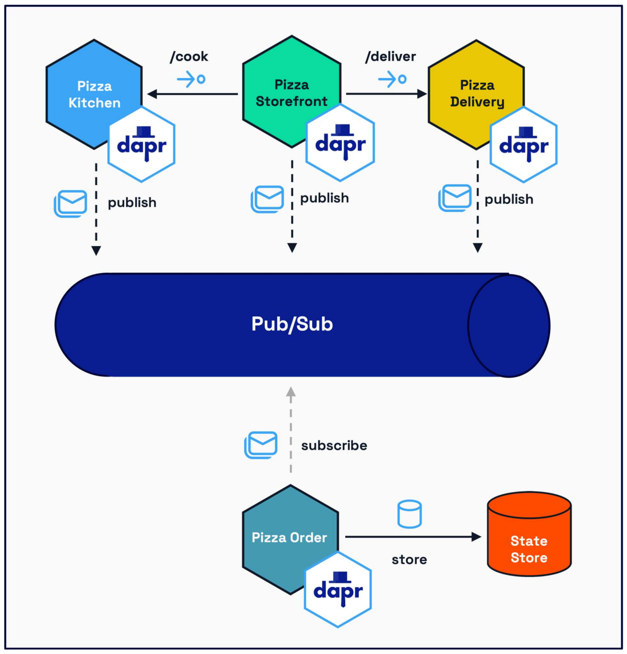
Challenge 3 - Pub/Sub

Overview

On the third challenge, your goal is to update the state store with all the events from pizza order that we are generating from the storefront, kitchen, and delivery services. For that, you will:

- Send all the generated events to a new Dapr component, a pub/sub message broker
- Update the storefront, kitchen, and delivery services to publish a message to the pub/sub.
- Subscribe to these events in the order service, which is already managing the order state in our state store.



To learn more about the Publish & Subscribe building block, refer to the [Dapr docs](https://docs.dapr.io).

Create the Pub/Sub component

Open the `/resources` folder and create a file called `pubsub.yaml`. Add the following content:

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

```
- pizza-storefront
- pizza-kitchen
- pizza-delivery
- pizza-order
```

Similar to the `statestore.yaml` file, this new definition creates a Dapr component called *pizzapubsub* of type *pubsub.redis* pointing to the local Redis instance, using Redis Streams. Each app will initialize this component to interact with it.

Create a subscription

Still inside the `/resources` folder, create a new file called `subscription.yaml`. Add the following content to it:

```yaml
apiVersion: dapr.io/v1alpha1
kind: Subscription
metadata:
  name: pizza-subscription
spec:
  topic: orders
  route: /orders-sub
  pubsubname: pizzapubsub
scopes:
- pizza-order
```

This file of kind `Subscription` specifies that every time the Pub/Sub `pizzapubsub` component receives a message in the `orders` topic, this message will be sent to a route called `/orders-sub` on the scoped `pizza-order` service. By setting `pizza-storefront` as the only scope, we guarantee that this subscription rule will only apply to this service and will be ignored by others. Finally, the `/orders-sub` endpoint needs to be created in the `pizza-order` service in order to receive the events.

Install the dependencies

Navigate to root of your solution. Before you start coding, install the Dapr dependencies to the `pizza-kitchen` and the `pizza-delivery` services. The `pizza-storefront` service already has the dependencies from challenge 2.

```
# Navigate to the service folder and add the Dapr package
cd PizzaKitchen
dotnet add package Dapr.Client
dotnet add package Dapr.AspNetCore

cd ..

# Navigate to the service folder and add the Dapr package
cd PizzaDelivery
dotnet add package Dapr.Client
dotnet add package Dapr.AspNetCore

cd ..
```

Register the DaprClient

Inside the `pizza-kitchen` and the `pizza-delivery` services, open `Program.cs` and add the `DaprClient` registration to the `ServiceCollection`:

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

Update the Kitchen service to publish messages to the message broker

1. Inside the `PizzaKitchen` folder, navigate to `/Services/CookService.cs`. Import the DaprClient:

```
using Dapr.Client;
```

2. Add a private readonly DaprClient reference:

```
private readonly DaprClient _daprClient;
```

3. Add two constants to hold the names of the pub/sub component and topic you will be publishing the messages to:

```
private const string PUBSUB_NAME = "pizzapubsub";
private const string TOPIC_NAME = "orders";
```

4. Update the class constructor to add the DaprClient dependency:

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

5. Finally, update the `CookPizzaAsync` try-catch block with the following code:

```
try
{
    foreach (var (status, duration) in stages)
    {
        order.Status = status;
        _logger.LogInformation("Order {OrderId} - {Status}", order.OrderId, status);

        await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);
        await Task.Delay(TimeSpan.FromSeconds(duration));
    }

    order.Status = "cooked";
    await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);
    return order;
}
catch (Exception ex)
{
    _logger.LogError(ex, "Error cooking order {OrderId}", order.OrderId);
    order.Status = "cooking_failed";
    order.Error = ex.Message;
    await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);
    return order;
}
```

The just like the previous challenges, we are using the Dapr Client to call the building block API: `await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);`.

In this case, `publishEventAsync` will publish the message `order` to the `PUBSUB_NAME` and `TOPIC_NAME` you've declared above.

Let's do the same for the Delivery and Storefront services.

Update the Delivery service to publish messages to the message broker

1. Inside the `PizzaDelivery` folder, navigate to `/Services/DeliveryService.cs`. Import the DaprClient:

```
using Dapr.Client;
```

2. Add a private readonly DaprClient reference:

```
private readonly DaprClient _daprClient;
```

3. Add two constants to hold the names of the pub/sub component and topic you will be publishing the messages to:

```
private const string PUBSUB_NAME = "pizzapubsub";
private const string TOPIC_NAME = "orders";
```

4. Update the class constructor to add the DaprClient dependency:

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

5. Finally, update the `DeliverPizzaAsync` try-catch block with the following code:

```
try
{
    foreach (var (status, duration) in stages)
    {
        order.Status = status;
        _logger.LogInformation("Order {OrderId} - {Status}", order.OrderId, status);

        await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);
        await Task.Delay(TimeSpan.FromSeconds(duration));
    }

    order.Status = "delivered";
    await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);
    return order;
}
catch (Exception ex)
{
    _logger.LogError(ex, "Error delivering order {OrderId}", order.OrderId);
    order.Status = "delivery_failed";
    order.Error = ex.Message;
    await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);
    return order;
}
```

Update the Storefront service to publish messages to the message broker

1. Inside the `PizzaStorefront` service folder, navigate to `/Services/Storefront.cs` and add the pub/sub constants:

```
private const string PUBSUB_NAME = "pizzapubsub";
private const string TOPIC_NAME = "orders";
```

2. Inside `ProcessOrderAsync` update the try-catch block with:

```
try
{
    // Set pizza order status
    foreach (var (status, duration) in stages)
    {
        order.Status = status;
        _logger.LogInformation("Order {OrderId} - {Status}", order.OrderId, status);

        await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);
        await Task.Delay(TimeSpan.FromSeconds(duration));
    }

    _logger.LogInformation("Starting cooking process for order {OrderId}",
order.OrderId);

    // Use the Service Invocation building block to invoke the endpoint in the
pizza-kitchen service
    var response = await _daprClient.InvokeMethodAsync<Order, Order>(
        HttpMethod.Post,
        "pizza-kitchen",
        "cook",
        order);

    _logger.LogInformation("Order {OrderId} cooked with status {Status}",
        order.OrderId, response.Status);

    // Use the Service Invocation building block to invoke the endpoint in the
pizza-delivery service
    _logger.LogInformation("Starting delivery process for order {OrderId}",
order.OrderId);

    response = await _daprClient.InvokeMethodAsync<Order, Order>(
        HttpMethod.Post,
        "pizza-delivery",
        "delivery",
        order);

    _logger.LogInformation("Order {OrderId} delivered with status {Status}",
        order.OrderId, response.Status);

    return order;
}
catch (Exception ex)
{
    _logger.LogError(ex, "Error processing order {OrderId}", order.OrderId);
    order.Status = "failed";
    order.Error = ex.Message;

    await _daprClient.PublishEventAsync(PUBSUB_NAME, TOPIC_NAME, order);
    return order;
}
```

Subscribe to events

Now that you've published the events to the topic `orders` in the message broker, you will subscribe to the same topic in the `pizza-order` service:

Navigate to the `PizzaOrder` service folder. Inside `/Controllers/OrderController.cs` find the the route `/order-sub`:

```
[HttpPost("/orders-sub")]
public async Task<IActionResult> HandleOrderUpdate(CloudEvent<Order> cloudEvent)
{
    _logger.LogInformation("Received order update for order {OrderId}",
        cloudEvent.Data.OrderId);

    var result = await _orderStateService.UpdateOrderStateAsync(cloudEvent.Data);
    return Ok();
}
```

Following the `subscription.yaml` file spec, every time a new message lands in the `orders` topic within the `pizzapubsub` pub/sub, it will be routed to this `/orders-sub` topic. The message will then be sent to the previously created function that creates or updates the message in the state store, created in the first challenge.

```
spec:
  topic: orders
  route: /orders-sub
  pubsubname: pizzapubsub
```

Run the application

It's time to run all four applications. If the `pizza-storefront`, `pizza-kitchen`, `pizza-delivery`, and the `pizza-store`
services are still running, press **CTRL+C** in each terminal window to stop them.

1. Open a new terminal window, navigate to the `/PizzaOrder` folder and run the command below:

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

2. In a new terminal, navigate to the `/PizzaStorefront` folder and run the command below:

```
dapr run --app-id pizza-storefront --app-protocol http --app-port 8002 --dapr-http-
port 3502 --resources-path ../resources -- dotnet run
```

3. Open a new terminal window and navigate to `/PizzaKitchen` folder. Run the command below:

```
dapr run --app-id pizza-kitchen --app-protocol http --app-port 8003 --dapr-http-port
3503 --resources-path ../resources -- dotnet run
```

4. Open a fourth terminal window and navigate to `/PizzaDelivery` folder. Run the command below:

```
dapr run --app-id pizza-delivery --app-protocol http --app-port 8004 --dapr-http-port 3504 --resources-path ../resources -- dotnet run
```

> **Important**
> If you are using Consul as a naming resolution service, add `--config ../resources/config/config.yaml` before `-- dotnet run` on your Dapr run command.

Check the Dapr and application logs for all four services. You should now see the pubsub component loaded in the Dapr logs:

```
INFO[0000] Component loaded: pizzapubsub (pubsub.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 `Endpoints.http` and create a new order sending the request on `Direct Pizza Store Endpoint (for testing)`, similar to what was done previous challenge.

Navigate to the `pizza-order` terminal, where you should see the following logs pop up with all the events being updated:

```
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: validating
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: processing
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: confirmed
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: cooking_preparing_ingredients
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: cooking_making_dough
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: cooking_adding_toppings
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: cooking_baking
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: cooking_quality_check
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: cooked
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: delivery_finding_driver
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: delivery_driver_assigned
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: delivery_picked_up
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: delivery_on_the_way
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: delivery_arriving
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: delivery_at_location
== APP == info: PizzaOrder.Controllers.OrderController[0]
== APP ==       Received order update for order 123
== APP == info: PizzaOrder.Services.OrderStateService[0]
== APP ==       Updated state for order 123 - Status: delivered
```

Use *cURL*

Open a fourth terminal window and create a new order using cURL:

```
curl -H 'Content-Type: application/json' \
   -d '{ "orderId": "1", "pizzaType": "pepperoni", "size": "large", "customer": {
"name": "John Doe", "address": "123 Main St", "phone": "555-0123" } }' \
    -X POST \
     http://localhost:8002/storefront/order
```

Dapr multi-app run

Instead of opening multiple terminals to run the services, you can take advantage of a great Dapr CLI feature: multi-app run. This enables you run all three services with just one command!

In the parent folder, create a new file called `dapr.yaml`. Add the following content to it:

```yaml
version: 1
common:
  resourcesPath: ./resources
  # Uncomment the following line if you are running Consul for service naming resolution
  # configFilePath: ./resources/config/config.yaml
apps:
  - appDirPath: ./PizzaOrder/
    appID: pizza-order
    daprHTTPPort: 3501
    appPort: 8001
    command: ["dotnet", "run"]
  - appDirPath: ./PizzaStore/
    appID: pizza-storefront
    daprHTTPPort: 3502
    appPort: 8002
    command: ["dotnet", "run"]
  - appDirPath: ./PizzaKitchen/
    appID: pizza-kitchen
    appPort: 8003
    daprHTTPPort: 3503
    command: ["dotnet", "run"]
  - appDirPath: ./PizzaDelivery/
    appID: pizza-delivery
    appPort: 8004
    daprHTTPPort: 3504
    command: ["dotnet", "run"]
```

Stop the services, if they are running, and enter the following command in the terminal:

```
dapr run -f .
```

All four services will run at the same time and log events at the same terminal window.

Next steps

In the next challenge we will orchestrate the pizza ordering, cooking, and delivering process leberaging Dapr's Workflow API. Once you are ready, navigate to Challenge 4: Workflows!