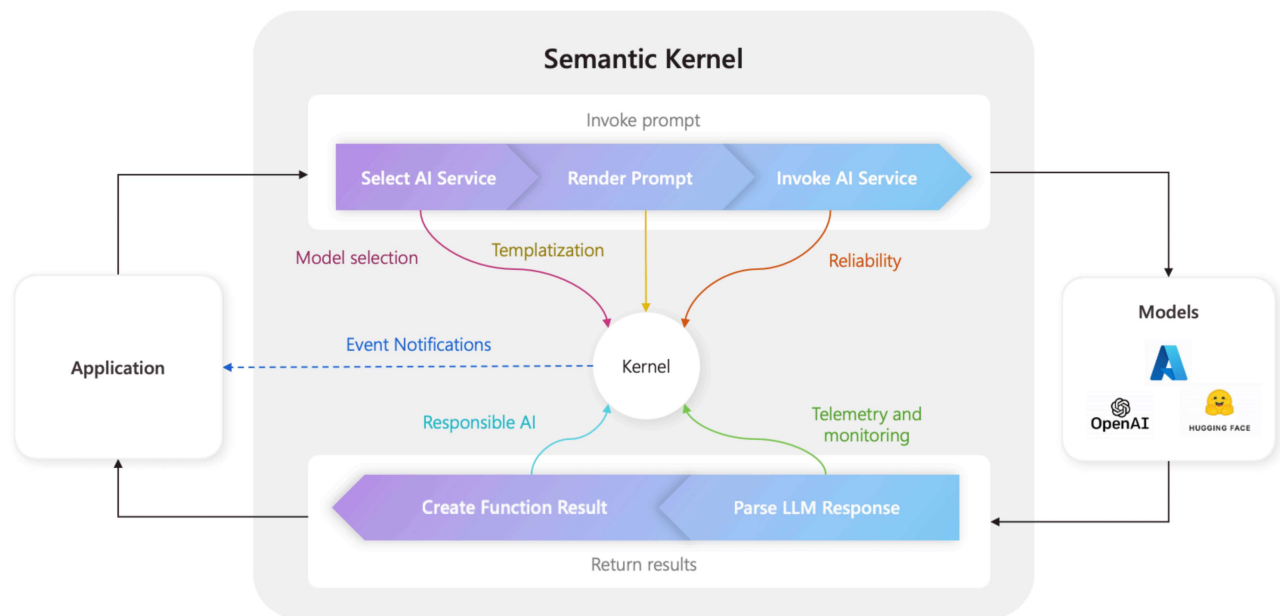
# Understanding the kernel

Article • 07/25/2024

The kernel is the central component of Semantic Kernel. At its simplest, the kernel is a Dependency Injection container that manages all of the services and plugins necessary to run your AI application. If you provide all of your services and plugins to the kernel, they will then be seamlessly used by the AI as needed.

## The kernel is at the center

Because the kernel has all of the services and plugins necessary to run both native code and AI services, it is used by nearly every component within the Semantic Kernel SDK to power your agents. This means that if you run any prompt or code in Semantic Kernel, the kernel will always be available to retrieve the necessary services and plugins.




This is extremely powerful, because it means you as a developer have a single place where you can configure, and most importantly monitor, your AI agents. Take for example, when you invoke a prompt from the kernel. When you do so, the kernel will...

1. Select the best AI service to run the prompt.
2. Build the prompt using the provided prompt template.
3. Send the prompt to the AI service.
4. Receive and parse the response.
5. And finally return the response from the LLM to your application.

Throughout this entire process, you can create events and middleware that are triggered at each of these steps. This means you can perform actions like logging, provide status updates to users, and most importantly responsible AI. All from a single place.

## Build a kernel with services and plugins

Before building a kernel, you should first understand the two types of components that exist:

 Expand table

Components	Description
1 Services	These consist of both AI services (e.g., chat completion) and other services (e.g., logging and HTTP clients) that are necessary to run your application. This was modelled after the Service Provider pattern in .NET so that we could support dependency injection across all languages.
2 Plugins	These are the components that are used by your AI services and prompt templates to perform work. AI services, for example, can use plugins to retrieve data from a database or call an external API to perform actions.

To start creating a kernel, import the necessary packages at the top of your file:

C#

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Plugins.Core;
```

Next, you can add services and plugins. Below is an example of how you can add an Azure OpenAI chat completion, a logger, and a time plugin.

C#

```
// Create a kernel with a logger and Azure OpenAI chat completion service
var builder = Kernel.CreateBuilder();
builder.AddAzureOpenAIChatCompletion(modelId, endpoint, apiKey);
builder.Services.AddLogging(c => c.AddDebug().SetMinimumLevel(LogLevel.Trace));
builder.Plugins.AddFromType<TimePlugin>();
Kernel kernel = builder.Build();
```

# Using Dependency Injection

In C#, you can use Dependency Injection to create a kernel. This is done by creating a `ServiceCollection` and adding services and plugins to it. Below is an example of how you can create a kernel using Dependency Injection.

## Tip

We recommend that you create a kernel as a transient service so that it is disposed of after each use because the plugin collection is mutable. The kernel is extremely lightweight (since it's just a container for services and plugins), so creating a new kernel for each use is not a performance concern.

C#

```
using Microsoft.SemanticKernel;

var builder = Host.CreateApplicationBuilder(args);

// Add the OpenAI chat completion service as a singleton
builder.Services.AddOpenAIChatCompletion(
    modelId: "gpt-4",
    apiKey: "YOUR_API_KEY",
    orgId: "YOUR_ORG_ID", // Optional; for OpenAI deployment
    serviceId: "YOUR_SERVICE_ID" // Optional; for targeting specific services
    within Semantic Kernel
);

// Create singletons of your plugins
builder.Services.AddSingleton(() => new LightsPlugin());
builder.Services.AddSingleton(() => new SpeakerPlugin());

// Create the plugin collection (using the KernelPluginFactory to create plug-
// ins from objects)
builder.Services.AddSingleton<KernelPluginCollection>((serviceProvider) =>
[
    KernelPluginFactory.CreateFromObject(serviceProvider.GetRequiredService<LightsP
    lugin>()),

    KernelPluginFactory.CreateFromObject(serviceProvider.GetRequiredService<Speaker
    Plugin>())
]);

// Finally, create the Kernel service with the service provider and plugin col-
```

```
lection
builder.Services.AddTransient((serviceProvider)=> {
    KernelPluginCollection pluginCollection =
    serviceProvider.GetRequiredService<KernelPluginCollection>();

    return new Kernel(serviceProvider, pluginCollection);
});
```

### Tip

For more samples on how to use dependency injection in C#, refer to the [concept samples](#).

## Next steps

Now that you understand the kernel, you can learn about all the different AI services that you can add to it.

[Learn about AI services](#)