Implementing Eternal Orchestrations

Mark Heath
MICROSOFT MVP

@mark_heath www.markheath.net

Looping in Orchestrator Functions

```
public static async Task MyOrchestrator(
      [OrchestrationTrigger] DurableOrchestrationContext ctx)
{
   while (true)
   {
      await ctx.CallActivityAsync<string>("MyActivity");
      await ctx.WaitForExternalEvent<string>("MyEvent");
   }
}
```

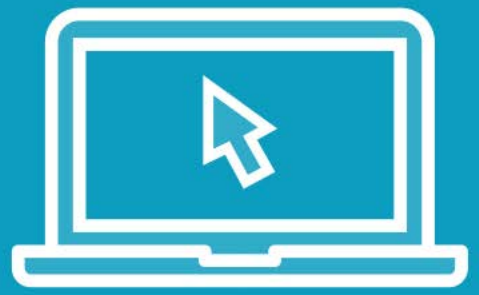
Orchestrator history is stored using “event sourcing”

The list of events will grow indefinitely

Call **ContinueAsNew** to restart the orchestration from the beginning

Can pass input data to the new instance

Demo



Implement a periodic clean-up task

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

Exiting Eternal Orchestrations

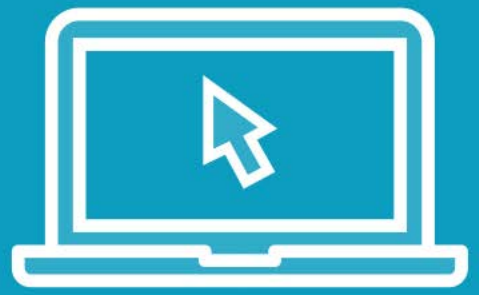
Restarting the Function App won't stop the orchestration

Don't call ContinueAsNew

Unhandled exception

Use the termination API

Demo



Terminate an eternal orchestration

Eternal Orchestrations Versus Timers

Eternal Orchestrations

Pass data from the previous invocation to the next

Can exit the orchestration if required

Can vary the interval between invocations

Allow multiple concurrent instances of the workflow

Alternative Uses of CreateAsNew

```
public static async Task MyOrchestrator(
      [OrchestrationTrigger] DurableOrchestrationContext ctx)
{
   await ctx.CallActivityAsync<string>("MyActivity");
   var ev = await ctx.WaitForExternalEvent<string>("MyEvent");
   ctx.ContinueAsNew(ev);
}
```

Alternative Uses of CreateAsNew

```
public static async Task<int> StatefulSingleton(
      [OrchestrationTrigger] DurableOrchestrationContext ctx)
{
   var state = ctx.GetInput<int>();
   var op = await ctx.WaitForExternalEvent<string>("MyEvent");
   if (op == "inc") state++;
   else if (op == "dec") state--;
   if (op != "exit")
   {
      ctx.ContinueAsNew(state);
   }
   return state;
}
```

Beware! Some external events can be missed – avoid this pattern for now

Message loss due to race conditions with ContinueAsNew #67

New issue

Open **cgillum** opened this issue on 22 Sep 2017 · 12 comments

cgillum commented on 22 Sep 2017 — Collaborator

Summary

Message loss or instance termination may be observed if an orchestrator completes after calling `ContinueAsNew` and subsequently processes any other message before restarting.

Details

When the orchestrator completes execution after `ContinueAsNew` is called, the status is internally marked as "completed" and a new message is enqueued to restart it. During this window between completion and restart, it's possible for other messages to arrive (for example, raised events or termination messages). Because the internal state of the orchestration is completed, those messages will be dropped. It's also possible for the DTFx runtime to terminate the instance claiming possible state corruption.

Repro

1. Start with the counter sample
2. Create a new instance
3. Call RaiseEventAsync("operation", "incr") multiple times without waiting

Assignees

No one assigned

Labels

bug

dtfx

Projects

None yet

Milestone

No milestone

8 participants

https://github.com/Azure/azure-functions-durable-extension/issues/67

Summary

ContinueAsNew

- Long-running or eternal workflows

Implementing periodic tasks

- Exiting the periodic workflow
- Maintain state between invocations

Combine with WaitForExternalEvent

- Avoid stateful singleton pattern

Up next ...

Using Durable Functions in Production