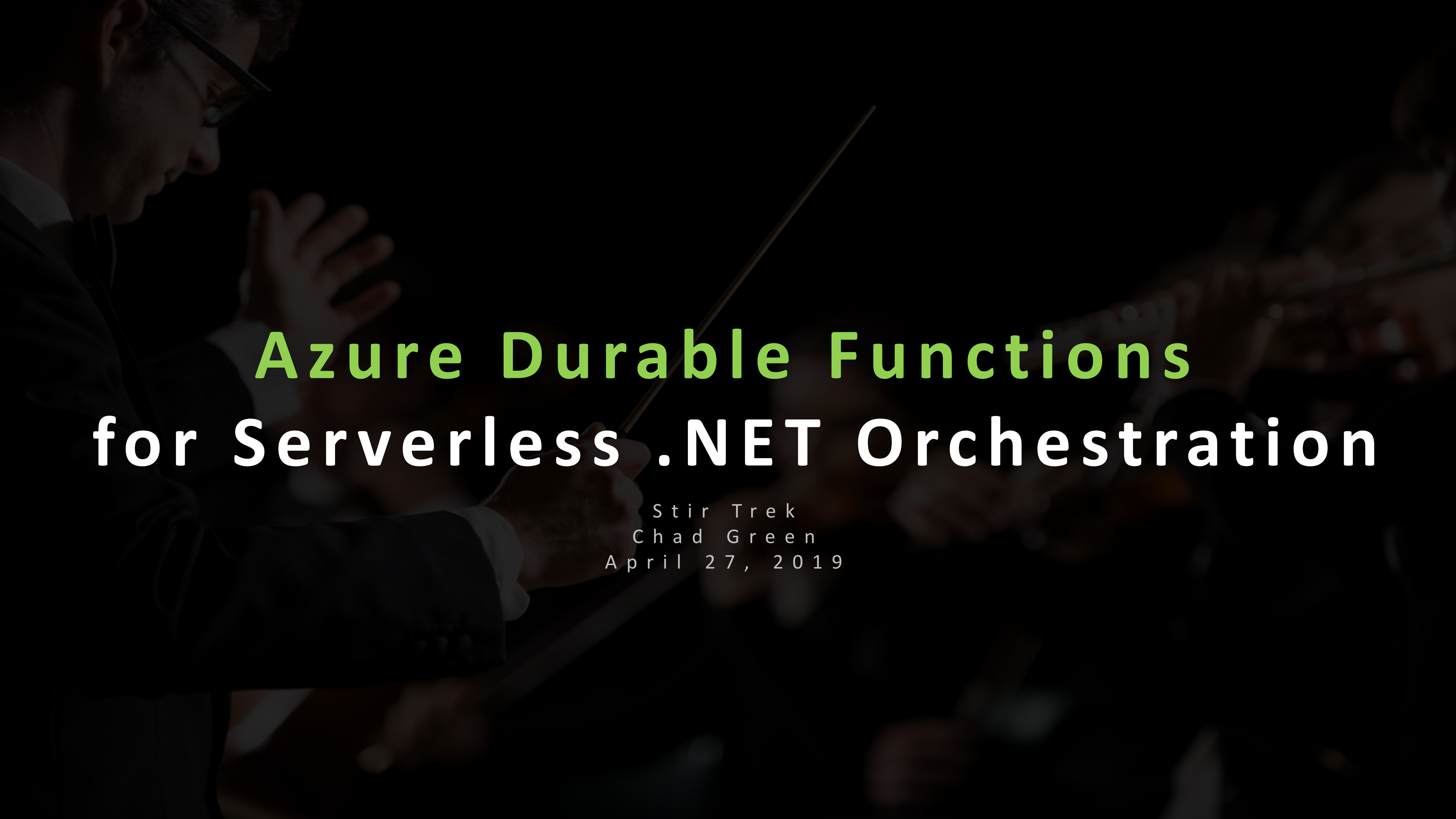


Azure Durable Functions for Serverless .NET Orchestration

Stir Trek
Chad Green
April 27, 2019

Who is Chad Green

Director of Software Development
ScholarRx

chadgreen@chadgreen.com

chadgreen.com

ChadGreen

ChadwickEGreen

Agenda

Azure Durable Functions for Stateless .NET Orchestration

ASK QUESTIONS DURING THE SESSIONS!

slack

THERE IS A SEPARATE CHANNEL FOR EACH TRACK:

#2019--HMB	#2019--LEADINGEDJE	#2019--MANIFEST	#2019--MPW
#2019--OCLC	#2019--PILLAR	#2019--ROOT	#2019--SOGETI

Function as a Service (FaaS) Core Tenets

What if my task isn't short lived, simple, and stateless?

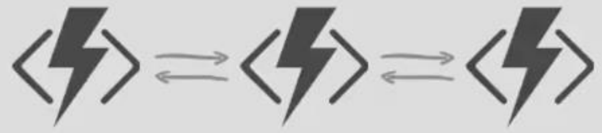
Single Responsibility

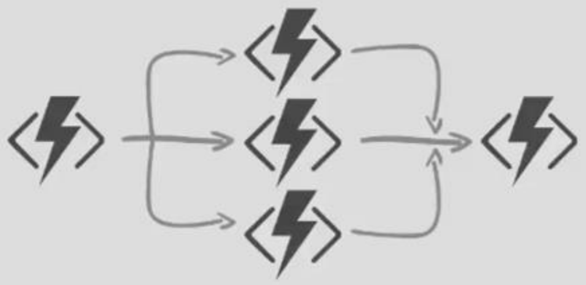
Shorted Lived

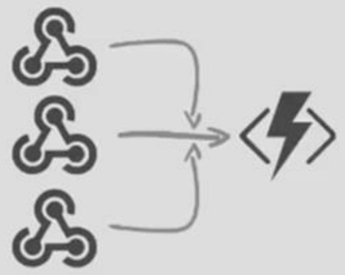
Stateless

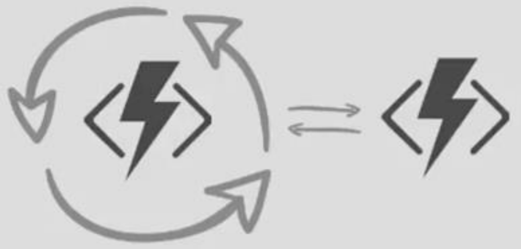
Event Driven & Scalable

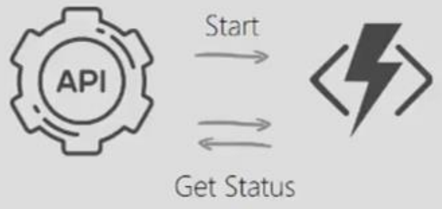
What is still hard?


Manageable Sequencing
+ Error Handling / Compensation

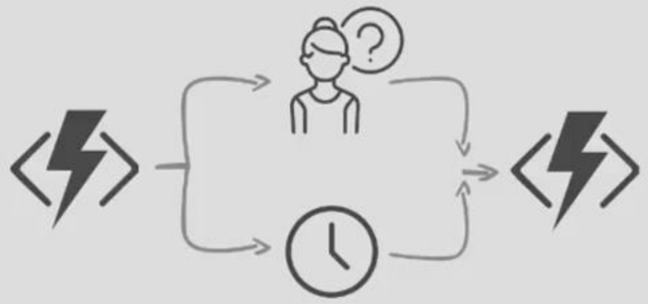

Fanning-out & Fanning-in


External Events Correlation


Flexible Automated Long-running
Process Monitoring



Http-based
Async Long-running APIs



Human Interaction

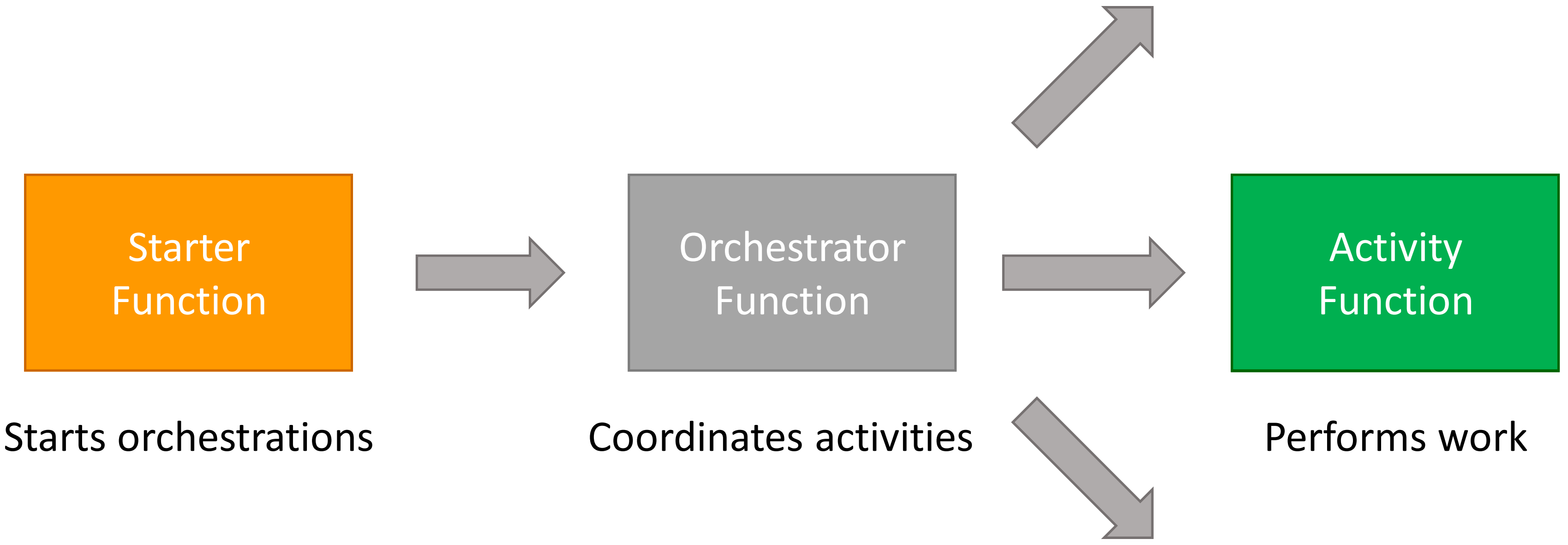
Introducing Durable Functions

- Write long-running orchestrations as a single function while maintaining local state.
- Simplify complex transactions and coordination (chaining, etc.). Easily call a Function from another Function, synchronously or asynchronously.
- All of the above using code-only. No JSON schemas. No graphical designer.

Durable Functions Benefits

- Define workflows in code
    - Easy to understand the big picture
    - Good separation of concerns
- Easy to implement complex workflows
    - Fan-out / Fan-in
    - Wait for human interaction
- Consolidate exception handling
- Check on progress or cancel workflows
- Manage state for you

Durable Functions Concepts



Starts orchestrations

Coordinates activities

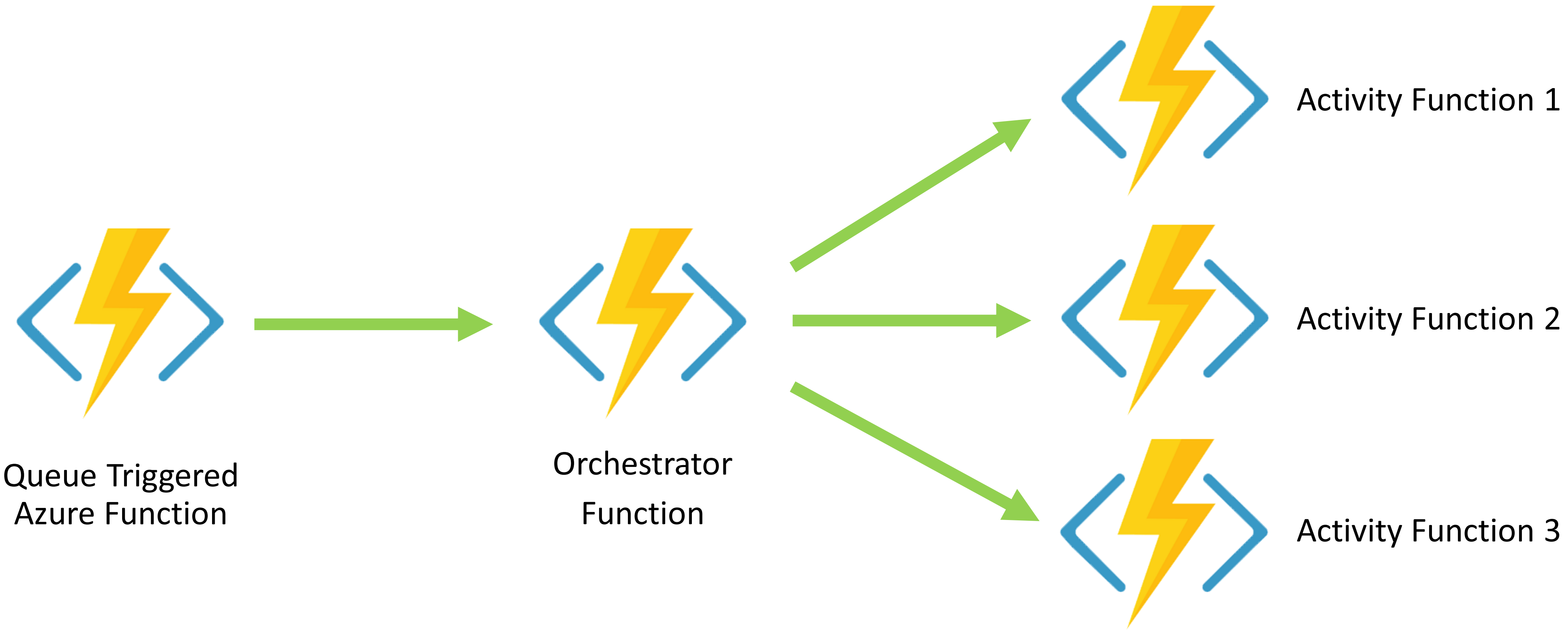
Performs work

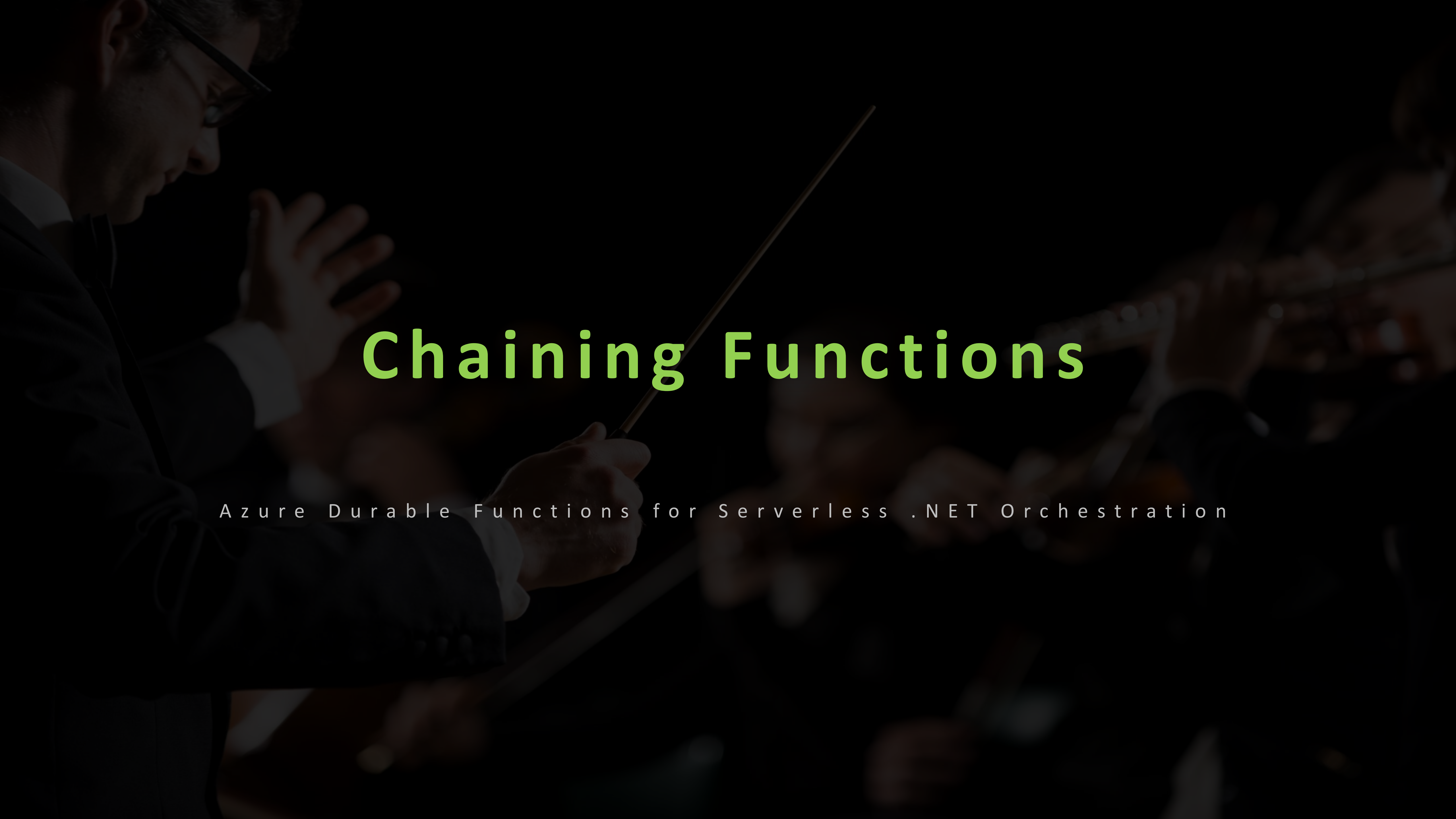
OrchestrationClient

OrchestrationContextTrigger

ActivityTrigger

Example Durable Function Workflow

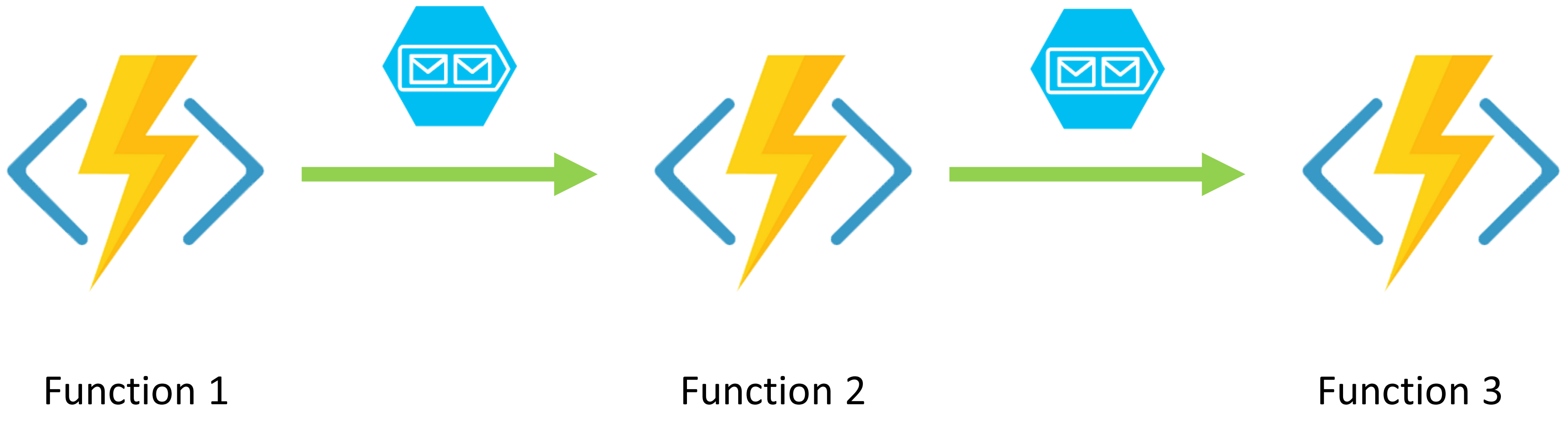




Chaining Functions

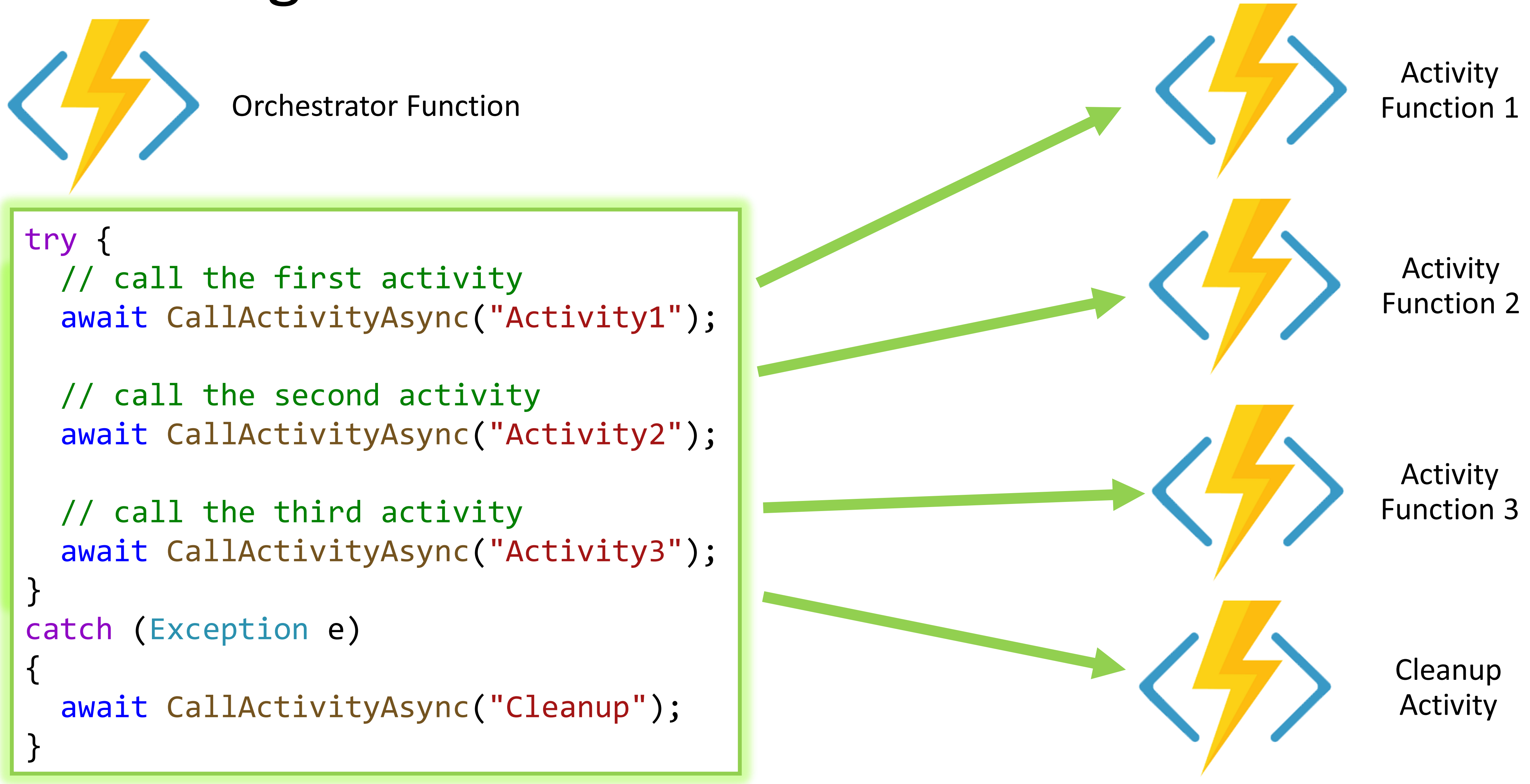
Azure Durable Functions for Serverless .NET Orchestration

Chaining Functions



No single place to see the whole workflow

Chaining with Durable Functions

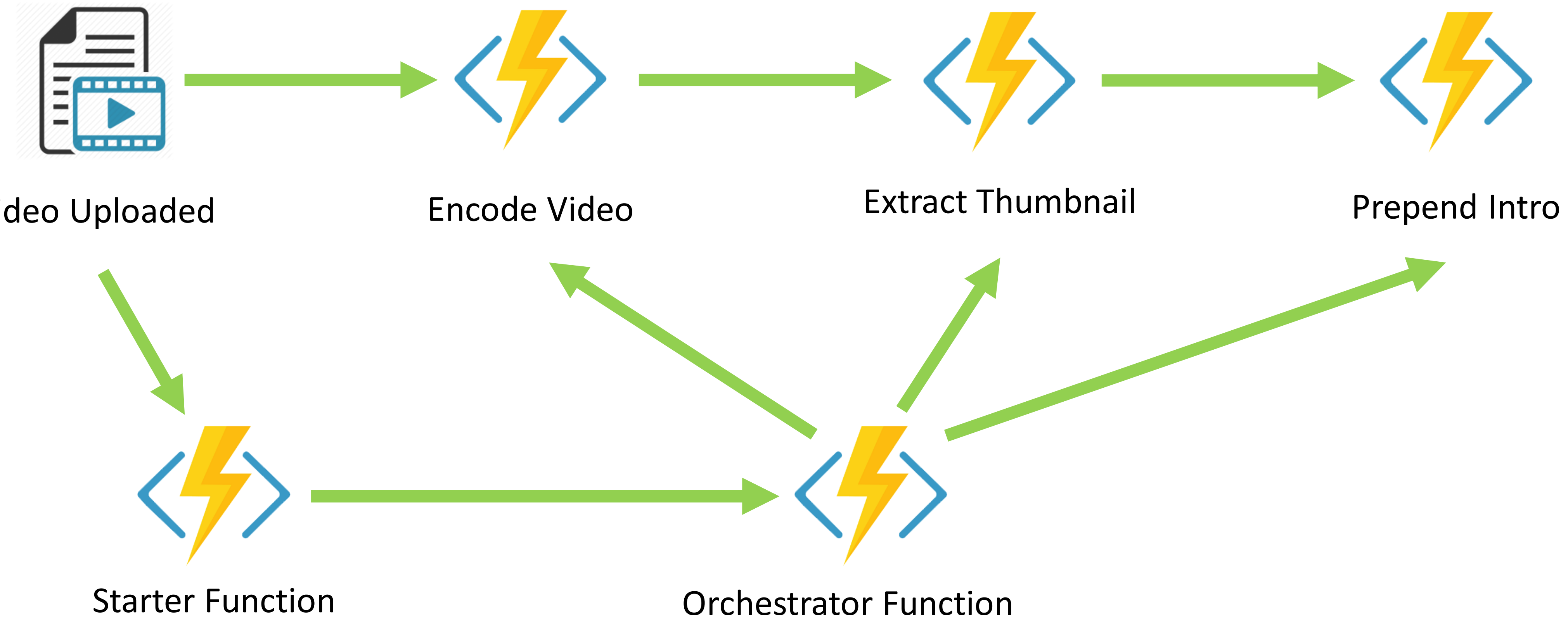


```
try {
  // call the first activity
  await CallActivityAsync("Activity1");

  // call the second activity
  await CallActivityAsync("Activity2");

  // call the third activity
  await CallActivityAsync("Activity3");
}
catch (Exception e)
{
  await CallActivityAsync("Cleanup");
}
```

Create a function chain with Durable Functions

- Create an orchestrator function
- Create activity functions
- Start a new orchestration with the OrchestrationClient binding
- Test locally

Demo Scenario – Video Publishing Workflow



First, we will create the activities

```
public static class Activities
{

 [FunctionName("EncodeVideo")]
 public static async Task<string> EncodeVideo([ActivityTrigger] string inputVideo, ILogger log)
  {



  }









}
```

```
public static class Activities
{

  [FunctionName("EncodeVideo")]
  public static async Task<string> EncodeVideo([ActivityTrigger] string inputVideo, ILogger log)
  {



  }

















}
```

```
public static class Activities
{

  [FunctionName("EncodeVideo")]
  public static async Task<string> EncodeVideo([ActivityTrigger] string inputVideo, ILogger log)
  {



  }
















}
```

```
public static class Activities
{

  [FunctionName("EncodeVideo")]
  public static async Task<string> EncodeVideo([ActivityTrigger] string inputVideo, ILogger log)
  {
    log.LogInformation($“Encoding {inputVideo}");
    await Task.Delay(5000); // Simulate doing the activity
    return “EncodedVideo.mp4";
  }















}
```

```
public static class Activities
{

  [FunctionName("EncodeVideo")]
  public static async Task<string> EncodeVideo([ActivityTrigger] string inputVideo, ILogger log)
  {
    log.LogInformation($"Encoding {inputVideo}");
    await Task.Delay(5000); // Simulate doing the activity
    return "EncodedVideo.mp4";
  }

  [FunctionName("ExtractThumbnail")]
  public static async Task<string> ExtractThumbnail([ActivityTrigger] string inputVideo, ILogger log)
  {
    log.LogInformation($"Extracting Thumbnail {inputVideo}");
    await Task.Delay(5000); // Simulate doing the activity
    return "thumbnail.png";
  }









}
```

```
public static class Activities
{

  [FunctionName("EncodeVideo")]
  public static async Task<string> EncodeVideo([ActivityTrigger] string inputVideo, ILogger log)
  {
    log.LogInformation($“Encoding {inputVideo}");
    await Task.Delay(5000); // Simulate doing the activity
    return “EncodedVideo.mp4";
  }

  [FunctionName("ExtractThumbnail")]
  public static async Task<string> ExtractThumbnail([ActivityTrigger] string inputVideo, ILogger log)
  {
    log.LogInformation($"Extracting Thumbnail {inputVideo}");
    await Task.Delay(5000); // Simulate doing the activity
    return "thumbnail.png";
  }

 [FunctionName("PrependIntro")]
  public static async Task<string> PrependIntro([ActivityTrigger] string inputVideo, ILogger log)
  {
    log.LogInformation($"Appending intro to video {inputVideo}");
    var introLocation = Environment.GetEnvironmentVariable("IntroLocation");
    await Task.Delay(5000); // Simulate doing the activity
    return “EncodedVideowithIntro.mp4";
  }

}
```

Now we create the orchestrator function

```
public class Orchestrator
{

  [FunctionName("ProcessVideoOrchestrator")]
  public static async Task<object> ProcessVideo([OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
  {















  }

}
```

```
public class Orchestrator
{

  [FunctionName("ProcessVideoOrchestrator")]
  public static async Task<object> ProcessVideo([OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
  {















  }

}
```

```
public class Orchestrator
{

  [FunctionName("ProcessVideoOrchestrator")]
  public static async Task<object> ProcessVideo([OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
  {

    var videoLocation = context.GetInput<string>();

    var encodedVideoLocation = await context.CallActivityAsync<string>("EncodeVideo", videoLocation);

    var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);

    return new
    {
      Encoded = encodedVideoLocation,
      Thumbnail = thumbnailLocation,
      WithInto = withIntroLocation
    };

  }

}
```

```
public class Orchestrator
{

  [FunctionName("ProcessVideoOrchestrator")]
  public static async Task<object> ProcessVideo([OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
  {

    var videoLocation = context.GetInput<string>();

    var encodedVideoLocation = await context.CallActivityAsync<string>("EncodeVideo", videoLocation);









  }

}
```

```
public class Orchestrator
{

  [FunctionName("ProcessVideoOrchestrator")]
  public static async Task<object> ProcessVideo([OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
  {

    var videoLocation = context.GetInput<string>();

    var encodedVideoLocation = await context.CallActivityAsync<string>("EncodeVideo", videoLocation);

    var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);

    return new





  }

}
```

```
public class Orchestrator
{

  [FunctionName("ProcessVideoOrchestrator")]
  public static async Task<object> ProcessVideo([OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
  {

    var videoLocation = context.GetInput<string>();

    var encodedVideoLocation = await context.CallActivityAsync<string>("EncodeVideo", videoLocation);

    var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);

    return new
    {
      Encoded = encodedVideoLocation,
      Thumbnail = thumbnailLocation,
      WithInto = withIntroLocation
    };

  }

}
```

Orchestrator Function Constraints

- **Must be deterministic**
    - The whole function will be “replayed”

```
public class Orchestrator
{

  [FunctionName("ProcessVideoOrchestrator")]
  public static async Task<object> ProcessVideo([OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
  {

    var videoLocation = context.GetInput<string>();

    var encodedVideoLocation = await context.CallActivityAsync<string>("EncodeVideo", videoLocation);

    var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);

    return new
    {
      Encoded = encodedVideoLocation,
      Thumbnail = thumbnailLocation,
      WithInto = withIntroLocation
    };

  }

}
```

Orchestrator Function Constraints

- **Must be deterministic**
    - The whole function will be “replayed”
- Things not to do
    - Use current date time
    - Generate random numbers or GUIDs
    - Access data stores (i.e. database, configuration)
- Things to do
    - Use DurableOrchestrationContext.CurrentUtcDateTime
    - Pass configuration into your orchestrator function
    - Retrieve data in activity functions

Orchestrator Function Constraints

- **Must be deterministic**
- **Must be non-blocking**
    - No I/O to disk to network
    - No Thread.Sleep
- **Do not initiate async operations**
    - Except on DurableOrchestrationContext API
    - No Task.Run, Task.Delay, HttpClient.SendAsync
- **Do not create infinite loops**
    - Event history needs to be replayed
    - ContinueAsNew should be used instaed

Logging in Orchestrator Functions

- **Use the built-in Ilogger**
- **Log messages get written on every replay**
    - Avoid with DurableOrchestrationContext.IsReplaying

```
if (!context.IsReplaying)
  log.LogInformation("About to call encode video activity");
```

Finally, we create the starter function

```
public static class Starter
{

  [FunctionName("ProcessVideoStarter")]
  public static async Task<HttpResponseMessage> Run(
        [HttpTrigger(AuthorizationLevel.Anonymous, "get")] HttpRequestMessage req,
        [OrchestrationClient]DurableOrchestrationClient starter,
        ILogger log)
  {










  }

}
```

```
public static class Starter
{

  [FunctionName("ProcessVideoStarter")]
  public static async Task<HttpResponseMessage> Run(
        [HttpTrigger(AuthorizationLevel.Anonymous, "get")] HttpRequestMessage req,
        [OrchestrationClient]DurableOrchestrationClient starter,
        ILogger log)
  {









  }

}
```

```
public static class Starter
{

    [FunctionName("ProcessVideoStarter")]
    public static async Task<HttpResponseMessage> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get")] HttpRequestMessage req,
            [OrchestrationClient]DurableOrchestrationClient starter,
            ILogger log)
    {










    }

}
```

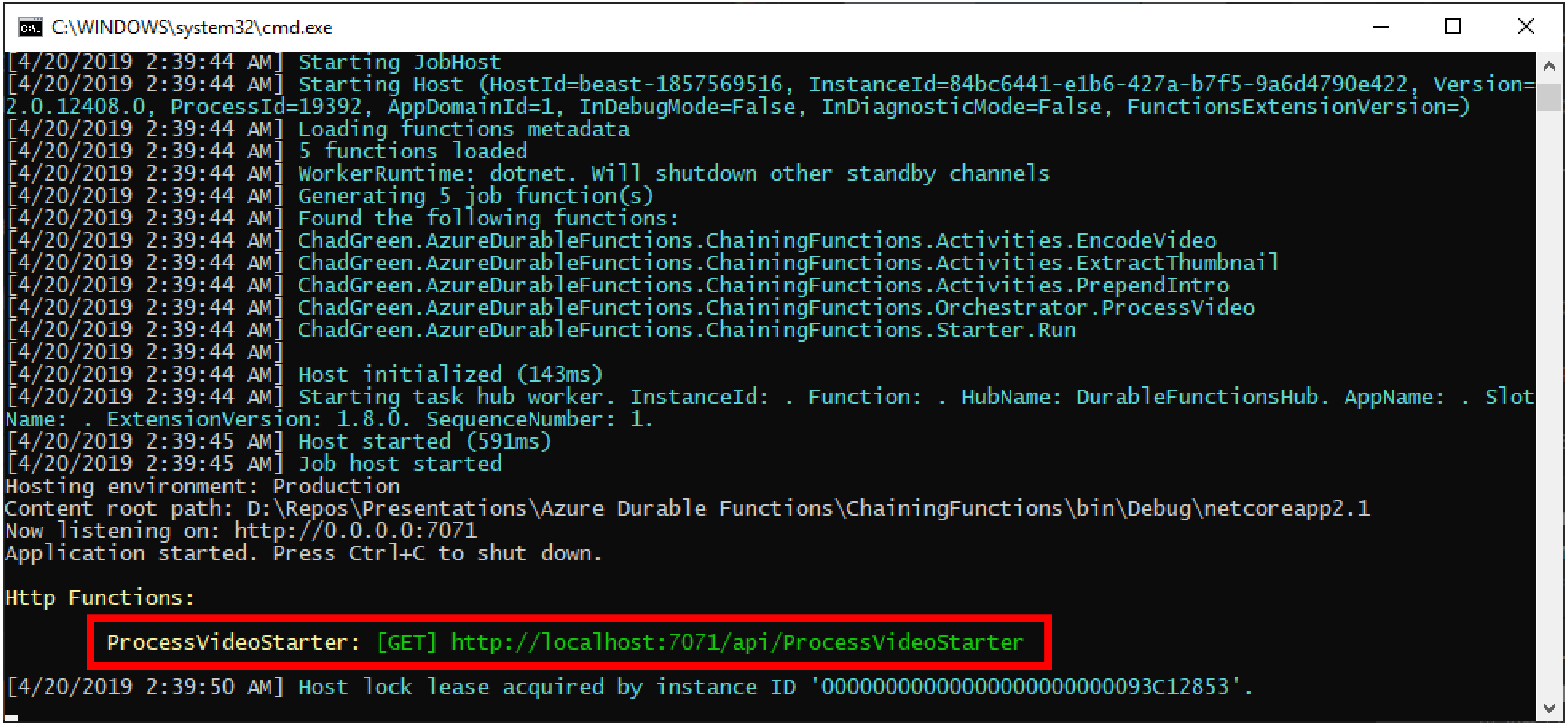
```
public static class Starter
{

  [FunctionName("ProcessVideoStarter")]
  public static async Task<HttpResponseMessage> Run(
        [HttpTrigger(AuthorizationLevel.Anonymous, "get")] HttpRequestMessage req,
        [OrchestrationClient]DurableOrchestrationClient starter,
        ILogger log)
  {

    string video = req.RequestUri.Query;









  }

}
```

```csharp
public static class Starter
{

  [FunctionName("ProcessVideoStarter")]
  public static async Task<HttpResponseMessage> Run(
        [HttpTrigger(AuthorizationLevel.Anonymous, "get")] HttpRequestMessage req,
        [OrchestrationClient]DurableOrchestrationClient starter,
        ILogger log)
  {

    string video = req.RequestUri.Query;

    var orchestrationId = await starter.StartNewAsync("ProcessVideoOrchestrator", video);





  }

}
```

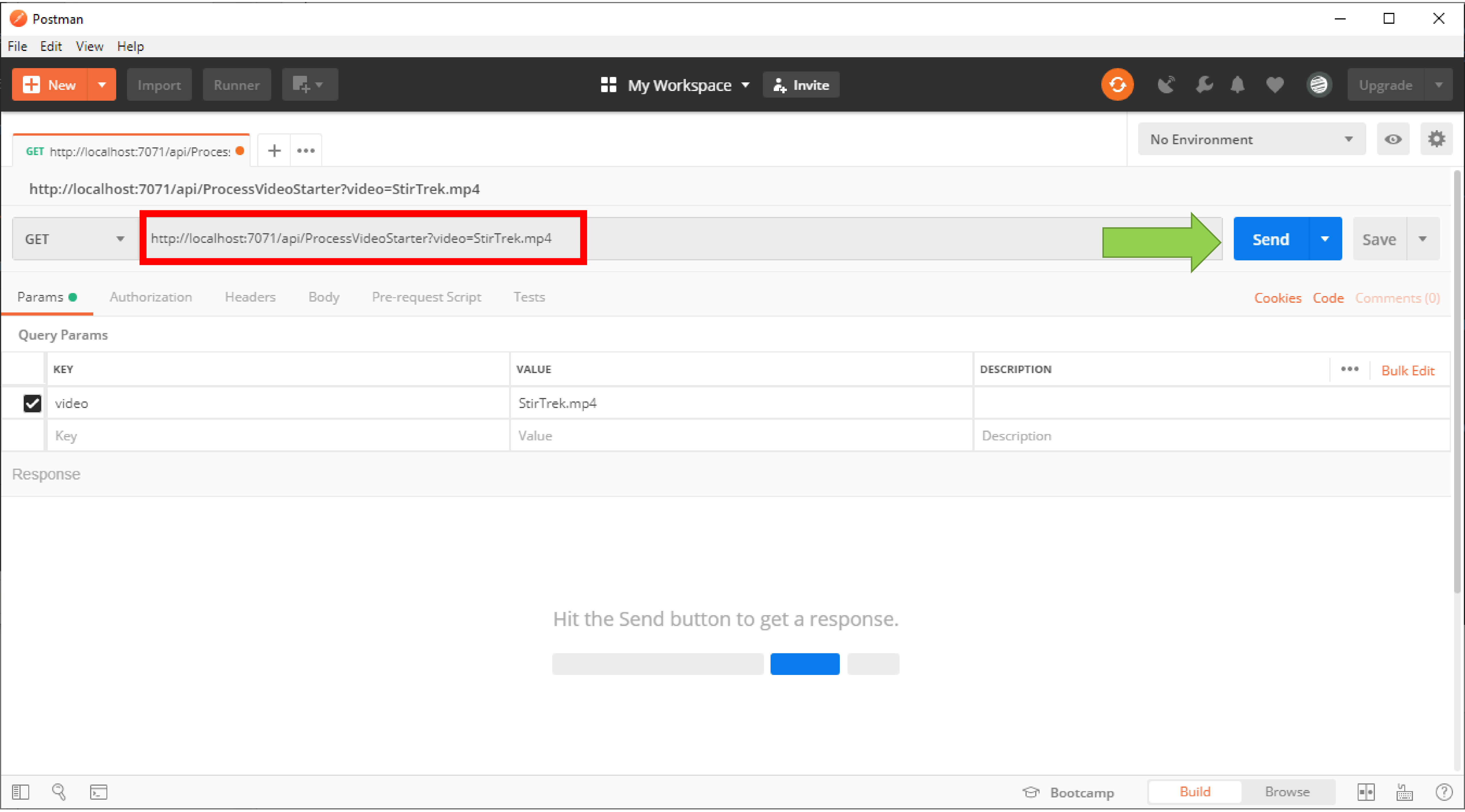
```
public static class Starter
{

  [FunctionName("ProcessVideoStarter")]
  public static async Task<HttpResponseMessage> Run(
        [HttpTrigger(AuthorizationLevel.Anonymous, "get")] HttpRequestMessage req,
        [OrchestrationClient]DurableOrchestrationClient starter,
        ILogger log)
  {

    string video = req.RequestUri.Query;

    var orchestrationId = await starter.StartNewAsync("ProcessVideoOrchestrator", video);

    log.LogInformation($"Started orchestration with ID = '{orchestrationId}'.");



  }

}
```

```
public static class Starter {

  [FunctionName("ProcessVideoStarter")]
  public static async Task<HttpResponseMessage> Run(
        [HttpTrigger(AuthorizationLevel.Anonymous, "get")] HttpRequestMessage req,
        [OrchestrationClient]DurableOrchestrationClient starter,
        ILogger log)
  {

    string video = req.RequestUri.Query;

    var orchestrationId = await starter.StartNewAsync("ProcessVideoOrchestrator", video);

    log.LogInformation($"Started orchestration with ID = '{orchestrationId}'.");

    return starter.CreateCheckStatusResponse(req, orchestrationId);

  }

}
```

Now that see this thing work



```
[4/20/2019 2:39:44 AM] Starting JobHost
[4/20/2019 2:39:44 AM] Starting Host (HostId=beast-1857569516, InstanceId=84bc6441-e1b6-427a-b7f5-9a6d4790e422, Version=
2.0.12408.0, ProcessId=19392, AppDomainId=1, InDebugMode=False, InDiagnosticMode=False, FunctionsExtensionVersion=)
[4/20/2019 2:39:44 AM] Loading functions metadata
[4/20/2019 2:39:44 AM] 5 functions loaded
[4/20/2019 2:39:44 AM] WorkerRuntime: dotnet. Will shutdown other standby channels
[4/20/2019 2:39:44 AM] Generating 5 job function(s)
[4/20/2019 2:39:44 AM] Found the following functions:
[4/20/2019 2:39:44 AM] ChadGreen.AzureDurableFunctions.ChainingFunctions.Activities.EncodeVideo
[4/20/2019 2:39:44 AM] ChadGreen.AzureDurableFunctions.ChainingFunctions.Activities.ExtractThumbnail
[4/20/2019 2:39:44 AM] ChadGreen.AzureDurableFunctions.ChainingFunctions.Activities.PrependIntro
[4/20/2019 2:39:44 AM] ChadGreen.AzureDurableFunctions.ChainingFunctions.Orchestrator.ProcessVideo
[4/20/2019 2:39:44 AM] ChadGreen.AzureDurableFunctions.ChainingFunctions.Starter.Run
[4/20/2019 2:39:44 AM]
[4/20/2019 2:39:44 AM] Host initialized (143ms)
[4/20/2019 2:39:44 AM] Starting task hub worker. InstanceId: . Function: . HubName: DurableFunctionsHub. AppName: . Slot
Name: . ExtensionVersion: 1.8.0. SequenceNumber: 1.
[4/20/2019 2:39:45 AM] Host started (591ms)
[4/20/2019 2:39:45 AM] Job host started
Hosting environment: Production
Content root path: D:\Repos\Presentations\Azure Durable Functions\ChainingFunctions\bin\Debug\netcoreapp2.1
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.

Http Functions:

        ProcessVideoStarter: [GET] http://localhost:7071/api/ProcessVideoStarter

[4/20/2019 2:39:50 AM] Host lock lease acquired by instance ID '000000000000000000000000093C12853'.
```



Postman

File Edit View Help

GET http://localhost:7071/api/Proces:

http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4

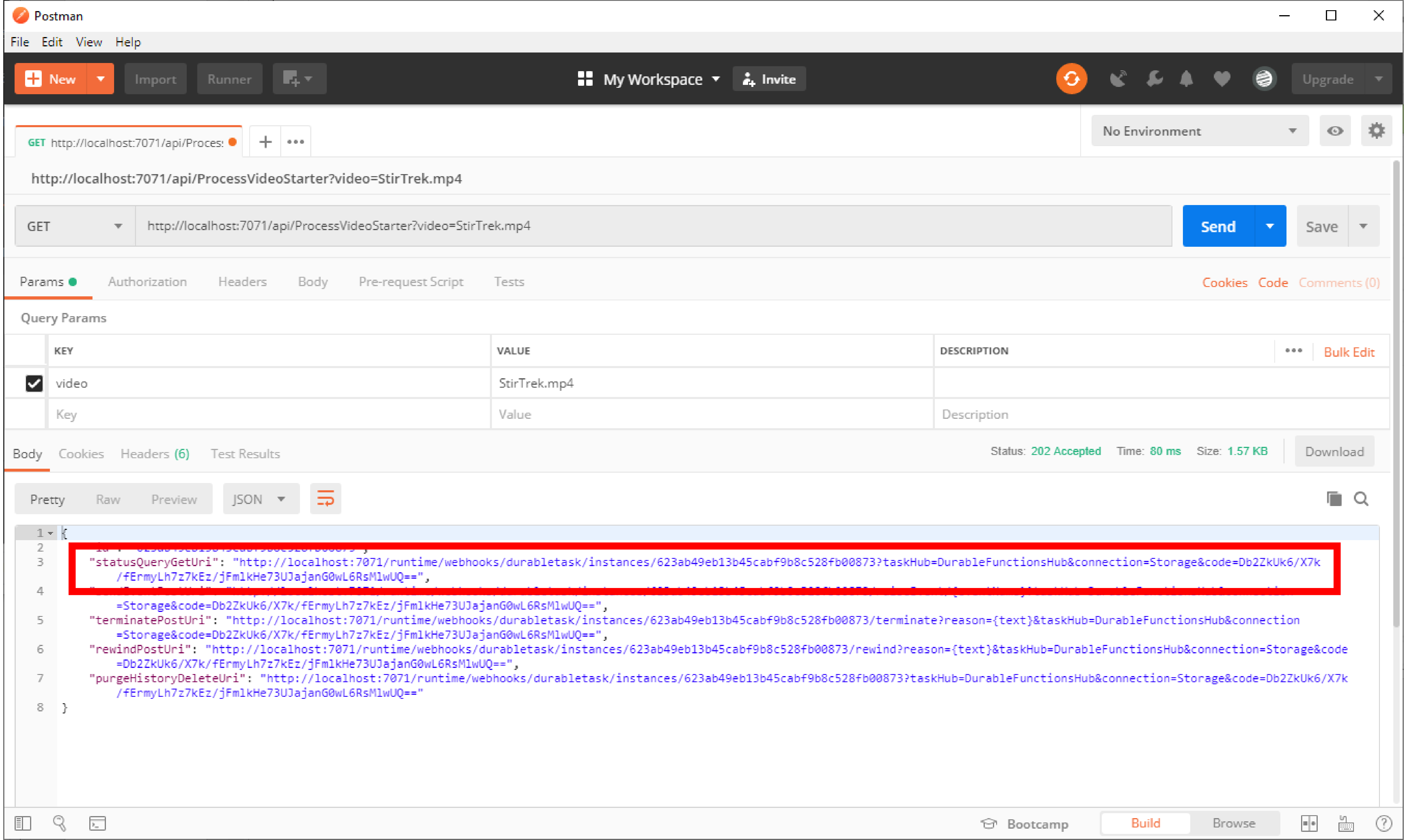
```
GET  http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4
```

Query Params

	KEY	VALUE	DESCRIPTION
✓	video	StirTrek.mp4	
	Key	Value	Description

Response

Hit the Send button to get a response.



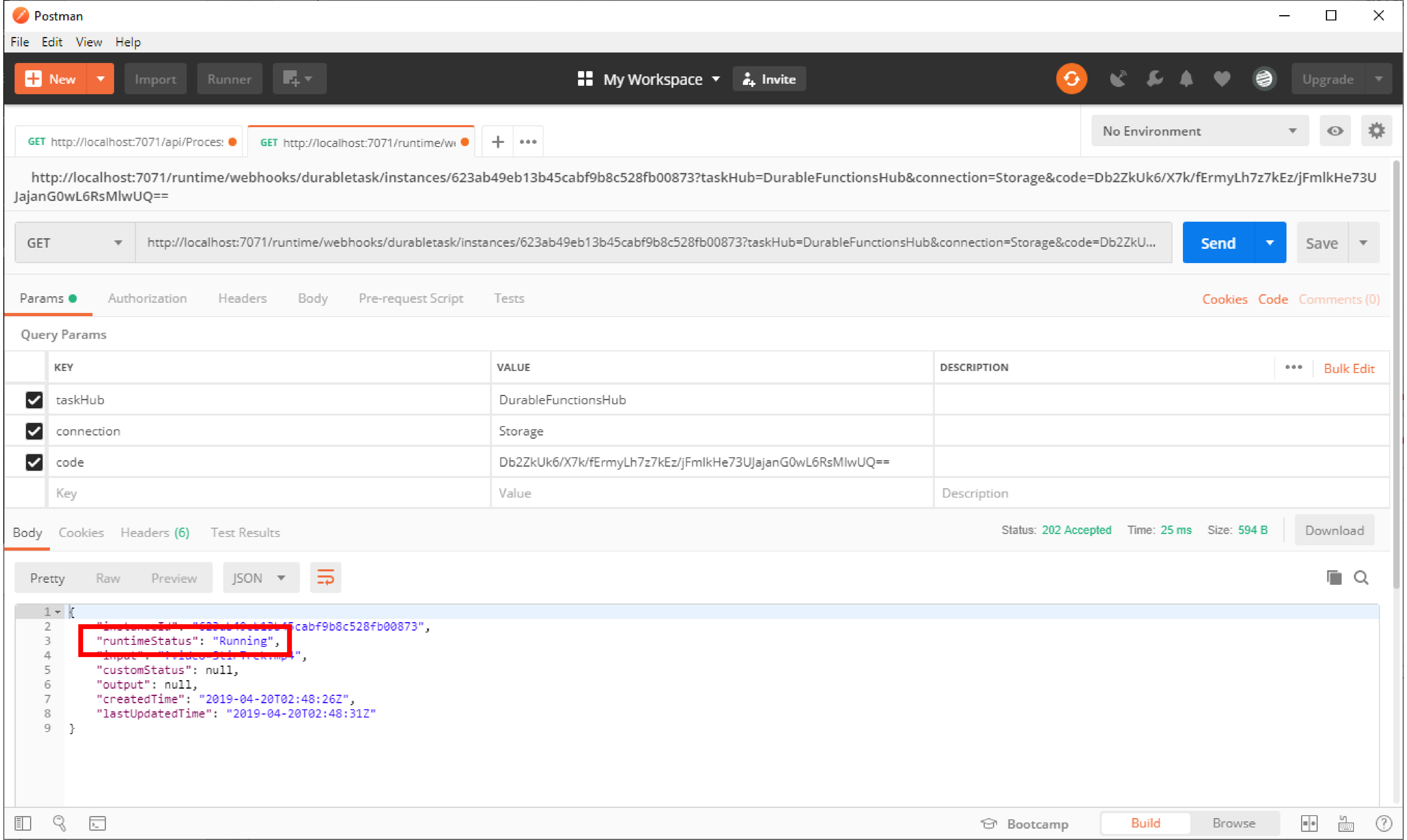
http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4

Query Params

	KEY	VALUE	DESCRIPTION
✓	video	StirTrek.mp4	
	Key	Value	Description

Status: 202 Accepted Time: 80 ms Size: 1.57 KB

```
{
    [illegible]
    "statusQueryGetUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/623ab49eb13b45cabf9b8c528fb00873?taskHub=DurableFunctionsHub&connection=Storage&code=Db2ZkUk6/X7k
        /fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ==",
    [illegible]
        =Storage&code=Db2ZkUk6/X7k/fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ==",
    "terminatePostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/623ab49eb13b45cabf9b8c528fb00873/terminate?reason={text}&taskHub=DurableFunctionsHub&connection
        =Storage&code=Db2ZkUk6/X7k/fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ==",
    "rewindPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/623ab49eb13b45cabf9b8c528fb00873/rewind?reason={text}&taskHub=DurableFunctionsHub&connection=Storage&code
        =Db2ZkUk6/X7k/fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ==",
    "purgeHistoryDeleteUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/623ab49eb13b45cabf9b8c528fb00873?taskHub=DurableFunctionsHub&connection=Storage&code=Db2ZkUk6/X7k
        /fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ=="
}
```


http://localhost:7071/runtime/webhooks/durabletask/instances/623ab49eb13b45cabf9b8c528fb00873?taskHub=DurableFunctionsHub&connection=Storage&code=Db2ZkUk6/X7k/fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ==

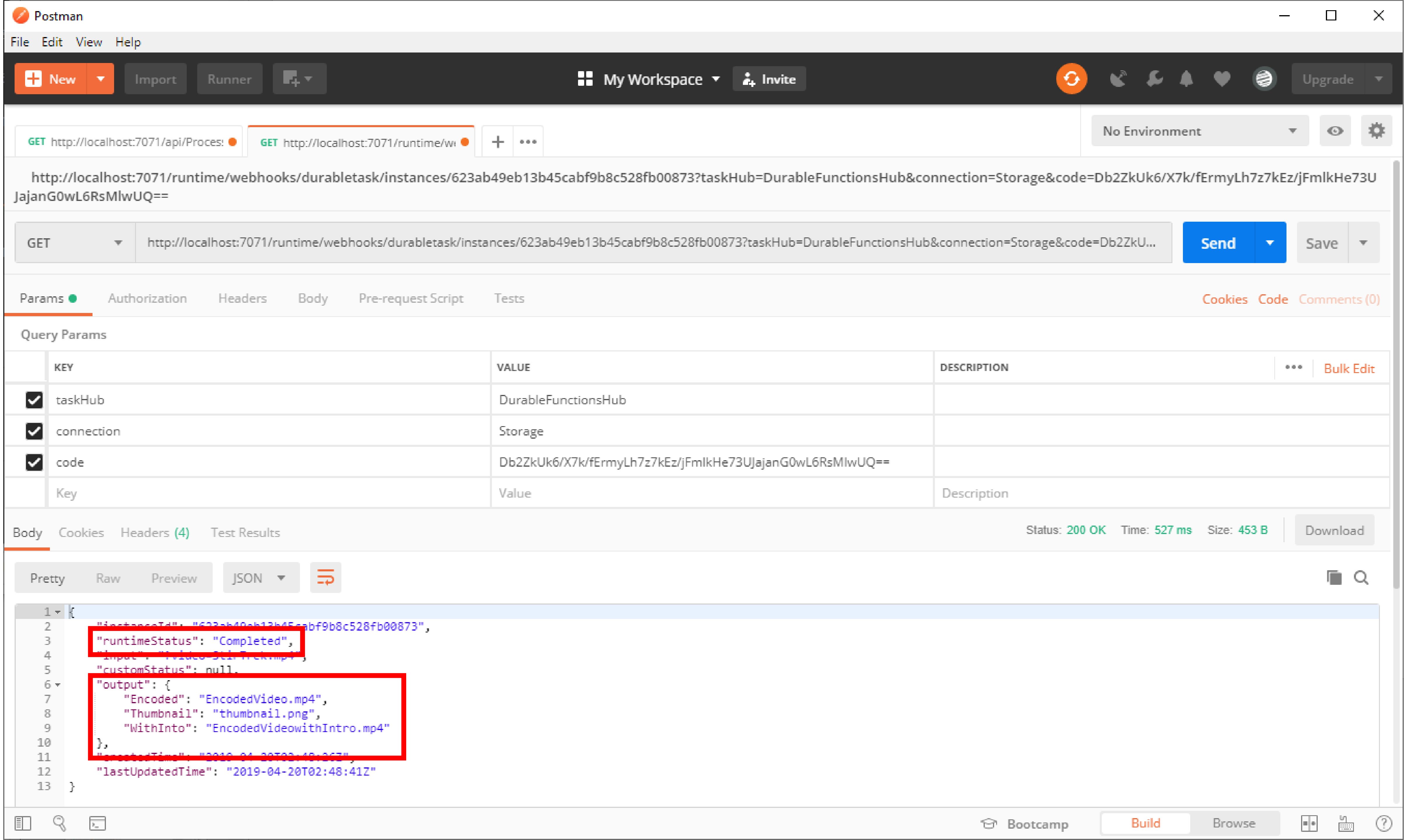
GET http://localhost:7071/runtime/webhooks/durabletask/instances/623ab49eb13b45cabf9b8c528fb00873?taskHub=DurableFunctionsHub&connection=Storage&code=Db2ZkU...

Query Params

	KEY	VALUE	DESCRIPTION
✓	taskHub	DurableFunctionsHub	
✓	connection	Storage	
✓	code	Db2ZkUk6/X7k/fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ==	
	Key	Value	Description

Status: 202 Accepted Time: 25 ms Size: 594 B

```
{
    "instanceId": "623ab49eb13b45cabf9b8c528fb00873",
    "runtimeStatus": "Running",
    "input": "Video-StarTrek.mp4",
    "customStatus": null,
    "output": null,
    "createdTime": "2019-04-20T02:48:26Z",
    "lastUpdatedTime": "2019-04-20T02:48:31Z"
}
```



http://localhost:7071/runtime/webhooks/durabletask/instances/623ab49eb13b45cabf9b8c528fb00873?taskHub=DurableFunctionsHub&connection=Storage&code=Db2ZkUk6/X7k/fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ==

GET http://localhost:7071/runtime/webhooks/durabletask/instances/623ab49eb13b45cabf9b8c528fb00873?taskHub=DurableFunctionsHub&connection=Storage&code=Db2ZkU...

Query Params

	KEY	VALUE	DESCRIPTION
✓	taskHub	DurableFunctionsHub	
✓	connection	Storage	
✓	code	Db2ZkUk6/X7k/fErmyLh7z7kEz/jFmlkHe73UJajanG0wL6RsMlwUQ==	
	Key	Value	Description

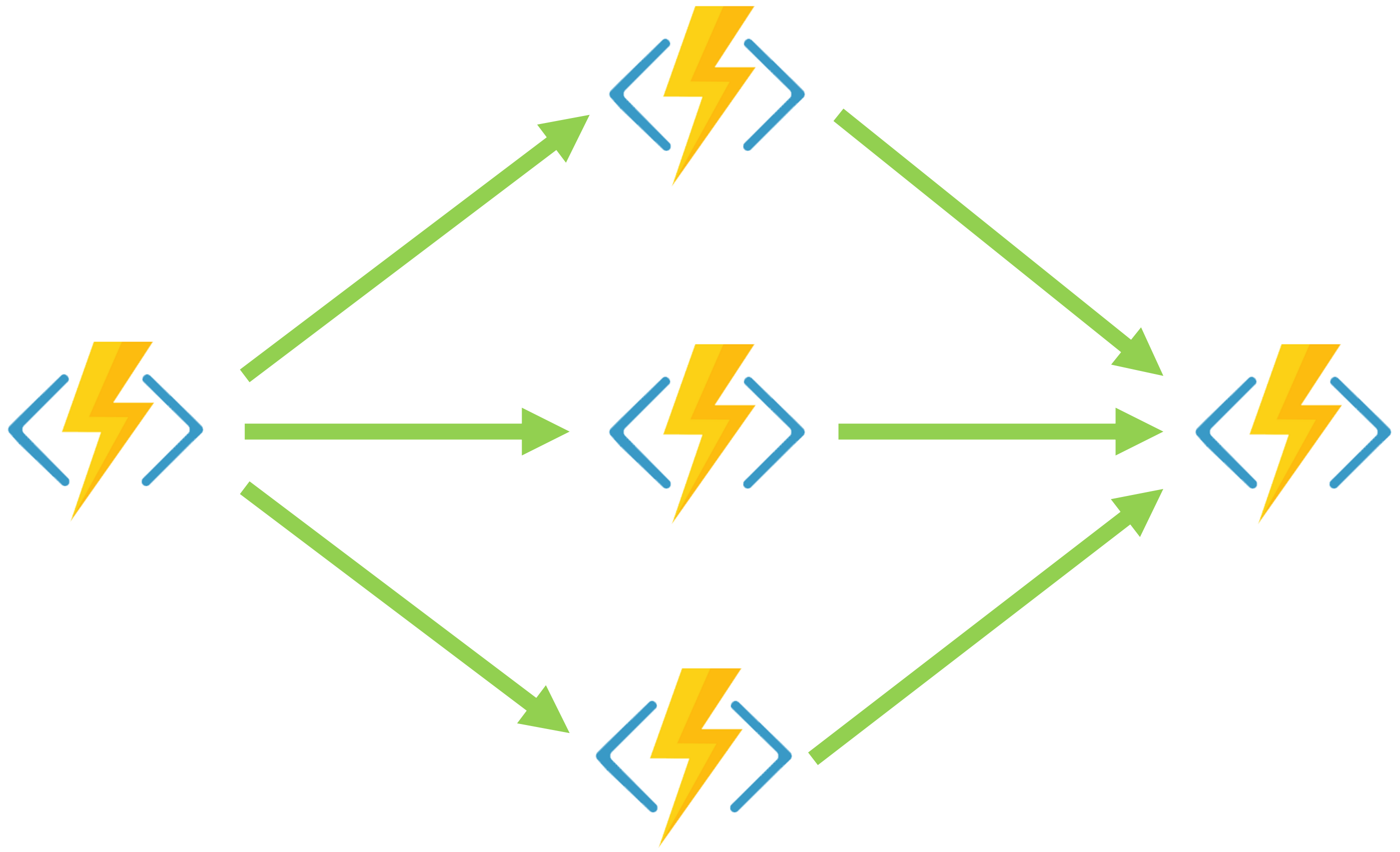
Status: 200 OK Time: 527 ms Size: 453 B

```
{
    "instanceId": "623ab49eb13b45cabf9b8c528fb00873",
    "runtimeStatus": "Completed",
    "input": "Video-StarTrek.mp4",
    "customStatus": null,
    "output": {
        "Encoded": "EncodedVideo.mp4",
        "Thumbnail": "thumbnail.png",
        "WithInto": "EncodedVideowithIntro.mp4"
    },
    "createdTime": "2019-04-20T02:48:26Z",
    "lastUpdatedTime": "2019-04-20T02:48:41Z"
}
```

Fan-Out / Fan-In Pattern

Azure Durable Functions for Serverless .NET Orchestration

Fan-Out / Fan-In Pattern



Extract Thumbnail Encode Video Prepend Intro

Implement the fan-out/fan-in pattern

- Encoding activity takes bitrate input
- Orchestrator function calls activity with multiple bitrates
- Orchestrator waits for all encoding activities to finish
- Call an orchestrator from another orchestration

```csharp
public class VideoFileInfo
{
  public string Location { get; set; }
  public int BitRate { get; set; }
}
```

```csharp
[FunctionName("EncodeVideo")]
public static async Task<string> EncodeVideo([ActivityTrigger] string inputVideo, ILogger log)
{
  log.LogInformation($"Encoding {inputVideo}");
  await Task.Delay(5000); // Simulate doing the activity
  return "EncodedVideo.mp4";
}
```

```csharp
[FunctionName("EncodeVideo")]
public static async Task<VideoFileInfo> EncodeVideo([ActivityTrigger] VideoFileInfo inputVideo, ILogger log)
{

  log.LogInformation($"Encoding {inputVideo.Location} to {inputVideo.BitRate}");

  await Task.Delay(5000); // Simulate doing the activity

  var encodedLocation = $"{Path.GetFileNameWithoutExtension(inputVideo.Location)}-{inputVideo.BitRate}kps.mp4";

  return new VideoFileInfo { Location = encodedLocation, BitRate = inputVideo.BitRate };

}
```

```
[FunctionName("GetEncodeBitrates")]
public static int[] GetTranscodeBitrates([ActivityTrigger] object input, ILogger log)
{
  return Environment.GetEnvironmentVariable("EncodingBitrates")
                  .Split(',’)
                  .Select(int.Parse)
                  .ToArray();
}
```

```
[FunctionName("EncodeVideoOrchestrator")]
public static async Task<VideoFileInfo[]> EncodeVideoOrchestrator(
                [OrchestrationTrigger] DurableOrchestrationContext context,
                ILogger log)
{

















}
```

```
[FunctionName("EncodeVideoOrchestrator")]
public static async Task<VideoFileInfo[]> EncodeVideoOrchestrator(
              [OrchestrationTrigger] DurableOrchestrationContext context,
              ILogger log)
{
  var videoLocation = context.GetInput<string>();















}
```

```csharp
[FunctionName("EncodeVideoOrchestrator")]
public static async Task<VideoFileInfo[]> EncodeVideoOrchestrator(
              [OrchestrationTrigger] DurableOrchestrationContext context,
              ILogger log)
{
  var videoLocation = context.GetInput<string>();
  var bitRates = await context.CallActivityAsync<int[]>("GetEncodeBitrates", null);


















}
```

```
[FunctionName("EncodeVideoOrchestrator")]
public static async Task<VideoFileInfo[]> EncodeVideoOrchestrator(
              [OrchestrationTrigger] DurableOrchestrationContext context,
              ILogger log)
{
  var videoLocation = context.GetInput<string>();
  var bitRates = await context.CallActivityAsync<int[]>("GetEncodeBitrates", null);
  var encodeTasks = new List<Task<VideoFileInfo>>();
















}
```

```
[FunctionName("EncodeVideoOrchestrator")]
public static async Task<VideoFileInfo[]> EncodeVideoOrchestrator(
              [OrchestrationTrigger] DurableOrchestrationContext context,
              ILogger log)
{
  var videoLocation = context.GetInput<string>();
  var bitRates = await context.CallActivityAsync<int[]>("GetEncodeBitrates", null);
  var encodeTasks = new List<Task<VideoFileInfo>>();

  foreach (var bitRate in bitRates)
  {
    var info = new VideoFileInfo()
    {
      Location = videoLocation,
      BitRate = bitrate
    };
    var task = context.CallActivityAsync<VideoFileInfo>("EncodeVideo", info);
    encodeTasks.Add(task);
  }



}
```

```
[FunctionName("EncodeVideoOrchestrator")]
public static async Task<VideoFileInfo[]> EncodeVideoOrchestrator(
              [OrchestrationTrigger] DurableOrchestrationContext context,
              ILogger log)
{
  var videoLocation = context.GetInput<string>();
  var bitRates = await context.CallActivityAsync<int[]>("GetEncodeBitrates", null);
  var encodeTasks = new List<Task<VideoFileInfo>>();

  foreach (var bitRate in bitRates)
  {
    var info = new VideoFileInfo()
    {
      Location = videoLocation,
      BitRate = bitrate
    };
    var task = context.CallActivityAsync<VideoFileInfo>("EncodeVideo", info);
    encodeTasks.Add(task);
  }

  var encodeResults = await Task.WhenAll(encodeTasks);

}
```

```
[FunctionName("EncodeVideoOrchestrator")]
public static async Task<VideoFileInfo[]> EncodeVideoOrchestrator(
              [OrchestrationTrigger] DurableOrchestrationContext context,
              ILogger log)
{
  var videoLocation = context.GetInput<string>();
  var bitRates = await context.CallActivityAsync<int[]>("GetEncodeBitrates", null);
  var encodeTasks = new List<Task<VideoFileInfo>>();

  foreach (var bitRate in bitRates)
  {
    var info = new VideoFileInfo()
    {
      Location = videoLocation,
      BitRate = bitrate
    };
    var task = context.CallActivityAsync<VideoFileInfo>("EncodeVideo", info);
    encodeTasks.Add(task);
  }

  var encodeResults = await Task.WhenAll(encodeTasks);
  return encodeResults;
}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");
  var encodedVideoLocation = await context.CallActivityAsync<string>("EncodeVideo", videoLocation);

  if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
  var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

  if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
  var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");
  var encodedVideoLocation = await context.CallActivityAsync<string>("EncodeVideo", videoLocation);

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
  var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

  if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
  var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
    var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);
  }
  catch (Exception e)
  {
  }

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
    var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);
  }
  catch (Exception e)
  {
  }

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);

    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();

    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
    var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);
  }
  catch (Exception e)
  {
  }

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);

    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();

    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    var thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
    var withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);
  }
  catch (Exception e)
  {
  }

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

```
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);

    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();

    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);


  }
  catch (Exception e)
  {
  }

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);

    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();

    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);

  }
  catch (Exception e)
  {
    if (!context.IsReplaying) log.LogInformation($"Caught an error from an activity: {e.Message}");
  }

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);

    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();

    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);

  }
  catch (Exception e)
  {
    if (!context.IsReplaying) log.LogInformation($"Caught an error from an activity: {e.Message}");
    await context.CallActivityAsync<string>("Cleanup", new[] { encodedLocation, thumbnailLocation, withIntroLocation });
  }

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };

}
```

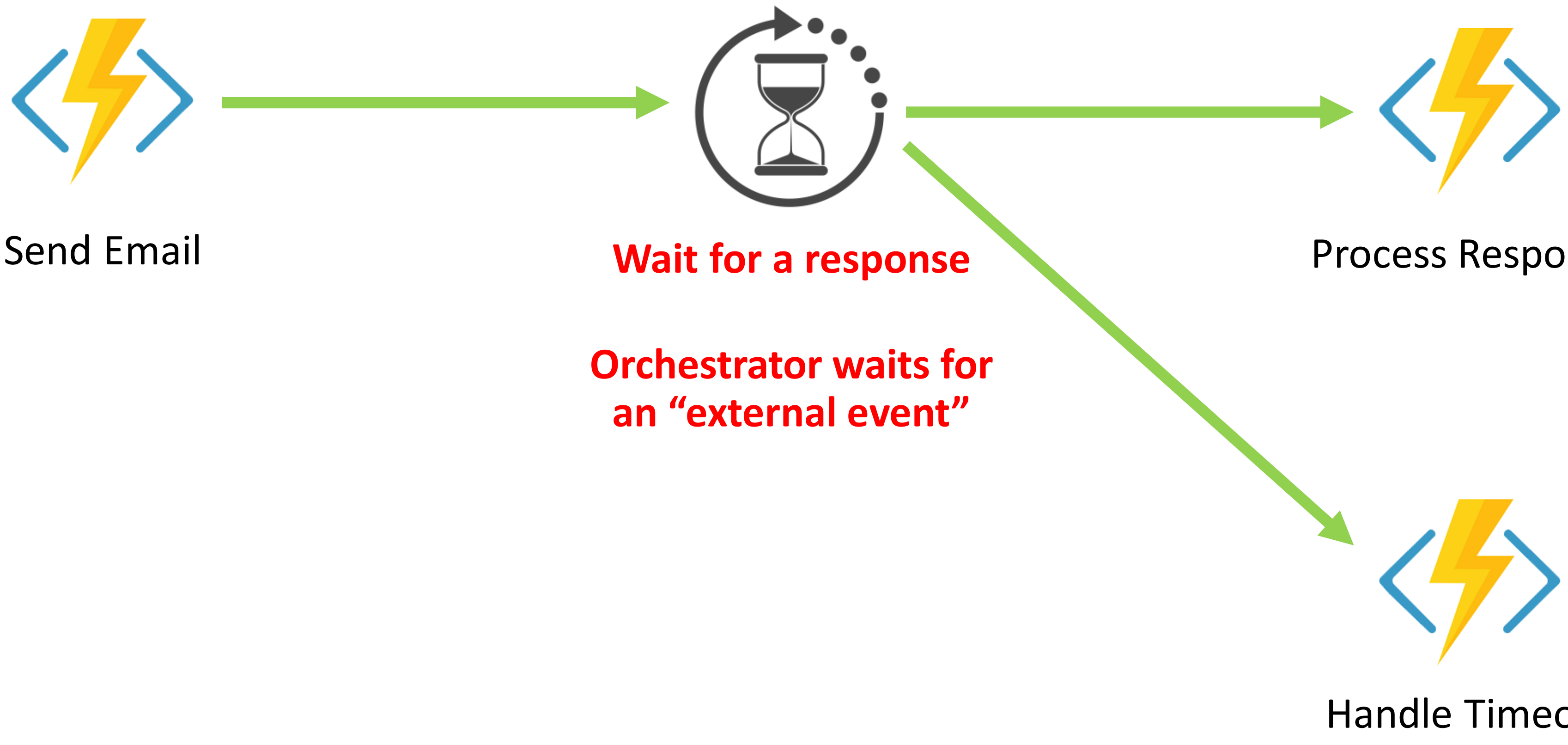
```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context, ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);

    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();

    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedVideoLocation);

    if (!context.IsReplaying) log.LogInformation("About to call prepend intro");
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedVideoLocation);
   }
  catch (Exception e)
  {
    if (!context.IsReplaying) log.LogInformation($"Caught an error from an activity: {e.Message}");
    await context.CallActivityAsync<string>("Cleanup", new[] { encodedLocation, thumbnailLocation, withIntroLocation });
    return new
    {
      Error = "Failed to process uploaded video",
      Message = e.Message
    }
  }

  return new
  {
    Encoded = encodedVideoLocation,
    Thumbnail = thumbnailLocation,
    WithInto = withIntroLocation
  };
}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context,  ILogger log)
{

  var videoLocation = context.GetInput<string>();

  if (!context.IsReplaying) log.LogInformation("About to call encode video activity");

  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);
    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();

    if (!context.IsReplaying) log.LogInformation("About to call extract thumbnail");
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedLocation);

    if (!context.IsReplaying) log.LogInformation("PrependIntro", encodedLocation);
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedLocation);

  }
  catch (Exception e)
  {
    if (!context.IsReplaying) log.LogInformation($"Caught an error from an activity: {e.Message}");
    await context.CallActivityAsync<string>("Cleanup", new[] { encodedLocation, thumbnailLocation, withIntroLocation });
    return new
    {
      Error = "Failed to process uploaded video",
      Message = e.Message
    };
  }

  return new
  {
    Encoded = encodedLocation,
    Thumbnail = thumbnailLocation,
    WithIntro = withIntroLocation
  };

}
```

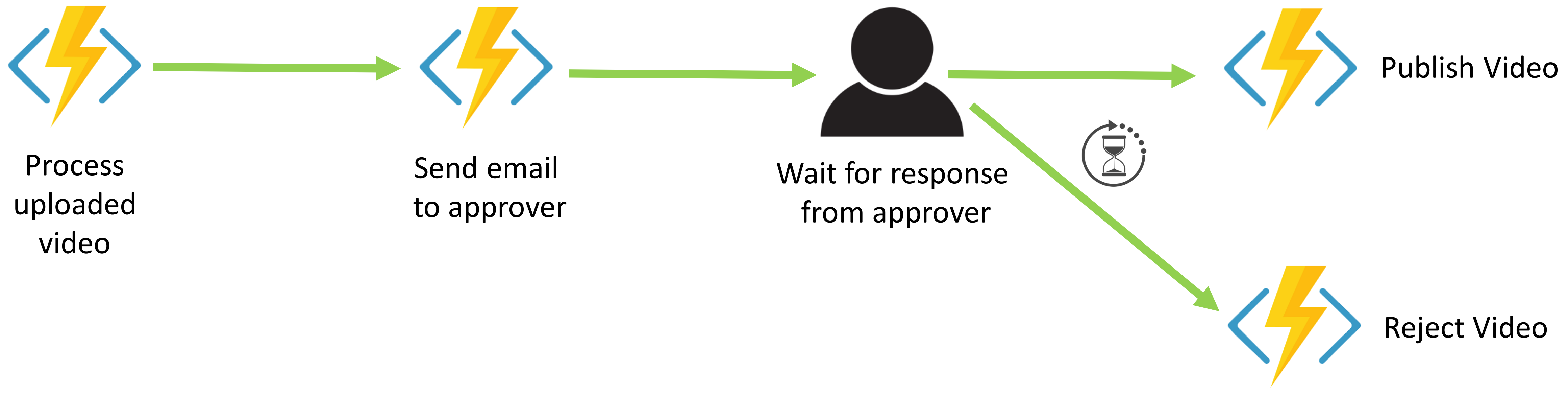
Waiting for Human Interaction

Azure Durable Functions for Serverless .NET Orchestration

Waiting for External Events



Demo Scenario



Waiting for an “external event”

- Send email activity function
- WaitForExternalEvent

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context,  ILogger log)
{

  var videoLocation = context.GetInput<string>();
  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);
    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedLocation);
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedLocation);

    await context.CallActivityAsync("SendApprovalRequestEmail", withIntroLocation);

  }
  catch (Exception e)
  {
    await context.CallActivityAsync<string>("Cleanup", new[] { encodedLocation, thumbnailLocation, withIntroLocation });
    return new { Error = "Failed to process uploaded video", Message = e.Message };
  }

  return new
  {
    Encoded = encodedLocation,
    Thumbnail = thumbnailLocation,
    WithIntro = withIntroLocation
  };

}
```

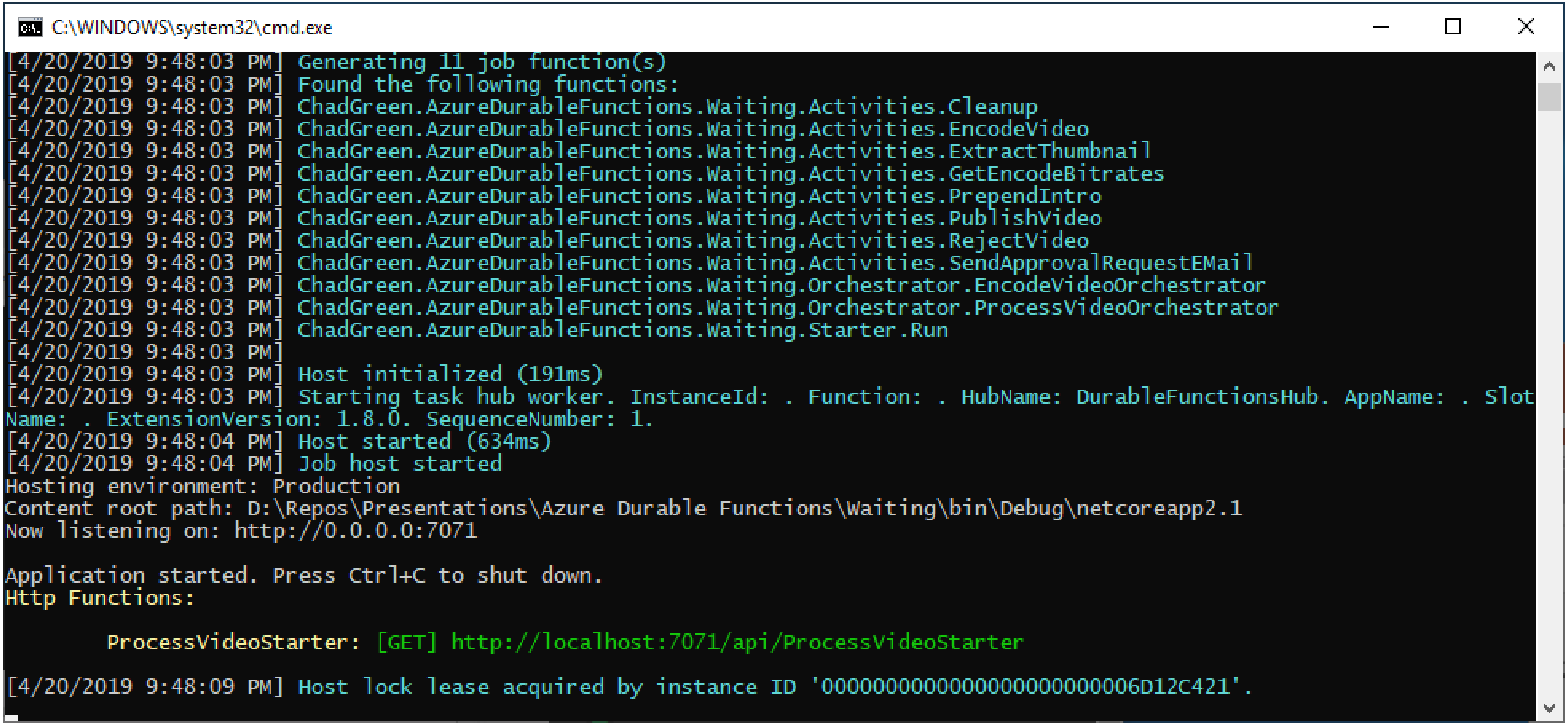
```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context,  ILogger log)
{

  var videoLocation = context.GetInput<string>();
  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;
  string approvalResult = "Unknown";

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);
    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedLocation);
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedLocation);

    await context.CallActivityAsync("SendApprovalRequestEmail", withIntroLocation);

    approvalResult = await context.WaitForExternalEvent<string>(“ApprovalResult");
  }
  catch (Exception e)
  {
    await context.CallActivityAsync<string>("Cleanup", new[] { encodedLocation, thumbnailLocation, withIntroLocation });
    return new { Error = "Failed to process uploaded video", Message = e.Message };
  }

  return new
  {
    Encoded = encodedLocation,
    Thumbnail = thumbnailLocation,
    WithIntro = withIntroLocation
  };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context,  ILogger log)
{

  var videoLocation = context.GetInput<string>();
  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;
  string approvalResult = "Unknown";

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);
    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedLocation);
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedLocation);

    await context.CallActivityAsync("SendApprovalRequestEmail", withIntroLocation);

    approvalResult = await context.WaitForExternalEvent<string>(“ApprovalResult");
    if (approvalResult == "Approved")
      await context.CallActivityAsync("PublishVideo", withIntroLocation);
    else
      await context.CallActivityAsync("RejectVideo", withIntroLocation);

  }
  catch (Exception e)
  {
    await context.CallActivityAsync<string>("Cleanup", new[] { encodedLocation, thumbnailLocation, withIntroLocation });
    return new { Error = "Failed to process uploaded video", Message = e.Message };
  }

  return new
  {
    Encoded = encodedLocation,
    Thumbnail = thumbnailLocation,
    WithIntro = withIntroLocation
  };

}
```

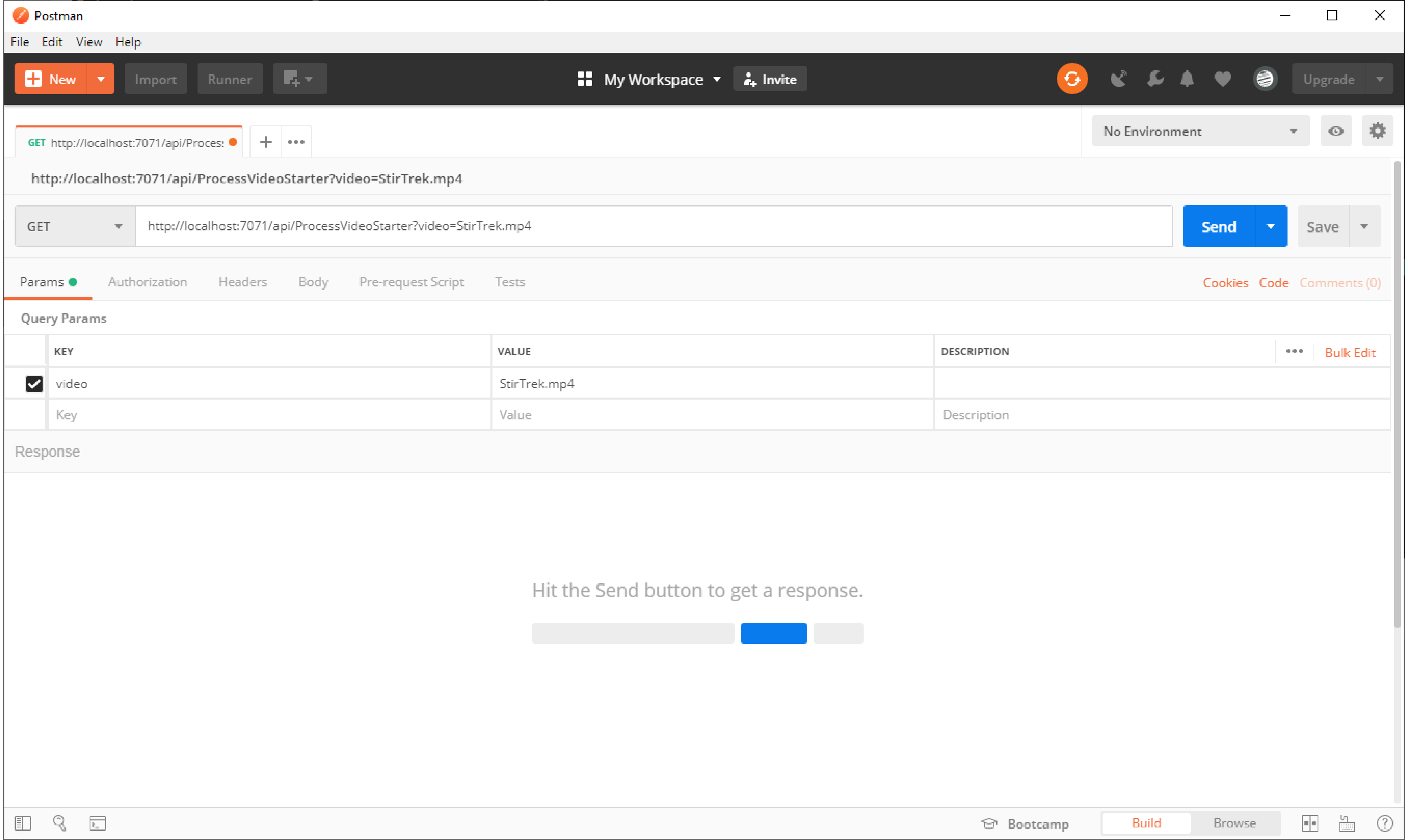
```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator( [OrchestrationTrigger] DurableOrchestrationContext context,  ILogger log)
{

  var videoLocation = context.GetInput<string>();
  string encodedLocation = null;
  string thumbnailLocation = null;
  string withIntroLocation = null;
  string approvalResult = "Unknown";

  try
  {
    var encodedResults = await context.CallSubOrchestratorAsync<VideoFileInfo[]>("EncodeVideoOrchestrator", videoLocation);
    encodedLocation = encodedResults.OrderByDescending(r => r.BitRate).Select(r => r.Location).First();
    thumbnailLocation = await context.CallActivityAsync<string>("ExtractThumbnail", encodedLocation);
    withIntroLocation = await context.CallActivityAsync<string>("PrependIntro", encodedLocation);

    await context.CallActivityAsync("SendApprovalRequestEmail", withIntroLocation);

    approvalResult = await context.WaitForExternalEvent<string>(“ApprovalResult");
    if (approvalResult == "Approved")
      await context.CallActivityAsync("PublishVideo", withIntroLocation);
    else
      await context.CallActivityAsync("RejectVideo", withIntroLocation);

  }
  catch (Exception e)
  {
    await context.CallActivityAsync<string>("Cleanup", new[] { encodedLocation, thumbnailLocation, withIntroLocation });
    return new { Error = "Failed to process uploaded video", Message = e.Message };
  }

  return new
  {
    Encoded = encodedLocation,
    Thumbnail = thumbnailLocation,
    WithIntro = withIntroLocation,
    ApprovalResult = approvalResult
  };

}
```

```
[FunctionName("SendApprovalRequestEmail")] public static async Task SendApprovalRequestEMail(
                   [ActivityTrigger] string inputVideo,
                   ILogger log)
{
  log.LogInformation($"Requesting approval for {inputVideo}");
  await Task.Delay(1000); // Simulate performing the activity
}
```

```csharp
[FunctionName("PublishVideo")]
public static async Task PublishVideo(
                   [ActivityTrigger] string inputVideo,
                   ILogger log)
{
  log.LogInformation($"Publishing {inputVideo}");
   await Task.Delay(1000); // Simulate performing the activity }

[FunctionName("RejectVideo")]
public static async Task RejectVideo(
                  [ActivityTrigger] string inputVideo,
                  ILogger log)
{
  log.LogInformation($"Rejecting {inputVideo}");
  await Task.Delay(1000); // Simulate performing the activity
}
```


C:\WINDOWS\system32\cmd.exe

```
[4/20/2019 9:48:03 PM] Generating 11 job function(s)
[4/20/2019 9:48:03 PM] Found the following functions:
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Activities.Cleanup
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Activities.EncodeVideo
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Activities.ExtractThumbnail
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Activities.GetEncodeBitrates
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Activities.PrependIntro
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Activities.PublishVideo
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Activities.RejectVideo
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Activities.SendApprovalRequestEMail
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Orchestrator.EncodeVideoOrchestrator
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Orchestrator.ProcessVideoOrchestrator
[4/20/2019 9:48:03 PM] ChadGreen.AzureDurableFunctions.Waiting.Starter.Run
[4/20/2019 9:48:03 PM]
[4/20/2019 9:48:03 PM] Host initialized (191ms)
[4/20/2019 9:48:03 PM] Starting task hub worker. InstanceId: . Function: . HubName: DurableFunctionsHub. AppName: . Slot
Name: . ExtensionVersion: 1.8.0. SequenceNumber: 1.
[4/20/2019 9:48:04 PM] Host started (634ms)
[4/20/2019 9:48:04 PM] Job host started
Hosting environment: Production
Content root path: D:\Repos\Presentations\Azure Durable Functions\Waiting\bin\Debug\netcoreapp2.1
Now listening on: http://0.0.0.0:7071

Application started. Press Ctrl+C to shut down.
Http Functions:

        ProcessVideoStarter: [GET] http://localhost:7071/api/ProcessVideoStarter

[4/20/2019 9:48:09 PM] Host lock lease acquired by instance ID '0000000000000000000000006D12C421'.
```



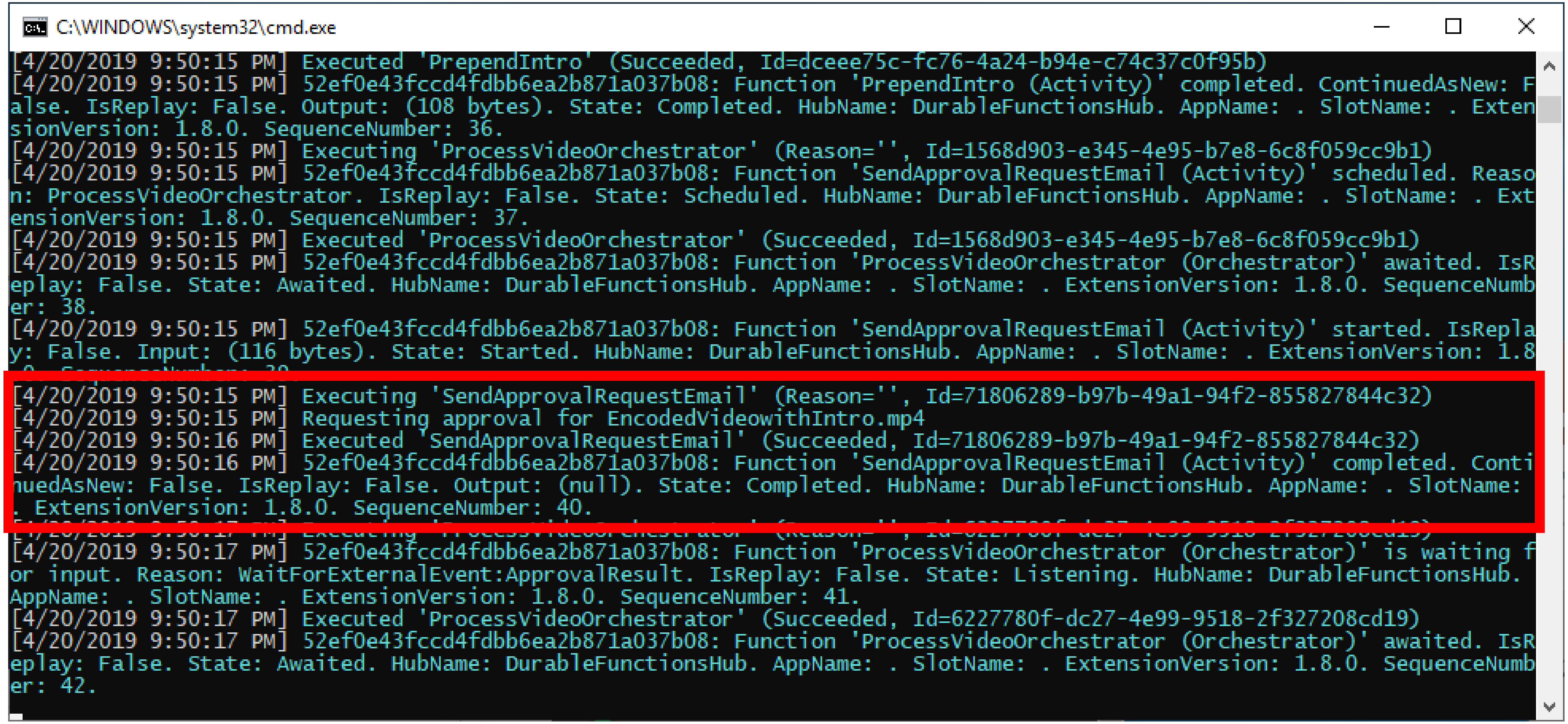
http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4

GET http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4

	KEY	VALUE	DESCRIPTION
✓	video	StirTrek.mp4	
	Key	Value	Description

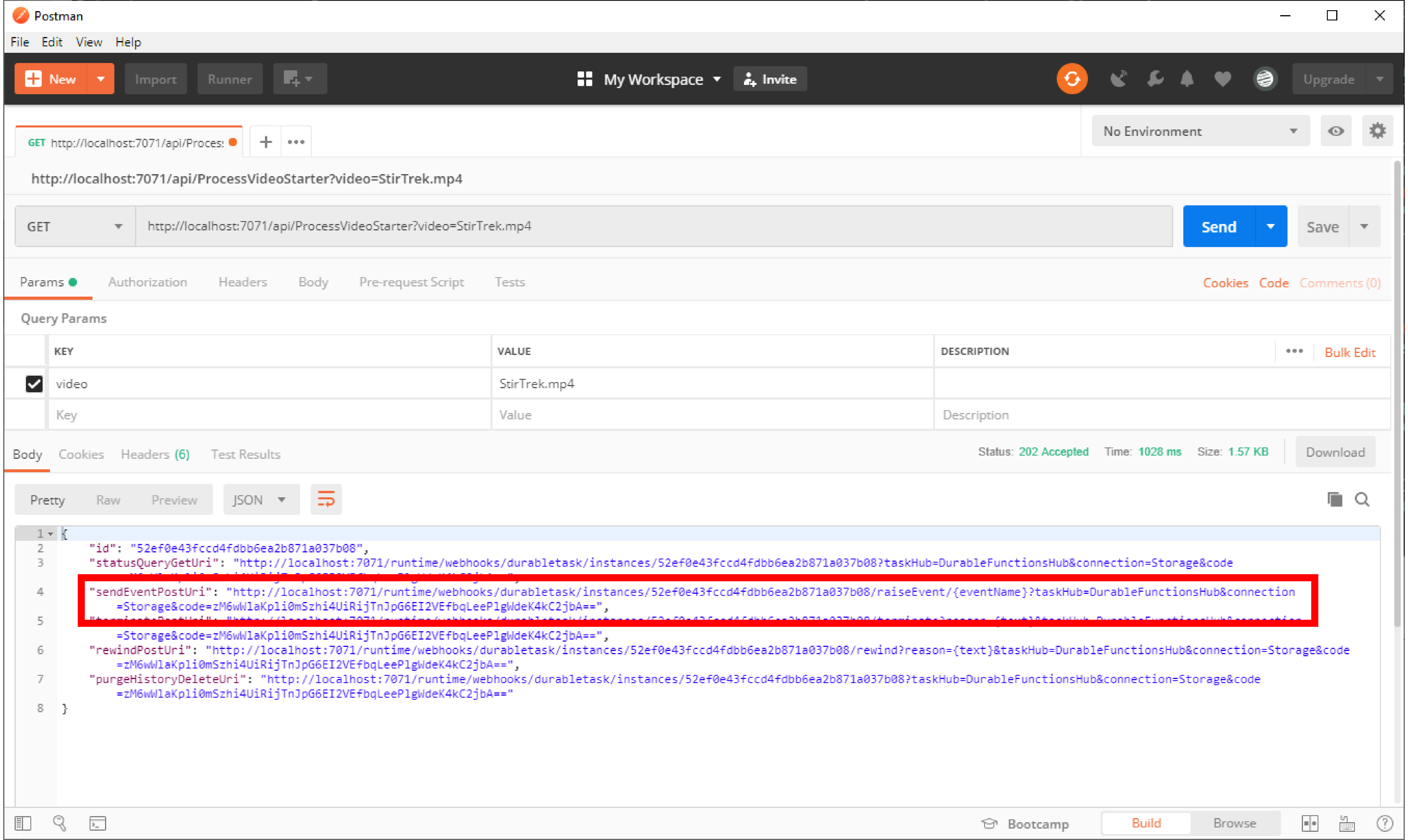
Response

Hit the Send button to get a response.



C:\WINDOWS\system32\cmd.exe

```
[4/20/2019 9:50:15 PM] Executed 'PrependIntro' (Succeeded, Id=dceee75c-fc76-4a24-b94e-c74c37c0f95b)
[4/20/2019 9:50:15 PM] 52ef0e43fccd4fdbb6ea2b871a037b08: Function 'PrependIntro (Activity)' completed. ContinuedAsNew: F
alse. IsReplay: False. Output: (108 bytes). State: Completed. HubName: DurableFunctionsHub. AppName: . SlotName: . Exten
sionVersion: 1.8.0. SequenceNumber: 36.
[4/20/2019 9:50:15 PM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=1568d903-e345-4e95-b7e8-6c8f059cc9b1)
[4/20/2019 9:50:15 PM] 52ef0e43fccd4fdbb6ea2b871a037b08: Function 'SendApprovalRequestEmail (Activity)' scheduled. Reaso
n: ProcessVideoOrchestrator. IsReplay: False. State: Scheduled. HubName: DurableFunctionsHub. AppName: . SlotName: . Ext
ensionVersion: 1.8.0. SequenceNumber: 37.
[4/20/2019 9:50:15 PM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=1568d903-e345-4e95-b7e8-6c8f059cc9b1)
[4/20/2019 9:50:15 PM] 52ef0e43fccd4fdbb6ea2b871a037b08: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 38.
[4/20/2019 9:50:15 PM] 52ef0e43fccd4fdbb6ea2b871a037b08: Function 'SendApprovalRequestEmail (Activity)' started. IsRepla
y: False. Input: (116 bytes). State: Started. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8
.0. SequenceNumber: 39.
[4/20/2019 9:50:15 PM] Executing 'SendApprovalRequestEmail' (Reason='', Id=71806289-b97b-49a1-94f2-855827844c32)
[4/20/2019 9:50:15 PM] Requesting approval for EncodedVideowithIntro.mp4
[4/20/2019 9:50:16 PM] Executed 'SendApprovalRequestEmail' (Succeeded, Id=71806289-b97b-49a1-94f2-855827844c32)
[4/20/2019 9:50:16 PM] 52ef0e43fccd4fdbb6ea2b871a037b08: Function 'SendApprovalRequestEmail (Activity)' completed. Conti
nuedAsNew: False. IsReplay: False. Output: (null). State: Completed. HubName: DurableFunctionsHub. AppName: . SlotName:
. ExtensionVersion: 1.8.0. SequenceNumber: 40.
[4/20/2019 9:50:17 PM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=6227780f-dc27-4e99-9518-2f327208cd19)
[4/20/2019 9:50:17 PM] 52ef0e43fccd4fdbb6ea2b871a037b08: Function 'ProcessVideoOrchestrator (Orchestrator)' is waiting f
or input. Reason: WaitForExternalEvent:ApprovalResult. IsReplay: False. State: Listening. HubName: DurableFunctionsHub.
AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumber: 41.
[4/20/2019 9:50:17 PM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=6227780f-dc27-4e99-9518-2f327208cd19)
[4/20/2019 9:50:17 PM] 52ef0e43fccd4fdbb6ea2b871a037b08: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 42.
```



GET http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4

Query Params

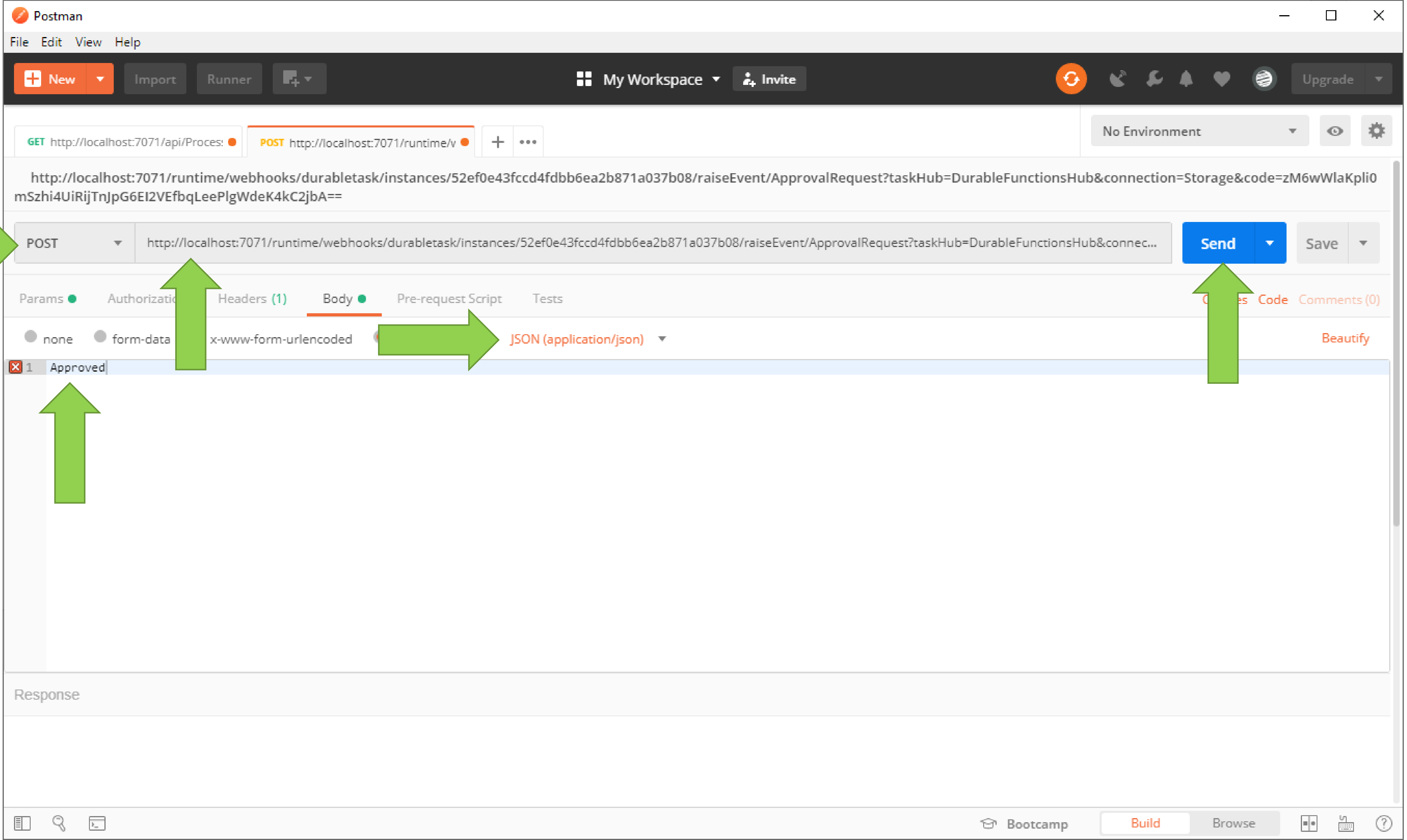
	KEY	VALUE	DESCRIPTION
✓	video	StirTrek.mp4	
	Key	Value	Description

Status: 202 Accepted Time: 1028 ms Size: 1.57 KB

```
{
    "id": "52ef0e43fccd4fdbb6ea2b871a037b08",
    "statusQueryGetUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08?taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "sendEventPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/raiseEvent/{eventName}?taskHub=DurableFunctionsHub&connection
        =Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "terminatePostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/terminate?reason={text}&taskHub=DurableFunctionsHub&connection
        =Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "rewindPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/rewind?reason={text}&taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "purgeHistoryDeleteUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08?taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA=="
}
```

http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/raiseEvent/{eventName}?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==

http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/raiseEvent/ApprovalResult?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==

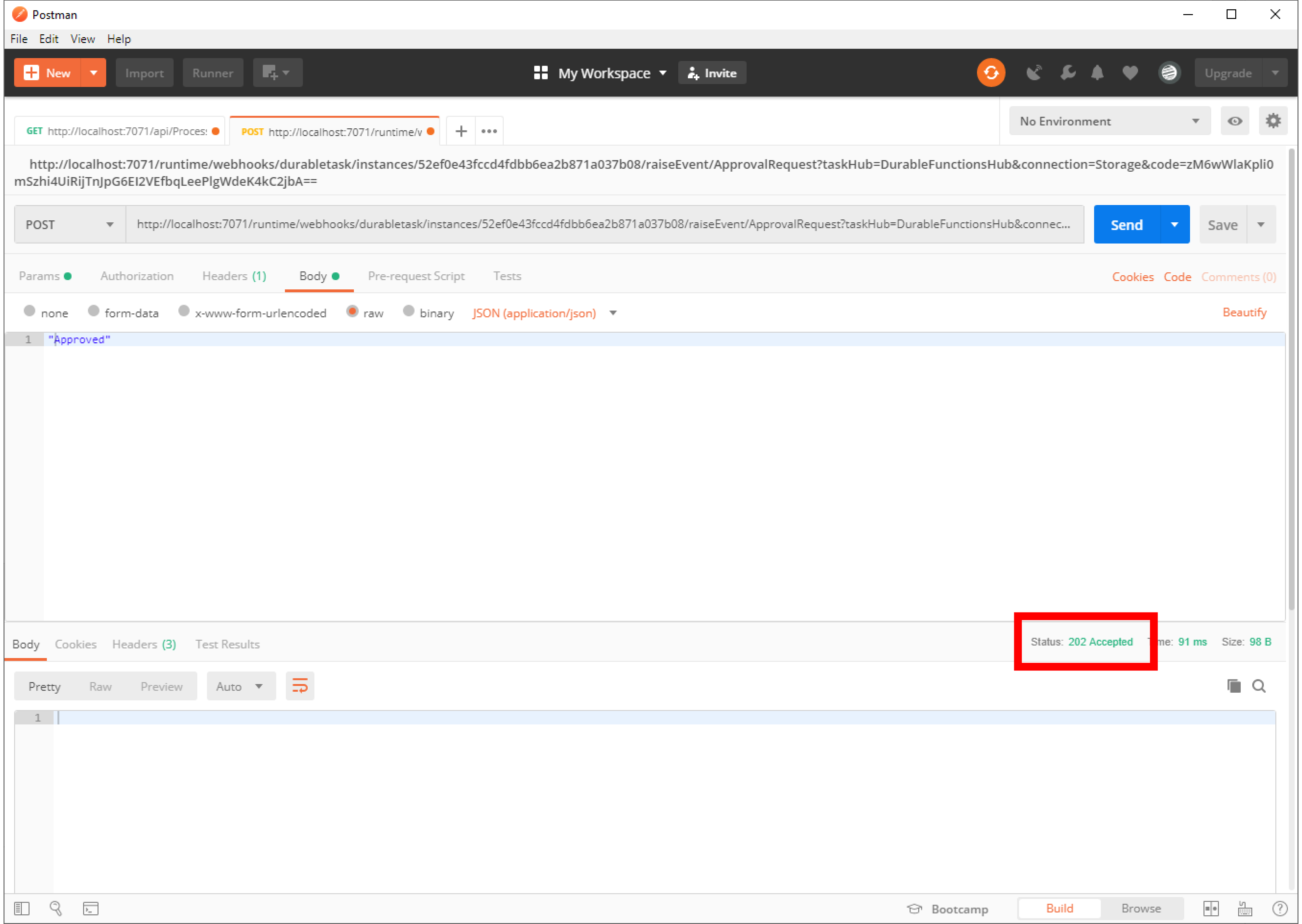


http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/raiseEvent/ApprovalRequest?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==

POST http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/raiseEvent/ApprovalRequest?taskHub=DurableFunctionsHub&connec...

JSON (application/json)

```
Approved
```

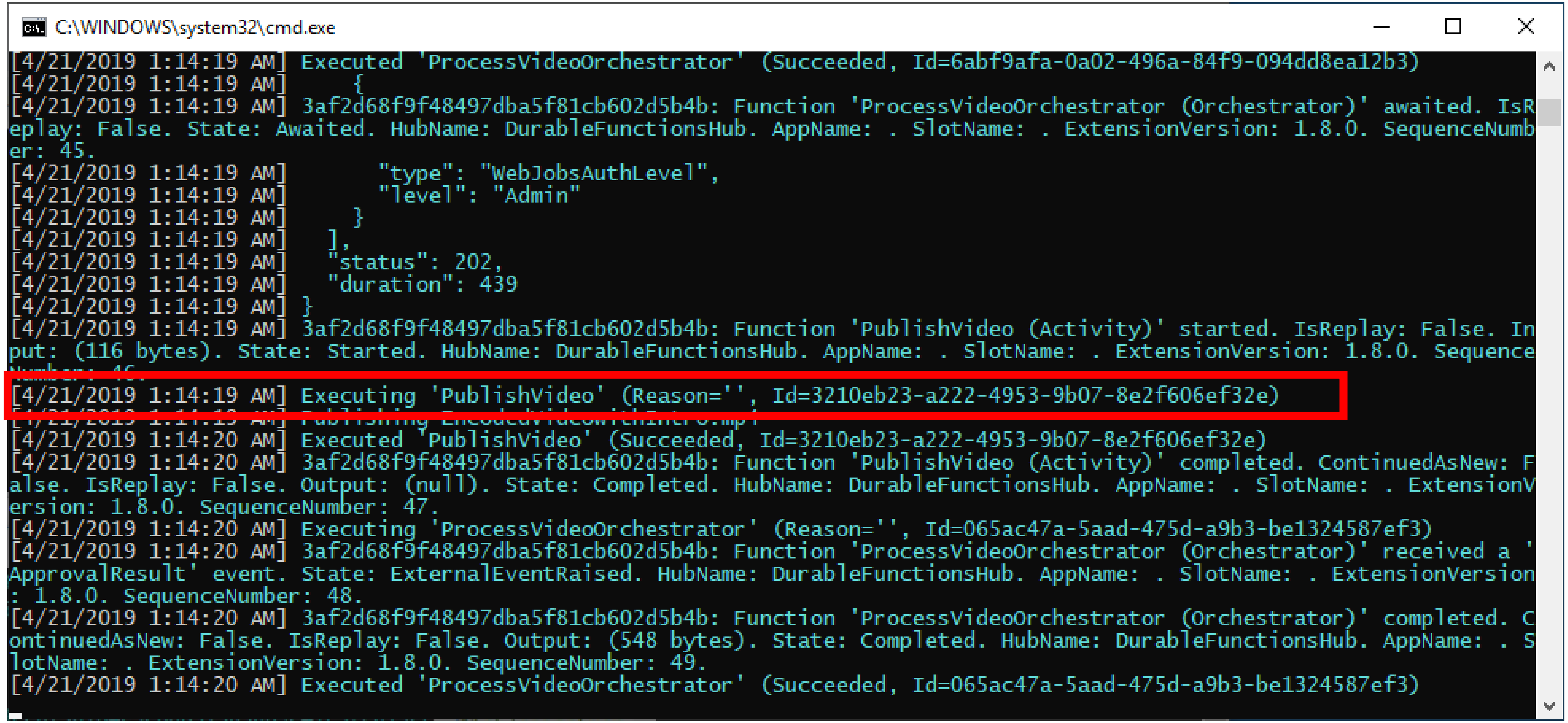


http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/raiseEvent/ApprovalRequest?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==

POST http://localhost:7071/runtime/webhooks/durabletask/instances/52ef0e43fccd4fdbb6ea2b871a037b08/raiseEvent/ApprovalRequest?taskHub=DurableFunctionsHub&connec...

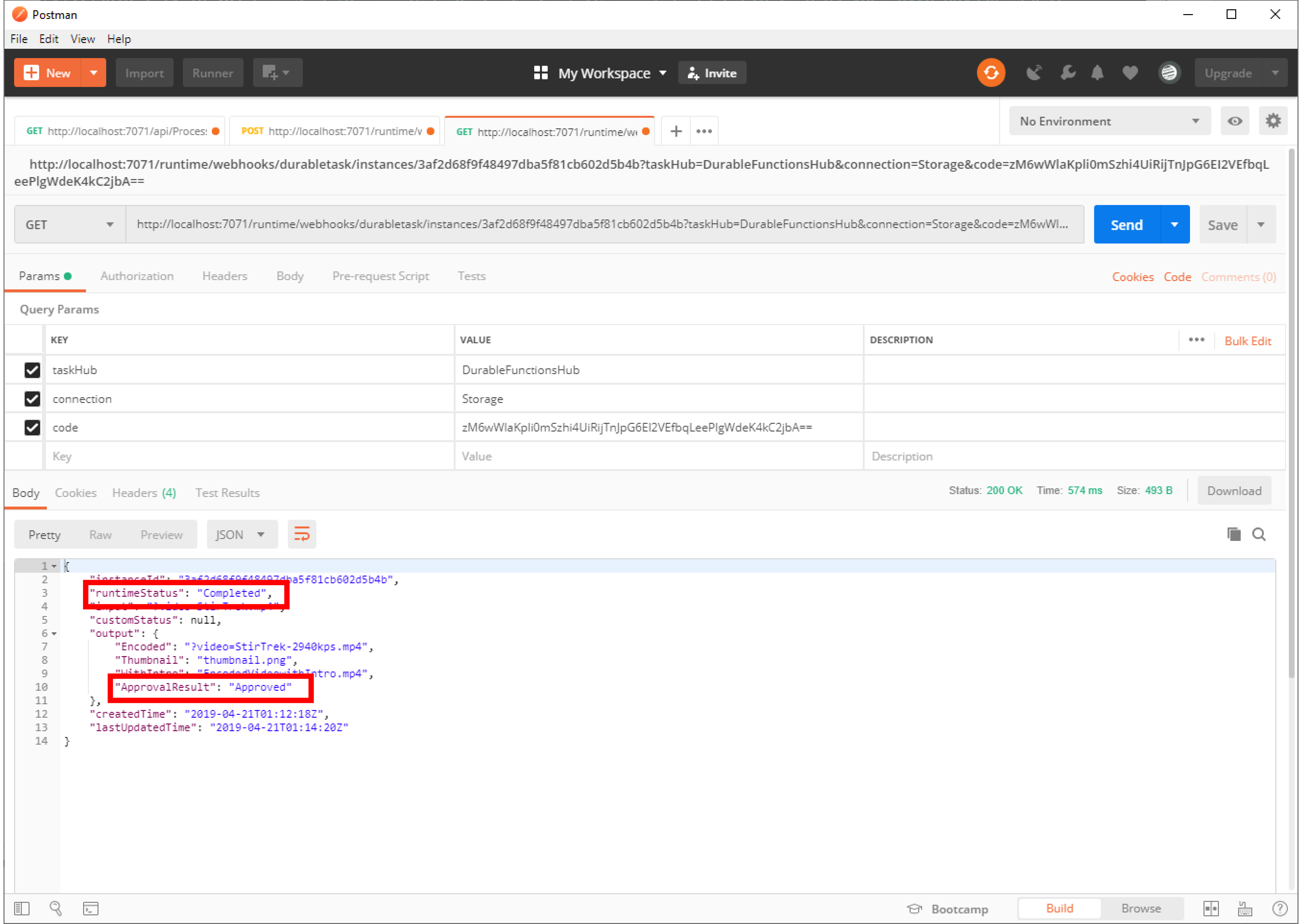
```
"Approved"
```

Status: 202 Accepted Time: 91 ms Size: 98 B



C:\WINDOWS\system32\cmd.exe

```
[4/21/2019 1:14:19 AM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=6abf9afa-0a02-496a-84f9-094dd8ea12b3)
[4/21/2019 1:14:19 AM]     {
[4/21/2019 1:14:19 AM] 3af2d68f9f48497dba5f81cb602d5b4b: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 45.
[4/21/2019 1:14:19 AM]       "type": "WebJobsAuthLevel",
[4/21/2019 1:14:19 AM]       "level": "Admin"
[4/21/2019 1:14:19 AM]     }
[4/21/2019 1:14:19 AM]   ],
[4/21/2019 1:14:19 AM]   "status": 202,
[4/21/2019 1:14:19 AM]   "duration": 439
[4/21/2019 1:14:19 AM] }
[4/21/2019 1:14:19 AM] 3af2d68f9f48497dba5f81cb602d5b4b: Function 'PublishVideo (Activity)' started. IsReplay: False. In
put: (116 bytes). State: Started. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. Sequence
Number: 46.
[4/21/2019 1:14:19 AM] Executing 'PublishVideo' (Reason='', Id=3210eb23-a222-4953-9b07-8e2f606ef32e)
[4/21/2019 1:14:19 AM] Publishing EncodedVideoWithIntro.mp4
[4/21/2019 1:14:20 AM] Executed 'PublishVideo' (Succeeded, Id=3210eb23-a222-4953-9b07-8e2f606ef32e)
[4/21/2019 1:14:20 AM] 3af2d68f9f48497dba5f81cb602d5b4b: Function 'PublishVideo (Activity)' completed. ContinuedAsNew: F
alse. IsReplay: False. Output: (null). State: Completed. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionV
ersion: 1.8.0. SequenceNumber: 47.
[4/21/2019 1:14:20 AM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=065ac47a-5aad-475d-a9b3-be1324587ef3)
[4/21/2019 1:14:20 AM] 3af2d68f9f48497dba5f81cb602d5b4b: Function 'ProcessVideoOrchestrator (Orchestrator)' received a '
ApprovalResult' event. State: ExternalEventRaised. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion
: 1.8.0. SequenceNumber: 48.
[4/21/2019 1:14:20 AM] 3af2d68f9f48497dba5f81cb602d5b4b: Function 'ProcessVideoOrchestrator (Orchestrator)' completed. C
ontinuedAsNew: False. IsReplay: False. Output: (548 bytes). State: Completed. HubName: DurableFunctionsHub. AppName: . S
lotName: . ExtensionVersion: 1.8.0. SequenceNumber: 49.
[4/21/2019 1:14:20 AM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=065ac47a-5aad-475d-a9b3-be1324587ef3)
```



Postman

File Edit View Help

New Import Runner My Workspace Invite Upgrade

GET http://localhost:7071/api/Proces: POST http://localhost:7071/runtime/v GET http://localhost:7071/runtime/w

No Environment

http://localhost:7071/runtime/webhooks/durabletask/instances/3af2d68f9f48497dba5f81cb602d5b4b?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==

GET http://localhost:7071/runtime/webhooks/durabletask/instances/3af2d68f9f48497dba5f81cb602d5b4b?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wWl... Send Save

Params Authorization Headers Body Pre-request Script Tests Cookies Code Comments (0)

Query Params

	KEY	VALUE	DESCRIPTION
✓	taskHub	DurableFunctionsHub	
✓	connection	Storage	
✓	code	zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==	
	Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK Time: 574 ms Size: 493 B Download

Pretty Raw Preview JSON

```
{
    "instanceId": "3af2d68f9f48497dba5f81cb602d5b4b",
    "runtimeStatus": "Completed",
    "input": "?video=StirTrek.mp4",
    "customStatus": null,
    "output": {
        "Encoded": "?video=StirTrek-2940kps.mp4",
        "Thumbnail": "thumbnail.png",
        "WithIntro": "EncodedVideowithIntro.mp4",
        "ApprovalResult": "Approved"
    },
    "createdTime": "2019-04-21T01:12:18Z",
    "lastUpdatedTime": "2019-04-21T01:14:20Z"
}
```

Bootcamp Build Browse

Sending Events to Workflows

- **Send using the ‘raiseEvent’ API**
    - This endpoint includes a secret key
- **Human interaction triggers a regular Azure Function**
    - HTTP trigger
    - Queue trigger
- **External systems might send webhooks**
    - Receive webhook and pass on event to workflow
- **DurableOrchestrationClient.RaiseEventAsync**

Sending events to orchestrations

- Approval and rejection links
- HTTP triggered function
- Send event to orchestration

```csharp
[FunctionName("SendApprovalRequestEmail")]
public static async Task SendApprovalRequestEMail(
                  [ActivityTrigger] string inputVideo,
                  ILogger log)
{
  log.LogInformation($"Requesting approval for {inputVideo}");
  await Task.Delay(1000); // Simulate performing the activity
}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $"http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>"
          + $"<a href=\"{approvedLink}\">Approve</a><br/>"
          + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $"http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>"
           + $"<a href=\"{approvedLink}\">Approve</a><br/>"
           + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $"http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>"
           + $"<a href=\"{approvedLink}\">Approve</a><br/>"
           + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $“http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>“
           + $"<a href=\"{approvedLink}\">Approve</a><br/>“
           + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $“http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>“
           + $"<a href=\"{approvedLink}\">Approve</a><br/>“
           + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $"http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>"
           + $"<a href=\"{approvedLink}\">Approve</a><br/>"
           + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $“http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>“
          + $"<a href=\"{approvedLink}\">Approve</a><br/>“
          + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $"http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>"
          + $"<a href=\"{approvedLink}\">Approve</a><br/>"
          + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $“http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>“
           + $"<a href=\"{approvedLink}\">Approve</a><br/>“
           + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("SendApprovalRequestEmail")]
[return: Table("Approvals")]
public static Approval SendApprovalRequestEMail(
                  [ActivityTrigger] ApprovalInfo approvalInfo,
                  [SendGrid(ApiKey = "SendGridKey")] out SendGridMessage message,
                  ILogger log)
{
  var approvalCode = Guid.NewGuid().ToString("N");

  log.LogInformation($"Sending approval request for {approvalInfo.VideoLocation}");

  var host = Environment.GetEnvironmentVariable("Host");
  var functionAddress = $“http://{host}/api/SubmitVideoApproval/{approvalCode}";
  var approvedLink = functionAddress + "?result=Approved";
  var rejecetedLink = functionAddress + "?result=Rejected";
  var body = $"Please review {approvalInfo.VideoLocation}<br/>“
          + $"<a href=\"{approvedLink}\">Approve</a><br/>“
          + $"<a href=\"{rejecetedLink}\">Reject</a>";

  message = new SendGridMessage();
  message.AddTo(Environment.GetEnvironmentVariable("ApproverEmail"));
  message.SetFrom(Environment.GetEnvironmentVariable("SenderEmail"));
  message.AddContent("text/html", body);
  message.SetSubject("A video is awaiting approval");

  log.LogInformation(body);

  return new Approval
  {
    PartitionKey = "Approval",
    RowKey = approvalCode,
    OrchestrationId = approvalInfo.OrchestrationId
   };

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{


    await context.CallActivityAsync("SendApprovalRequestEmail", withIntroLocation);


}
```

```csharp
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{


    await context.CallActivityAsync("SendApprovalRequestEmail", new ApprovalInfo()
    {
      OrchestrationId = context.InstanceId,
      VideoLocation = withIntroLocation
    });


}
```

```
[FunctionName("SubmitVideoApproval")]
public static async Task<HttpResponseMessage> SubmitVideoApproval(
                    [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "SubmitVideoApproval/{id}")] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    [Table("Approvals", "Approval", "{id}")] Approval approval,
                    ILogger log)
{

  string result = GetQueryStringValue(req, "result");
  if (string.IsNullOrEmpty(result))
    return req.CreateResponse(HttpStatusCode.BadRequest, "Need an approval result");

  log.LogWarning($"Sending approval result to {approval.OrchestrationId} of {result}");

  await client.RaiseEventAsync(approval.OrchestrationId, "ApprovalResult", result);

  return req.CreateResponse(HttpStatusCode.OK);

}
```

```
[FunctionName("SubmitVideoApproval")]
public static async Task<HttpResponseMessage> SubmitVideoApproval(
                    [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "SubmitVideoApproval/{id}")] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    [Table("Approvals", "Approval", "{id}")] Approval approval,
                    ILogger log)
{

  string result = GetQueryStringValue(req, "result");
  if (string.IsNullOrEmpty(result))
    return req.CreateResponse(HttpStatusCode.BadRequest, "Need an approval result");

  log.LogWarning($"Sending approval result to {approval.OrchestrationId} of {result}");

  await client.RaiseEventAsync(approval.OrchestrationId, "ApprovalResult", result);

  return req.CreateResponse(HttpStatusCode.OK);

}
```

```
[FunctionName("SubmitVideoApproval")]
public static async Task<HttpResponseMessage> SubmitVideoApproval(
                    [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "SubmitVideoApproval/{id}")] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    [Table("Approvals", "Approval", "{id}")] Approval approval,
                    ILogger log)
{

  string result = GetQueryStringValue(req, "result");
  if (string.IsNullOrEmpty(result))
    return req.CreateResponse(HttpStatusCode.BadRequest, "Need an approval result");

  log.LogWarning($"Sending approval result to {approval.OrchestrationId} of {result}");

  await client.RaiseEventAsync(approval.OrchestrationId, "ApprovalResult", result);

  return req.CreateResponse(HttpStatusCode.OK);

}
```

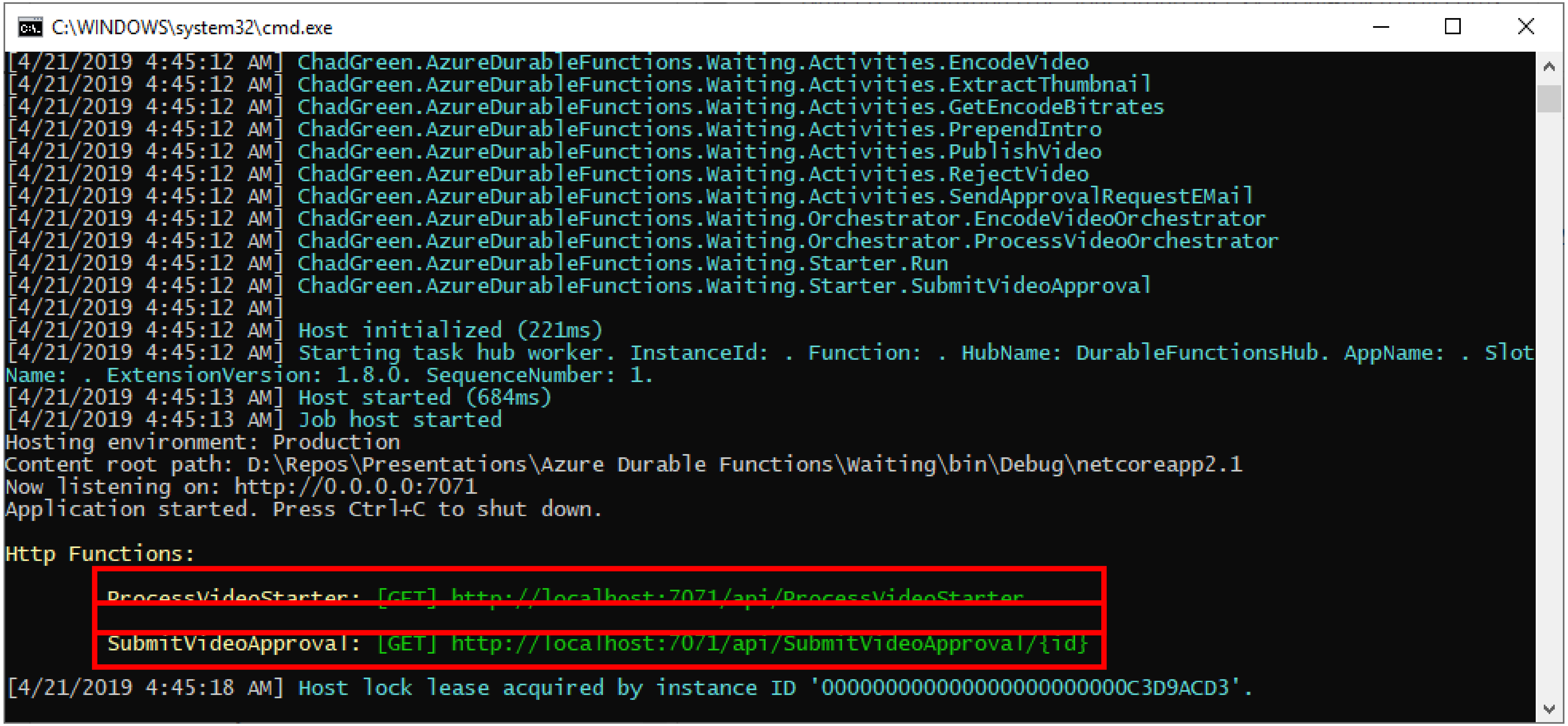
```
[FunctionName("SubmitVideoApproval")]
public static async Task<HttpResponseMessage> SubmitVideoApproval(
                    [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "SubmitVideoApproval/{id}")] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    [Table("Approvals", "Approval", "{id}")] Approval approval,
                    ILogger log)
{

  string result = GetQueryStringValue(req, "result");
  if (string.IsNullOrEmpty(result))
    return req.CreateResponse(HttpStatusCode.BadRequest, "Need an approval result");

  log.LogWarning($"Sending approval result to {approval.OrchestrationId} of {result}");

  await client.RaiseEventAsync(approval.OrchestrationId, "ApprovalResult", result);

  return req.CreateResponse(HttpStatusCode.OK);

}
```

```
[FunctionName("SubmitVideoApproval")]
public static async Task<HttpResponseMessage> SubmitVideoApproval(
                    [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "SubmitVideoApproval/{id}")] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    [Table("Approvals", "Approval", "{id}")] Approval approval,
                    ILogger log)
{

  string result = GetQueryStringValue(req, "result");
  if (string.IsNullOrEmpty(result))
    return req.CreateResponse(HttpStatusCode.BadRequest, "Need an approval result");

  log.LogWarning($"Sending approval result to {approval.OrchestrationId} of {result}");

  await client.RaiseEventAsync(approval.OrchestrationId, "ApprovalResult", result);

  return req.CreateResponse(HttpStatusCode.OK);

}
```

```
[FunctionName("SubmitVideoApproval")]
public static async Task<HttpResponseMessage> SubmitVideoApproval(
                    [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "SubmitVideoApproval/{id}")] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    [Table("Approvals", "Approval", "{id}")] Approval approval,
                    ILogger log)
{

  string result = GetQueryStringValue(req, "result");
  if (string.IsNullOrEmpty(result))
    return req.CreateResponse(HttpStatusCode.BadRequest, "Need an approval result");

  log.LogWarning($"Sending approval result to {approval.OrchestrationId} of {result}");

  await client.RaiseEventAsync(approval.OrchestrationId, "ApprovalResult", result);

  return req.CreateResponse(HttpStatusCode.OK);

}
```

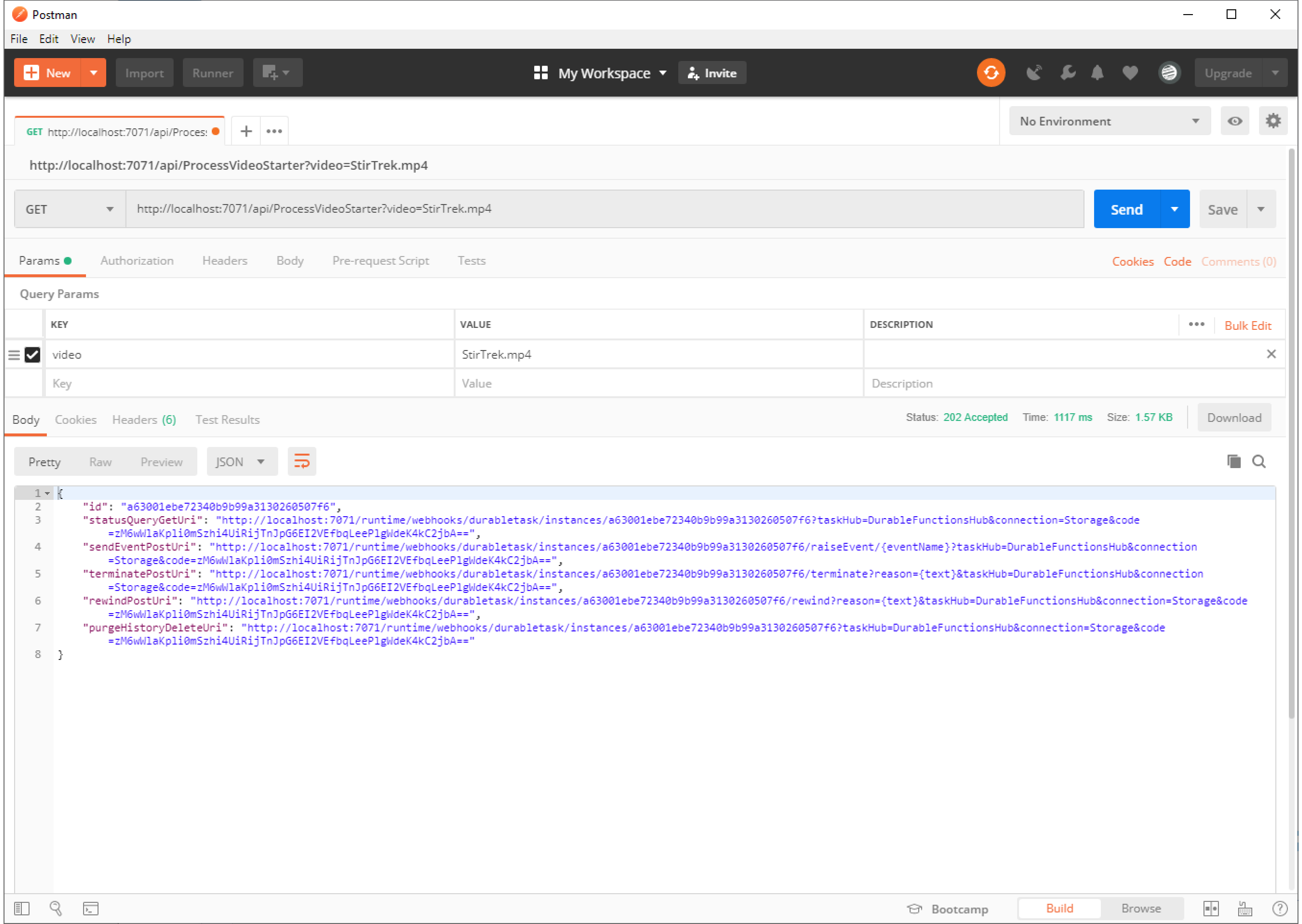
```
[FunctionName("SubmitVideoApproval")]
public static async Task<HttpResponseMessage> SubmitVideoApproval(
                    [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "SubmitVideoApproval/{id}")] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    [Table("Approvals", "Approval", "{id}")] Approval approval,
                    ILogger log)
{

  string result = GetQueryStringValue(req, "result");
  if (string.IsNullOrEmpty(result))
    return req.CreateResponse(HttpStatusCode.BadRequest, "Need an approval result");

  log.LogWarning($"Sending approval result to {approval.OrchestrationId} of {result}");

  await client.RaiseEventAsync(approval.OrchestrationId, "ApprovalResult", result);

  return req.CreateResponse(HttpStatusCode.OK);

}
```

```
[FunctionName("SubmitVideoApproval")]
public static async Task<HttpResponseMessage> SubmitVideoApproval(
                    [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "SubmitVideoApproval/{id}")] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    [Table("Approvals", "Approval", "{id}")] Approval approval,
                    ILogger log)
{

  string result = GetQueryStringValue(req, "result");
  if (string.IsNullOrEmpty(result))
    return req.CreateResponse(HttpStatusCode.BadRequest, "Need an approval result");

  log.LogWarning($"Sending approval result to {approval.OrchestrationId} of {result}");

  await client.RaiseEventAsync(approval.OrchestrationId, "ApprovalResult", result);

  return req.CreateResponse(HttpStatusCode.OK);

}
```



C:\WINDOWS\system32\cmd.exe

```
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.EncodeVideo
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.ExtractThumbnail
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.GetEncodeBitrates
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.PrependIntro
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.PublishVideo
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.RejectVideo
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.SendApprovalRequestEMail
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Orchestrator.EncodeVideoOrchestrator
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Orchestrator.ProcessVideoOrchestrator
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Starter.Run
[4/21/2019 4:45:12 AM] ChadGreen.AzureDurableFunctions.Waiting.Starter.SubmitVideoApproval
[4/21/2019 4:45:12 AM]
[4/21/2019 4:45:12 AM] Host initialized (221ms)
[4/21/2019 4:45:12 AM] Starting task hub worker. InstanceId: . Function: . HubName: DurableFunctionsHub. AppName: . Slot
Name: . ExtensionVersion: 1.8.0. SequenceNumber: 1.
[4/21/2019 4:45:13 AM] Host started (684ms)
[4/21/2019 4:45:13 AM] Job host started
Hosting environment: Production
Content root path: D:\Repos\Presentations\Azure Durable Functions\Waiting\bin\Debug\netcoreapp2.1
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.

Http Functions:

        ProcessVideoStarter: [GET] http://localhost:7071/api/ProcessVideoStarter

        SubmitVideoApproval: [GET] http://localhost:7071/api/SubmitVideoApproval/{id}

[4/21/2019 4:45:18 AM] Host lock lease acquired by instance ID '0000000000000000000000000C3D9ACD3'.
```

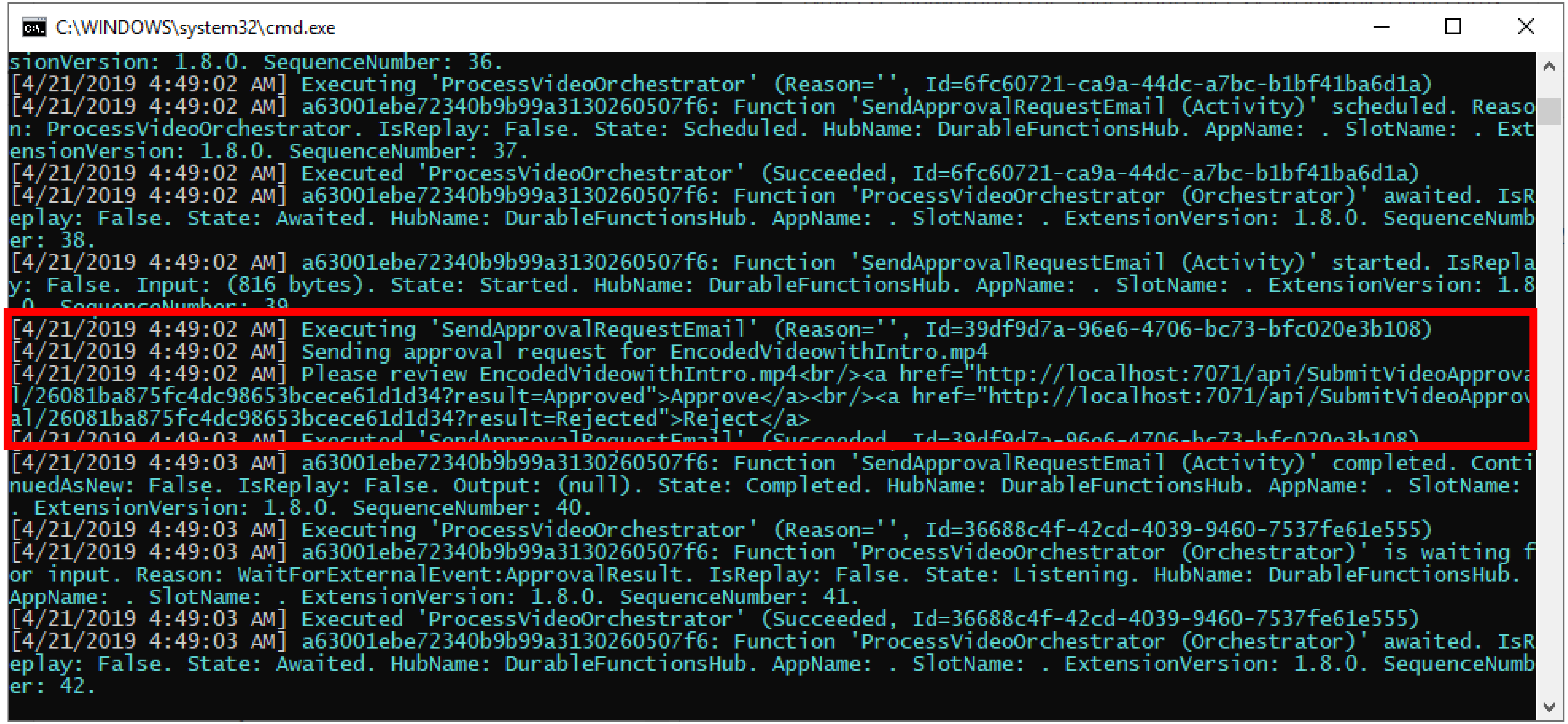


http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4

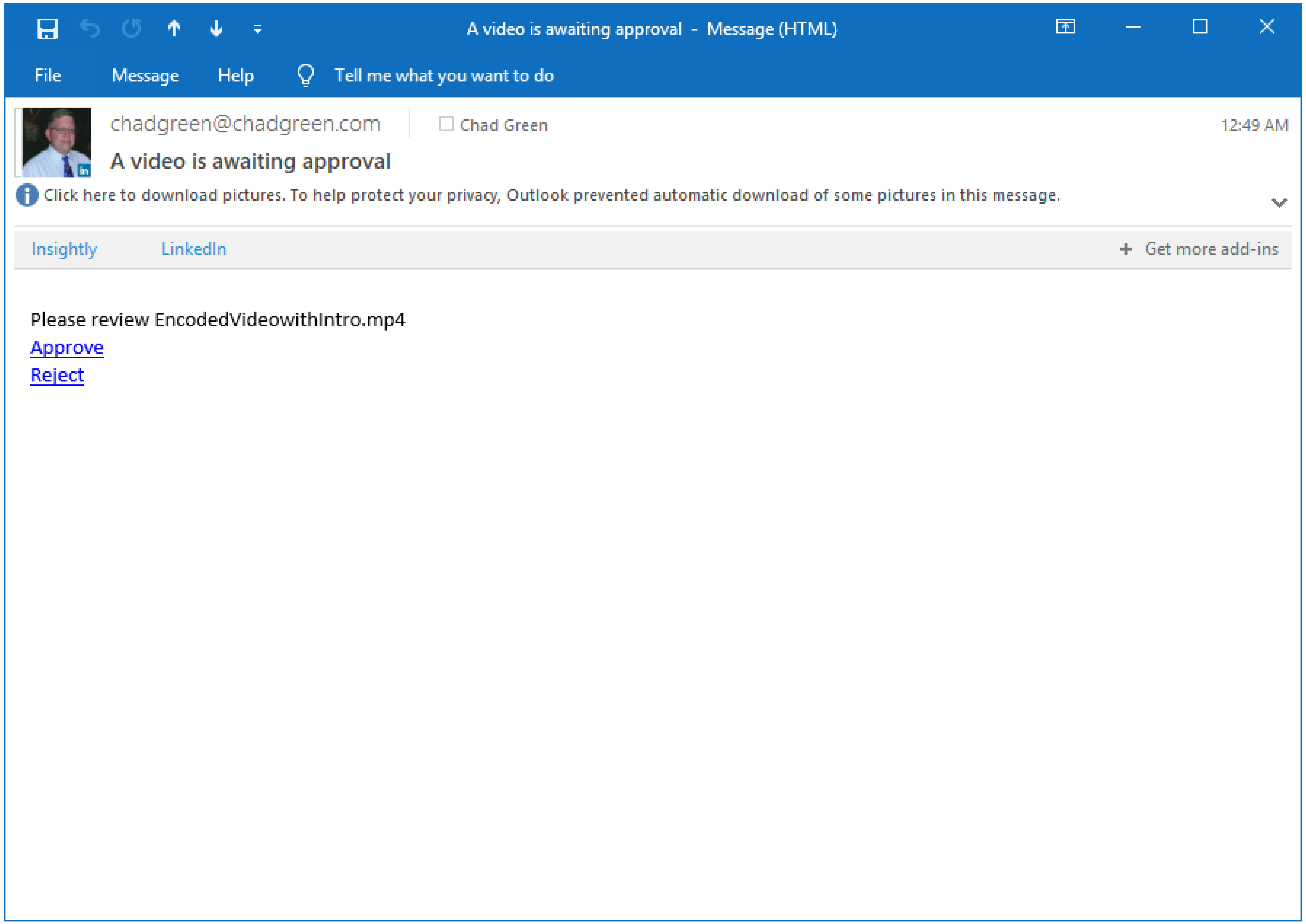
KEY	VALUE	DESCRIPTION
video	StirTrek.mp4	
Key	Value	Description

Status: 202 Accepted Time: 1117 ms Size: 1.57 KB

```
{
    "id": "a63001ebe72340b9b99a3130260507f6",
    "statusQueryGetUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/a63001ebe72340b9b99a3130260507f6?taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "sendEventPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/a63001ebe72340b9b99a3130260507f6/raiseEvent/{eventName}?taskHub=DurableFunctionsHub&connection
        =Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "terminatePostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/a63001ebe72340b9b99a3130260507f6/terminate?reason={text}&taskHub=DurableFunctionsHub&connection
        =Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "rewindPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/a63001ebe72340b9b99a3130260507f6/rewind?reason={text}&taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "purgeHistoryDeleteUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/a63001ebe72340b9b99a3130260507f6?taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA=="
}
```



```
C:\WINDOWS\system32\cmd.exe
sionVersion: 1.8.0. SequenceNumber: 36.
[4/21/2019 4:49:02 AM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=6fc60721-ca9a-44dc-a7bc-b1bf41ba6d1a)
[4/21/2019 4:49:02 AM] a63001ebe72340b9b99a3130260507f6: Function 'SendApprovalRequestEmail (Activity)' scheduled. Reaso
n: ProcessVideoOrchestrator. IsReplay: False. State: Scheduled. HubName: DurableFunctionsHub. AppName: . SlotName: . Ext
ensionVersion: 1.8.0. SequenceNumber: 37.
[4/21/2019 4:49:02 AM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=6fc60721-ca9a-44dc-a7bc-b1bf41ba6d1a)
[4/21/2019 4:49:02 AM] a63001ebe72340b9b99a3130260507f6: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 38.
[4/21/2019 4:49:02 AM] a63001ebe72340b9b99a3130260507f6: Function 'SendApprovalRequestEmail (Activity)' started. IsRepla
y: False. Input: (816 bytes). State: Started. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8
.0. SequenceNumber: 39.
[4/21/2019 4:49:02 AM] Executing 'SendApprovalRequestEmail' (Reason='', Id=39df9d7a-96e6-4706-bc73-bfc020e3b108)
[4/21/2019 4:49:02 AM] Sending approval request for EncodedVideowithIntro.mp4
[4/21/2019 4:49:02 AM] Please review EncodedVideowithIntro.mp4<br/><a href="http://localhost:7071/api/SubmitVideoApprova
l/26081ba875fc4dc98653bcece61d1d34?result=Approved">Approve</a><br/><a href="http://localhost:7071/api/SubmitVideoApprov
al/26081ba875fc4dc98653bcece61d1d34?result=Rejected">Reject</a>
[4/21/2019 4:49:03 AM] Executed 'SendApprovalRequestEmail' (Succeeded, Id=39df9d7a-96e6-4706-bc73-bfc020e3b108)
[4/21/2019 4:49:03 AM] a63001ebe72340b9b99a3130260507f6: Function 'SendApprovalRequestEmail (Activity)' completed. Conti
nuedAsNew: False. IsReplay: False. Output: (null). State: Completed. HubName: DurableFunctionsHub. AppName: . SlotName:
. ExtensionVersion: 1.8.0. SequenceNumber: 40.
[4/21/2019 4:49:03 AM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=36688c4f-42cd-4039-9460-7537fe61e555)
[4/21/2019 4:49:03 AM] a63001ebe72340b9b99a3130260507f6: Function 'ProcessVideoOrchestrator (Orchestrator)' is waiting f
or input. Reason: WaitForExternalEvent:ApprovalResult. IsReplay: False. State: Listening. HubName: DurableFunctionsHub.
AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumber: 41.
[4/21/2019 4:49:03 AM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=36688c4f-42cd-4039-9460-7537fe61e555)
[4/21/2019 4:49:03 AM] a63001ebe72340b9b99a3130260507f6: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 42.
```


A video is awaiting approval - Message (HTML)
File
Message
Help
Tell me what you want to do

chadgreen@chadgreen.com | Chad Green 12:49 AM

A video is awaiting approval

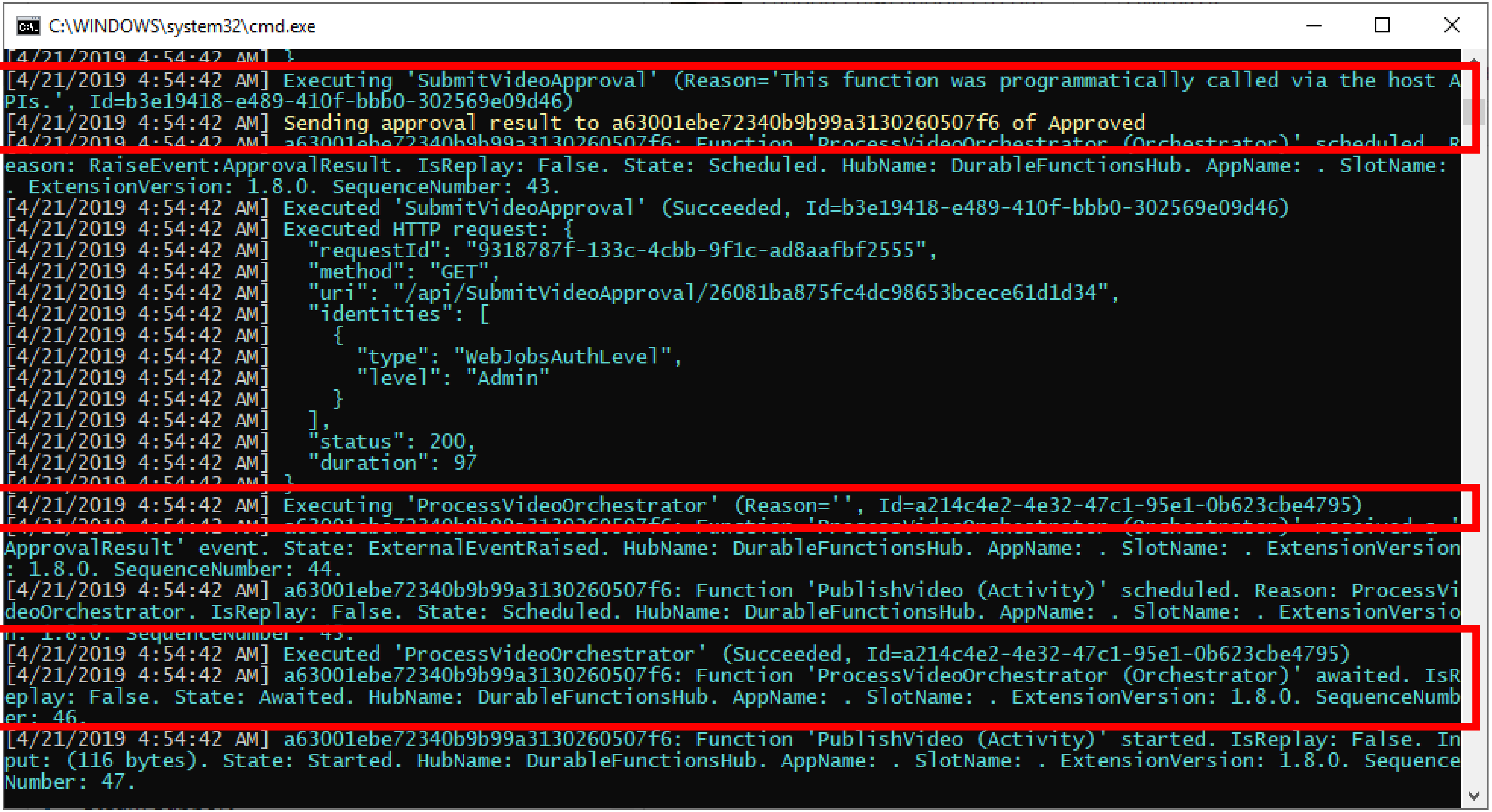
Click here to download pictures. To help protect your privacy, Outlook prevented automatic download of some pictures in this message.

Insightly
LinkedIn
Get more add-ins

Please review EncodedVideowithIntro.mp4

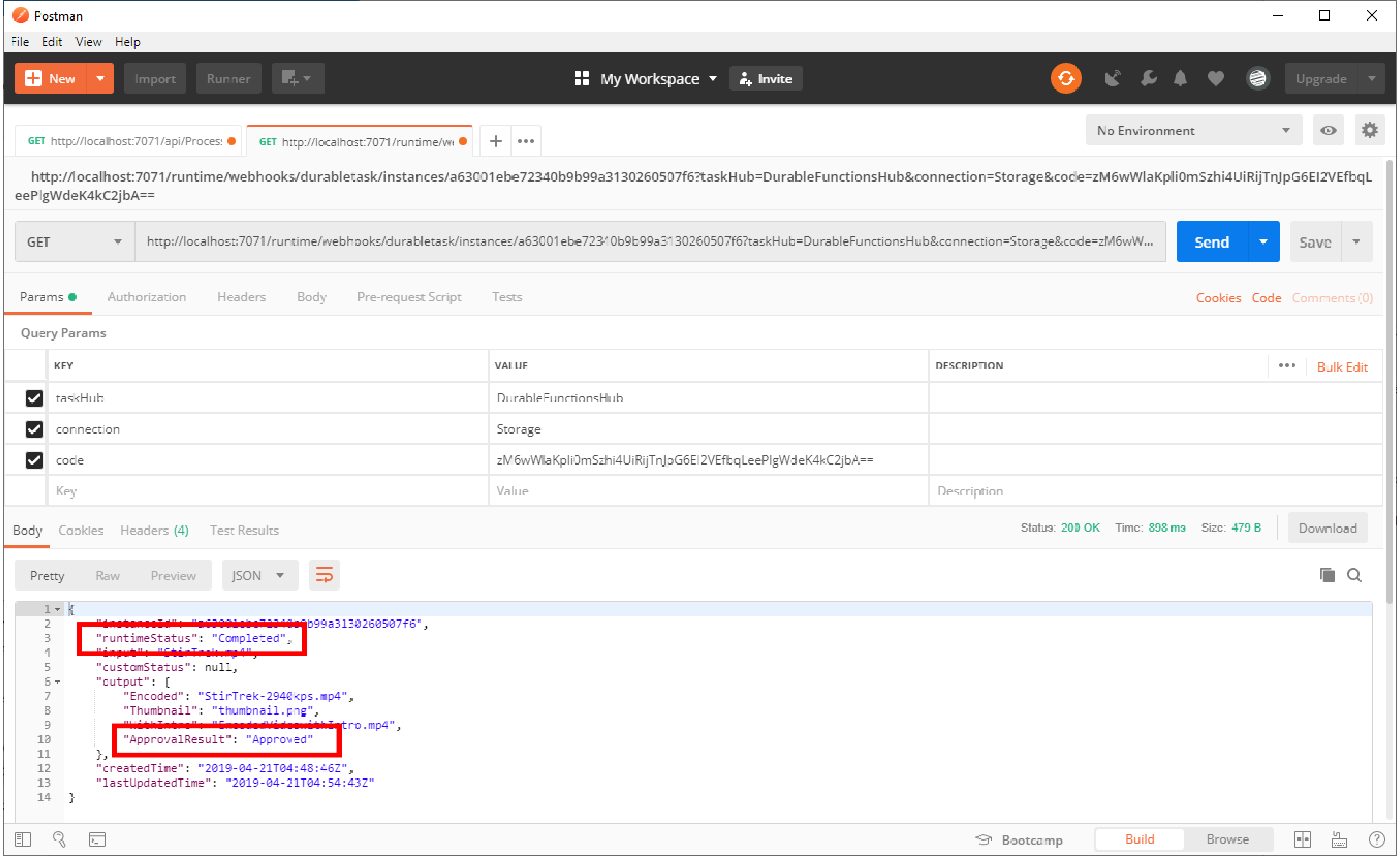
Approve

Reject



C:\WINDOWS\system32\cmd.exe

```
[4/21/2019 4:54:42 AM] }
[4/21/2019 4:54:42 AM] Executing 'SubmitVideoApproval' (Reason='This function was programmatically called via the host A
PIs.', Id=b3e19418-e489-410f-bbb0-302569e09d46)
[4/21/2019 4:54:42 AM] Sending approval result to a63001ebe72340b9b99a3130260507f6 of Approved
[4/21/2019 4:54:42 AM] a63001ebe72340b9b99a3130260507f6: Function 'ProcessVideoOrchestrator (Orchestrator)' scheduled. R
eason: RaiseEvent:ApprovalResult. IsReplay: False. State: Scheduled. HubName: DurableFunctionsHub. AppName: . SlotName:
. ExtensionVersion: 1.8.0. SequenceNumber: 43.
[4/21/2019 4:54:42 AM] Executed 'SubmitVideoApproval' (Succeeded, Id=b3e19418-e489-410f-bbb0-302569e09d46)
[4/21/2019 4:54:42 AM] Executed HTTP request: {
[4/21/2019 4:54:42 AM]   "requestId": "9318787f-133c-4cbb-9f1c-ad8aafbf2555",
[4/21/2019 4:54:42 AM]   "method": "GET",
[4/21/2019 4:54:42 AM]   "uri": "/api/SubmitVideoApproval/26081ba875fc4dc98653bcece61d1d34",
[4/21/2019 4:54:42 AM]   "identities": [
[4/21/2019 4:54:42 AM]     {
[4/21/2019 4:54:42 AM]       "type": "WebJobsAuthLevel",
[4/21/2019 4:54:42 AM]       "level": "Admin"
[4/21/2019 4:54:42 AM]     }
[4/21/2019 4:54:42 AM]   ],
[4/21/2019 4:54:42 AM]   "status": 200,
[4/21/2019 4:54:42 AM]   "duration": 97
[4/21/2019 4:54:42 AM] }
[4/21/2019 4:54:42 AM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=a214c4e2-4e32-47c1-95e1-0b623cbe4795)
[4/21/2019 4:54:42 AM] a63001ebe72340b9b99a3130260507f6: Function 'ProcessVideoOrchestrator (Orchestrator)' received a '
ApprovalResult' event. State: ExternalEventRaised. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion
: 1.8.0. SequenceNumber: 44.
[4/21/2019 4:54:42 AM] a63001ebe72340b9b99a3130260507f6: Function 'PublishVideo (Activity)' scheduled. Reason: ProcessVi
deoOrchestrator. IsReplay: False. State: Scheduled. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersio
n: 1.8.0. SequenceNumber: 45.
[4/21/2019 4:54:42 AM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=a214c4e2-4e32-47c1-95e1-0b623cbe4795)
[4/21/2019 4:54:42 AM] a63001ebe72340b9b99a3130260507f6: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 46.
[4/21/2019 4:54:42 AM] a63001ebe72340b9b99a3130260507f6: Function 'PublishVideo (Activity)' started. IsReplay: False. In
put: (116 bytes). State: Started. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. Sequence
Number: 47.
```


GET http://localhost:7071/runtime/webhooks/durabletask/instances/a63001ebe72340b9b99a3130260507f6?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==

Query Params

KEY	VALUE	DESCRIPTION
taskHub	DurableFunctionsHub	
connection	Storage	
code	zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==	

Status: 200 OK Time: 898 ms Size: 479 B

```
{
    "instanceId": "a63001ebe72340b9b99a3130260507f6",
    "runtimeStatus": "Completed",
    "input": "StirTrek.mp4",
    "customStatus": null,
    "output": {
        "Encoded": "StirTrek-2940kps.mp4",
        "Thumbnail": "thumbnail.png",
        "WithIntro": "EncodedVideoWithIntro.mp4",
        "ApprovalResult": "Approved"
    },
    "createdTime": "2019-04-21T04:48:46Z",
    "lastUpdatedTime": "2019-04-21T04:54:43Z"
}
```

Timing Out External Events

- **Do-it-yourself approach**
  - Send a future scheduled message
  - When it arrives, check if activity has completed
- **Durable Functions makes it easy!**

Timing out external events

- Create a timer
- Determine what happened first

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                  [OrchestrationTrigger] DurableOrchestrationContext context,
                  ILogger log)
{

     approvalResult = await context.WaitForExternalEvent<string>("ApprovalResult");

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                  [OrchestrationTrigger] DurableOrchestrationContext context,
                  ILogger log)
{

     using (var cancellationToken = new CancellationTokenSource())
     {
     }

     approvalResult = await context.WaitForExternalEvent<string>("ApprovalResult");

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                  [OrchestrationTrigger] DurableOrchestrationContext context,
                  ILogger log)
{

     using (var cancellationToken = new CancellationTokenSource())
     {
       var timeoutAt = context.CurrentUtcDateTime.AddSeconds(30);
     }

     approvalResult = await context.WaitForExternalEvent<string>("ApprovalResult");

}
```

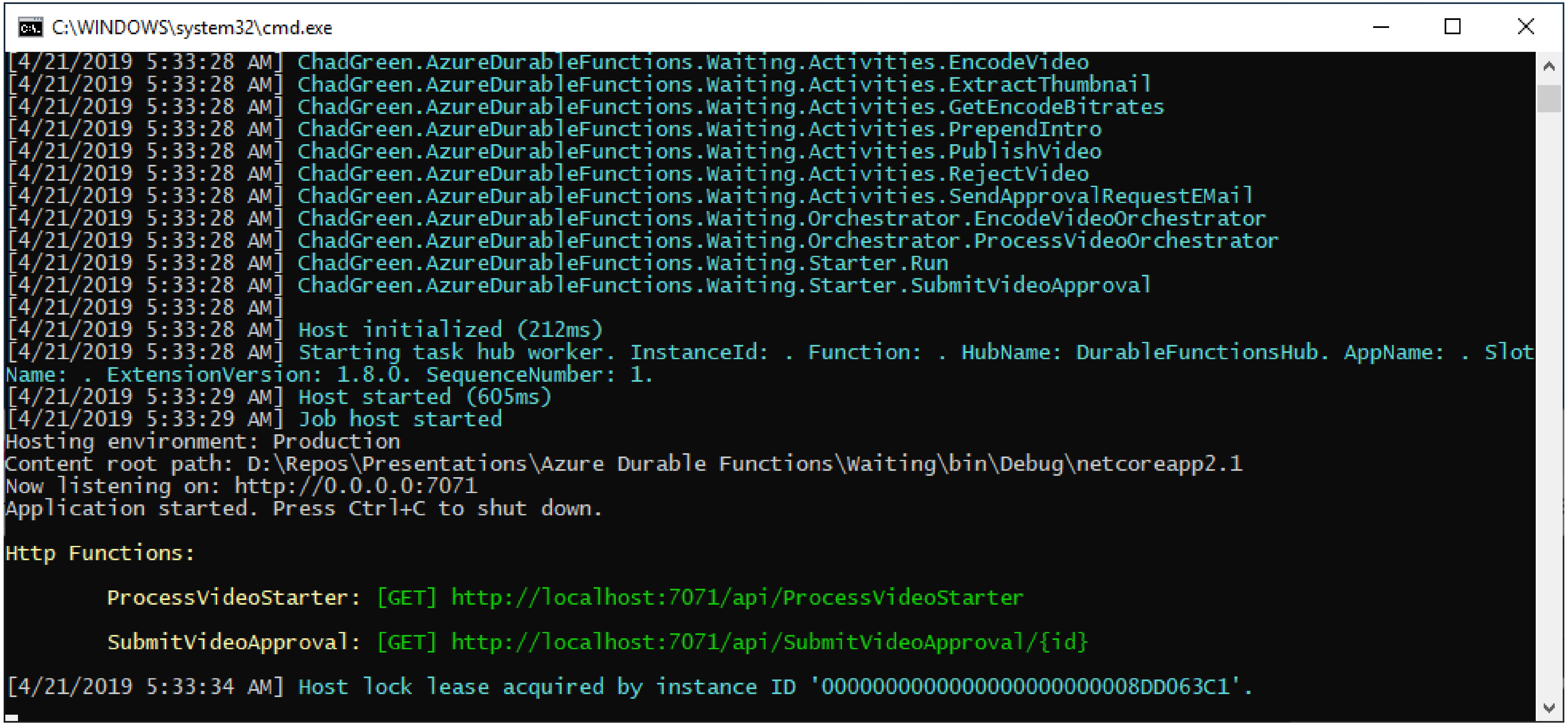
```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                  [OrchestrationTrigger] DurableOrchestrationContext context,
                  ILogger log)
{

     using (var cancellationToken = new CancellationTokenSource())
     {
       var timeoutAt = context.CurrentUtcDateTime.AddSeconds(30);
       var timeoutTask = context.CreateTimer(timeoutAt, cancellationToken.Token);
     }

     approvalResult = await context.WaitForExternalEvent<string>("ApprovalResult");

}
```

```csharp
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{

      using (var cancellationToken = new CancellationTokenSource())
      {
        var timeoutAt = context.CurrentUtcDateTime.AddSeconds(30);
        var timeoutTask = context.CreateTimer(timeoutAt, cancellationToken.Token);
        var approvalTask = context.WaitForExternalEvent<string>("ApprovalResult");
      }

}
```

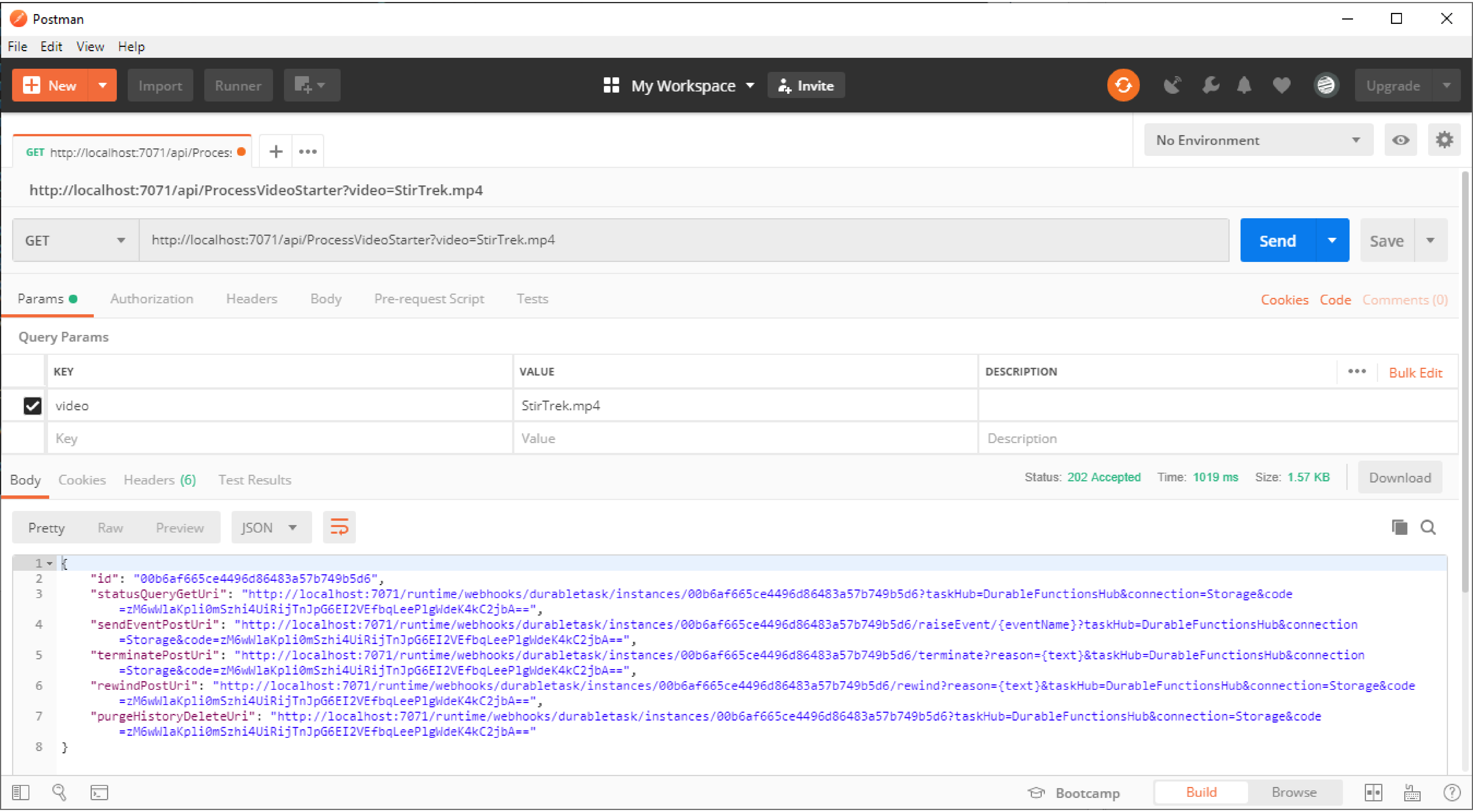
```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{

      using (var cancellationToken = new CancellationTokenSource())
      {
        var timeoutAt = context.CurrentUtcDateTime.AddSeconds(30);
        var timeoutTask = context.CreateTimer(timeoutAt, cancellationToken.Token);
        var approvalTask = context.WaitForExternalEvent<string>("ApprovalResult");
        var winner = await Task.WhenAny(approvalTask, timeoutTask);
      }

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                  [OrchestrationTrigger] DurableOrchestrationContext context,
                  ILogger log)
{

     using (var cancellationToken = new CancellationTokenSource())
     {
       var timeoutAt = context.CurrentUtcDateTime.AddSeconds(30);
       var timeoutTask = context.CreateTimer(timeoutAt, cancellationToken.Token);
       var approvalTask = context.WaitForExternalEvent<string>("ApprovalResult");
       var winner = await Task.WhenAny(approvalTask, timeoutTask);
       if (winner == approvalTask)
       {
         approvalResult = approvalTask.Result;
         cancellationToken.Cancel();
       }
     }

}
```

```
[FunctionName("ProcessVideoOrchestrator")]
public static async Task<object> ProcessVideoOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{

      using (var cancellationToken = new CancellationTokenSource())
      {
        var timeoutAt = context.CurrentUtcDateTime.AddSeconds(30);
        var timeoutTask = context.CreateTimer(timeoutAt, cancellationToken.Token);
        var approvalTask = context.WaitForExternalEvent<string>("ApprovalResult");
        var winner = await Task.WhenAny(approvalTask, timeoutTask);
        if (winner == approvalTask)
        {
          approvalResult = approvalTask.Result;
          cancellationToken.Cancel();
        }
        else
        {
          approvalResult = "Timed Out";
          log.LogWarning("Approval request timed out");
        }
      }

}
```



```
C:\WINDOWS\system32\cmd.exe
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.EncodeVideo
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.ExtractThumbnail
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.GetEncodeBitrates
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.PrependIntro
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.PublishVideo
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.RejectVideo
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Activities.SendApprovalRequestEMail
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Orchestrator.EncodeVideoOrchestrator
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Orchestrator.ProcessVideoOrchestrator
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Starter.Run
[4/21/2019 5:33:28 AM] ChadGreen.AzureDurableFunctions.Waiting.Starter.SubmitVideoApproval
[4/21/2019 5:33:28 AM]
[4/21/2019 5:33:28 AM] Host initialized (212ms)
[4/21/2019 5:33:28 AM] Starting task hub worker. InstanceId: . Function: . HubName: DurableFunctionsHub. AppName: . Slot
Name: . ExtensionVersion: 1.8.0. SequenceNumber: 1.
[4/21/2019 5:33:29 AM] Host started (605ms)
[4/21/2019 5:33:29 AM] Job host started
Hosting environment: Production
Content root path: D:\Repos\Presentations\Azure Durable Functions\Waiting\bin\Debug\netcoreapp2.1
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.

Http Functions:

        ProcessVideoStarter: [GET] http://localhost:7071/api/ProcessVideoStarter

        SubmitVideoApproval: [GET] http://localhost:7071/api/SubmitVideoApproval/{id}

[4/21/2019 5:33:34 AM] Host lock lease acquired by instance ID '0000000000000000000000008DD063C1'.
```



GET http://localhost:7071/api/Proces:

http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4

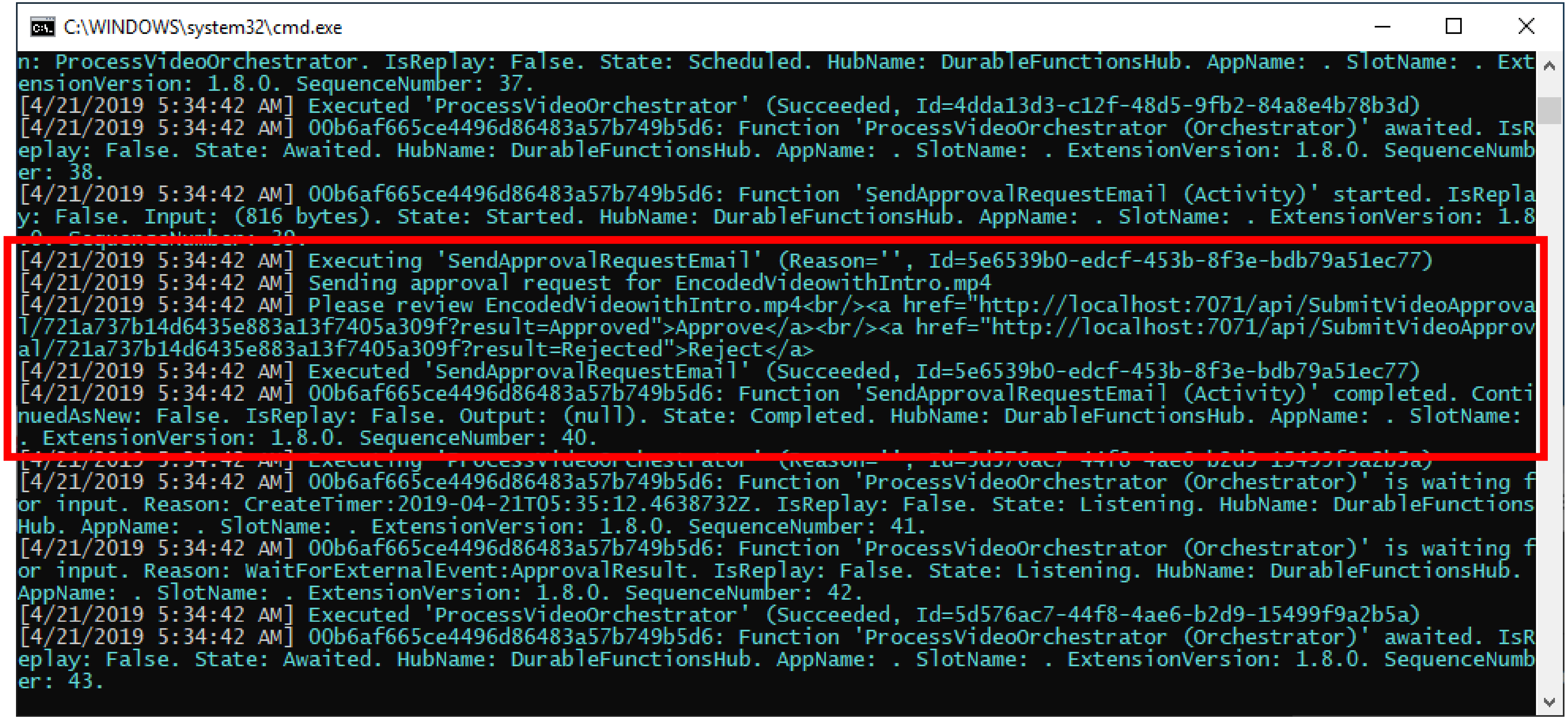
GET http://localhost:7071/api/ProcessVideoStarter?video=StirTrek.mp4

Query Params

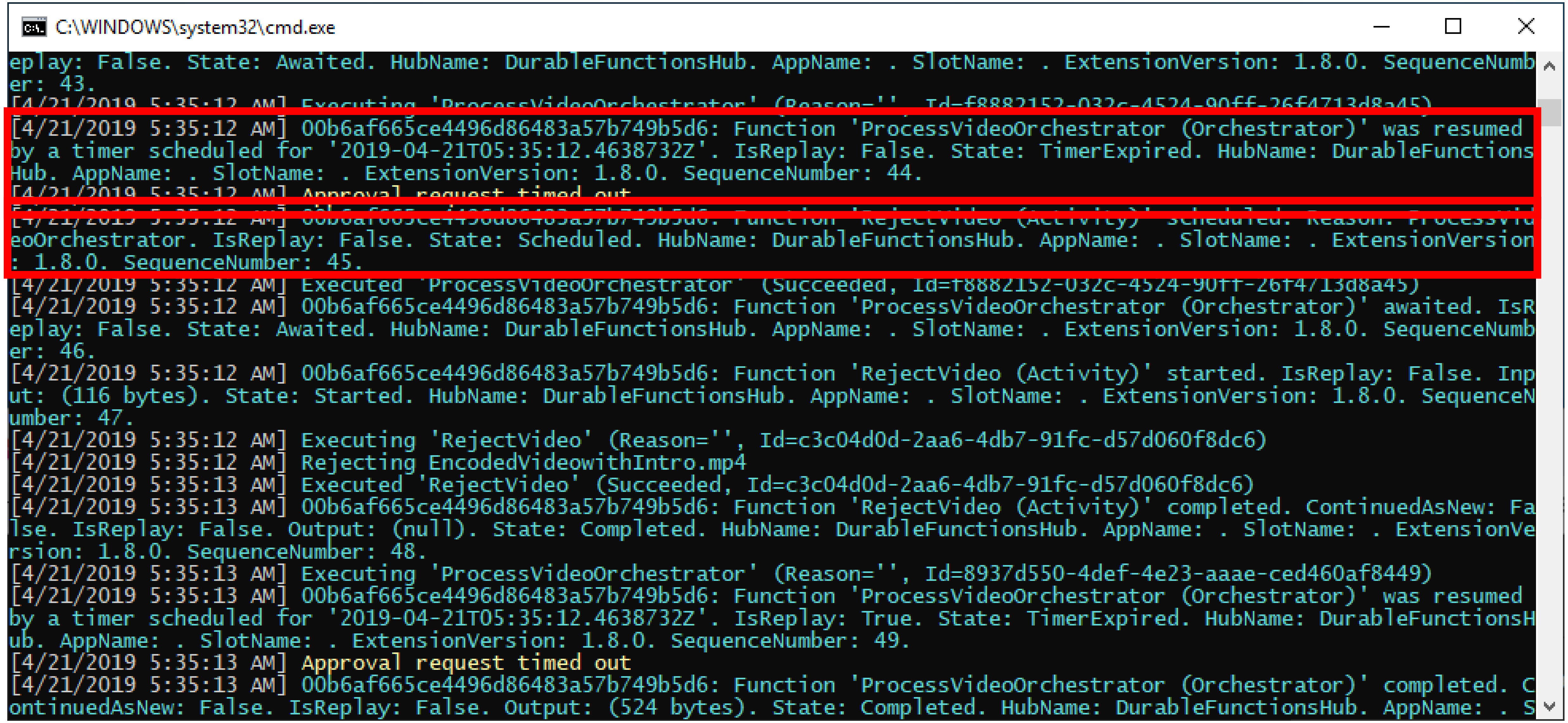
	KEY	VALUE	DESCRIPTION
✓	video	StirTrek.mp4	
	Key	Value	Description

Status: 202 Accepted Time: 1019 ms Size: 1.57 KB

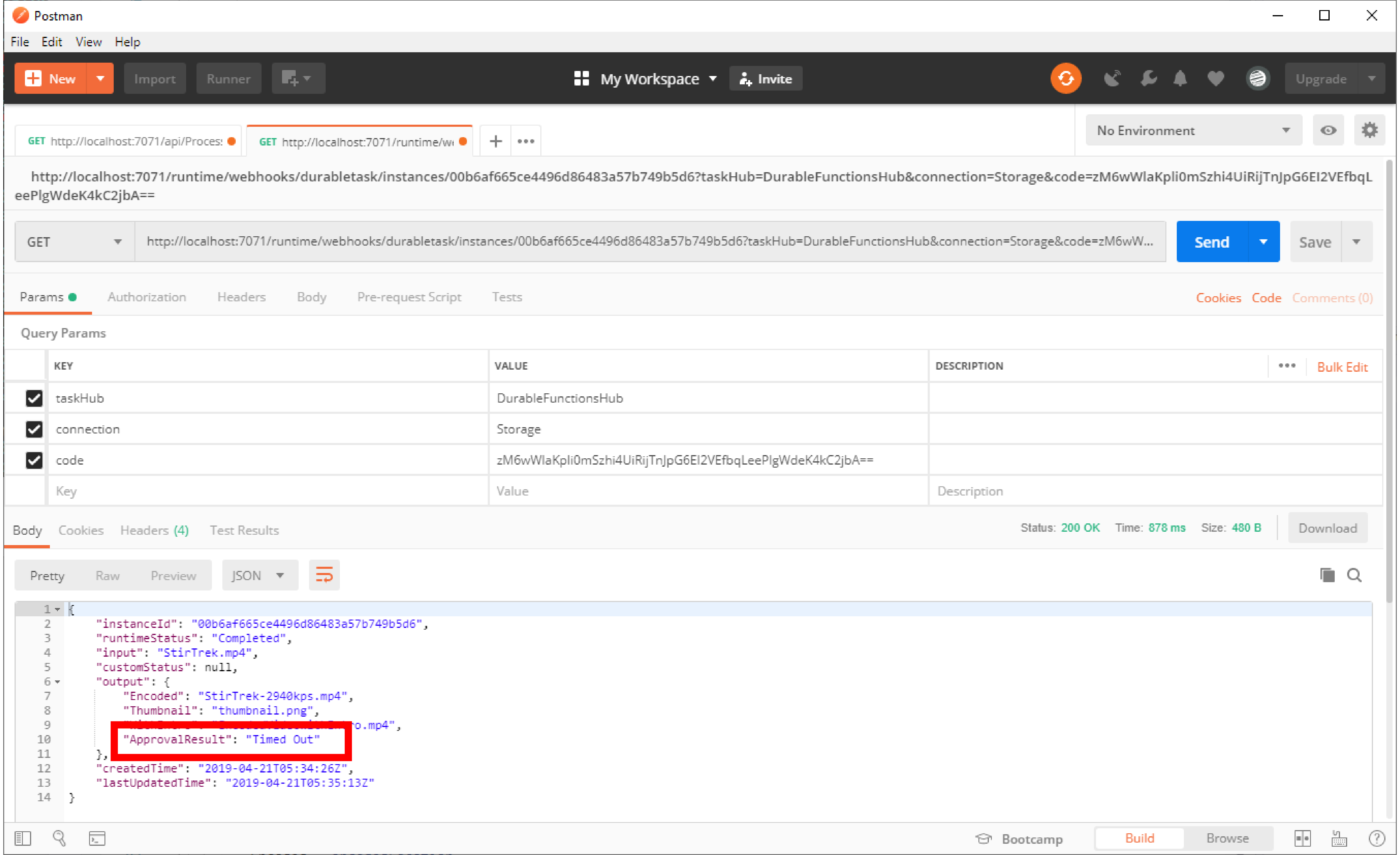
```
{
    "id": "00b6af665ce4496d86483a57b749b5d6",
    "statusQueryGetUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/00b6af665ce4496d86483a57b749b5d6?taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "sendEventPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/00b6af665ce4496d86483a57b749b5d6/raiseEvent/{eventName}?taskHub=DurableFunctionsHub&connection
        =Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "terminatePostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/00b6af665ce4496d86483a57b749b5d6/terminate?reason={text}&taskHub=DurableFunctionsHub&connection
        =Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "rewindPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/00b6af665ce4496d86483a57b749b5d6/rewind?reason={text}&taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==",
    "purgeHistoryDeleteUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/00b6af665ce4496d86483a57b749b5d6?taskHub=DurableFunctionsHub&connection=Storage&code
        =zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA=="
}
```



```
n: ProcessVideoOrchestrator. IsReplay: False. State: Scheduled. HubName: DurableFunctionsHub. AppName: . SlotName: . Ext
ensionVersion: 1.8.0. SequenceNumber: 37.
[4/21/2019 5:34:42 AM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=4dda13d3-c12f-48d5-9fb2-84a8e4b78b3d)
[4/21/2019 5:34:42 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 38.
[4/21/2019 5:34:42 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'SendApprovalRequestEmail (Activity)' started. IsRepla
y: False. Input: (816 bytes). State: Started. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8
.0. SequenceNumber: 39.
[4/21/2019 5:34:42 AM] Executing 'SendApprovalRequestEmail' (Reason='', Id=5e6539b0-edcf-453b-8f3e-bdb79a51ec77)
[4/21/2019 5:34:42 AM] Sending approval request for EncodedVideowithIntro.mp4
[4/21/2019 5:34:42 AM] Please review EncodedVideowithIntro.mp4<br/><a href="http://localhost:7071/api/SubmitVideoApprova
l/721a737b14d6435e883a13f7405a309f?result=Approved">Approve</a><br/><a href="http://localhost:7071/api/SubmitVideoApprov
al/721a737b14d6435e883a13f7405a309f?result=Rejected">Reject</a>
[4/21/2019 5:34:42 AM] Executed 'SendApprovalRequestEmail' (Succeeded, Id=5e6539b0-edcf-453b-8f3e-bdb79a51ec77)
[4/21/2019 5:34:42 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'SendApprovalRequestEmail (Activity)' completed. Conti
nuedAsNew: False. IsReplay: False. Output: (null). State: Completed. HubName: DurableFunctionsHub. AppName: . SlotName:
. ExtensionVersion: 1.8.0. SequenceNumber: 40.
[4/21/2019 5:34:42 AM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=5d576ac7-44f8-4ae6-b2d9-15499f9a2b5a)
[4/21/2019 5:34:42 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'ProcessVideoOrchestrator (Orchestrator)' is waiting f
or input. Reason: CreateTimer:2019-04-21T05:35:12.4638732Z. IsReplay: False. State: Listening. HubName: DurableFunctions
Hub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumber: 41.
[4/21/2019 5:34:42 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'ProcessVideoOrchestrator (Orchestrator)' is waiting f
or input. Reason: WaitForExternalEvent:ApprovalResult. IsReplay: False. State: Listening. HubName: DurableFunctionsHub.
AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumber: 42.
[4/21/2019 5:34:42 AM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=5d576ac7-44f8-4ae6-b2d9-15499f9a2b5a)
[4/21/2019 5:34:42 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 43.
```


C:\WINDOWS\system32\cmd.exe

```
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 43.
[4/21/2019 5:35:12 AM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=f8882152-032c-4524-90ff-26f4713d8a45)
[4/21/2019 5:35:12 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'ProcessVideoOrchestrator (Orchestrator)' was resumed
by a timer scheduled for '2019-04-21T05:35:12.4638732Z'. IsReplay: False. State: TimerExpired. HubName: DurableFunctions
Hub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumber: 44.
[4/21/2019 5:35:12 AM] Approval request timed out
[4/21/2019 5:35:12 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'RejectVideo (Activity)' scheduled. Reason: ProcessVid
eoOrchestrator. IsReplay: False. State: Scheduled. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion
: 1.8.0. SequenceNumber: 45.
[4/21/2019 5:35:12 AM] Executed 'ProcessVideoOrchestrator' (Succeeded, Id=f8882152-032c-4524-90ff-26f4713d8a45)
[4/21/2019 5:35:12 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'ProcessVideoOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 46.
[4/21/2019 5:35:12 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'RejectVideo (Activity)' started. IsReplay: False. Inp
ut: (116 bytes). State: Started. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceN
umber: 47.
[4/21/2019 5:35:12 AM] Executing 'RejectVideo' (Reason='', Id=c3c04d0d-2aa6-4db7-91fc-d57d060f8dc6)
[4/21/2019 5:35:12 AM] Rejecting EncodedVideowithIntro.mp4
[4/21/2019 5:35:13 AM] Executed 'RejectVideo' (Succeeded, Id=c3c04d0d-2aa6-4db7-91fc-d57d060f8dc6)
[4/21/2019 5:35:13 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'RejectVideo (Activity)' completed. ContinuedAsNew: Fa
lse. IsReplay: False. Output: (null). State: Completed. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVe
rsion: 1.8.0. SequenceNumber: 48.
[4/21/2019 5:35:13 AM] Executing 'ProcessVideoOrchestrator' (Reason='', Id=8937d550-4def-4e23-aaae-ced460af8449)
[4/21/2019 5:35:13 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'ProcessVideoOrchestrator (Orchestrator)' was resumed
by a timer scheduled for '2019-04-21T05:35:12.4638732Z'. IsReplay: True. State: TimerExpired. HubName: DurableFunctionsH
ub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumber: 49.
[4/21/2019 5:35:13 AM] Approval request timed out
[4/21/2019 5:35:13 AM] 00b6af665ce4496d86483a57b749b5d6: Function 'ProcessVideoOrchestrator (Orchestrator)' completed. C
ontinuedAsNew: False. IsReplay: False. Output: (524 bytes). State: Completed. HubName: DurableFunctionsHub. AppName: . S
```



http://localhost:7071/runtime/webhooks/durabletask/instances/00b6af665ce4496d86483a57b749b5d6?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==

GET http://localhost:7071/runtime/webhooks/durabletask/instances/00b6af665ce4496d86483a57b749b5d6?taskHub=DurableFunctionsHub&connection=Storage&code=zM6wW...

Query Params

	KEY	VALUE	DESCRIPTION
✓	taskHub	DurableFunctionsHub	
✓	connection	Storage	
✓	code	zM6wWlaKpli0mSzhi4UiRijTnJpG6EI2VEfbqLeePlgWdeK4kC2jbA==	
	Key	Value	Description

Status: 200 OK Time: 878 ms Size: 480 B

```
{
    "instanceId": "00b6af665ce4496d86483a57b749b5d6",
    "runtimeStatus": "Completed",
    "input": "StirTrek.mp4",
    "customStatus": null,
    "output": {
        "Encoded": "StirTrek-2940kps.mp4",
        "Thumbnail": "thumbnail.png",
        [illegible]o.mp4",
        "ApprovalResult": "Timed Out"
    },
    "createdTime": "2019-04-21T05:34:26Z",
    "lastUpdatedTime": "2019-04-21T05:35:13Z"
}
```

Eternal Orchestrations

Azure Durable Functions for Serverless .NET Orchestration

Implement a periodic clean-up task

- Call clean-up activity function
- Sleep for a while
- Call ContinueAsNew
- Loop indefinitely

```csharp
[FunctionName("PeriodicTaskOrchestrator")]
public static async Task<int> PeriodicTaskOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{
  var executionCount = context.GetInput<int>();
  executionCount++;
  if (!context.IsReplaying)
    log.LogInformation($"Starting the PeriodicTask activity {context.InstanceId}, {executionCount}");
  await context.CallActivityAsync("PeriodicActivity", executionCount);
  var nextExecution = context.CurrentUtcDateTime.AddSeconds(30);
  await context.CreateTimer(nextExecution, CancellationToken.None);
  context.ContinueAsNew(executionCount);
  return executionCount;
}
```

```
[FunctionName("PeriodicTaskOrchestrator")]
public static async Task<int>  PeriodicTaskOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{
  var executionCount = context.GetInput<int>();
  executionCount++;
  if (!context.IsReplaying)
    log.LogInformation($"Starting the PeriodicTask activity {context.InstanceId}, {executionCount}");
  await context.CallActivityAsync("PeriodicActivity", executionCount);
  var nextExecution = context.CurrentUtcDateTime.AddSeconds(30);
  await context.CreateTimer(nextExecution, CancellationToken.None);
  context.ContinueAsNew(executionCount);
  return executionCount;
}
```

```
[FunctionName(" PeriodicTaskOrchestrator ")]
public static async Task<int>  PeriodicTaskOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{
  var executionCount = context.GetInput<int>();
  executionCount++;
  if (!context.IsReplaying)
    log.LogInformation($"Starting the PeriodicTask activity {context.InstanceId}, {executionCount}");
  await context.CallActivityAsync("PeriodicActivity", executionCount);
  var nextExecution = context.CurrentUtcDateTime.AddSeconds(30);
  await context.CreateTimer(nextExecution, CancellationToken.None);
  context.ContinueAsNew(executionCount);
  return executionCount;
}
```

```
[FunctionName(" PeriodicTaskOrchestrator ")]
public static async Task<int>  PeriodicTaskOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{
  var executionCount = context.GetInput<int>();
  executionCount++;
  if (!context.IsReplaying)
    log.LogInformation($"Starting the PeriodicTask activity {context.InstanceId}, {executionCount}");
  await context.CallActivityAsync("PeriodicActivity", executionCount);
  var nextExecution = context.CurrentUtcDateTime.AddSeconds(30);
  await context.CreateTimer(nextExecution, CancellationToken.None);
  context.ContinueAsNew(executionCount);
  return executionCount;
}
```

```
[FunctionName("PeriodicActivity")]
public static void PeriodicActivity(
                    [ActivityTrigger] int executionCount,
                    ILogger log)
{
  log.LogWarning($"Running the periodic activity; executions: {executionCount}");
}
```

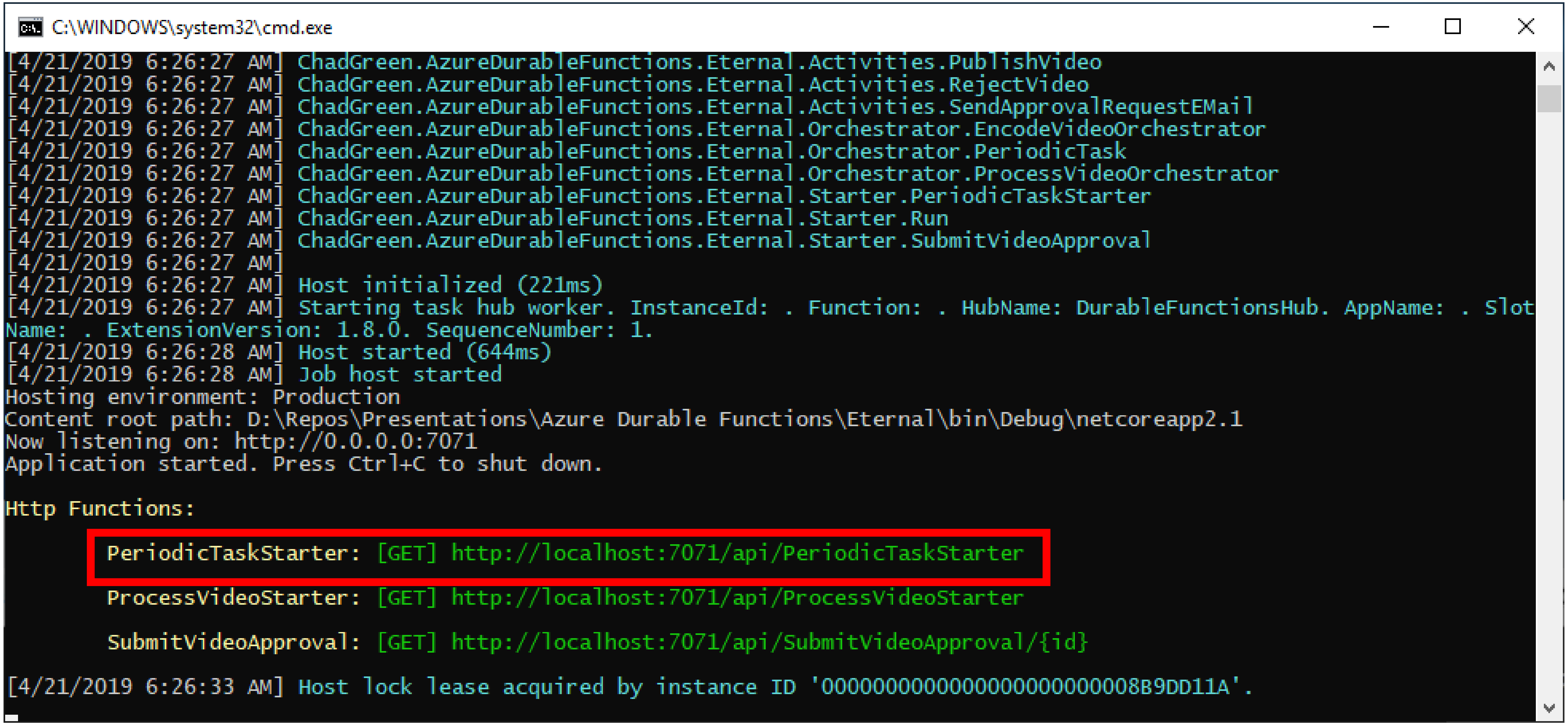
```
[FunctionName(" PeriodicTaskOrchestrator ")]
public static async Task<int>  PeriodicTaskOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{
  var executionCount = context.GetInput<int>();
  executionCount++;
  if (!context.IsReplaying)
    log.LogInformation($"Starting the PeriodicTask activity {context.InstanceId}, {executionCount}");
  await context.CallActivityAsync("PeriodicActivity", executionCount);
  var nextExecution = context.CurrentUtcDateTime.AddSeconds(30);
  await context.CreateTimer(nextExecution, CancellationToken.None);
  context.ContinueAsNew(executionCount);
  return executionCount;
}
```

```
[FunctionName(" PeriodicTaskOrchestrator ")]
public static async Task<int>  PeriodicTaskOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{
  var executionCount = context.GetInput<int>();
  executionCount++;
  if (!context.IsReplaying)
    log.LogInformation($"Starting the PeriodicTask activity {context.InstanceId}, {executionCount}");
  await context.CallActivityAsync("PeriodicActivity", executionCount);
  var nextExecution = context.CurrentUtcDateTime.AddSeconds(30);
  await context.CreateTimer(nextExecution, CancellationToken.None);
  context.ContinueAsNew(executionCount);
  return executionCount;
}
```

```
[FunctionName(" PeriodicTaskOrchestrator ")]
public static async Task<int>  PeriodicTaskOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{
  var executionCount = context.GetInput<int>();
  executionCount++;
  if (!context.IsReplaying)
    log.LogInformation($"Starting the PeriodicTask activity {context.InstanceId}, {executionCount}");
  await context.CallActivityAsync("PeriodicActivity", executionCount);
  var nextExecution = context.CurrentUtcDateTime.AddSeconds(30);
  await context.CreateTimer(nextExecution, CancellationToken.None);
  context.ContinueAsNew(executionCount);
  return executionCount;
}
```

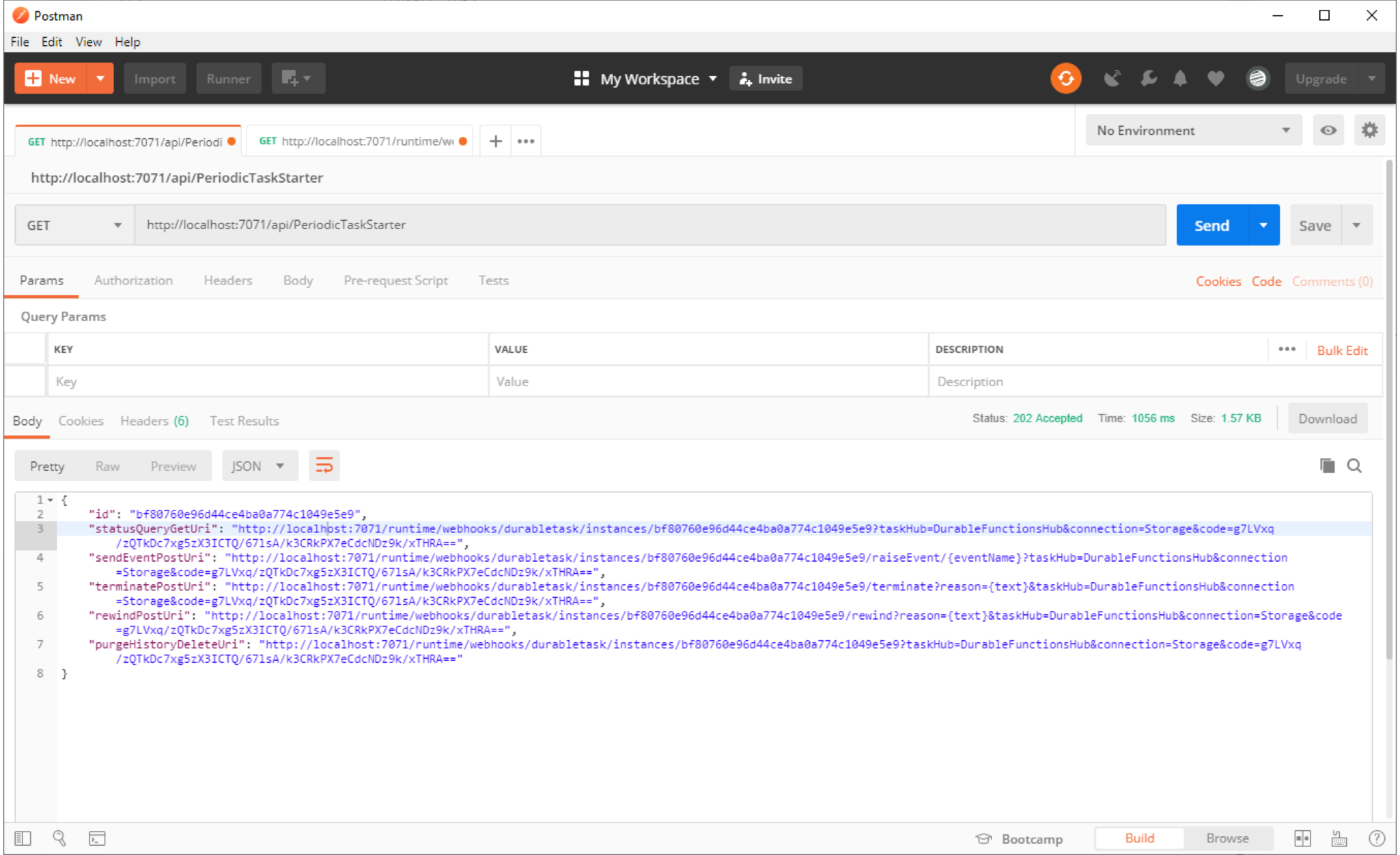
```
[FunctionName(" PeriodicTaskOrchestrator ")]
public static async Task<int>  PeriodicTaskOrchestrator(
                    [OrchestrationTrigger] DurableOrchestrationContext context,
                    ILogger log)
{
  var executionCount = context.GetInput<int>();
  executionCount++;
  if (!context.IsReplaying)
    log.LogInformation($"Starting the PeriodicTask activity {context.InstanceId}, {executionCount}");
  await context.CallActivityAsync("PeriodicActivity", executionCount);
  var nextExecution = context.CurrentUtcDateTime.AddSeconds(30);
  await context.CreateTimer(nextExecution, CancellationToken.None);
  context.ContinueAsNew(executionCount);
  return executionCount;
}
```

```
[FunctionName("PeriodicTaskStarter")]
public static async Task<HttpResponseMessage> PeriodicTaskStarter(
                    [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequestMessage req,
                    [OrchestrationClient] DurableOrchestrationClient client,
                    ILogger log)
{
  var instanceId = await client.StartNewAsync("PeriodicTaskOrchestrator", 0);
  return client.CreateCheckStatusResponse(req, instanceId);
}
```

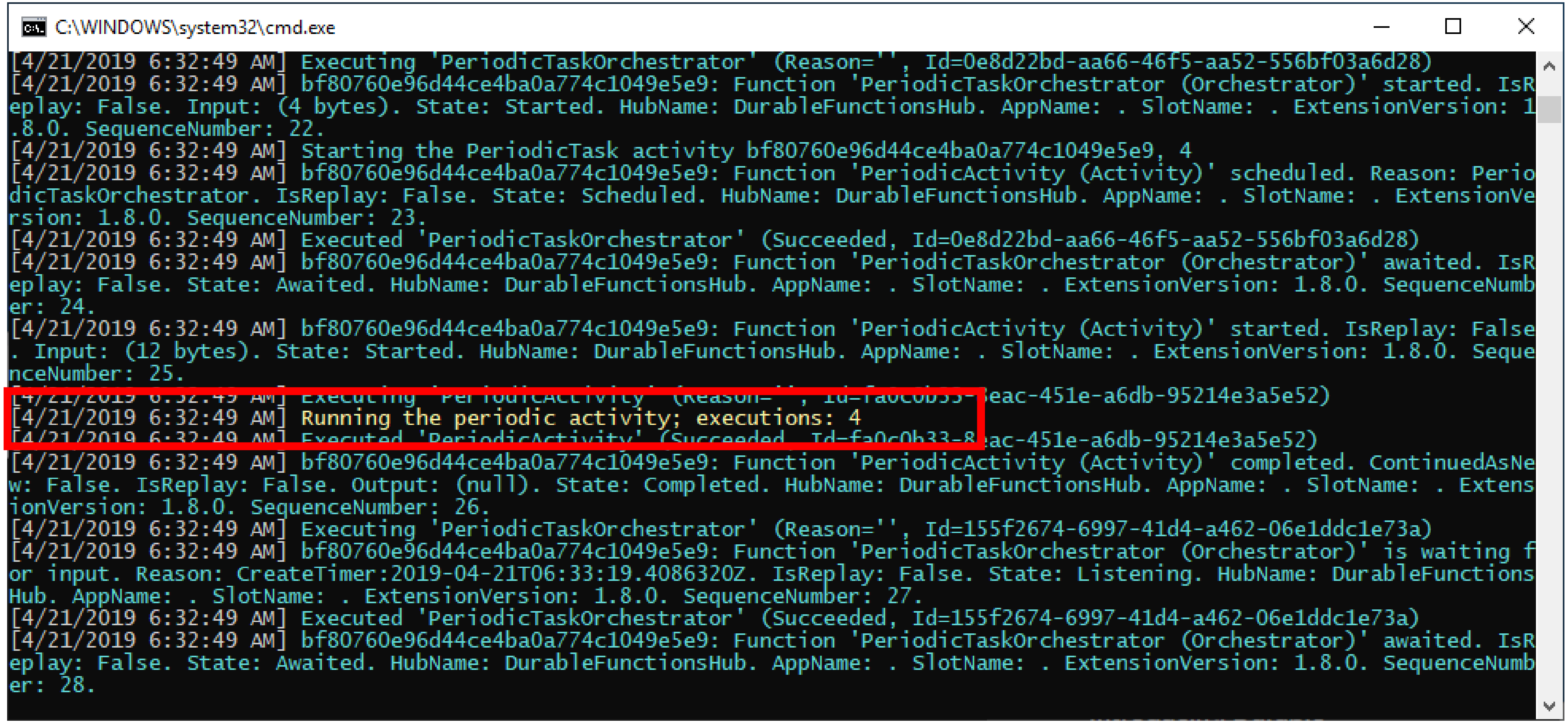


C:\WINDOWS\system32\cmd.exe

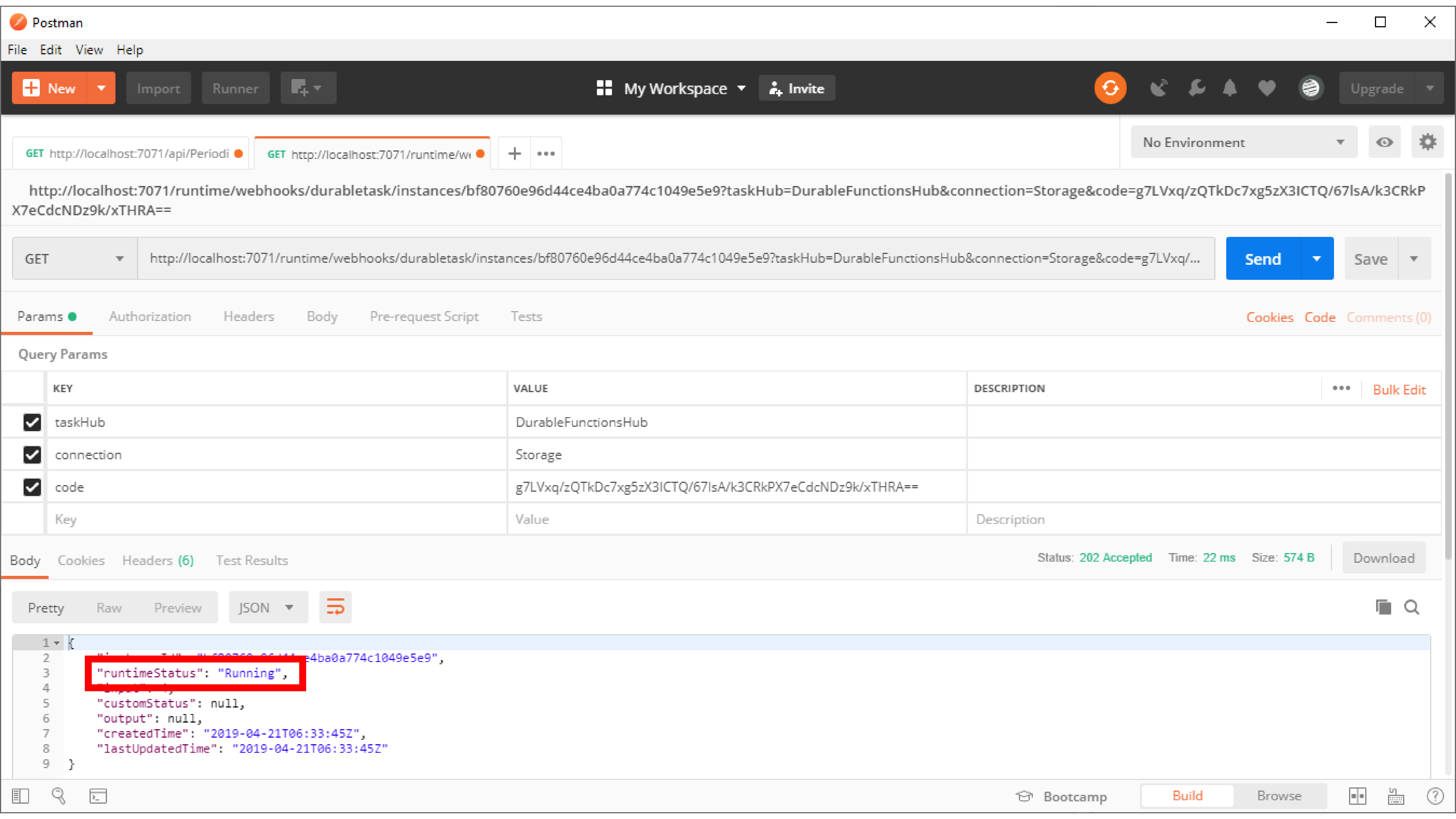
```
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Activities.PublishVideo
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Activities.RejectVideo
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Activities.SendApprovalRequestEMail
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Orchestrator.EncodeVideoOrchestrator
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Orchestrator.PeriodicTask
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Orchestrator.ProcessVideoOrchestrator
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Starter.PeriodicTaskStarter
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Starter.Run
[4/21/2019 6:26:27 AM] ChadGreen.AzureDurableFunctions.Eternal.Starter.SubmitVideoApproval
[4/21/2019 6:26:27 AM]
[4/21/2019 6:26:27 AM] Host initialized (221ms)
[4/21/2019 6:26:27 AM] Starting task hub worker. InstanceId: . Function: . HubName: DurableFunctionsHub. AppName: . Slot
Name: . ExtensionVersion: 1.8.0. SequenceNumber: 1.
[4/21/2019 6:26:28 AM] Host started (644ms)
[4/21/2019 6:26:28 AM] Job host started
Hosting environment: Production
Content root path: D:\Repos\Presentations\Azure Durable Functions\Eternal\bin\Debug\netcoreapp2.1
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.

Http Functions:

        PeriodicTaskStarter: [GET] http://localhost:7071/api/PeriodicTaskStarter

        ProcessVideoStarter: [GET] http://localhost:7071/api/ProcessVideoStarter

        SubmitVideoApproval: [GET] http://localhost:7071/api/SubmitVideoApproval/{id}

[4/21/2019 6:26:33 AM] Host lock lease acquired by instance ID '000000000000000000000000008B9DD11A'.
```



```
{
    "id": "bf80760e96d44ce4ba0a774c1049e5e9",
    "statusQueryGetUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9?taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq
        /zQTkDc7xg5zX3ICTQ/671sA/k3CRkPX7eCdcNDz9k/xTHRA==",
    "sendEventPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9/raiseEvent/{eventName}?taskHub=DurableFunctionsHub&connection
        =Storage&code=g7LVxq/zQTkDc7xg5zX3ICTQ/671sA/k3CRkPX7eCdcNDz9k/xTHRA==",
    "terminatePostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9/terminate?reason={text}&taskHub=DurableFunctionsHub&connection
        =Storage&code=g7LVxq/zQTkDc7xg5zX3ICTQ/671sA/k3CRkPX7eCdcNDz9k/xTHRA==",
    "rewindPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9/rewind?reason={text}&taskHub=DurableFunctionsHub&connection=Storage&code
        =g7LVxq/zQTkDc7xg5zX3ICTQ/671sA/k3CRkPX7eCdcNDz9k/xTHRA==",
    "purgeHistoryDeleteUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9?taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq
        /zQTkDc7xg5zX3ICTQ/671sA/k3CRkPX7eCdcNDz9k/xTHRA=="
}
```



```
[4/21/2019 6:32:49 AM] Executing 'PeriodicTaskOrchestrator' (Reason='', Id=0e8d22bd-aa66-46f5-aa52-556bf03a6d28)
[4/21/2019 6:32:49 AM] bf80760e96d44ce4ba0a774c1049e5e9: Function 'PeriodicTaskOrchestrator (Orchestrator)' started. IsR
eplay: False. Input: (4 bytes). State: Started. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1
.8.0. SequenceNumber: 22.
[4/21/2019 6:32:49 AM] Starting the PeriodicTask activity bf80760e96d44ce4ba0a774c1049e5e9, 4
[4/21/2019 6:32:49 AM] bf80760e96d44ce4ba0a774c1049e5e9: Function 'PeriodicActivity (Activity)' scheduled. Reason: Perio
dicTaskOrchestrator. IsReplay: False. State: Scheduled. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVe
rsion: 1.8.0. SequenceNumber: 23.
[4/21/2019 6:32:49 AM] Executed 'PeriodicTaskOrchestrator' (Succeeded, Id=0e8d22bd-aa66-46f5-aa52-556bf03a6d28)
[4/21/2019 6:32:49 AM] bf80760e96d44ce4ba0a774c1049e5e9: Function 'PeriodicTaskOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 24.
[4/21/2019 6:32:49 AM] bf80760e96d44ce4ba0a774c1049e5e9: Function 'PeriodicActivity (Activity)' started. IsReplay: False
. Input: (12 bytes). State: Started. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. Seque
nceNumber: 25.
[4/21/2019 6:32:49 AM] Executing 'PeriodicActivity' (Reason='', Id=fa0c0b33-8eac-451e-a6db-95214e3a5e52)
[4/21/2019 6:32:49 AM] Running the periodic activity; executions: 4
[4/21/2019 6:32:49 AM] Executed 'PeriodicActivity' (Succeeded, Id=fa0c0b33-8eac-451e-a6db-95214e3a5e52)
[4/21/2019 6:32:49 AM] bf80760e96d44ce4ba0a774c1049e5e9: Function 'PeriodicActivity (Activity)' completed. ContinuedAsNe
w: False. IsReplay: False. Output: (null). State: Completed. HubName: DurableFunctionsHub. AppName: . SlotName: . Extens
ionVersion: 1.8.0. SequenceNumber: 26.
[4/21/2019 6:32:49 AM] Executing 'PeriodicTaskOrchestrator' (Reason='', Id=155f2674-6997-41d4-a462-06e1ddc1e73a)
[4/21/2019 6:32:49 AM] bf80760e96d44ce4ba0a774c1049e5e9: Function 'PeriodicTaskOrchestrator (Orchestrator)' is waiting f
or input. Reason: CreateTimer:2019-04-21T06:33:19.4086320Z. IsReplay: False. State: Listening. HubName: DurableFunctions
Hub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumber: 27.
[4/21/2019 6:32:49 AM] Executed 'PeriodicTaskOrchestrator' (Succeeded, Id=155f2674-6997-41d4-a462-06e1ddc1e73a)
[4/21/2019 6:32:49 AM] bf80760e96d44ce4ba0a774c1049e5e9: Function 'PeriodicTaskOrchestrator (Orchestrator)' awaited. IsR
eplay: False. State: Awaited. HubName: DurableFunctionsHub. AppName: . SlotName: . ExtensionVersion: 1.8.0. SequenceNumb
er: 28.
```



http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9?taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq/zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==

GET http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9?taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq/...

Query Params

	KEY	VALUE	DESCRIPTION
✓	taskHub	DurableFunctionsHub	
✓	connection	Storage	
✓	code	g7LVxq/zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==	
	Key	Value	Description

Status: 202 Accepted Time: 22 ms Size: 574 B

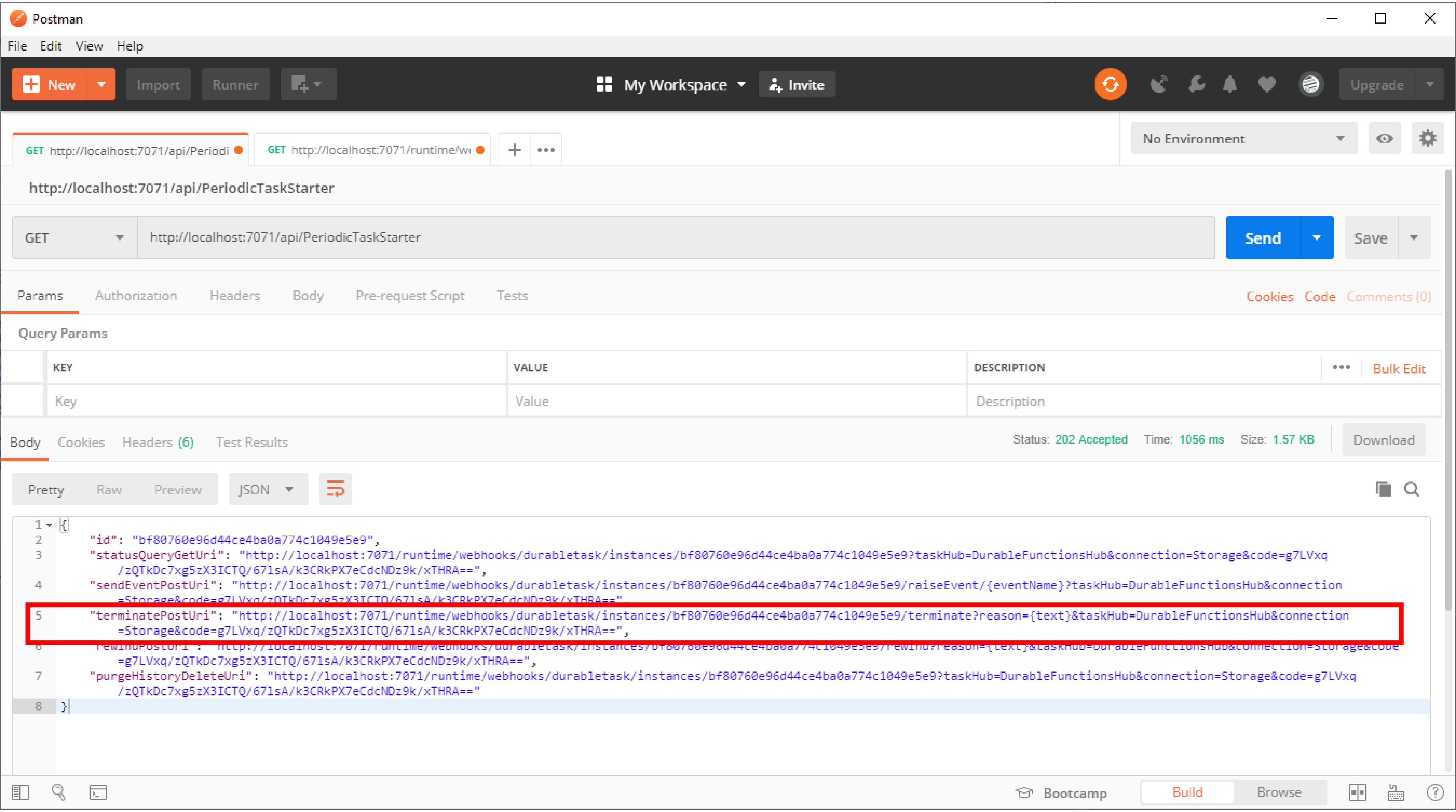
```
{
    "instanceId": "bf80760e96d44ce4ba0a774c1049e5e9",
    "runtimeStatus": "Running",
    "input": {},
    "customStatus": null,
    "output": null,
    "createdTime": "2019-04-21T06:33:45Z",
    "lastUpdatedTime": "2019-04-21T06:33:45Z"
}
```

Exiting Eternal Orchestrations

- **Restarting the Function App will not stop the orchestration**
- **Ways an eternal orchestration will stop**
    - Do not call ContinueAsNew
    - Unhandled excceptions
    - Use the termination API

Terminate an eternal orchestration

- Use the termination API to stop an eternal orchestration


Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
Upgrade
GET http://localhost:7071/api/Periodi
GET http://localhost:7071/runtime/w
No Environment
Send
Save
Params Authorization Headers Body Pre-request Script Tests
Cookies Code Comments (0)
Bulk Edit
Body Cookies Headers (6) Test Results
Download
Pretty Raw Preview JSON
Bootcamp
Build Browse

http://localhost:7071/api/PeriodicTaskStarter

GET http://localhost:7071/api/PeriodicTaskStarter

Query Params

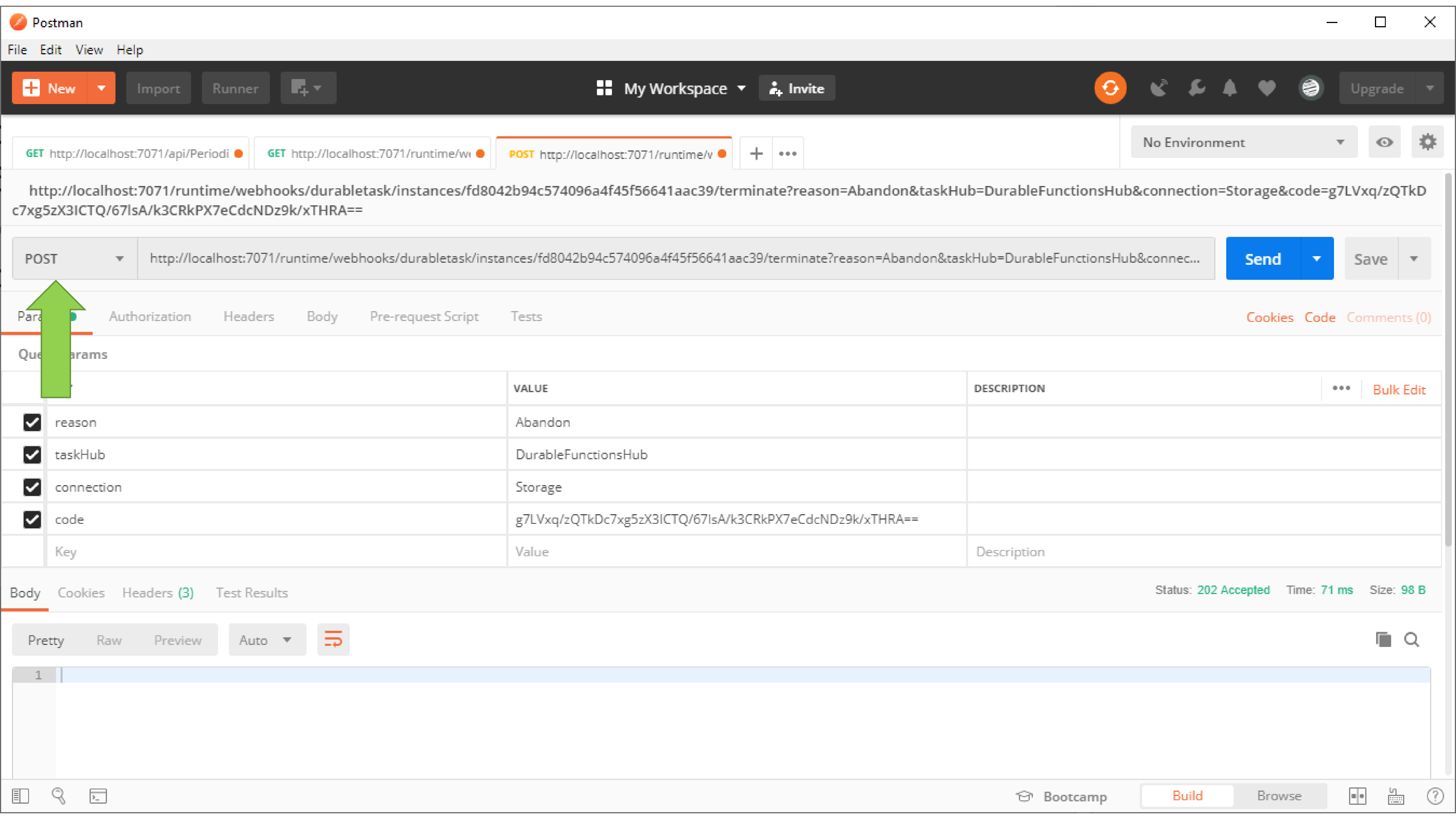
KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 202 Accepted Time: 1056 ms Size: 1.57 KB

```
{
    "id": "bf80760e96d44ce4ba0a774c1049e5e9",
    "statusQueryGetUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9?taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq
        /zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==",
    "sendEventPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9/raiseEvent/{eventName}?taskHub=DurableFunctionsHub&connection
        =Storage&code=g7LVxq/zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==",
    "terminatePostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9/terminate?reason={text}&taskHub=DurableFunctionsHub&connection
        =Storage&code=g7LVxq/zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==",
    "rewindPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9/rewind?reason={text}&taskHub=DurableFunctionsHub&connection=Storage&code
        =g7LVxq/zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==",
    "purgeHistoryDeleteUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bf80760e96d44ce4ba0a774c1049e5e9?taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq
        /zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA=="
}
```

http://localhost:7071/runtime/webhooks/durabletask/instances/fd8042b94c574096a4f45f56641aac39/terminate?reason={text}&taskHub=DurableFunctionsHub&connection=Storage&code=d7LV7LV/zQTkOTk7Dc5xX5LCB0C/T7l/s6A/k3AlRk3X7ePXlcNDz9kD/zDk/xA+RA==

Abandon&taskHub=DurableFunctionsHub&connection=Storage&code=[illegible]

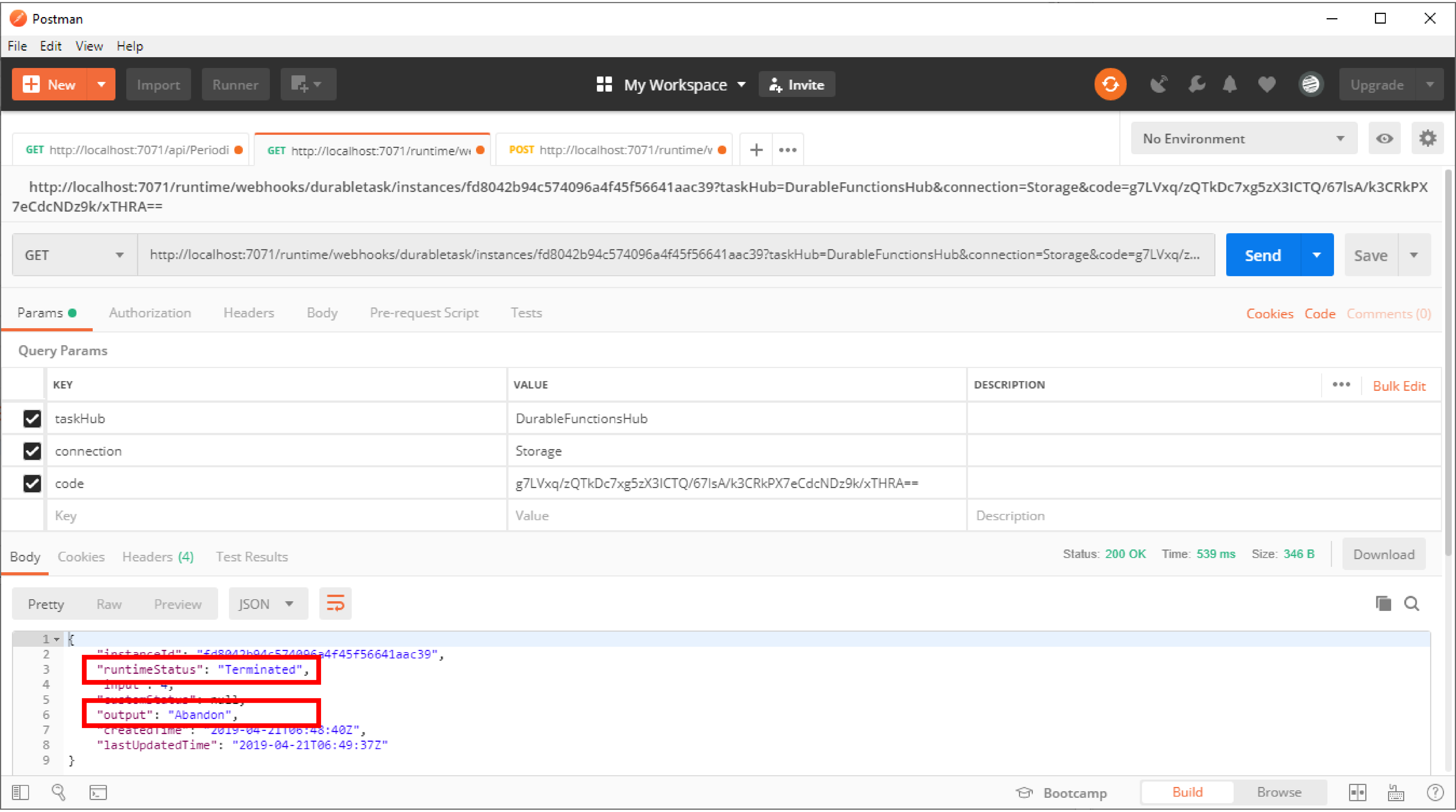

Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
Upgrade
GET http://localhost:7071/api/Periodi
GET http://localhost:7071/runtime/w
POST http://localhost:7071/runtime/v
No Environment

http://localhost:7071/runtime/webhooks/durabletask/instances/fd8042b94c574096a4f45f56641aac39/terminate?reason=Abandon&taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq/zQTkDc7xg5zX3lCTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==

POST http://localhost:7071/runtime/webhooks/durabletask/instances/fd8042b94c574096a4f45f56641aac39/terminate?reason=Abandon&taskHub=DurableFunctionsHub&connec...

	KEY	VALUE	DESCRIPTION
✓	reason	Abandon	
✓	taskHub	DurableFunctionsHub	
✓	connection	Storage	
✓	code	g7LVxq/zQTkDc7xg5zX3lCTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==	
	Key	Value	Description

Status: 202 Accepted Time: 71 ms Size: 98 B



Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
Upgrade
GET http://localhost:7071/api/Periodi
GET http://localhost:7071/runtime/w
POST http://localhost:7071/runtime/v
No Environment

http://localhost:7071/runtime/webhooks/durabletask/instances/fd8042b94c574096a4f45f56641aac39?taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq/zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==

GET http://localhost:7071/runtime/webhooks/durabletask/instances/fd8042b94c574096a4f45f56641aac39?taskHub=DurableFunctionsHub&connection=Storage&code=g7LVxq/z...

Query Params

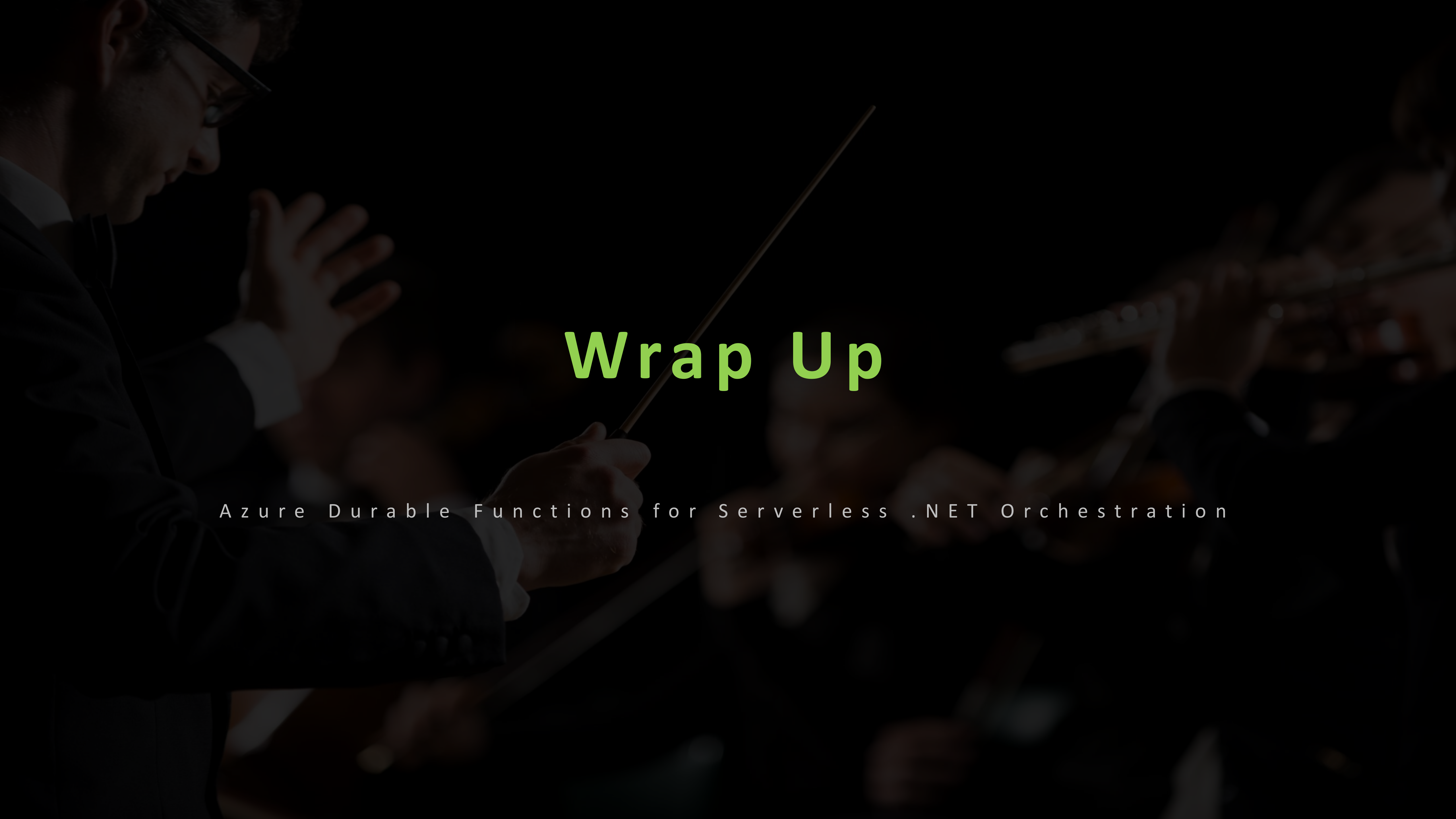
	KEY	VALUE	DESCRIPTION
✓	taskHub	DurableFunctionsHub	
✓	connection	Storage	
✓	code	g7LVxq/zQTkDc7xg5zX3ICTQ/67lsA/k3CRkPX7eCdcNDz9k/xTHRA==	
	Key	Value	Description

Status: 200 OK Time: 539 ms Size: 346 B

```
{
    "instanceId": "fd8042b94c574096a4f45f56641aac39",
    "runtimeStatus": "Terminated",
    "input": 4,
    "customStatus": null,
    "output": "Abandon",
    "createdTime": "2019-04-21T06:48:40Z",
    "lastUpdatedTime": "2019-04-21T06:49:37Z"
}
```

Eternal Orchestrations Versus Timers

- Pass data from the previous invocation to the next
- Can exit the orchestration if required
- Can very the interval between invocations
- Allow multiple concurrent instances of the workflow



Wrap Up

Azure Durable Functions for Serverless .NET Orchestration

Wrap Up – Durable Functions

- FaaS – Single Responsibility; Short Lived; Stateless; Event Driven & Scalable
- Define your workflows in code
- Other functions can be called both synchronously and asynchronously
- Output from called functions can be saved to local variables
- Progress is automatically checkpointed when the function awaits
- Application Patterns
  - Chaining
  - Fan-out/fan-in
  - Human Interaction/External
  - Eternal

Thank You

chadgreen@chadgreen.com

chadgreen.com

ChadGreen

ChadwickEGreen