

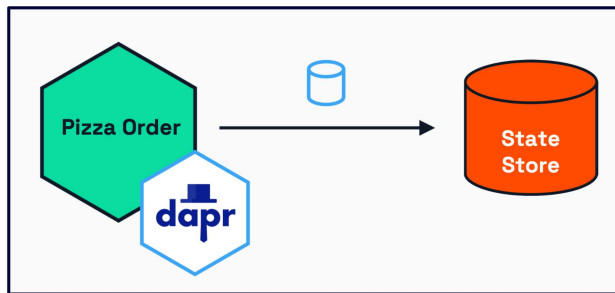
Challenge 1 - State Store

Overview

Ensure you have completed the [technical prerequisites](#) before starting the challenges.

In this challenge, you will:

- Configure a State Store component using a local Redis instance to save, get, and delete a pizza order.
- Update the `pizza-manager` application to use the Dapr State Management API.
- Run the app locally using the Dapr CLI.



To learn more about the Dapr State Management Building Block, refer to the [Dapr docs](#).

In your newly cloned `dapr-workshop-csharp` repository, navigate to the `start-here` folder

Configure the state store

Navigate to the `/resources` folder and create a new file called `statestore.yaml`. Add the content below to the file:

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: pizzastatestore
spec:
  type: state.redis
  version: v1
  metadata:
    - name: redisHost
      value: localhost:6379
    - name: redisPassword
      value: ""
```

This is a Dapr Component specification file named `pizzastatestore`. In the `spec` definition, note that the type of the component is `state.redis` and the metadata contains host and password information for the local Redis instance that was deployed as a container during Dapr's initialization process.

Install dependencies

Navigate to the `/PizzaOrder` directory. This folder contains all the files you need for your first service. Before beginning to code, install the Dapr dependencies by running the following in a new terminal window:

```
cd PizzaOrder

dotnet add package Dapr.Client
dotnet add package Dapr.AspNetCore
```

Register the DaprClient

Open `Program.cs` and add the `DaprClient` registration to the `ServiceCollection`:

```
builder.Services.AddControllers().AddDapr();
```

This enables the dependency injection of the `DaprClient` in other classes.

Create the service

Inside `Services/OrderStateService.cs` import the Dapr client.

```
using Dapr.Client;
```

This will import the *Dapr.Client* library from the [Dapr Dotnet SDK](#). That is what you will use to manage the state in the Redis instance.

Create a private field for the `DaprClient` inside the controller class:

```
private readonly DaprClient _daprClient;
```

Update the `OrderStateService` constructor to include the `DaprClient` and to set the private field:

```
public OrderStateService(DaprClient daprClient, ILogger<OrderStateService> logger)
{
    _daprClient = daprClient;
    _logger = logger;
}
```

Manage state

You will now update three existing functions: `UpdateOrderStateAsync`, `GetOrderAsync`, and `DeleteOrderAsync`.

1. Start by adding a readonly string to the `OrderStateService` class to represent the name of the Dapr state store component defined in the previous step. This name **must** be the same as the `metadata.name` in the Dapr component spec.

```
private readonly string STORE_NAME = "pizzastore";
```

2. `UpdateStateOrderAsync` checks for an existing order, then updates or creates it. Update it with the following code:

```
public async Task<Order> UpdateOrderStateAsync(Order order)
{
    try
    {
        var statekey = $"order_{order.OrderId}";

        // Try to get existing state
        var existingState = await _daprcClient.GetStateAsync<Order>(STORE_NAME,
statekey);
        if (existingState != null)
        {
            // Merge new data with existing state
            order = MergeOrderStates(existingState, order);
        }

        // Save updated state
        await _daprcClient.SaveStateAsync(STORE_NAME, statekey, order);
        _logger.LogInformation("Updated state for order {OrderId} - Status:
{Status}",
            order.OrderId, order.Status);

        return order;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error updating state for order {OrderId}",
order.OrderId);
        throw;
    }
}
```

3. `GetOrderAsync` gets an order from the state store by `orderId`. Update the function with the following:

```
public async Task<Order?> GetOrderAsync(string orderId)
{
    try
    {
        // Get order from state store by order ID
        var statekey = $"order_{orderId}";
        var order = await _daprcClient.GetStateAsync<Order>(STORE_NAME, statekey);

        if (order == null)
        {
            _logger.LogWarning("Order {OrderId} not found", orderId);
        }
    }
}
```

```

        return null;
    }

    return order;
}
catch (Exception ex)
{
    _logger.LogError(ex, "Error retrieving order {OrderId}", orderId);
    throw;
}
}

```

4. Finally, update `DeleteOrderAsync` deletes the order by `orderId`:

```

public async Task<string?> DeleteOrderAsync(string orderId)
{
    try
    {
        var stateKey = $"order_{orderId}";

        // Tries to delete the order from the state store
        await _daprcClient.DeleteStateAsync(STORE_NAME, stateKey);
        _logger.LogInformation("Deleted state for order {OrderId}", orderId);
        return orderId;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error deleting order {OrderId}", orderId);
        throw;
    }
}

```

The Dapr Client is responsible for the following, respectively:

1. `await _daprcClient.SaveStateAsync(STORE_NAME, stateKey, order);` saves the state to Redis using a key/value pair. It requires the state store name, the order id as a **key**, and a json representation of the order as a **value**.
2. `await _daprcClient.GetStateAsync<Order>(STORE_NAME, stateKey);` retrieves the state from the store. It requires a key and the state store name.
3. `await _daprcClient.DeleteStateAsync(STORE_NAME, stateKey);` deletes the state from the store. It also requires a key and the state store name.

Run the application

Open a new terminal and navigate to the `/Pizzaorder` folder. Use the Dapr CLI to run the following command:

```

dapr run --app-id pizza-order --app-protocol http --app-port 8001 --dapr-http-port 3501 --resources-path ../resources -- dotnet run

```

Important

If you are using Consul as a name resolution service, add `--config` `../resources/config/config.yaml` before `-- dotnet run` in your Dapr run command.

This command sets:

- the app-id as `pizza-storefront`
- the app-protocol to `http`
- an app-port of `8001` for Dapr communication into the app
- an http-port of `3501` for Dapr API communication from the app
- the resources-path, where the state store component definition file is located. This will guarantee that the Redis component is loaded when the app initializes.

Look for the log entry below to ensure that the state store component was loaded successfully:

```
...
INFO[0000] Component loaded: pizzastore (state.redis/v1) app_id=pizza-
storefront instance=diagrid.local scope=dapr.runtime.processor type=log ver=1.14.4
...
```

Test the service

Use VS Code REST Client

Open the `Endpoints.http` file located in the root of the repository and place a new order by clicking the button `Send request` under `Direct Pizza Order Endpoint (for testing)`:

```
### Direct Pizza Order Endpoint (for testing)
POST {{pizzaOrderUrl}}/order
Content-Type: application/json

{
  "orderId": "123",
  "pizzaType": "pepperoni",
  "size": "large",
  "customer": {
    "name": "John Doe",
    "address": "123 Main St",
    "phone": "555-0123"
  }
}
```

Run the `GET` and `DELETE` requests situated below to get and delete the order as well.

Use *cURL*

Run the command below to create a new order:

```
curl -H 'Content-Type: application/json' \
  -d '{ "orderId": "123", "pizzaType": "pepperoni", "size": "large", "customer": {
"name": "John Doe", "address": "123 Main St", "phone": "555-0123" } }' \
  -X POST \
  http://localhost:8001/order
```

Get:



```
curl -H 'Content-Type: application/json' \
  -X GET \
  http://localhost:8001/order/123
```

Finally, delete the order:

```
curl -H 'Content-Type: application/json' \
  -X DELETE \
  http://localhost:8001/order/123
```

Visualize the data

If you downloaded Database Client for VSCode or Redis Insight, you can visualize the new order there:

Field	Value	
data	{"orderId":"123","pizzaType":"pepperoni","size":"large","customer":{"name":"John Doe","address":"123 Main St","phone":"555-0123"},"status":"cooking_preparing_ingredients","error":null}	
version	4	

If you use Redis Insight from docker:

```
docker run -d --name redisinsight -p 5540:5540 redis/redisinsight:latest
```

When adding the Redis database, set the host to: host.docker.internal

If you need to reset the Redis Database, open a new terminal and run the following command:

```
docker exec -it dapr_redis redis-cli FLUSHALL
```

Next steps

Create a new service to create the order, cook, and deliver the pizza. In the next challenge, you will learn how to create a new API endpoint and how to invoke it using Dapr. When you are ready, go to Challenge 2: [Service Invocation](#)!