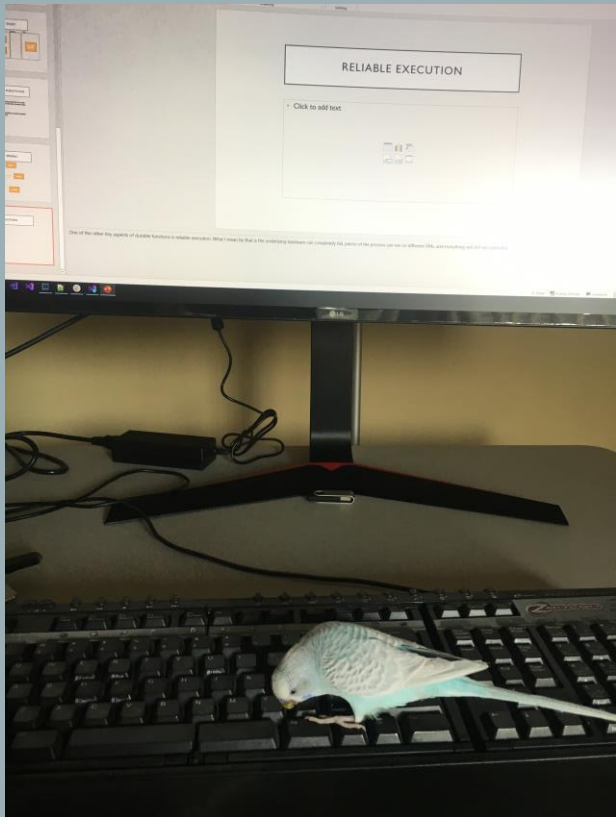


LONG RUNNING SERVERLESS TASKS WITH AZURE DURABLE FUNCTIONS

BRETT HAZEN

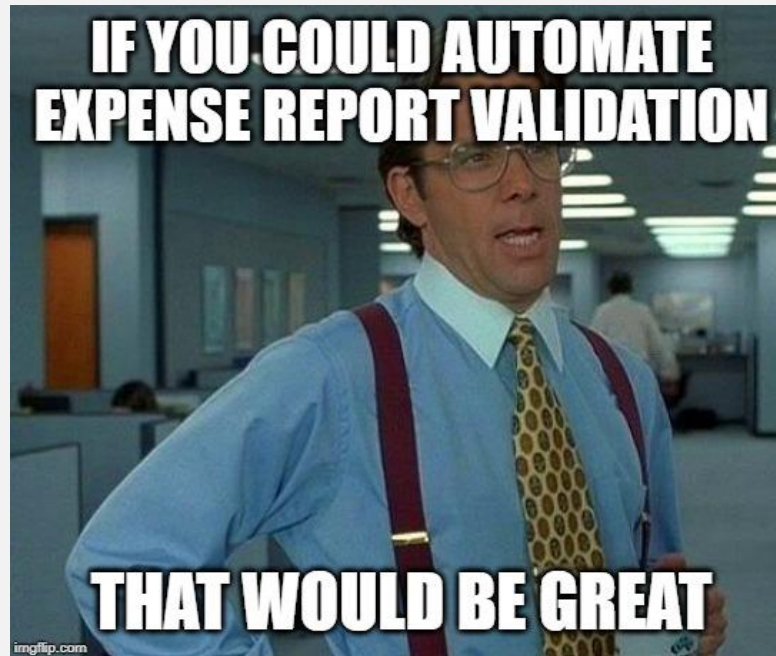


- Principal Consultant at ILM Professional Services
- @BrettEHazen
- brett.hazen@ilmservice.com

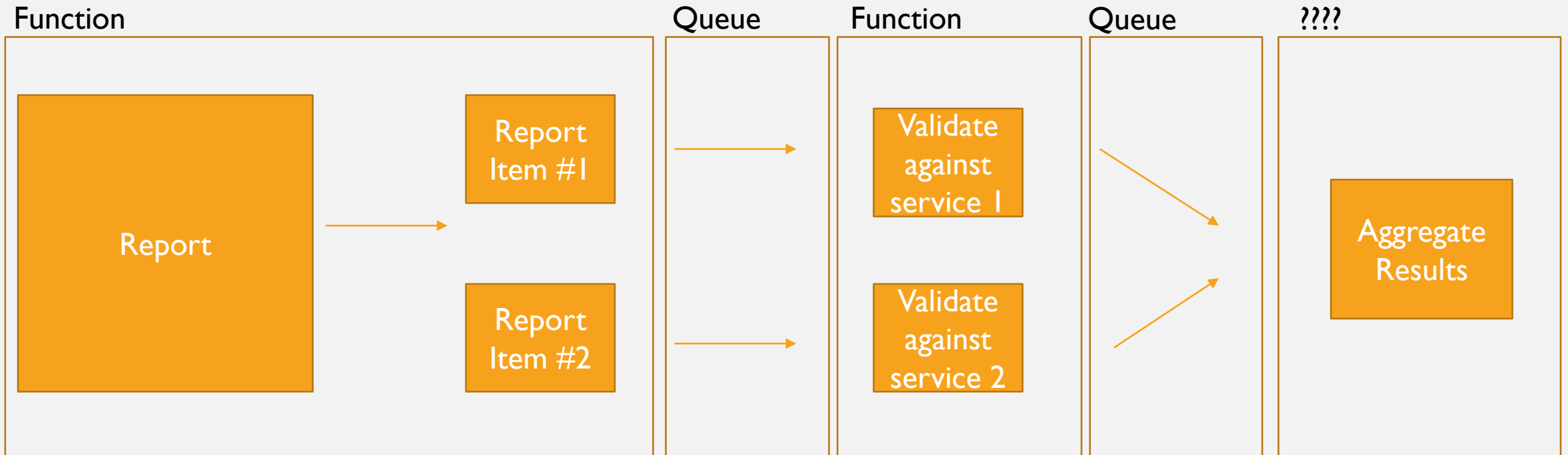
OUR SCENARIO

- Internal tool exists for tracking expense reports
- It's serverless and life is good

A NEW REQUIREMENT EMERGES!



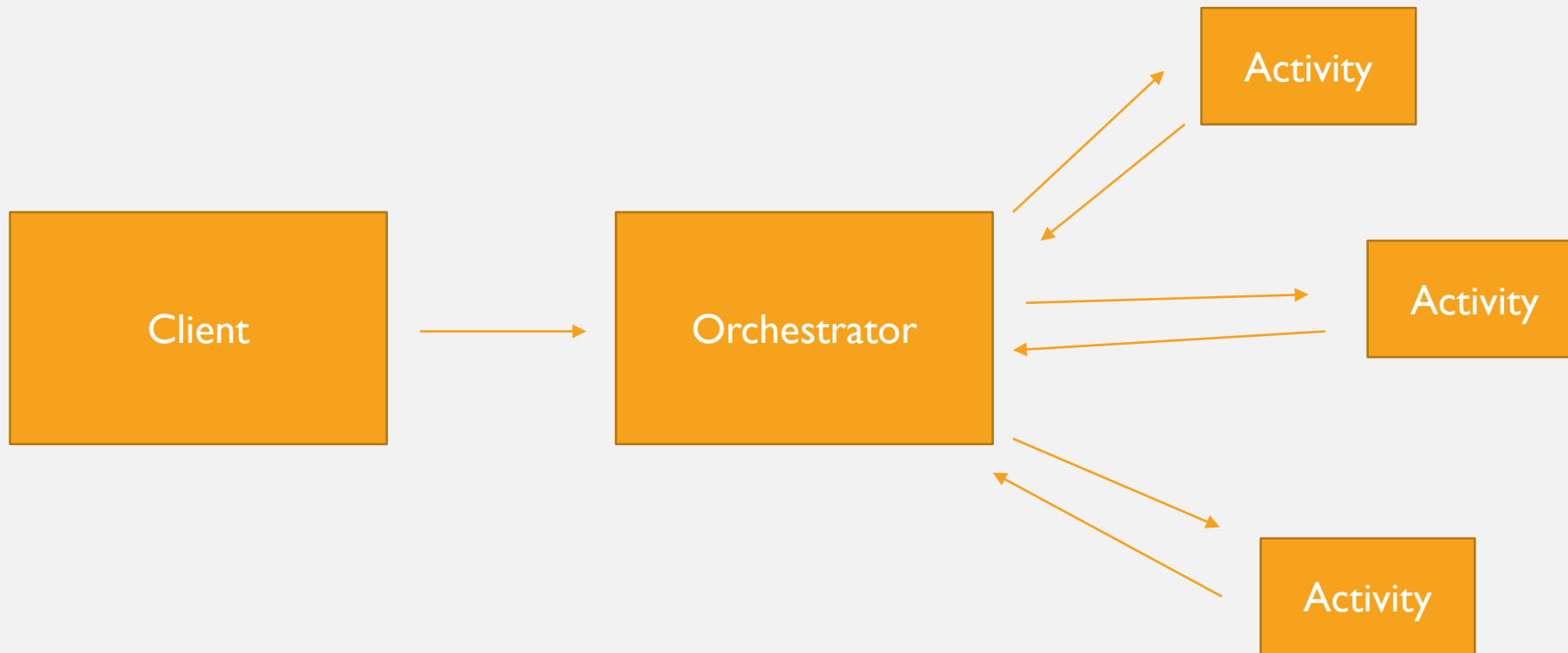
HOW COULD WE DO THAT?



ENTER AZURE DURABLE FUNCTIONS

- Durable Functions are an extension of Azure Functions that lets you write stateful functions in a serverless environment. The extension manages state, checkpoints, and restarts for you.
- The primary use case for Durable Functions is simplifying complex, stateful coordination requirements in serverless applications

ANATOMY OF A DURABLE FUNCTION



RELIABLE EXECUTION

```
[FunctionName("ReliableExecution")]  
0 references | 0 changes | 0 authors, 0 changes  
public static async Task<List<string>> RunOrchestrator(  
    [OrchestrationTrigger] DurableOrchestrationContext context)  
{  
    var outputs = new List<string>();  
  
    // Replace "hello" with the name of your Durable Activity Function.  
    outputs.Add(await context.CallActivityAsync<string>("ReliableExecution_Hello", "Tokyo"));  
    outputs.Add(await context.CallActivityAsync<string>("ReliableExecution_Hello", "Seattle"));  
    outputs.Add(await context.CallActivityAsync<string>("ReliableExecution_Hello", "London"));  
  
    // returns ["Hello Tokyo!", "Hello Seattle!", "Hello London!"]  
    return outputs;  
}
```

The diagram illustrates the flow of data from a code block to a History Table. On the left, four vertical orange arrows point downwards, indicating the execution of the code. On the right, three horizontal orange arrows point from the code block towards an orange box labeled "History Table", representing the storage of execution history.

History Table

ORCHESTRATOR CONSTRAINTS

- Must be deterministic
- Should be non-blocking
- Must never initiate async operations except using the `DurableOrchestrationContext` API or equivalent
- Infinite loops should be avoided
- JavaScript orchestrator functions cannot be async

TASK HUBS

- One history table
- One instance table
- One work-item queue
- One or more control queues
- One storage container containing one or more lease blobs

ERROR HANDLING

- Write try/catch blocks as you normally would
 - Be sure orchestrator is deterministic
- DurableOrchestrationContext API has built-in retry mechanism

OTHER FEATURES

- External Events
- Timers

VERSIONING

- Do nothing
- Stop all in-flight instances
- Side-by-side deployments
- Version in code

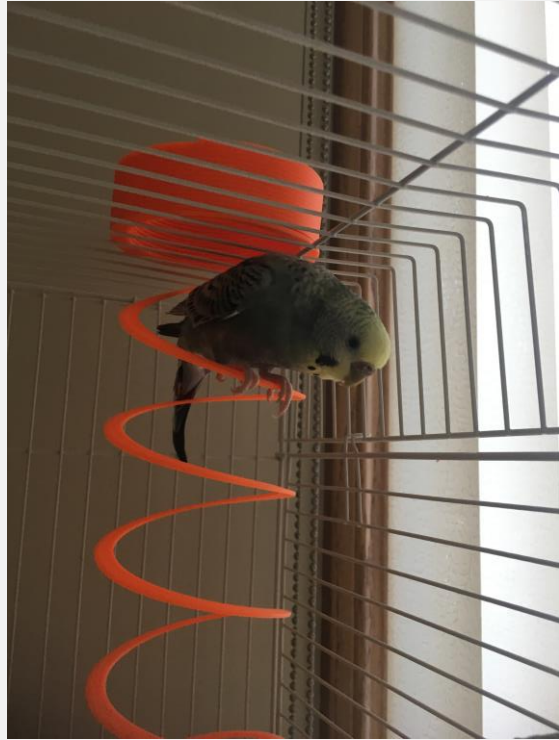
THINGS WE DIDN'T COVER

- Sub-orchestrators
- Eternal Orchestrators
- Unit testing
- Singleton Orchestrations
- Concurrency Throttling
- Instance Management

SOME FINAL THOUGHTS

- Be sure everyone understands orchestrators must be deterministic
- Figure out logging early
- Determine versioning strategy early
- Separate storage account or task hub per durable function

QUESTIONS



THANK YOU

@BrettEHazen

brett.hazen@ilmservice.com

<https://github.com/bhazen/durable-functions-talk>

RESOURCES

- <https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview>
- <https://mikhail.io/2018/12/making-sense-of-azure-durable-functions/>
- <https://medium.com/@tsuyoshiushio/durable-functions-101-35aa3919f182>
- <https://markheath.net/post/10-reasons-durable-functions>
- <https://www.freecodecamp.org/news/an-introduction-to-azure-durable-functions-patterns-and-best-practices-b1939ae6c717/>