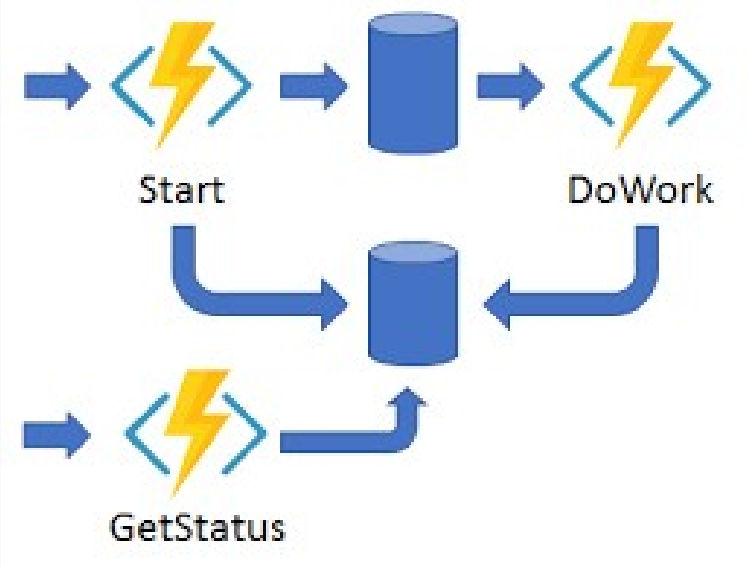
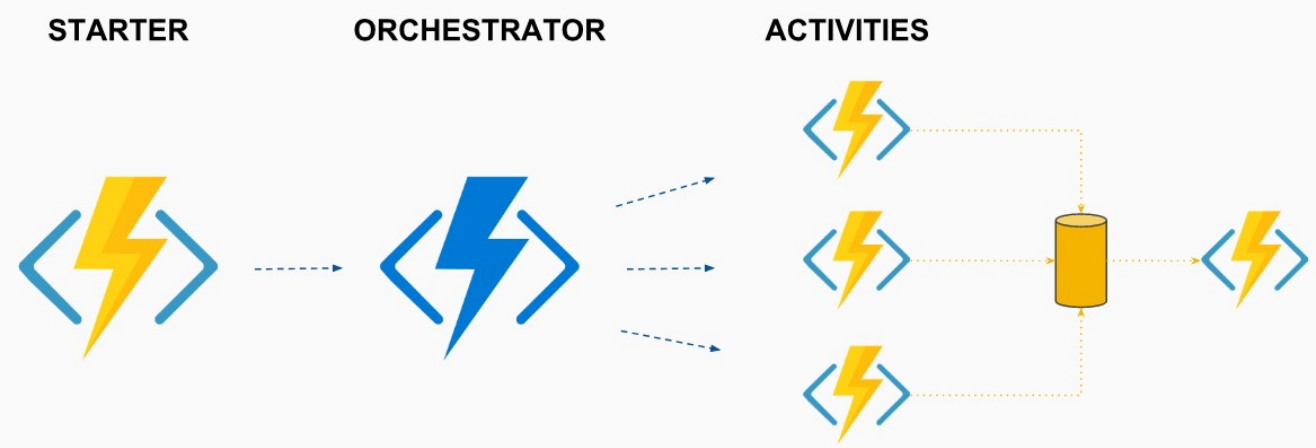


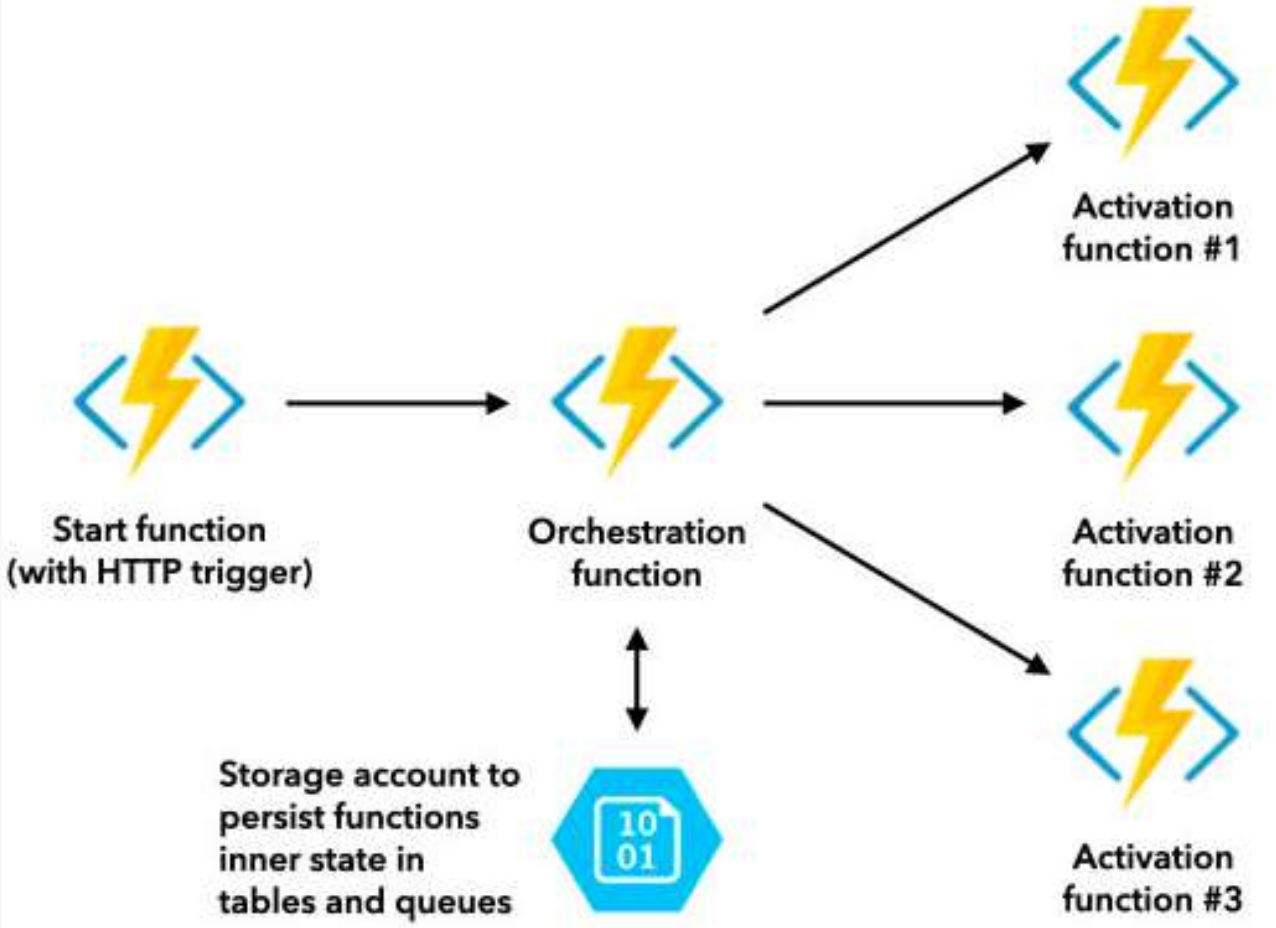

# Azure Durable Function

# Azure Durable Function

- An extension of Azure Functions
- Lets you write stateful functions in a serverless compute environment
- The extension lets you define stateful workflows



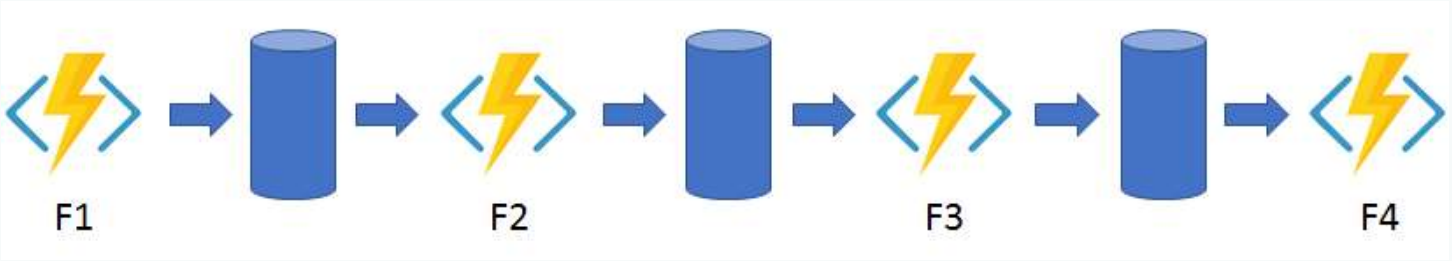
# Azure Function Runtime




## Execution logs:

| |
|---|
| Orchestration function triggered |
| Execution started |
| Activity function 1 task scheduled |
| Orchestration function completed |
| Activity function 1 task completed |
| Orchestration function started |
| Activity function 2 task scheduled |
| Orchestration function completed |
| Activity function 2 task completed |
| Orchestration function started |
| Activity function 3 task scheduled |
| Orchestration function completed |
| Activity function 3 task completed |
| Orchestration function started |
| Execution completed, Results |

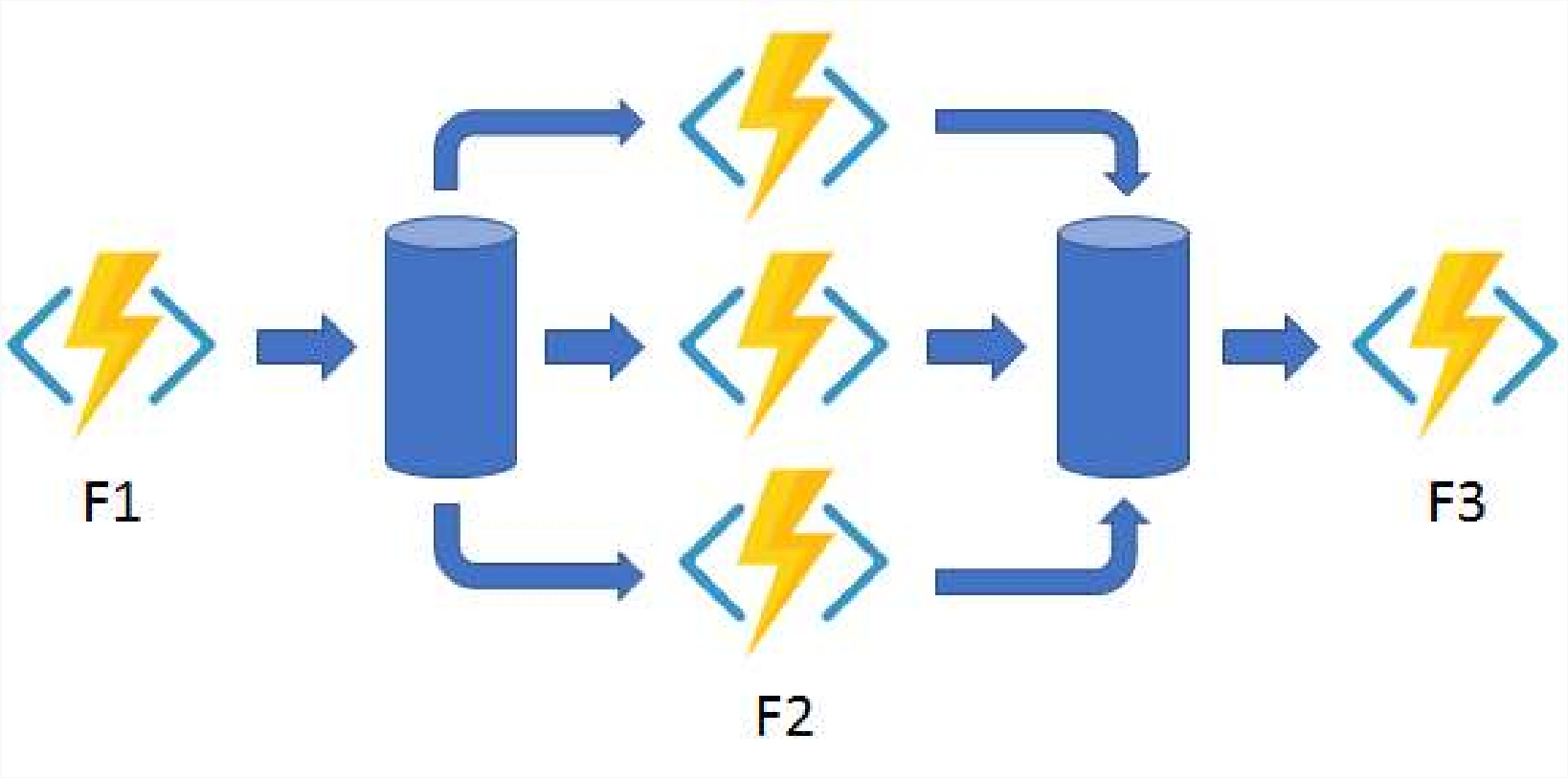
# Pattern #1: Function chaining



# Pattern #1: Function chaining

```
[FunctionName("Chaining")]
public static async Task<object> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    try
    {
        var x = await context.CallActivityAsync<object>("F1", null);
        var y = await context.CallActivityAsync<object>("F2", x);
        var z = await context.CallActivityAsync<object>("F3", y);
        return  await context.CallActivityAsync<object>("F4", z);
    }
    catch (Exception)
    {
        // Error handling or compensation goes here.
    }
}
```

# Pattern #2: Fan out/fan in



# Pattern #2: Fan out/fan in

```
[FunctionName("FanOutFanIn")]
public static async Task Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var parallelTasks = new List<Task<int>>();

    // Get a list of N work items to process in parallel.
    object[] workBatch = await context.CallActivityAsync<object[]>("F1", null);
    for (int i = 0; i < workBatch.Length; i++)
    {
        Task<int> task = context.CallActivityAsync<int>("F2", workBatch[i]);
        parallelTasks.Add(task);
    }

    await Task.WhenAll(parallelTasks);

    // Aggregate all N outputs and send the result to F3.
    int sum = parallelTasks.Sum(t => t.Result);
    await context.CallActivityAsync("F3", sum);
}
```

# Thank You