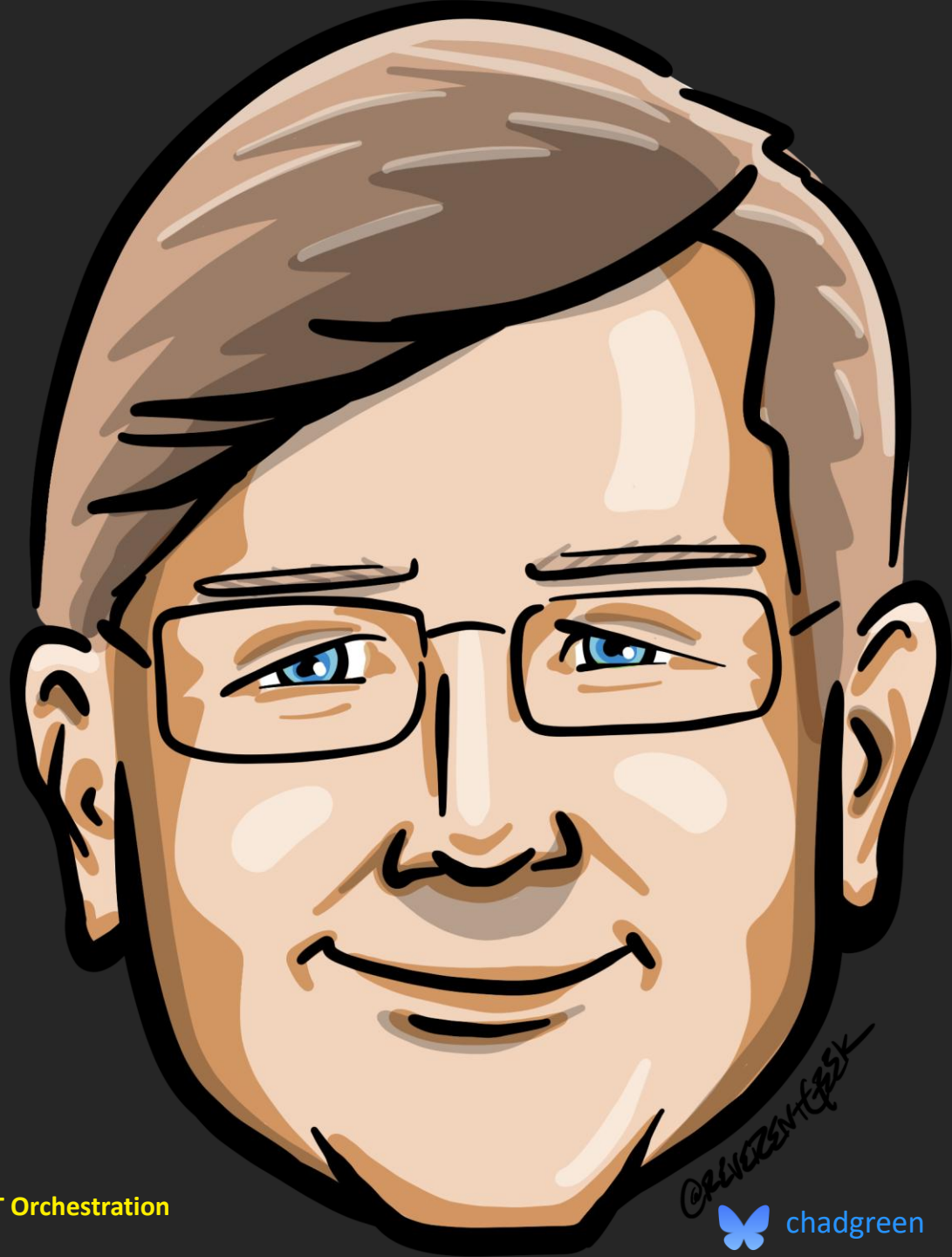
AZURE DURABLE FUNCTIONS
FOR SERVERLESS

.NET ORCHESTRATION

Who is Chad Green?



- chadgreen@chadgreen.com
- TaleLearnCode
- ChadGreen.com
- ChadGreen
- ChadwickEGreen

Microsoft® MVP Most Valuable Professional

Who is using Azure Functions?

Who is using Durable Functions?

Serverless & Azure Function Overview

Azure Durable Functions for Serverless .NET Orchestration

The evolution of application platforms

On-Premises

What media should I use to keep **backups?**

What **size** of **servers** should I **buy**?

How do I **deploy** new **code** to my **servers**?

What happens in case of server hardware **failure?**

How can I increase **server** utilization?

How can I **scale** my app?

Which packages should be on my **server**?

Who **monitors** my **apps**?

How often should I **backup** my **server?**

Are my **servers** in a **secure** location?

Do I need a secondary **Network connection?**

How do I keep the **operating system** up to date?

Who has **physical** access my **servers?**

Who **monitors** my **servers**?

What **storage** do I need to use?

How **many** servers do I need?

What happens in case of server hardware **failure?**

Do I need a **UPS**?

How can I dynamically configure my app?

It takes how long to **provision** a new **server?**

How often should I **patch** my **severs?**

The evolution of application platforms

IaaS

What media should I use to keep **backups?**

What **size** of **servers** should I **buy**?

What is the right **size** of **servers** for my business needs?

How do I **deploy** new **code** to my **server**?

How can I increase **server** utilization?

How many servers do I need?

How can I **scale** my application?

Who **monitors** my **apps**?

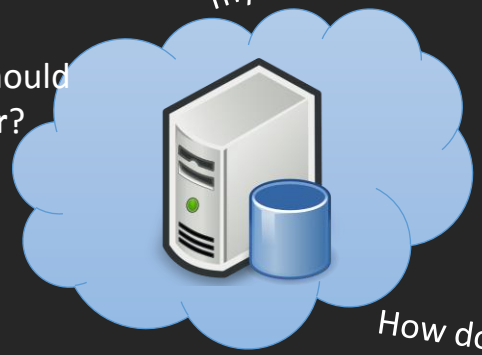
What happens in case of server hardware **failure?**

How can I increase **server** utilization?

How often should I **backup** my **server?**

How can I **scale** my app?

Which packages should be on my **server**?



Are my **servers** in a **secure** location?

Do I need a secondary **Network connection?**

How often should I **patch** my **servers**?

How often should I **backup** my **server**?

Which **packages** should be on my **server**?

How many servers do I need?

Who has **physical** access my **servers?**

Who **monitors** my **servers**?

Do I need a **UPS**?

How do I keep the **operating system** up to date?

How do I **deploy** new **code** to my **server**?

How do I keep the **operating system** up to date?

What **storage** do I need to use?

What happens in case of server hardware **failure?**

Who **monitors** my application?

How can I dynamically configure my app?

It takes how long to **provision** a new **server?**

How often should I **patch** my **severs?**

The evolution of application platforms

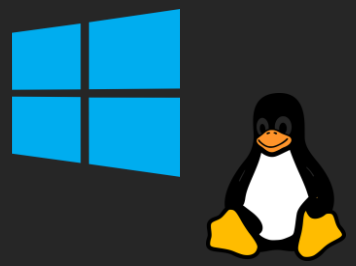
PaaS

What is the right **size** of **servers** for my business needs?

How can I increase **server** utilization?

How **many** servers do I need?

How can I **scale** my application?



How often should I **patch** my **servers**?

How often should I **backup** my **server**?

Which **packages** should be on my **server**?

How do I **deploy** new **code** to my **server**?

How do I keep the **operating system** up to date?

Who **monitors** my application?

The evolution of application platforms

Serverless

What is the right **size** of **servers** for my business needs?
How can I increase **server** utilization?
How **many** servers do I need?
How can I **scale** my application?

Not there isn't servers

Just, you can think about the servers less

~~Server Configuration~~

~~Server Scaling~~

Types of Serverless Architecture

Function as a Service (FaaS)

Backend as a Service (BaaS)

Function-as-a-Service

Event-Driven

Function-as-a-Service

Event-Driven

Short-Lived

Function-as-a-Service

Event-Driven

Short-Lived

Automatic Scaling

Function-as-a-Service

Event-Driven

Short-Lived

Automatic Scaling

Pay-Per-Execution

Function-as-a-Service

Event-Driven

Short-Lived

Automatic Scaling

Pay-Per-Execution

Abstraction of Infrastructure

Azure Functions

Code

Events + data

Choice of Language

Bring your own dependencies

Simplified Integration

RabbitMQ

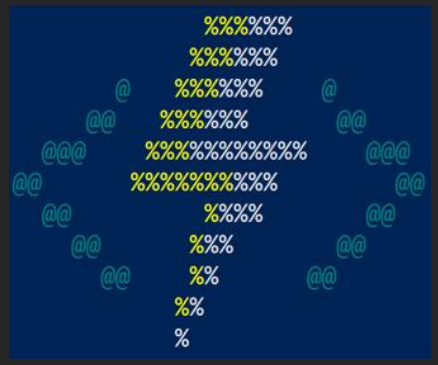
twilio

Supported Bindings

Third Party Bindings

Custom Bindings

Flexible Development



Many Hosting Options

Consumption

Serverless

Premium

Sort of Serverless

Dedicated

App Service Plan

Container Apps

Consumption, Dedicated

App Service Environment

Network Isolation

Azure Stack

On Premises

Functions Runtime

Your Server

Azure IoT Edge

On Devices

Many Hosting Options

Consumption

Serverless

Premium

Sort of Serverless

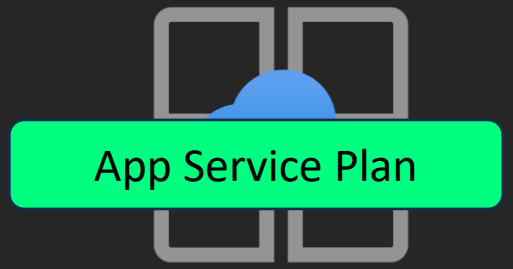
Dedicated

App Service Plan

Container Apps

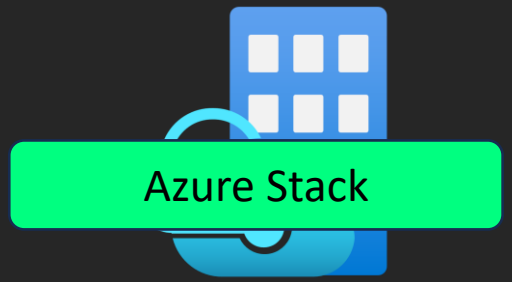
Consumption, Dedicated

App Service Environment



Network Isolation

Azure Stack

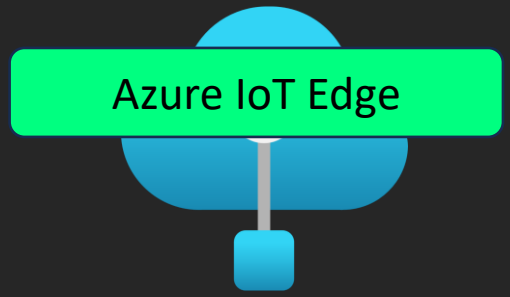


On Premises

Functions Runtime

Your Server

Azure IoT Edge

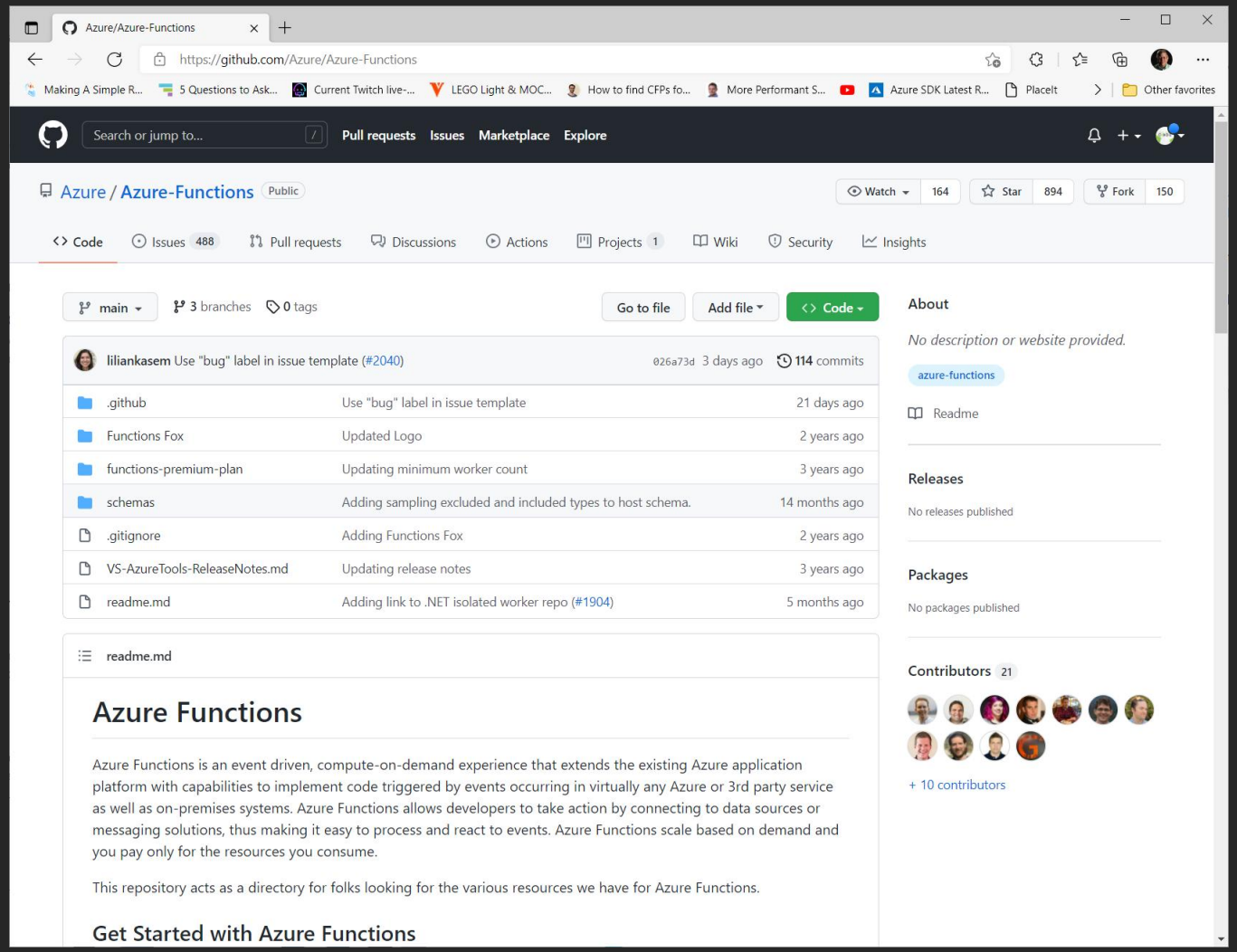


On Devices

Many Hosting Options

- Consumption
- Premium
- Dedicated
- Container Apps
- App Service Plan
- Azure Stack
- Functions Runtime
- Azure IoT Edge

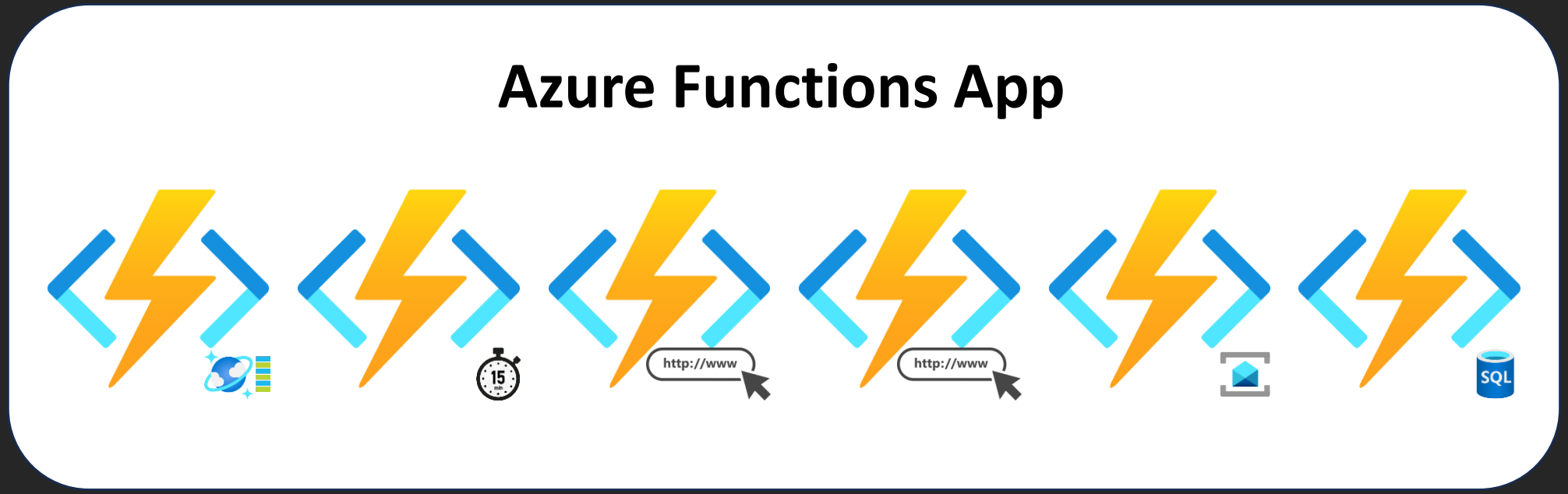
Open Source


Azure/Azure-Functions
https://github.com/Azure/Azure-Functions
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Azure / Azure-Functions
Public
Watch 164
Star 894
Fork 150
Code
Issues 488
Pull requests
Discussions
Actions
Projects 1
Wiki
Security
Insights
main
3 branches
0 tags
Go to file
Add file
Code
liliankasem Use "bug" label in issue template (#2040)
026a73d 3 days ago
114 commits
.github Use "bug" label in issue template 21 days ago
Functions Fox Updated Logo 2 years ago
functions-premium-plan Updating minimum worker count 3 years ago
schemas Adding sampling excluded and included types to host schema. 14 months ago
.gitignore Adding Functions Fox 2 years ago
VS-AzureTools-ReleaseNotes.md Updating release notes 3 years ago
readme.md Adding link to .NET isolated worker repo (#1904) 5 months ago
readme.md
Azure Functions
Azure Functions is an event driven, compute-on-demand experience that extends the existing Azure application platform with capabilities to implement code triggered by events occurring in virtually any Azure or 3rd party service as well as on-premises systems. Azure Functions allows developers to take action by connecting to data sources or messaging solutions, thus making it easy to process and react to events. Azure Functions scale based on demand and you pay only for the resources you consume.
This repository acts as a directory for folks looking for the various resources we have for Azure Functions.
Get Started with Azure Functions
About
No description or website provided.
azure-functions
Readme
Releases
No releases published
Packages
No packages published
Contributors 21
+ 10 contributors

Simple Azure Function Code

```
public class HelloWorld(ILogger<HelloWorld> logger)
{
  private readonly ILogger<HelloWorld> _logger = logger;

  [Function("HelloWorld")]
  public IActionResult Run([HttpTrigger(AuthorizationLevel.Function, "get“, Route = “hello-world”)] HttpRequest request)
  {
    _logger.LogInformation("C# HTTP trigger function processed a request.");
    return new OkObjectResult("Welcome to Azure Functions!");
  }
}
```

Anatomy of an Azure Function App



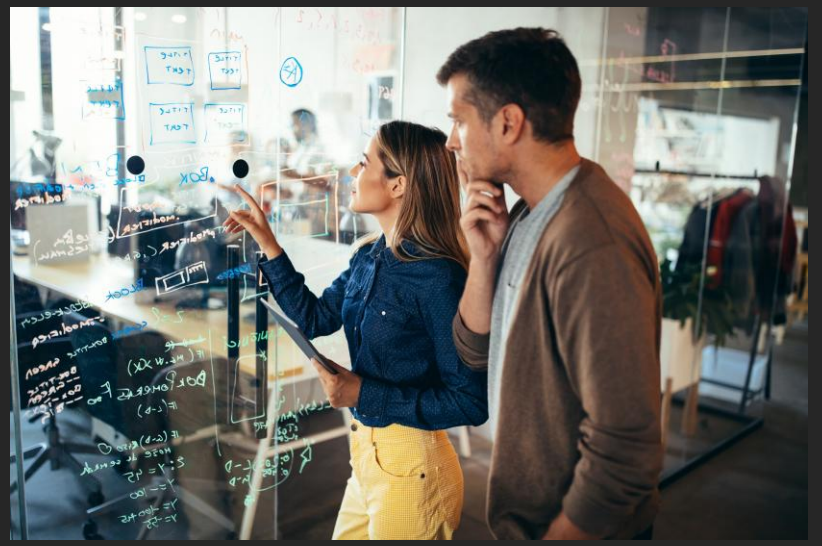
Azure Functions

Stateless, short-running, event-triggered serverless computing

Durable Functions

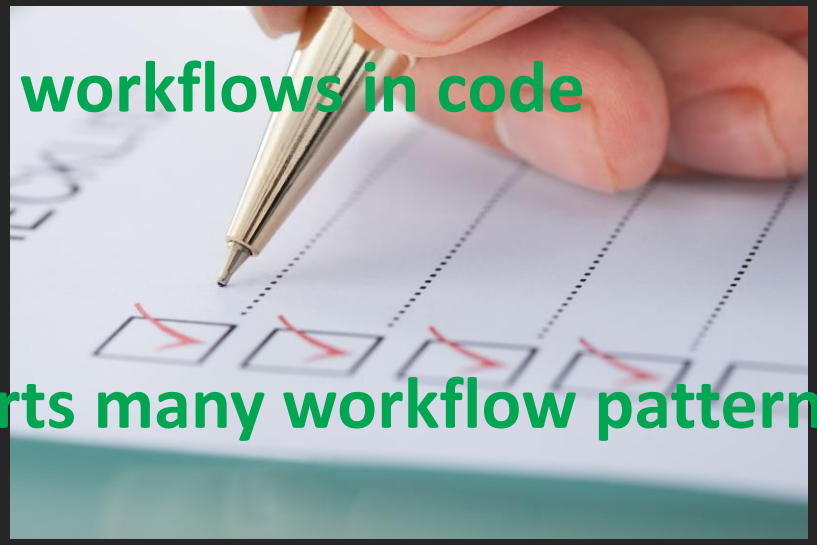
An extension to Azure Functions

Write “stateful” functions in a “serverless” environment



Define workflows in code

Durable Functions



Define workflows in code

Supports many workflow patterns

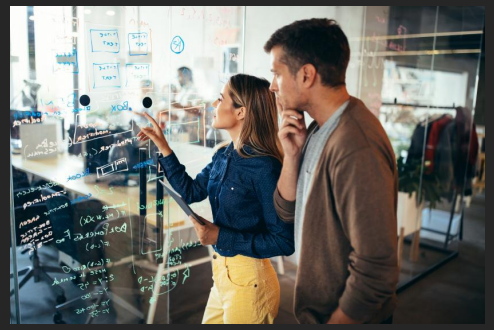
to
Azure Functions

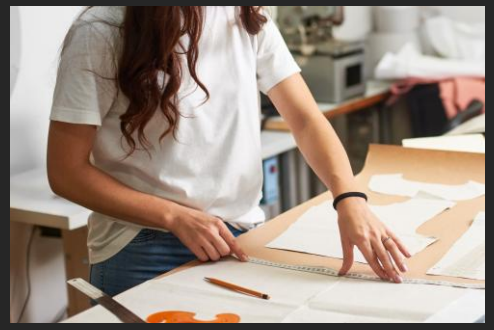
Write “stateful” functions
in a “serverless”
environment

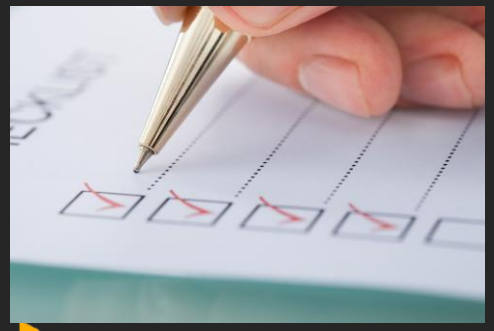
Define workflows
in code

Solves the state problem

Basics







Define workflows in code

- Parallel execution
- Error handling
- Easily understood “Orchestrator Function”

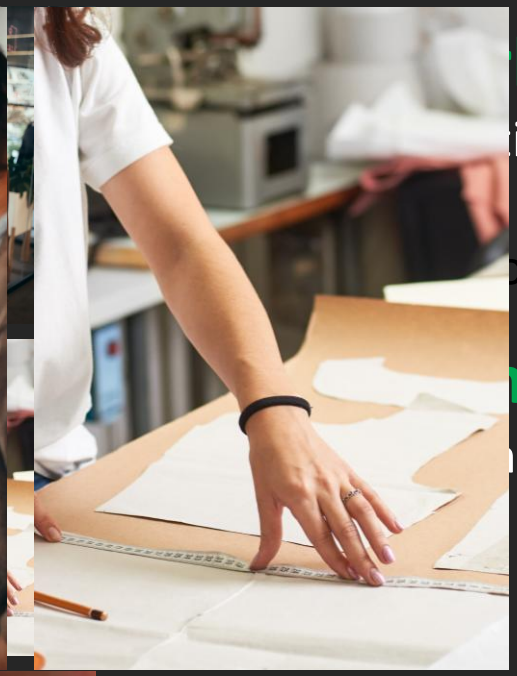
Supports many workflow patterns

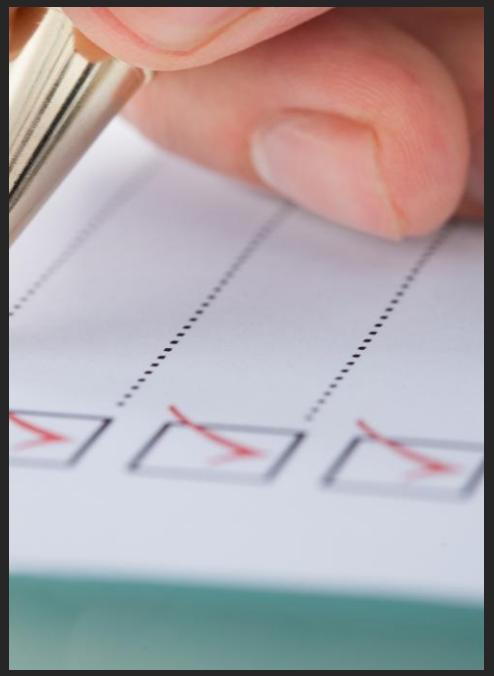
- Function chaining, fan-out/fan-in, human interaction, etc.

Solves the state problem

- Tracks workflow progress

Basics

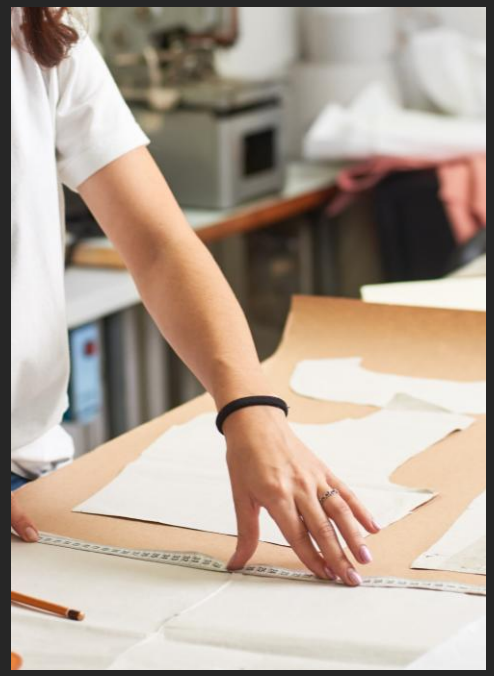




Solves the state problem

- Tracks workflow progress

Benefits

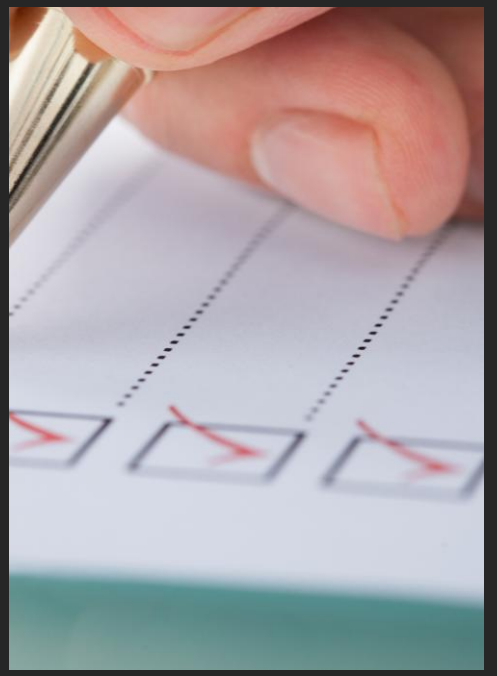
Define workflows in code



Easy to implement complex workflows

Consolidate exception handling

Check on progress or cancel workflows



Manage state for you

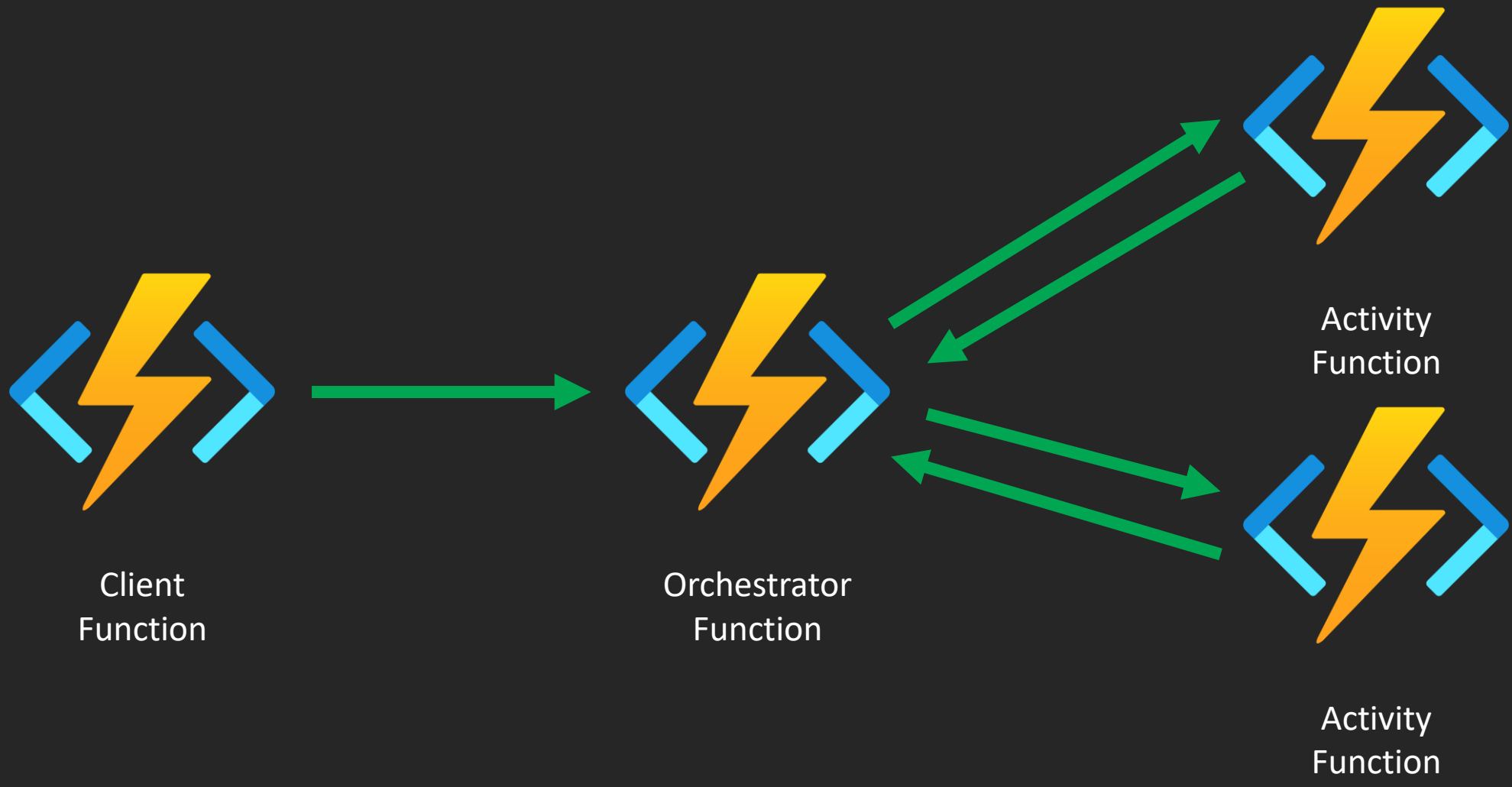
Key Concepts

Orchestrator Functions

Activity Functions

Starting Orchestrations

Durable Function Workflow



Constraints

Deterministic APIs

- Dates and Times
- GUIDs and UUIDs
- Random Numbers
- Bindings
- Static Variables
- Environment Variables
- Network and HTTP
- Thread-Blocking APIs
- Async APIs
- Threading APIs

Application Patterns

Azure Durable Functions for Serverless .NET Orchestration

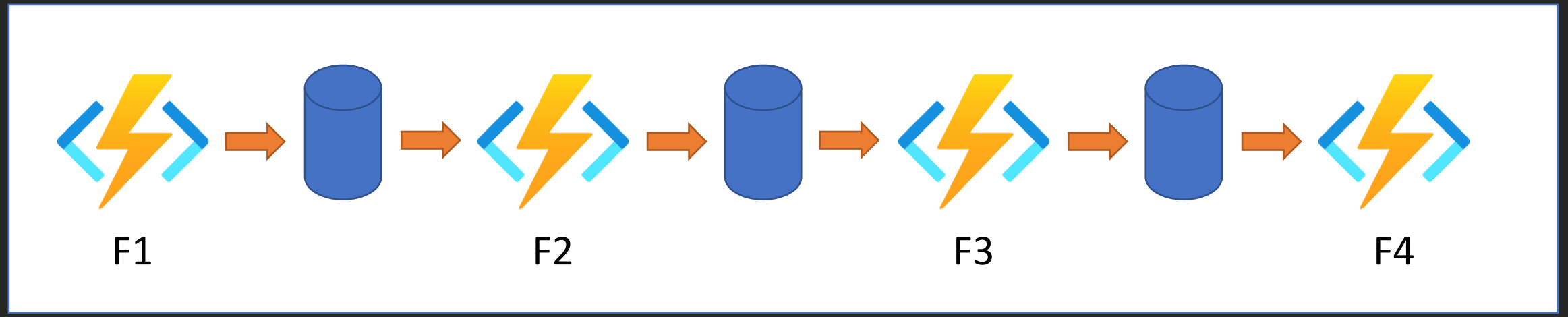
Application Patterns

- Function Chaining
- Fan-Out/Fan-In
- Async HTTP APIs
- Monitoring
- Human Interaction
- Aggregator

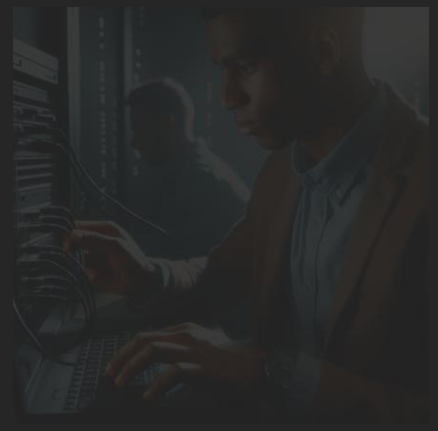
Application Patterns

- Function Chaining
- Fan-Out/Fan-In
- Async HTTP APIs
- Monitoring
- Human Interaction
- Aggregator

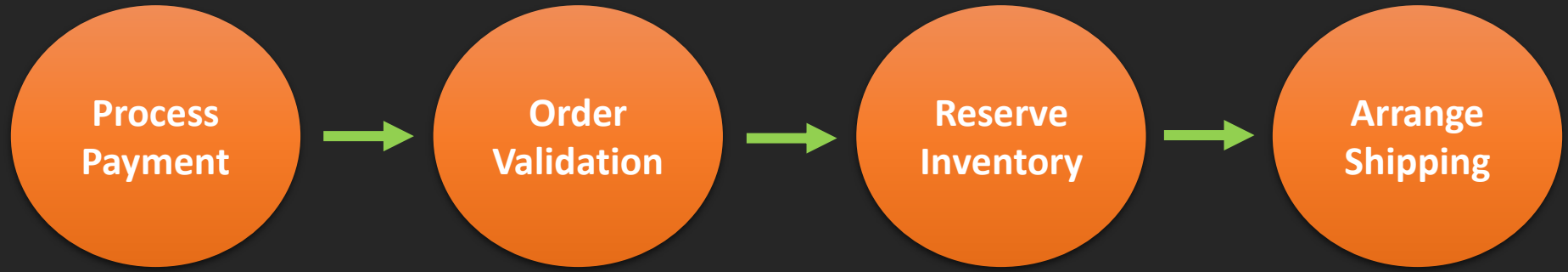
Function Chaining



Possible Scenarios

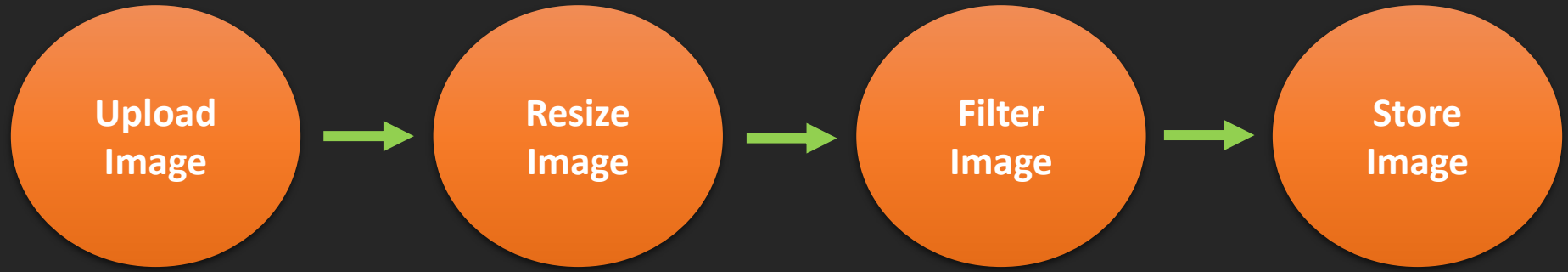



Order Processing Workflow



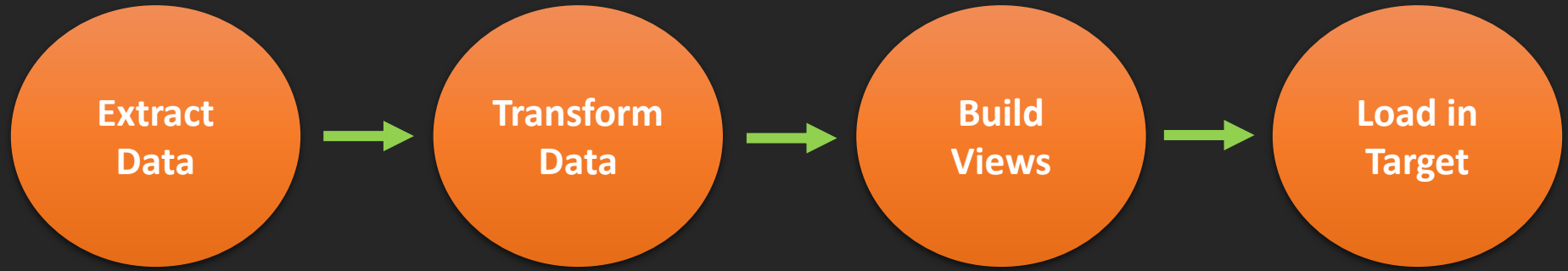
Possible Scenarios

Image Processing Pipeline



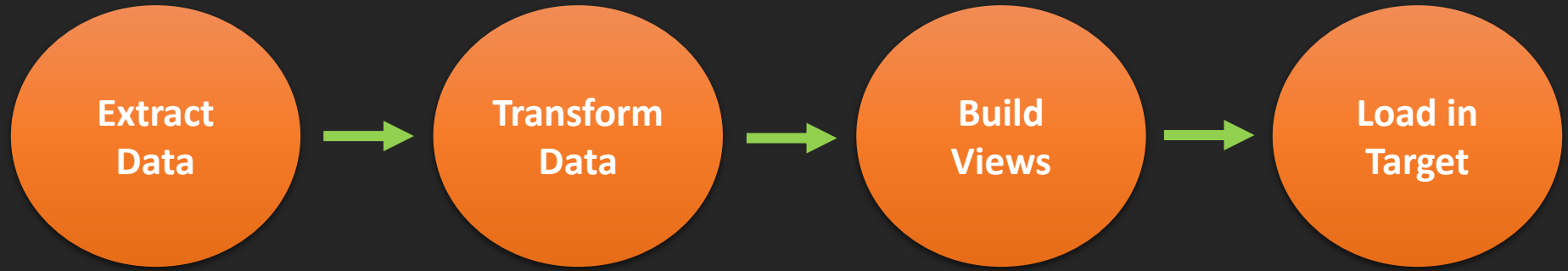
Possible Scenarios

Data Migration Workflow



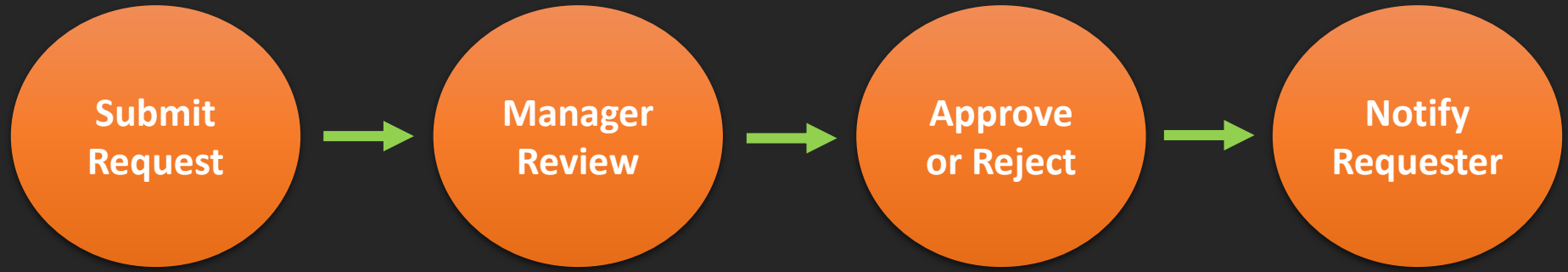
Possible Scenarios

Data Migration Workflow

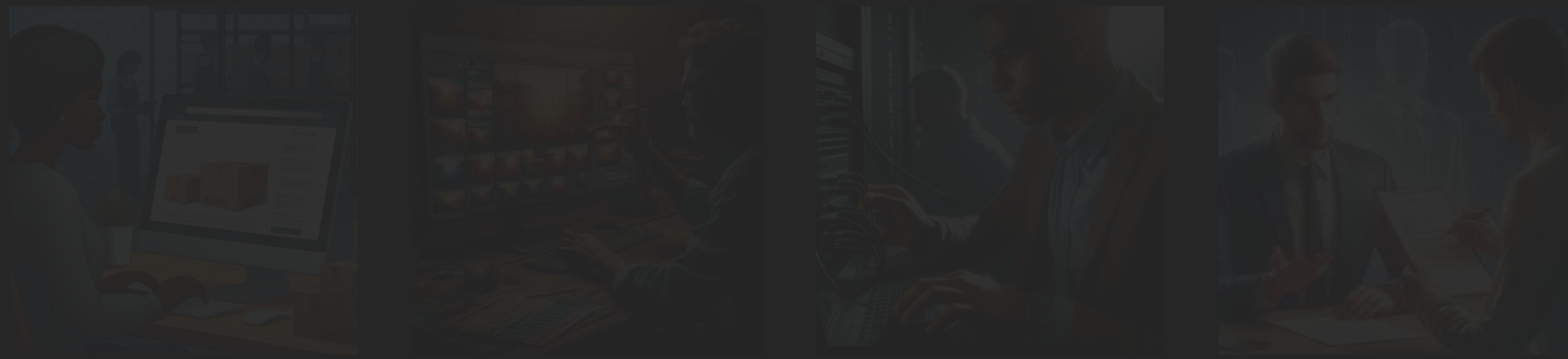


Possible Scenarios

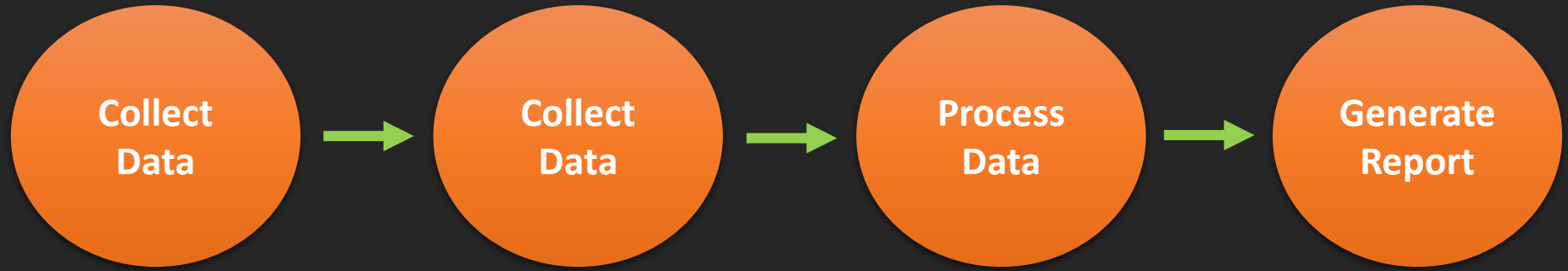
Approval Process



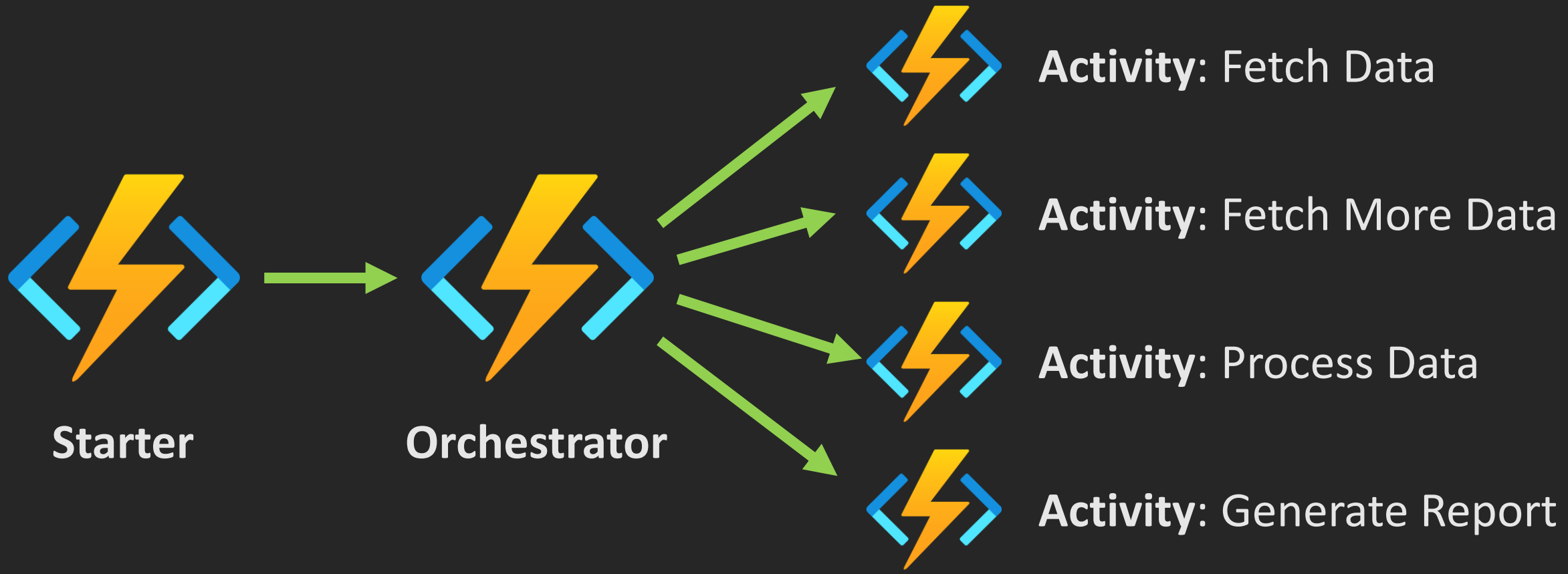
Possible Scenarios



Report Generation



Report Generation



Fetch Data

```
[Function(nameof(FetchDataFromLocation1))]
4 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<List<string>> FetchDataFromLocation1([ActivityTrigger] FunctionContext functionContext)
{
    ILogger logger = functionContext.GetLogger(nameof(FetchDataFromLocation1));
    logger.LogInformation("Fetching data from Location 1...");
    await Task.Delay(1000);
    return ["Data1_Location1", "Data2_Location1"];
}
```

Fetch More Data

```
[Function(nameof(FetchDataFromLocation2))]
4 references | 0 changes | 0 authors, 0 changes
public static async Task<List<string>> FetchDataFromLocation2([ActivityTrigger] FunctionContext functionContext)
{
    ILogger logger = functionContext.GetLogger(nameof(FetchDataFromLocation2));
    logger.LogInformation("Fetching data from Location 2...");
    await Task.Delay(1000);
    return ["Data1_Location2", "Data2_Location2"];
}
```

Process Data

```
[Function(nameof(ProcessData))]
4 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<string> ProcessData(
    [ActivityTrigger] List<List<string>> reportData, FunctionContext functionContext)
{
    ILogger logger = functionContext.GetLogger(nameof(ProcessData));
    logger.LogInformation("Processing the fetched data...");
    await Task.Delay(1000);
    string processedData = string.Join(",", reportData.SelectMany(x => x));
    return processedData;
}
```

Generate Report

```
[Function(nameof(GenerateReport))]
4 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<string> GenerateReport(
    [ActivityTrigger] string processedData, FunctionContext functionContext)
{

    ILogger logger = functionContext.GetLogger(nameof(GenerateReport));
    logger.LogInformation("Generating report...");

    // Simulate report generation
    await Task.Delay(1000);
    string reportContent = $"Report Content: {processedData}";

    // Save the report to Blob Storage
    BlobContainerClient containerClient = GetBlobContainerClient();
    string blobName = $"report_{Guid.NewGuid()}.txt";
    await containerClient.CreateIfNotExistsAsync(PublicAccessType.Blob);
    BlobClient blobClient = containerClient.GetBlobClient(blobName);
    using MemoryStream stream = new(Encoding.UTF8.GetBytes(reportContent));
    await blobClient.UploadAsync(stream, true);

    return blobClient.Uri.ToString();

}
```

Orchestrator

```
[Function(nameof(FunctionChainingOrchestrator))]
3 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<string> FunctionChainingOrchestrator(
    [OrchestrationTrigger] TaskOrchestrationContext context)
{
    ILogger logger = context.CreateReplaySafeLogger(nameof(Orchestrator));
    logger.LogInformation("Starting report generation.");

    List<string> data1 = await context.CallActivityAsync<List<string>>(nameof(FetchDataFromLocation1Activity.FetchDataFromLocation1));
    List<string> data2 = await context.CallActivityAsync<List<string>>(nameof(FetchDataFromLocation2Activity.FetchDataFromLocation2));
    string processedData = await context.CallActivityAsync<string>(nameof(ProcessDataActivity.ProcessData), new List<List<string>> { data1, data2 });
    string reportUrl = await context.CallActivityAsync<string>(nameof(GenerateReportActivity.GenerateReport), processedData);

    return reportUrl;

}
```

Starter

```
[Function(nameof(FunctionChainingHttpStarter))]
2 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<HttpResponseData> FunctionChainingHttpStarter(
    [HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "function-chaining")] HttpRequestData request,
    [DurableClient] DurableTaskClient client,
    FunctionContext executionContext)
{
    ILogger logger = executionContext.GetLogger("HttpStarter");
    string instanceId = await client.ScheduleNewOrchestrationInstanceAsync(nameof(Orchestrator.FunctionChainingOrchestrator));
    logger.LogInformation("Started orchestration with ID = {instanceId}.", instanceId);
    return await client.CreateCheckStatusResponseAsync(request, instanceId);
}
```

```
Azure Functions Core Tools
Core Tools Version:       4.0.6543 Commit hash: N/A +a4bf0e412b43c659740d8db346fc44d6cdd62e45 (64-bit)
Function Runtime Version: 4.1036.1.23224

[2024-11-09T16:57:23.123Z] Found C:\Presentations\ServerlessOrchestration\Demos\src\FunctionApp\FunctionApp.csproj. Usin
g for user secrets file configuration.
[2024-11-09T16:57:27.574Z] Azure Functions .NET Worker (PID: 18464) initialized in debug mode. Waiting for debugger to a
ttach...
[2024-11-09T16:57:27.679Z] Worker process started and initialized.

Functions:

        FunctionChainingHttpStarter: [POST] http://localhost:7071/orchestrations/function-chaining

        HelloWorld: [GET] http://localhost:7071/orchestrations/hello-world

        FetchDataFromLocation1: activityTrigger

        FetchDataFromLocation2: activityTrigger

        FunctionChainingOrchestrator: orchestrationTrigger

        GenerateReport: activityTrigger

        ProcessData: activityTrigger

For detailed output, run func with --verbose flag.
[2024-11-09T16:57:32.694Z] Host lock lease acquired by instance ID '000000000000000000000003F7F057D'.
```

POST http://localhost:7071/o

No environment

http://localhost:7071/orchestrations/function-chaining

Save | Share

POST | http://localhost:7071/orchestrations/function-chaining | Send

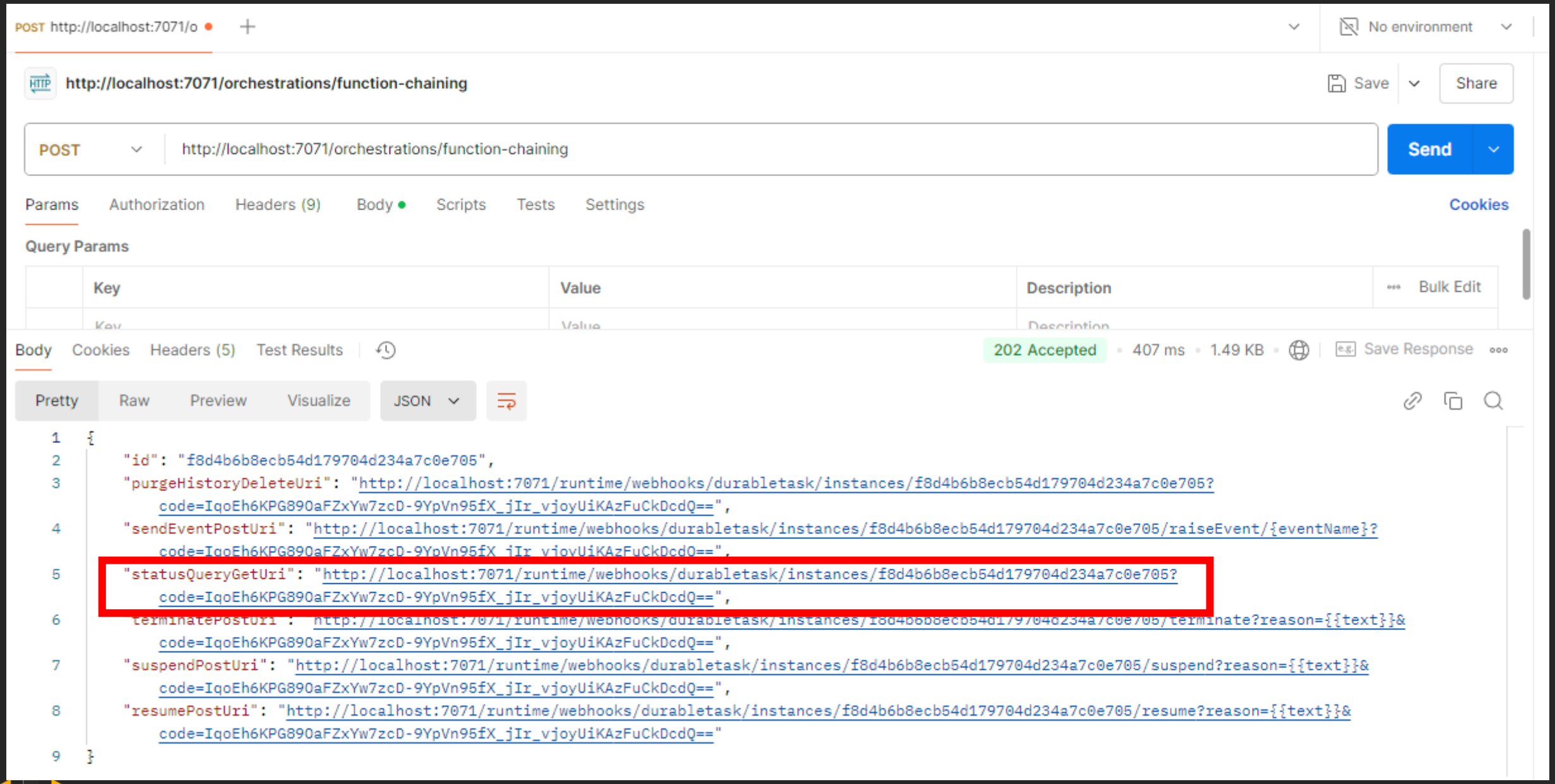
Params | Authorization | Headers (7) | Body | Scripts | Tests | Settings | Cookies

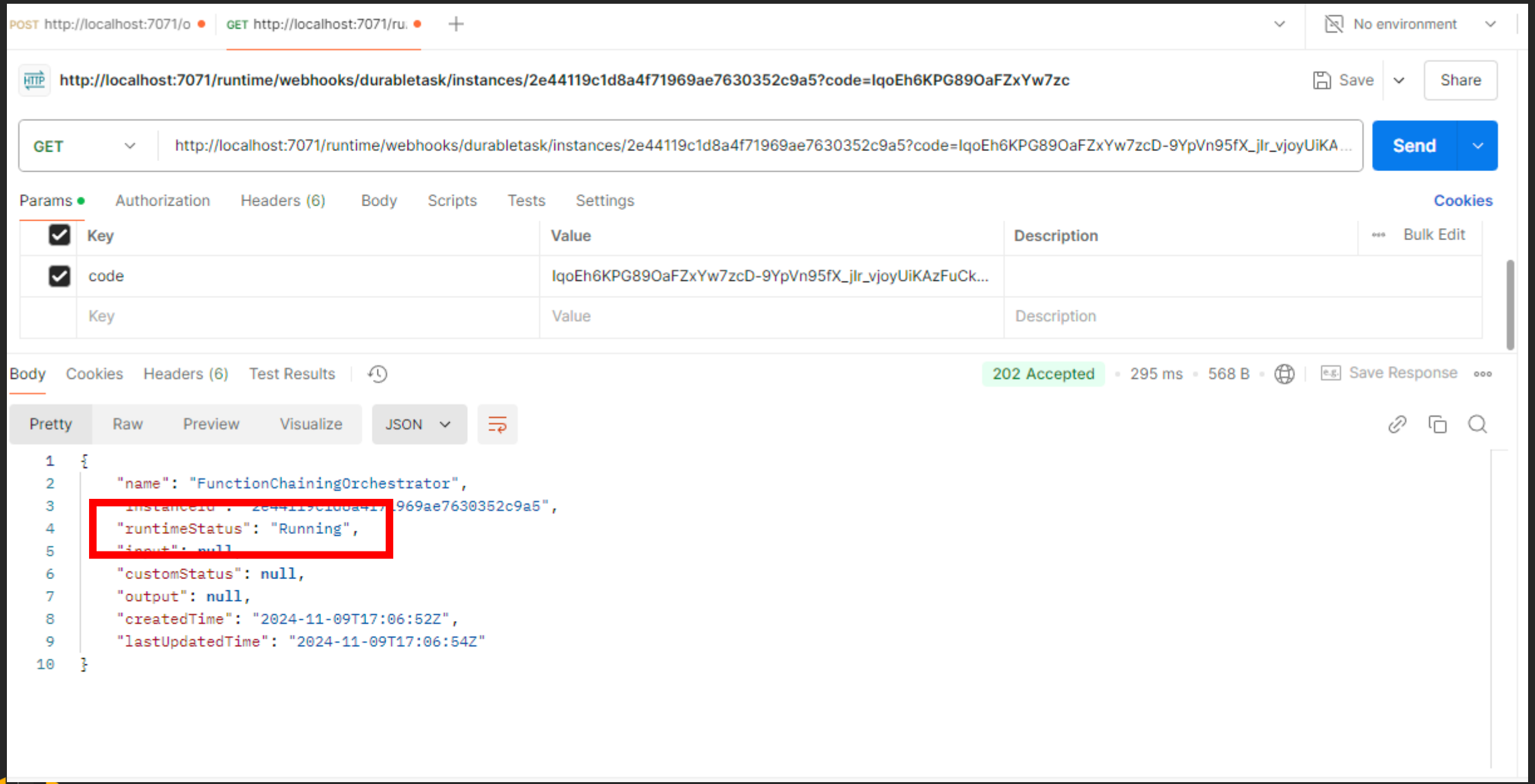
Query Params

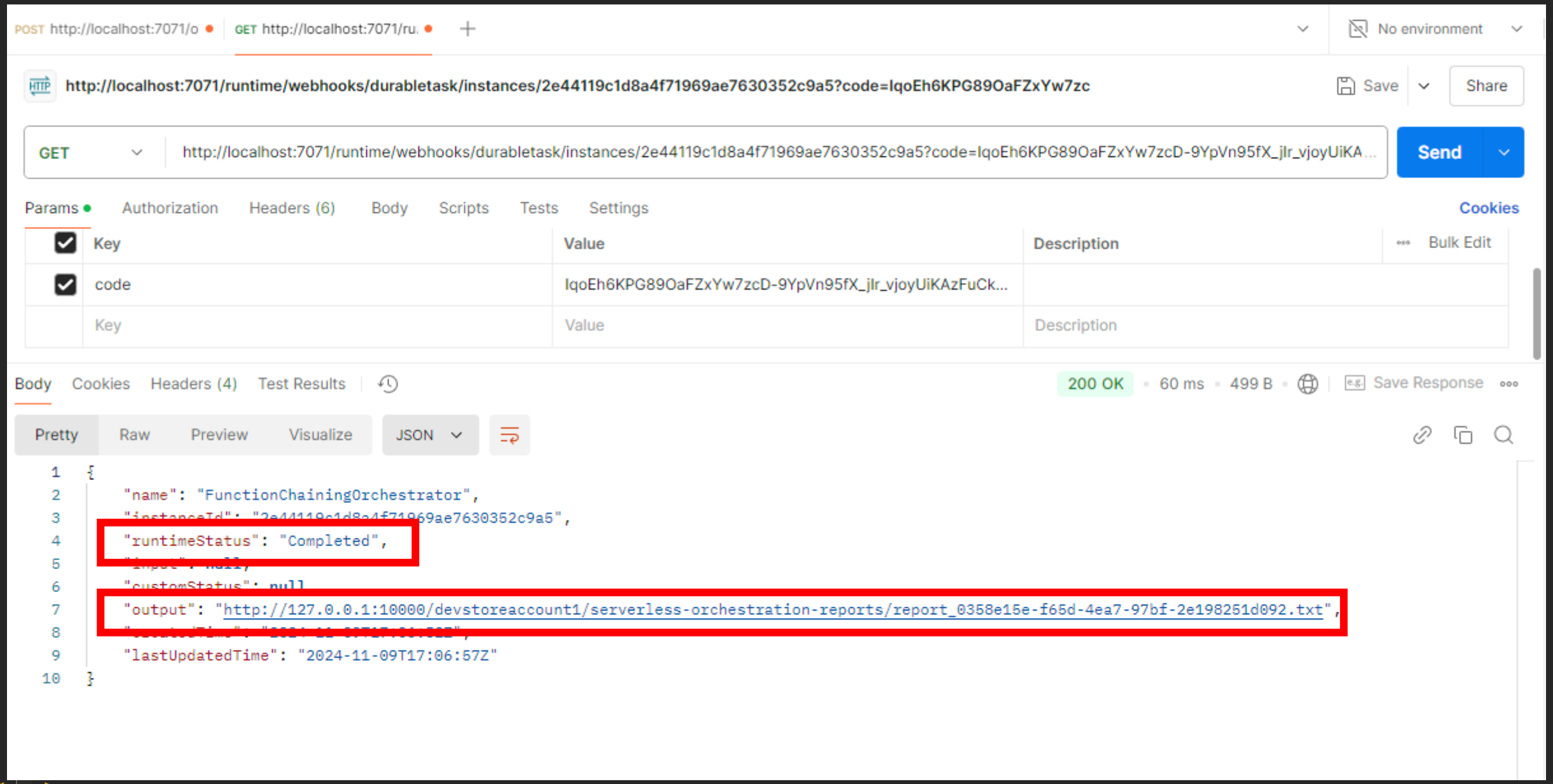
	Key	Value	Description	Bulk Edit
	Key	Value	Description	

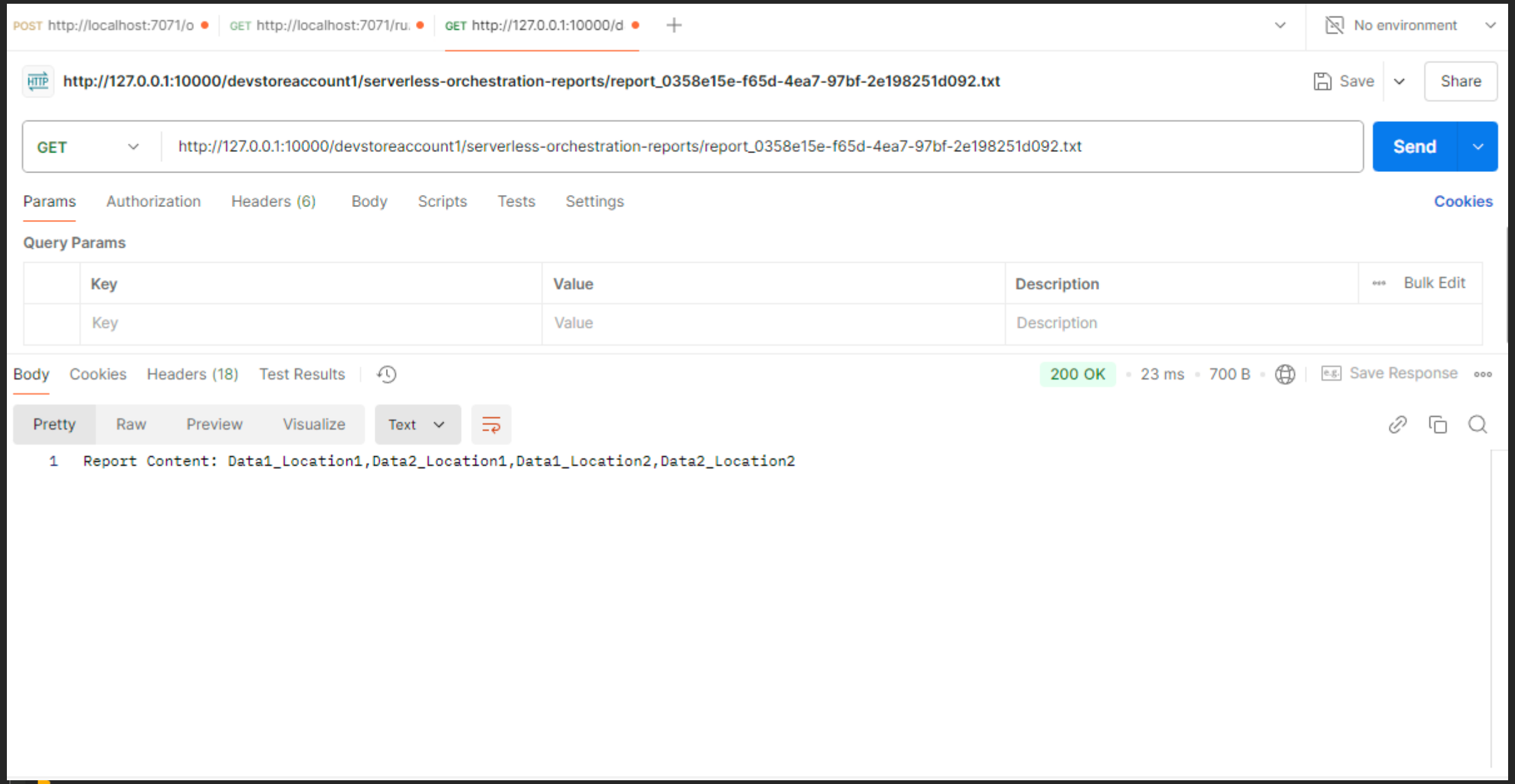
Response | History

Click Send to get a response

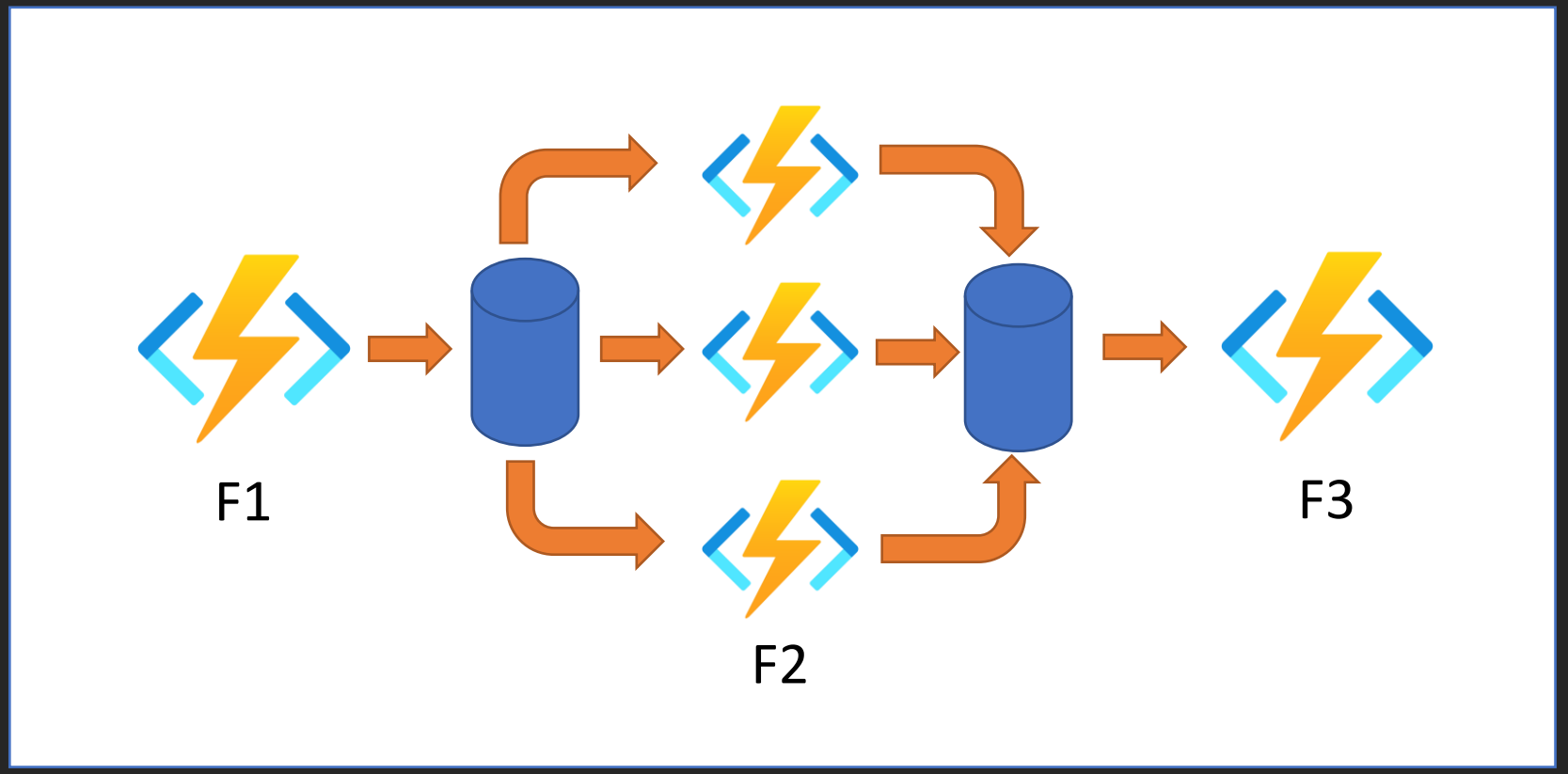

POST http://localhost:7071/o
No environment
http://localhost:7071/orchestrations/function-chaining
Save
Share
POST
http://localhost:7071/orchestrations/function-chaining
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (5)
Test Results
202 Accepted
407 ms
1.49 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": "f8d4b6b8ecb54d179704d234a7c0e705",
"purgeHistoryDeleteUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/f8d4b6b8ecb54d179704d234a7c0e705?code=IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jIr_vjoyUiKAzFuCkDcdQ==",
"sendEventPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/f8d4b6b8ecb54d179704d234a7c0e705/raiseEvent/{eventName}?code=IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jIr_vjoyUiKAzFuCkDcdQ==",
"statusQueryGetUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/f8d4b6b8ecb54d179704d234a7c0e705?code=IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jIr_vjoyUiKAzFuCkDcdQ==",
"terminatePostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/f8d4b6b8ecb54d179704d234a7c0e705/terminate?reason={{text}}&code=IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jIr_vjoyUiKAzFuCkDcdQ==",
"suspendPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/f8d4b6b8ecb54d179704d234a7c0e705/suspend?reason={{text}}&code=IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jIr_vjoyUiKAzFuCkDcdQ==",
"resumePostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/f8d4b6b8ecb54d179704d234a7c0e705/resume?reason={{text}}&code=IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jIr_vjoyUiKAzFuCkDcdQ=="
}


POST http://localhost:7071/o
GET http://localhost:7071/ru.
No environment
http://localhost:7071/runtime/webhooks/durabletask/instances/2e44119c1d8a4f71969ae7630352c9a5?code=IqoEh6KPG89OaFZxYw7zc
Save
Share
GET
http://localhost:7071/runtime/webhooks/durabletask/instances/2e44119c1d8a4f71969ae7630352c9a5?code=IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jlr_vjoyUiKA...
Send
Params
Authorization
Headers (6)
Body
Scripts
Tests
Settings
Cookies
Key
Value
Description
Bulk Edit
code
IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jlr_vjoyUiKAzFuCk...
Key
Value
Description
Body
Cookies
Headers (6)
Test Results
202 Accepted
295 ms
568 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"name": "FunctionChainingOrchestrator",
"runtimeStatus": "Running",
"customStatus": null,
"output": null,
"createdTime": "2024-11-09T17:06:52Z",
"lastUpdatedTime": "2024-11-09T17:06:54Z"
}


POST http://localhost:7071/o
GET http://localhost:7071/ru.
No environment
http://localhost:7071/runtime/webhooks/durabletask/instances/2e44119c1d8a4f71969ae7630352c9a5?code=IqoEh6KPG89OaFZxYw7zc
Save
Share
GET
http://localhost:7071/runtime/webhooks/durabletask/instances/2e44119c1d8a4f71969ae7630352c9a5?code=IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jlr_vjoyUiKA...
Send
Params
Authorization
Headers (6)
Body
Scripts
Tests
Settings
Cookies
Key
Value
Description
Bulk Edit
code
IqoEh6KPG89OaFZxYw7zcD-9YpVn95fX_jlr_vjoyUiKAzFuCk...
Key
Value
Description
Body
Cookies
Headers (4)
Test Results
200 OK
60 ms
499 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"name": "FunctionChainingOrchestrator",
"runtimeStatus": "Completed",
"output": "http://127.0.0.1:10000/devstoreaccount1/serverless-orchestration-reports/report_0358e15e-f65d-4ea7-97bf-2e198251d092.txt",
"lastUpdatedTime": "2024-11-09T17:06:57Z"
}


POST http://localhost:7071/o
GET http://localhost:7071/ru.
GET http://127.0.0.1:10000/d
No environment
http://127.0.0.1:10000/devstoreaccount1/serverless-orchestration-reports/report_0358e15e-f65d-4ea7-97bf-2e198251d092.txt
Save
Share
GET
http://127.0.0.1:10000/devstoreaccount1/serverless-orchestration-reports/report_0358e15e-f65d-4ea7-97bf-2e198251d092.txt
Send
Params
Authorization
Headers (6)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (18)
Test Results
200 OK
23 ms
700 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1 Report Content: Data1_Location1,Data2_Location1,Data1_Location2,Data2_Location2

Fan-out/Fan-in



Possible Scenarios

Batch Data Processing

Possible Scenarios

Stock Price Analysis

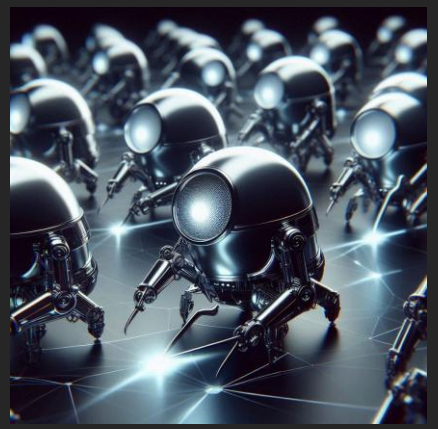
Possible Scenarios

Rendering Videos

Possible Scenarios

Concurrent API Calls

Possible Scenarios



Web Scrapping

Web Scrapping

Scrape Webpage

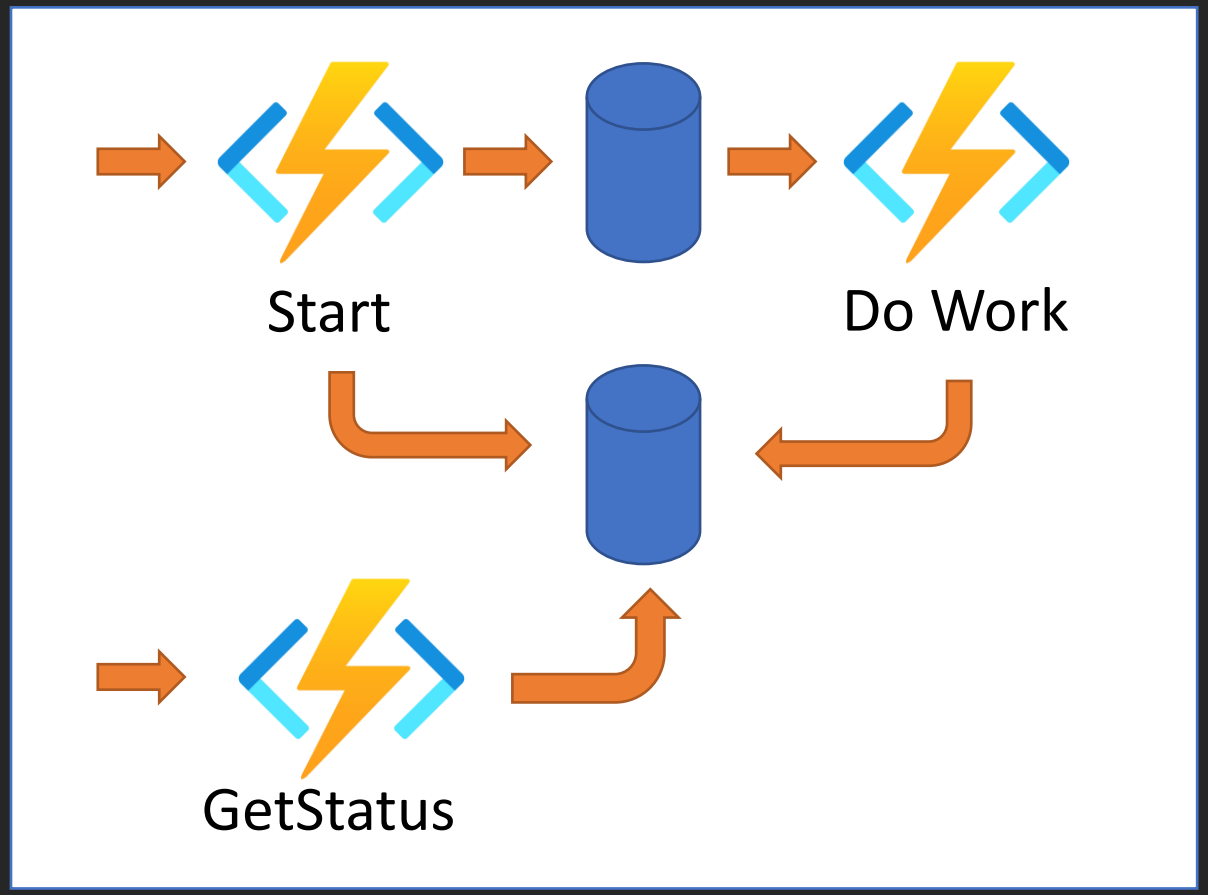
```
[Function(nameof(ScrapeWebpage))]
4 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<KeyValuePair<string, string>> ScrapeWebpage(
    [ActivityTrigger] string url, FunctionContext functionContext)
{
    ILogger logger = functionContext.GetLogger(nameof(ScrapeWebpage));
    logger.LogInformation("Scraping webpage: {url}", url);

    using HttpClient httpClient = HttpClientFactory();

    HttpResponseMessage response = await httpClient.GetAsync(url);
    response.EnsureSuccessStatusCode();

    string htmlContent = await response.Content.ReadAsStringAsync();
    HtmlDocument htmlDocument = new();
    htmlDocument.LoadHtml(htmlContent);

    HtmlNode titleNode = htmlDocument.DocumentNode.SelectSingleNode("//title");
    string title = titleNode != null ? titleNode.InnerText : "No Title";

    return new KeyValuePair<string, string>(url, title);

}
```

Starter

```
[Function(nameof(StartWebScrapping))]
2 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<HttpResponseData> StartWebScrapping(
  [HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "fan-out-fan-in")] HttpRequestData request,
  [DurableClient] DurableTaskClient client,
  FunctionContext executionContext)
{
  ILogger logger = executionContext.GetLogger("HttpStarter");

  string requestBody = await new StreamReader(request.Body).ReadToEndAsync();
  List<string>? urls = JsonConvert.DeserializeObject<List<string>>(requestBody);
  if (urls == null || urls.Count == 0)
    return request.CreateResponse(HttpStatusCode.BadRequest);

  string instanceId = await client.ScheduleNewOrchestrationInstanceAsync(nameof(Ochestrator.FanInFanOutOrcestrator), urls);
  logger.LogInformation("Started orchestration with ID = '{instanceId}'.", instanceId);
  return await client.CreateCheckStatusResponseAsync(request, instanceId);
}
```

```
[Function(nameof(FanInFanOutOrcestrator))]
3 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<Dictionary<string, string>> FanInFanOutOrcestrator(
[OrchestrationTrigger] TaskOrchestrationContext context)
{
  ILogger logger = context.CreateReplaySafeLogger(nameof(Orchestrator));

  List<string>? urls = context.GetInput<List<string>>();
  if (urls == null)
  {
    logger.LogError("Input URLs cannot be null");
    return [];
  }

  List<Task<KeyValuePair<string, string>>> tasks = [];
  foreach (string url in urls)
    tasks.Add(context.CallActivityAsync<KeyValuePair<string, string>>(nameof(ScrapeWebpageActivity.ScrapeWebpage), url));

  logger.LogInformation("Starting fan-out/fan-in activities...");
  KeyValuePair<string, string>[] results = await Task.WhenAll(tasks);
  logger.LogInformation("Fan-out/fan-in activities completed.");

  Dictionary<string, string> scrapedData = [];
  foreach (KeyValuePair<string, string> result in results)
    scrapedData.Add(result.Key, result.Value);

  return scrapedData;

}
```

Async HTTP APIs



Possible Scenarios

Video Processing

Possible Scenarios

Report Generation

Possible Scenarios

Image Recognition

Possible Scenarios

Machine Learning Model Training

Possible Scenarios

Long-Running Database Operations

Long-Running Database Operation

Starter

Orchestrator

Activity: Database Operation

Starter

```
[Function(nameof(AsyncApiHttpStarter))]
2 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task<HttpResponseData> AsyncApiHttpStarter(
  [HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "async-http-api")] HttpRequestData request,
  [DurableClient] DurableTaskClient client,
  FunctionContext executionContext)
{
  ILogger logger = executionContext.GetLogger("HttpStarter");

  string requestBody = await new StreamReader(request.Body).ReadToEndAsync();
  Dictionary<string, string>? data = JsonConvert.DeserializeObject<Dictionary<string, string>>(requestBody);
  string? operationId = data?["operationId"];
  if (string.IsNullOrWhiteSpace(operationId))
    return request.CreateResponse(HttpStatusCode.BadRequest);

  string instanceId = await client.ScheduleNewOrchestrationInstanceAsync(nameof(Orchestrator.DatabaseOperationOrchestrator), operationId);
  logger.LogInformation("Started orchestration with ID = '{instanceId}'.", instanceId);
  return await client.CreateCheckStatusResponseAsync(request, instanceId);
}
```

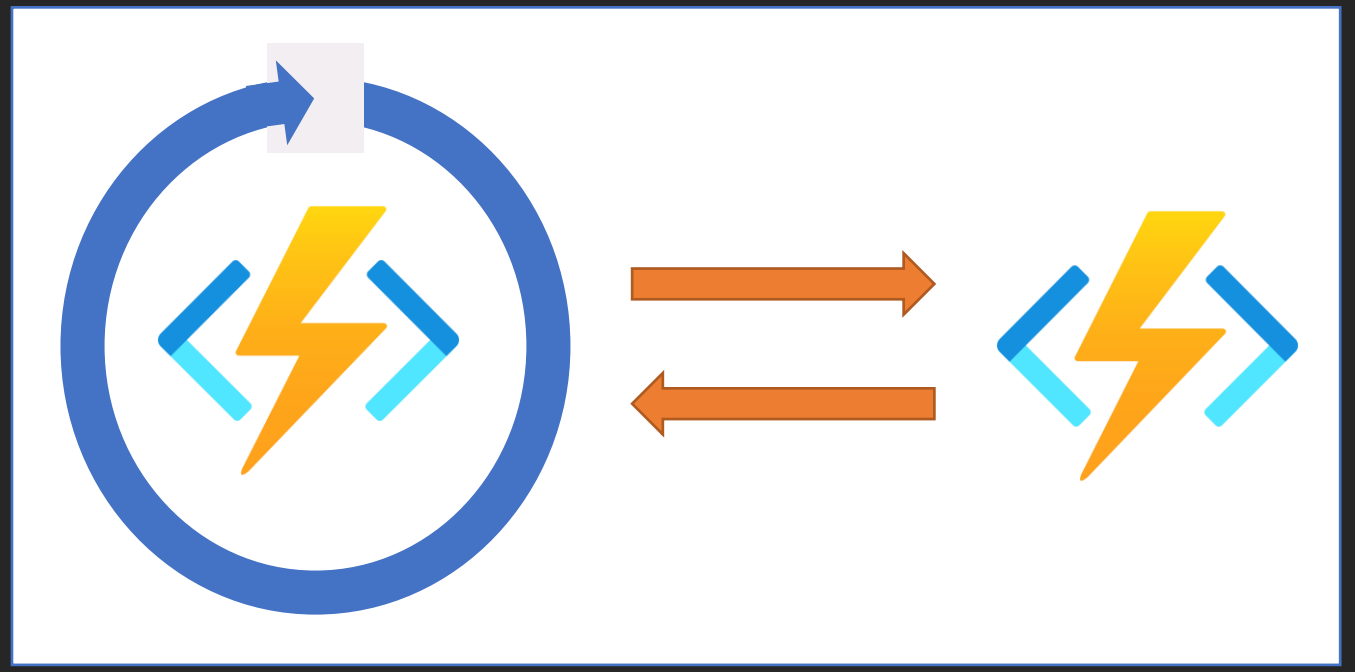
Orchestrator

```
[Function(nameof(DatabaseOperationOrchestrator))]
3 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task DatabaseOperationOrchestrator(
[OrchestrationTrigger] TaskOrchestrationContext context)
{
    ILogger logger = context.CreateReplaySafeLogger(nameof(Orchestrator));
    string? operationId = context.GetInput<string>();
    logger.LogInformation("Starting database operation for {operationId}...", operationId);
    await context.CallActivityAsync(nameof(PerformDatabaseOperationActivity.PerformDatabaseOperation), operationId);
    logger.LogInformation("Database operation for {operationId} completed.", operationId);
}
```

Long-Running Operation

```
[Function(nameof(PerformDatabaseOperation))]
4 references | Chad Green, 6 days ago | 1 author, 1 change
public static async Task PerformDatabaseOperation([ActivityTrigger] string operationId, FunctionContext functionContext)
{
    ILogger logger = functionContext.GetLogger(nameof(PerformDatabaseOperation));
    logger.LogInformation("Performing long-running database operation for {operationId}...", operationId);
    await Task.Delay(30000); // Simulate a 30-second database operation
    logger.LogInformation("Database operation for {operationId} completed.", operationId);
}
```

Monitoring



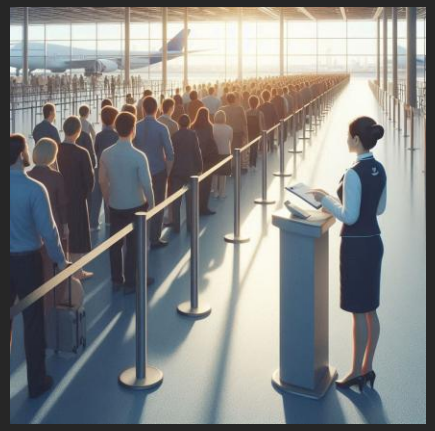
Possible Scenarios

Batch Job Monitoring

Possible Scenarios

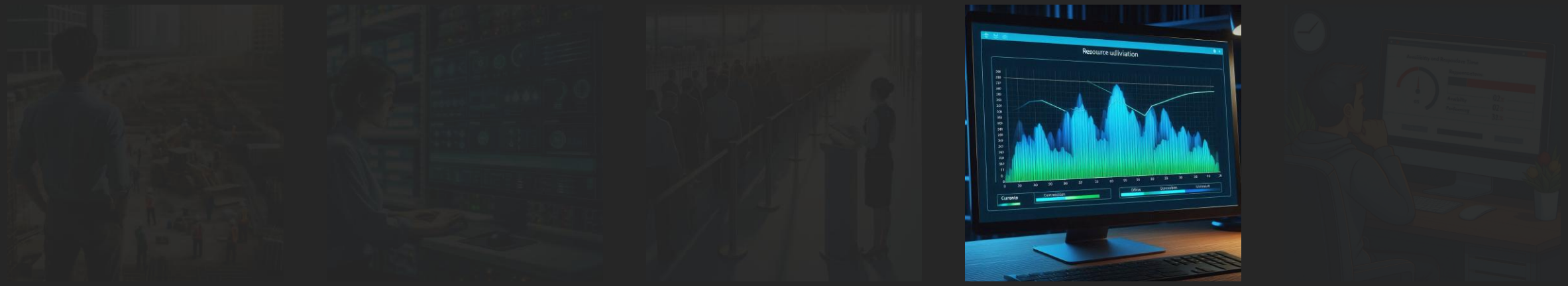
IoT Device Health Monitoring

Possible Scenarios



Queue Length Monitoring

Possible Scenarios



Resource Utilization Monitoring

Possible Scenarios

Website Availability Monitoring

Website Availability Monitoring

Activity: Check Website Status

Activity: Send Alert

Check

```
[Function(nameof(CheckWebsiteStatus))]
4 references | Chad Green, 2 days ago | 1 author, 1 change
public static async Task<bool> CheckWebsiteStatus([ActivityTrigger] string url, FunctionContext functionContext)
{

  ILogger logger = functionContext.GetLogger(nameof(CheckWebsiteStatus));
  logger.LogInformation("Checking Website Status: {url}", url);

  using HttpClient httpClient = HttpClientFactory();

  Uri uri = new(url);

  bool isWebsiteUp = false;
  string? errorMessage = null;
  string? statusCode = null;
  long responseTimeMs = 0;

  try
  {
    Stopwatch stopwatch = Stopwatch.StartNew();
    HttpResponseMessage response = await httpClient.GetAsync(uri);
    stopwatch.Stop();
    isWebsiteUp = response.IsSuccessStatusCode;
  }
  catch (Exception ex)
  {
    logger.LogError("Error checking website {url}: {errorMessage}", url, ex.Message);
    errorMessage = ex.Message;
  }

  try
  {
    TableClient tableClient = GetTableClient();

    await tableClient.AddEntityAsync(new TelemetryTableEntity
    {
      PartitionKey = uri.Host,
      RowKey = Guid.NewGuid().ToString(),
      Url = url,
      IsUp = isWebsiteUp,
      ErrorMessage = errorMessage,
      StatusCode = statusCode,
      ResponseTimeMs = responseTimeMs
    });
  }
  catch (Exception ex)
  {
    logger.LogError("Error saving telemetry for website {url}: {errorMessage}", url, ex.Message);
  }

  return isWebsiteUp;

}
```

Send Alert

```
[Function(nameof(SendAlert))]
4 references | Chad Green, 2 days ago | 1 author, 1 change
public static void SendAlert([ActivityTrigger] string url, FunctionContext functionContext)
{

  ILogger logger = functionContext.GetLogger(nameof(SendAlert));
  logger.LogInformation("Sending alert of down website - {url}...", url);

  EmailClient emailClient = GetEmailClient();
  EmailMessage emailMessage = new(
    senderAddress: Environment.GetEnvironmentVariable("SendEmailFrom")!,
    content: new EmailContent("Website Down Alert")
    {
      PlainText = $"The website `{url} is down.",
      Html = $@"<html><body><h1>Website Down Alert</h1><p>The website `{url}' is down.</body></html>"
    },
    recipients: new EmailRecipients([new EmailAddress("chadgreen@chadgreen.com")])
  );
  EmailSendOperation emailSendOperation = emailClient.Send(WaitUntil.Started, emailMessage);

}
```

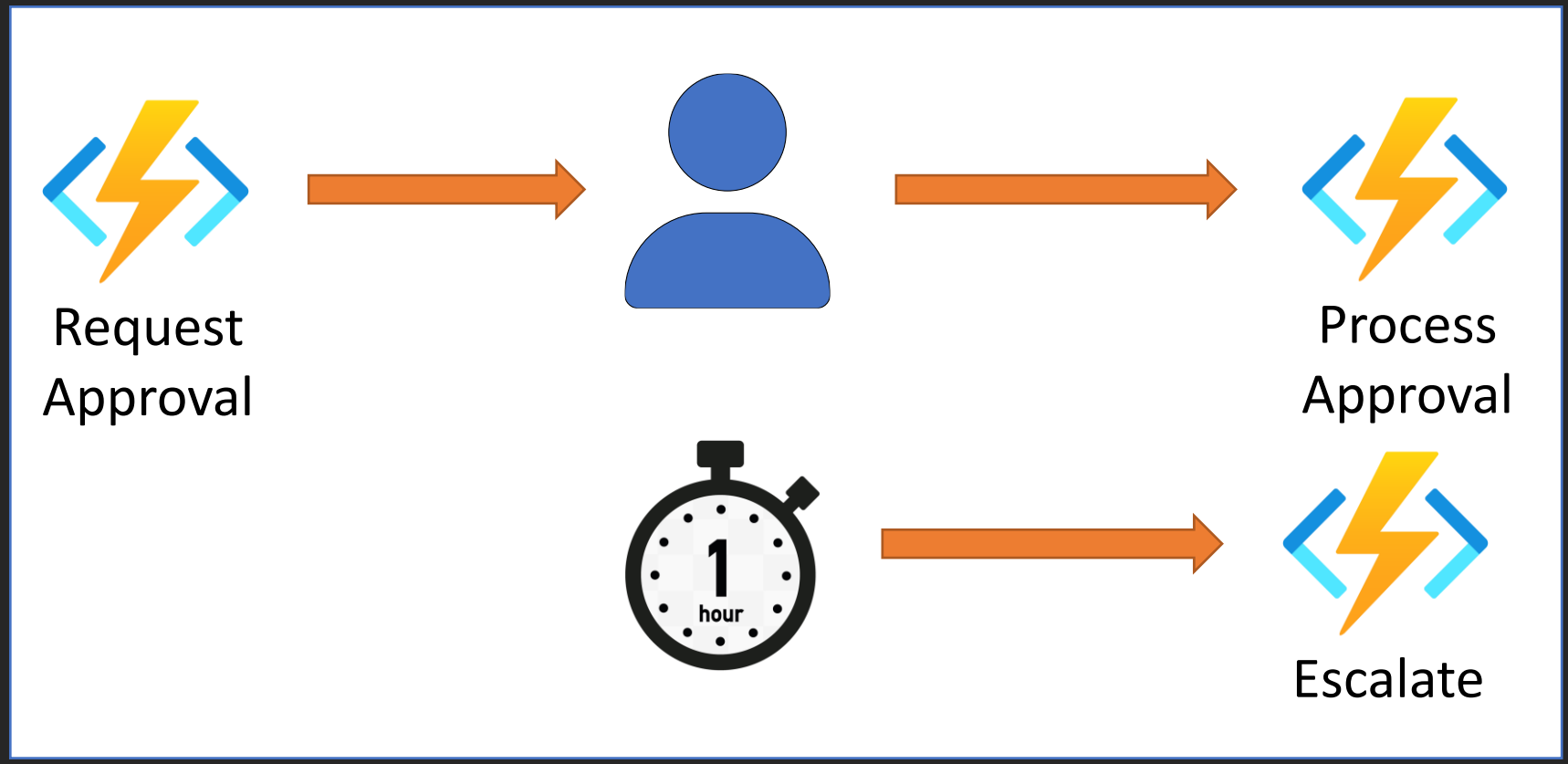
Starter

```
[Function(nameof(MonitorHttpStarter))]
2 references | Chad Green, 2 days ago | 1 author, 1 change
public static async Task<HttpResponseData> MonitorHttpStarter(
[HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "monitor")] HttpRequestData request,
[DurableClient] DurableTaskClient client,
FunctionContext executionContext)
{
  ILogger logger = executionContext.GetLogger("HttpStarter");
  string? websiteUrl = request.Query["url"];
  if (string.IsNullOrEmpty(websiteUrl))
    return request.CreateResponse(HttpStatusCode.BadRequest);
  string instanceId = await client.ScheduleNewOrchestrationInstanceAsync(nameof(Orchestrator.MonitorOrchestrator), websiteUrl);
  logger.LogInformation("Started orchestration with ID = '{instanceId}'.", instanceId);
  return await client.CreateCheckStatusResponseAsync(request, instanceId);
}
```

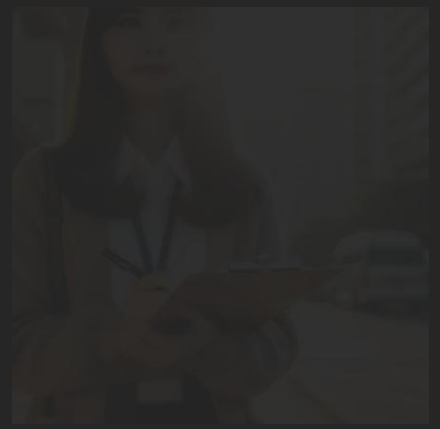
Orchestrator

```
[Function(nameof(MonitorOrchestrator))]
3 references | Chad Green, 2 days ago | 1 author, 1 change
public static async Task MonitorOrchestrator(
  [OrchestrationTrigger] TaskOrchestrationContext context)
{

  ILogger logger = context.CreateReplaySafeLogger(nameof(Orchestrator));
  logger.LogInformation("Starting website monitor service...");

  string? websiteUrl = context.GetInput<string>();
  TimeSpan checkInterval = TimeSpan.FromMinutes(Convert.ToDouble(Environment.GetEnvironmentVariable("WebsiteMonitoringDefaultInternval")!));

  while (context.CurrentUtcDateTime < context.CurrentUtcDateTime.AddHours(24)) // Monitor for 24 hours
  {
    logger.LogInformation("Checking website status for {websiteUrl}...", websiteUrl);
    bool isWebsiteUp = await context.CallActivityAsync<bool>(nameof(CheckWebsiteStatusActivity.CheckWebsiteStatus), websiteUrl);
    if (!isWebsiteUp)
    {
      logger.LogError("Website {websiteUrl} is down. Sending alert...", websiteUrl);
      await context.CallActivityAsync(nameof(SendAlertActivity.SendAlert), websiteUrl);
    }
    else
    {
      logger.LogInformation("Website {websiteUrl} is up. Waiting for next check...", websiteUrl);
    }
    DateTime nextCheck = context.CurrentUtcDateTime.Add(checkInterval);
    await context.CreateTimer(context.CurrentUtcDateTime.Add(checkInterval), CancellationToken.None);
  }

}
```

Human Interaction



Possible Scenarios



Approval Workflow

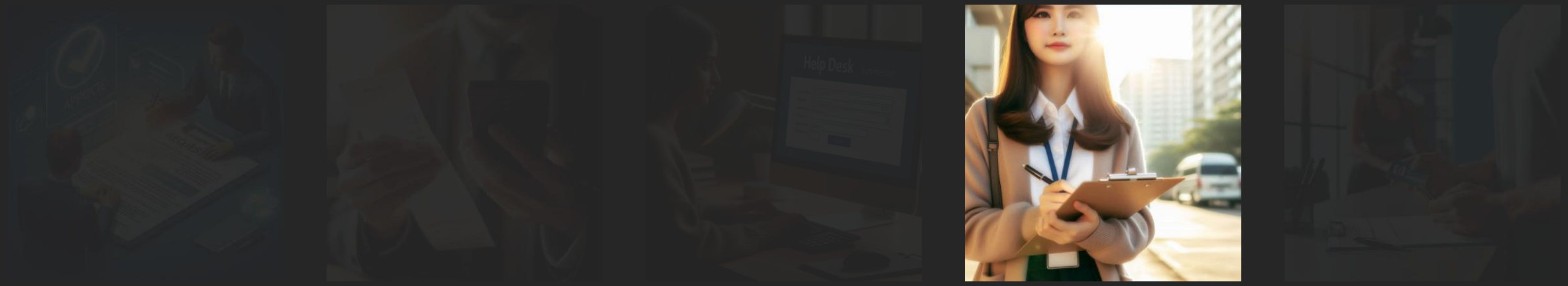
Possible Scenarios

Expense Report Submission

Possible Scenarios

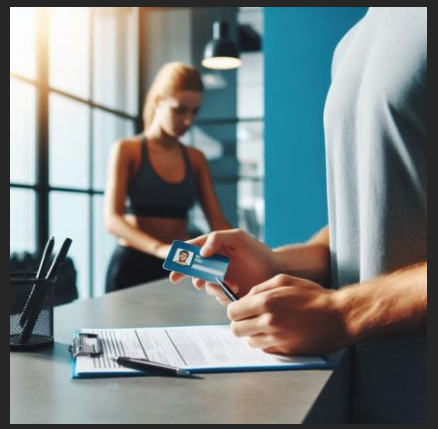
Support Ticket Escalation

Possible Scenarios



Survey or Feedback Collection

Possible Scenarios



User Registration

User Registration

Orch

```
[Function("HumanInteraction")]
2 references | Chad Green, 313 days ago | 1 author, 1 change
public async Task<bool> RunOrchestratorAsync([OrchestrationTrigger] TaskOrchestrationContext context)
{

  ILogger logger = context.CreateReplaySafeLogger(nameof(RunOrchestratorAsync));

  string? emailAddress = context.GetInput<string>();
  ArgumentException.ThrowIfNullOrWhiteSpace(emailAddress, nameof(emailAddress));

  ChallengeTableEntity challenge = await context.CallActivityAsync<ChallengeTableEntity>(nameof(SendChallengeAsync), emailAddress);

  using CancellationTokenSource timeoutCts = new();
  DateTime expiration = context.CurrentUtcDateTime.AddSeconds(_challengeTimeoutSeconds);
  Task timeoutTask = context.CreateTimer(expiration, timeoutCts.Token);

  bool authorized = false;
  for (int retryCount = 0; retryCount <= 3; retryCount++)
  {

    Task<int> challengeResponseTask = context.WaitForExternalEvent<int>("EmailChallengeResponse");

    Task winner = await Task.WhenAny(challengeResponseTask, timeoutTask);
    if (winner == challengeResponseTask)
    {
      // Response received; compare it to the challenge code
      if (challengeResponseTask.Result == Convert.ToInt32(challenge.ChallengeCode))
      {
        logger.LogInformation("Challenge successfully completed");
        authorized = true;
        challenge.ChallengeCompleted = true;
        challenge.ChallengePeriodEnd = context.CurrentUtcDateTime;
        await context.CallActivityAsync(nameof(RecordChallengeStatusAsync), challenge);
        break;
      }
    }
    else
    {
      // Timeout expired
      logger.LogInformation("Challenge timed out");
      challenge.ChallengePeriodEnd = context.CurrentUtcDateTime;
      await context.CallActivityAsync(nameof(RecordChallengeStatusAsync), challenge);
      break;
    }
  }

  if (!timeoutTask.IsCompleted)
    //  All pending times must be complete or cancelled before the function exits
    timeoutCts.Cancel();

  return authorized;

}
```

Orch

```
[Function("HumanInteraction")]
2 references | Chad Green, 313 days ago | 1 author, 1 change
public async Task<bool> RunOrchestratorAsync([OrchestrationTrigger] TaskOrchestrationContext context)
{

  ILogger logger = context.CreateReplaySafeLogger(nameof(RunOrchestratorAsync));

  string? emailAddress = context.GetInput<string>();
  ArgumentException.ThrowIfNullOrWhiteSpace(emailAddress, nameof(emailAddress));

  ChallengeTableEntity challenge = await context.CallActivityAsync<ChallengeTableEntity>(nameof(SendChallengeAsync), emailAddress);

  using CancellationTokenSource timeoutCts = new();
  DateTime expiration = context.CurrentUtcDateTime.AddSeconds(_challengeTimeoutSeconds);
  Task timeoutTask = context.CreateTimer(expiration, timeoutCts.Token);

  bool authorized = false;
  for (int retryCount = 0; retryCount <= 3; retryCount++)
  {

    Task<int> challengeResponseTask = context.WaitForExternalEvent<int>("EmailChallengeResponse");

    Task winner = await Task.WhenAny(challengeResponseTask, timeoutTask);
    if (winner == challengeResponseTask)
    {
      // Response received; compare it to the challenge code
      if (challengeResponseTask.Result == Convert.ToInt32(challenge.ChallengeCode))
      {
        logger.LogInformation("Challenge successfully completed");
        authorized = true;
        challenge.ChallengeCompleted = true;
        challenge.ChallengePeriodEnd = context.CurrentUtcDateTime;
        await context.CallActivityAsync(nameof(RecordChallengeStatusAsync), challenge);
        break;
      }
    }
    else
    {
      // Timeout expired
      logger.LogInformation("Challenge timed out");
      challenge.ChallengePeriodEnd = context.CurrentUtcDateTime;
      await context.CallActivityAsync(nameof(RecordChallengeStatusAsync), challenge);
      break;
    }
  }

  if (!timeoutTask.IsCompleted)
    //  All pending times must be complete or cancelled before the function exits
    timeoutCts.Cancel();

  return authorized;

}
```

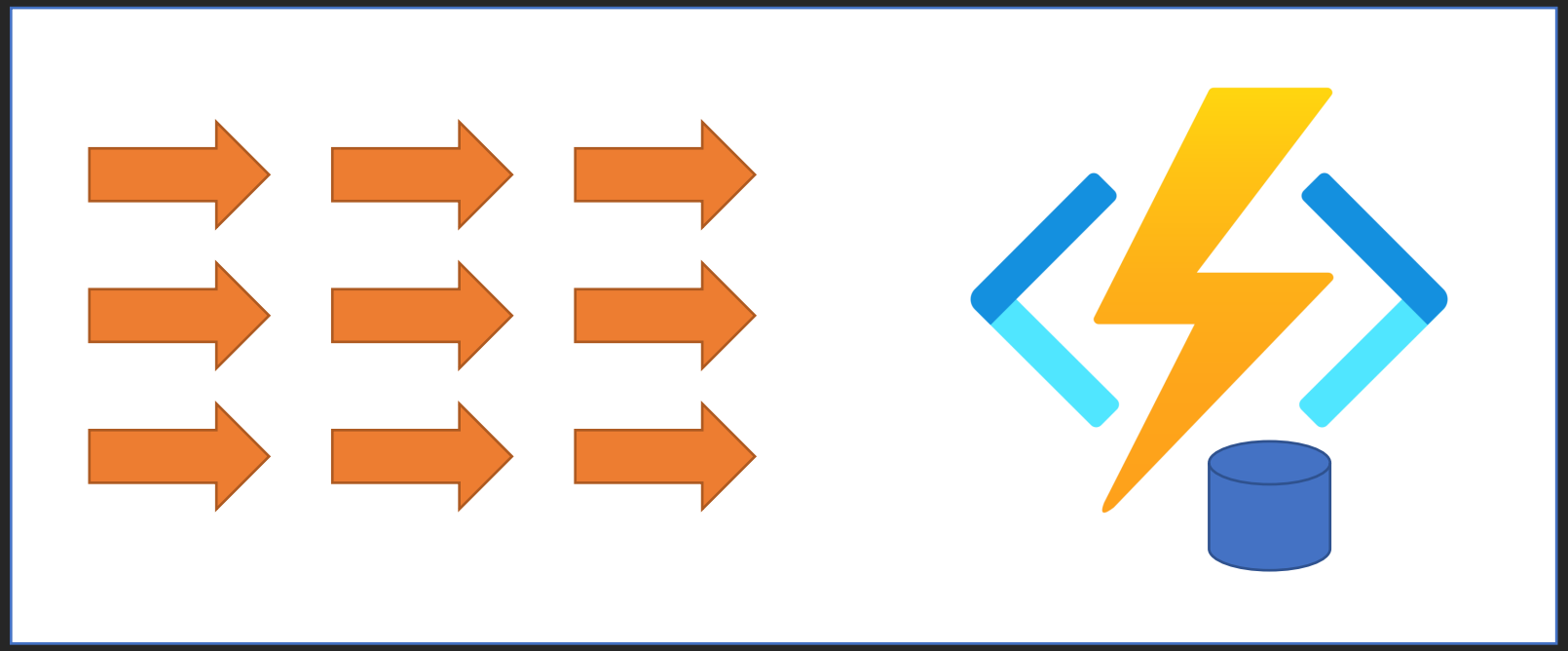
```
[Function("HumanInteraction")]
2 references | Chad Green, 313 days ago | 1 author, 1 change
public async Task<bool> RunOrchestratorAsync([OrchestrationTrigger] TaskOrchestrationContext context)
{

  ILogger logger = context.CreateReplaySafeLogger(nameof(RunOrchestratorAsync));

  string? emailAddress = context.GetInput<string>();
  Arg
  Cha
  usi
  Dat
  Tas
  boo
  for
  {
    Task<int> challengeResponseTask = context.WaitForExternalEvent<int>("EmailChallengeResponse");

    Task winner = await Task.WhenAny(challengeResponseTask, timeoutTask);
    if (winner == ch
    {
      // Response re
      if (challengeR
      {
        logger.LogIn
        authorized =
        challenge.Ch
        challenge.ChallengePeriodEnd = context.CurrentUtcDateTime;
        await context.CallActivityAsync(nameof(RecordChallengeStatusAsync), challenge);
        break;
      }
    }
    else
    {
      // Timeout expired
      logger.LogInformation("Challenge timed out");
      challenge.ChallengePeriodEnd = context.CurrentUtcDateTime;
      await context.CallActivityAsync(nameof(RecordChallengeStatusAsync), challenge);
      break;
    }
  }

  if (!timeoutTask.IsCompleted)
    //  All pending times must be complete or cancelled before the function exits
    timeoutCts.Cancel();

  return authorized;
}
```

```
ILogger logger = context.CreateReplaySafeLogger(nameof(RunOrchestratorAsync));

string? emailAddress = context.GetInput<string>();
ArgumentException.ThrowIfNullOrWhiteSpace(emailAddress, nameof(emailAddress));

ChallengeTableEntity challenge = await context.CallActivityAsync<ChallengeTableEntity>(nameof(SendChallengeAsync), emailAddress);
```

```
using CancellationTokenSource timeoutCts = new();
DateTime expiration = context.CurrentUtcDateTime.AddSeconds(_challengeTimeoutSeconds);
Task timeoutTask = context.CreateTimer(expiration, timeoutCts.Token);
```

```
bool authorized = false;
for (int retryCount = 0; retryCount <= 3; retryCount++)
{

  Task<int> challengeResponseTask = context.WaitForExternalEvent<int>("EmailChallengeResponse");

  Task winner = await Task.WhenAny(challengeResponseTask, timeoutTask);
  if (winner == challengeResponseTask)
  {
    // Response received; compare it to the challenge code
    if (challengeResponseTask.Result == Convert.ToInt32(challenge.ChallengeCode))
    {
      logger.LogInformation("Challenge successfully completed");
      authorized = true;
      challenge.ChallengeCompleted = true;
      challenge.ChallengePeriodEnd = context.CurrentUtcDateTime;
      await context.CallActivityAsync(nameof(RecordChallengeStatusAsync), challenge);
      break;
    }
  }
  else
  {
    // Timeout expired
    logger.LogInformation("Challenge timed out");
    challenge.ChallengePeriodEnd = context.CurrentUtcDateTime;
    await context.CallActivityAsync(nameof(RecordChallengeStatusAsync), challenge);
    break;
  }
}
```

Aggregator (Stateful Entities)



Possible Scenarios

Weather Data Aggregation

Possible Scenarios

Sales Data Aggregation

Possible Scenarios

Log Data Aggregation

Possible Scenarios

Social Media Mentions

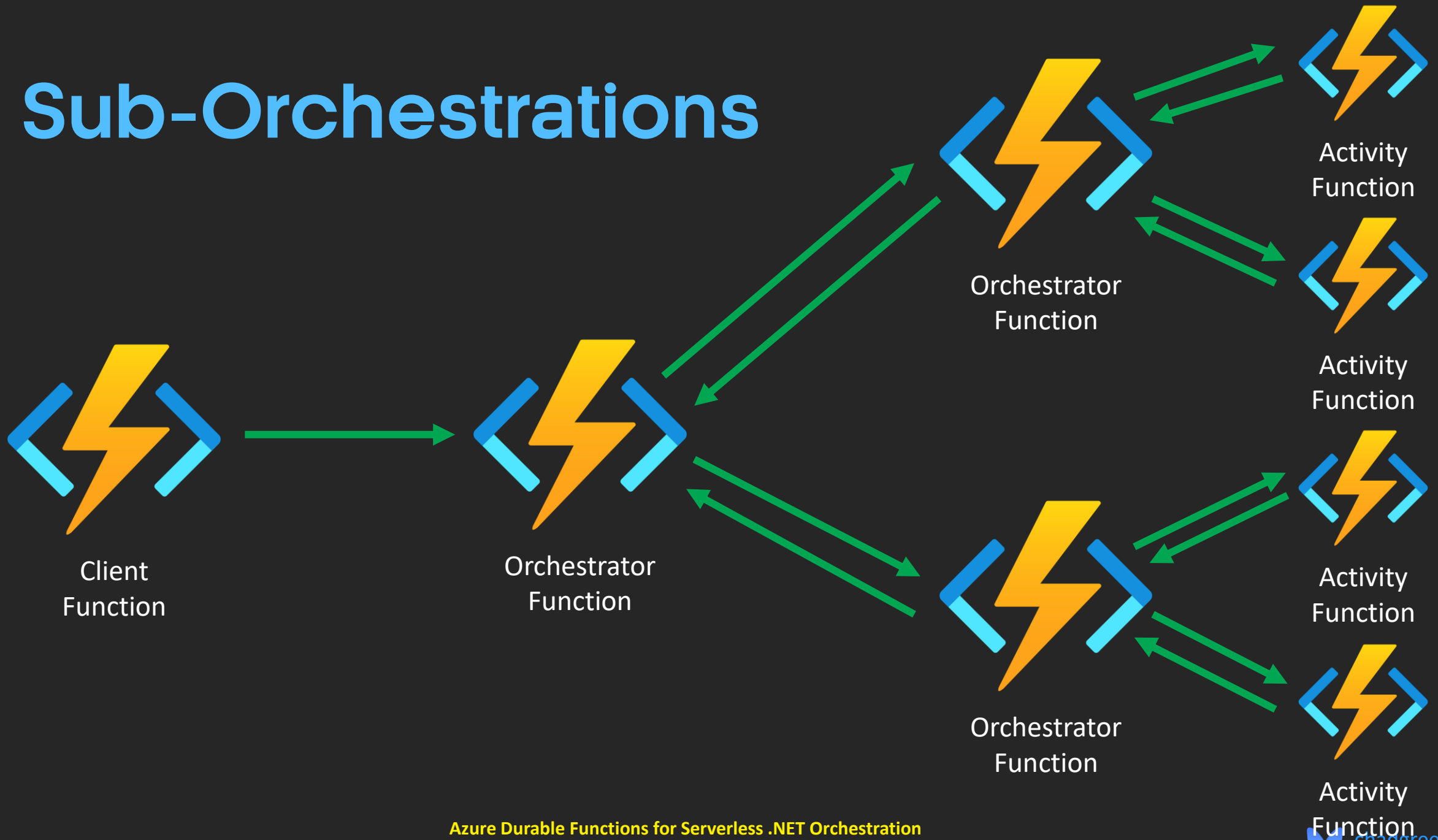
Possible Scenarios

Stock Market Data Aggregation

Additional Details

Azure Durable Functions for Serverless .NET Orchestration

Sub-Orchestrations



Custom Orchestration Status

```
[FunctionName("E1_HelloSequence")]
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
  var outputs = new List<string>();

  outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokyo"));
  outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
  outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "London"));


  // returns ["Hello Tokyo!", "Hello Seattle!", "Hello London!"]
  return outputs; }

[FunctionName("E1_SayHello")]
public static string SayHello([ActivityTrigger] string name)
{
  return $"Hello {name}!";
}
```

Custom Orchestration Status


```
{
  "runtimeStatus": "Running",
  "input": null,
  "output": null,
  "createdTime": "2019-10-06T18:30:24Z",
  "lastUpdatedTime": "2019-10-06T19:40:30Z"
}
```


Custom Orchestration Status

```
[FunctionName("E1_HelloSequence")]
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)

{
  var outputs = new List<string>();

  outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokyo"));
  context.SetCustomStatus("Tokyo");
  outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
  context.SetCustomStatus("Seattle");
  outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "London"));
  context.SetCustomStatus("London");


  // returns ["Hello Tokyo!", "Hello Seattle!", "Hello London!"]
  return outputs; }

[FunctionName("E1_SayHello")]
public static string SayHello([ActivityTrigger] string name)
{
  return $"Hello {name}!";
}
```

Custom Orchestration Status

```
{
  "runtimeStatus": "Running",
  "input": null,
  "customStatus": { “Seattle" },
  "output": null,
  "createdTime": "2019-10-06T18:30:24Z",
  "lastUpdatedTime": "2019-10-06T19:40:30Z"
}
```

Eternal Orchestrations

```
[FunctionName("Periodic_Cleanup_Loop")]
public static async Task Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
  await context.CallActivityAsync("DoCleanup", null);
  // sleep for one hour between cleanups
  DateTime nextCleanup = context.CurrentUtcDateTime.AddHours(1);
  await context.CreateTimer(nextCleanup, CancellationToken.None);

  context.ContinueAsNew(null);
}
```

Best Practices

Azure Durable Functions for Serverless .NET Orchestration

Best Practices

- Use the latest version of the Durable Functions extension and SDK

Best Practices

- Use the latest version of the Durable Functions extension and SDK
- Adhere to Durable Functions **code constraints**

Best Practices

- Use the latest version of the Durable Functions extension and SDK
- Adhere to Durable Functions **code constraints**
- Keep function inputs and outputs as small as possible

Best Practices

- Use the latest version of the Durable Functions extension and SDK
- Adhere to Durable Functions **code constraints**
- Keep function inputs and outputs as small as possible
- Keep Entity data small

Best Practices

- Use the latest version of the Durable Functions extension and SDK
- Adhere to Durable Functions **code constraints**
- Keep Entity data small
- Invest in stress testing

Exciting Enhancements Coming

Conclusion

Azure Durable Functions for Serverless .NET Orchestration

Conclusion

- What is Serverless Computing
- Function as a Service
- Azure Functions
- Durable Functions
- Application Patterns
- Additional Details
- Best Practices

Thank You

chadgreen@chadgreen.com

TaleLearnCode

ChadGreen.com

ChadGreen

ChadwickEGreen

Microsoft® Most Valuable Professional