**Dapr Bindings**

Bindings offer a standardized method to trigger an application with events from external systems or invoke an external system with optional data payloads. Bindings are highly effective in supporting event-driven, on demand compute and contribute to the reduction of boilerplate code. With the bindings; you can avoid the complexities of connecting and pooling from messaging systems, focus on business logic, handle retries and failure recovery, switch between bindings at runtime, build portable applications with environment-specific bindings setup and no required code changes.

**The problem Dapr Observability tackles**

Dapr resource bindings empower your services to seamlessly connect business operations with external resources beyond the immediate application scope. When an event occurs in an external system, it can initiate an operation in your service while providing relevant contextual information. Subsequently, your service can extend this operation by triggering an event in another external system, along with passing contextual payload information. This communication occurs without tight coupling or the need for your service to be aware of the external resource. The underlying infrastructure is encapsulated within predefined Dapr components, offering the flexibility to swap out components at runtime without requiring code modifications.

At first blush, resource binding behavior may appear similar to the Publish/Subscribe pattern. However, despite sharing some similarities, notable differences set them apart. The Publish/Subscribe mechanism centers on asynchronous communication within Dapr services. In contrast, resource binding extends its scope to facilitate system interoperability across various software platforms. It facilitates the exchange of information among distinct applications, datastores, and services outside the confines of your microservice application.

**Dapr Bindings operational mechanism**

Dapr resource binding initiates with a component configuration file, expressed in YAML format. This file outlines the resource type and its associated configuration settings to which you will establish a binding. After configuration, your service gains the capability to either receive events from the resource or initiate events on it.

A blue hexagon with white text

Description automatically generatedIn the diagram:

The input binding initiates a method within your application and execute output binding operations on the component.

**Input Bindings**

Input bindings activate your code by responding to incoming events from external resources. To capture events and data, you register a public endpoint from your service, which serves as the event handler.

Dapr input binding flow:

1. The Dapr sidecar processes the binding configuration file, subscribing to the event specified for the external resource.
2. When the event source publish an event the binding component running in the Dapr sidecar picks it up and triggers an event.
3. The Dapr sidecar invokes the endpoint (that is, event handler) configured for the binding
4. After handling the event, the service returns an HTTP status code 200 OK.

Example Configuration of Input Binding (Cron Binding)

apiVersion: dapr.io/v1alpha1

kind: Component

metadata:

name: cronBinding

namespace: default

spec:

type: bindings.cron

version: v1

metadata:

- name: schedule

value: "@every 30s"

**Output Bindings**

Dapr also has output binding functionalities, allowing your service to initiate an event that invokes an external resource. Begin by setting up a YAML file for configuring the output binding. Once this configuration is established, proceed to initiate an event, prompting the invocation of the bindings API within the Dapr sidecar of your application. Once this configuration is configured, proceed to trigger an event, thereby invoking the bindings API within the Dapr sidecar of your application.

Dapr output binding flow:

1. The Dapr sidecar reads the binding configuration file, which contains details on how to establish a connection with the external resource.
2. Application invokes the endpoint on the Dapr sidecar.
3. The binding component running in the Dapr sidecar calls the external source.

Example Configuration of Output Binding (Http Binding)

apiVersion: dapr.io/v1alpha1

kind: Component

metadata:

name: httpBinding

namespace: default

spec:

type: bindings.http

version: v1

metadata:

- name: url

value: http://worldtimeapi.org/api/timezone/Europe/Istanbul

Using the configured input and output bindings in the examples above, let’s create a controller that writes current time to terminal every 30 seconds.

[Route("api/[controller]")]

[ApiController]

public class ScheduleController : ControllerBase

{

private readonly ILogger<ScheduleController> \_logger;

private readonly DaprClient \_daprClient;

public ScheduleController(ILogger<ScheduleController> logger, DaprClient daprClient)

{

\_logger = logger;

\_daprClient = daprClient;

}

[HttpPost]

public async Task HttpAndCronBindings()

{

\_logger.LogInformation($"{nameof(ScheduleController)} called!");

var response = await \_daprClient.InvokeBindingAsync(new BindingRequest("httpBinding", "get"));

var timeModel = JsonSerializer.Deserialize<TimeModel>(response.Data.Span);

\_logger.LogInformation($"Time in Istanbul {timeModel?.DateTimeUTC}");

}

}

In this example, the controller is called by cronBinding binding using Http Post and the controller uses the httpBinding binding to call the external service.

To run the application and the Dapr sidecar, the following command can be executed.

dapr run --app-id worker --app-port 5001 --dapr-http-port 3501 --app-ssl --components-path .\dapr\components\ dotnet run -- --urls=https://localhost:5001/ -p Worker/Worker.csproj.

You can access the code on GitHub in the https://github.com/adessoTurkey-dotNET/dapr-samples/tree/main/Bindings

The table below enumerates the supported generic input and output bindings provided by the Dapr bindings building block.

A screenshot of a computer

Description automatically generated

Additionally, bindings are available for other resources such as AWS and Microsoft Azure. You can find a detailed list on the Dapr docs.