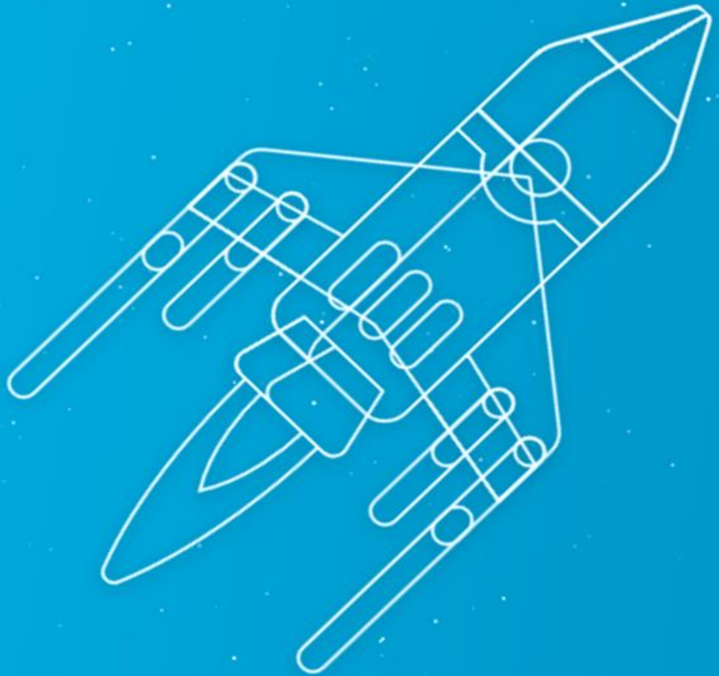
Serverless Side-by-Side Extensions with Azure Durable Functions

When stateful meets stateless



www.minnosphere.com info@minnosphere.com

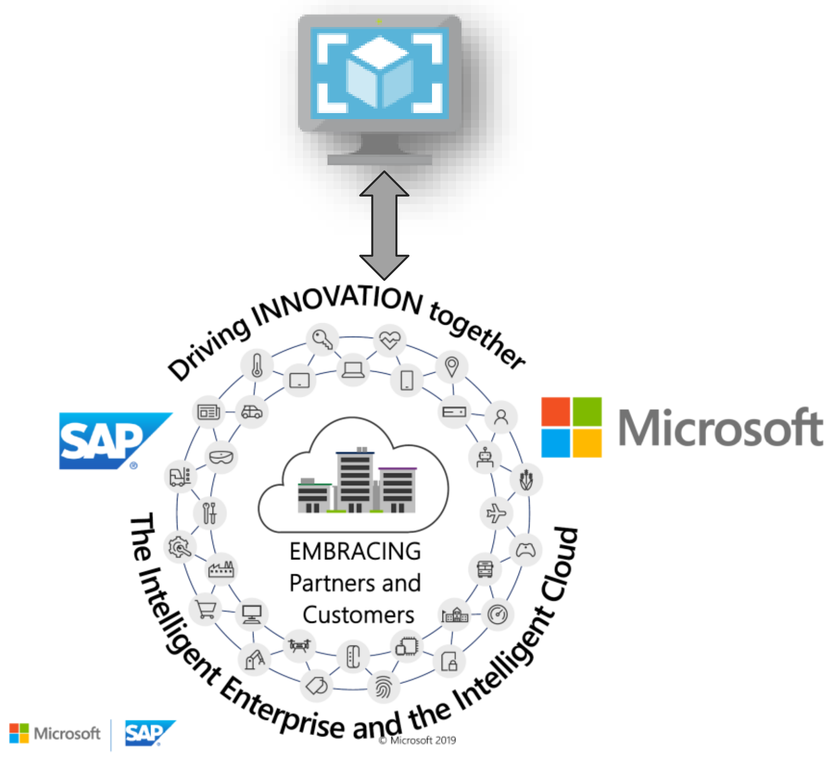
About me

- Heading the Microsoft Azure team@minnosphere
- In the SAP ecosystem since 2005
- Kind of active in the community
- Focus Topics: Extensibility, Cloud Native Development, Serverless

@lechnerc77

Why should we care about Microsoft at all?

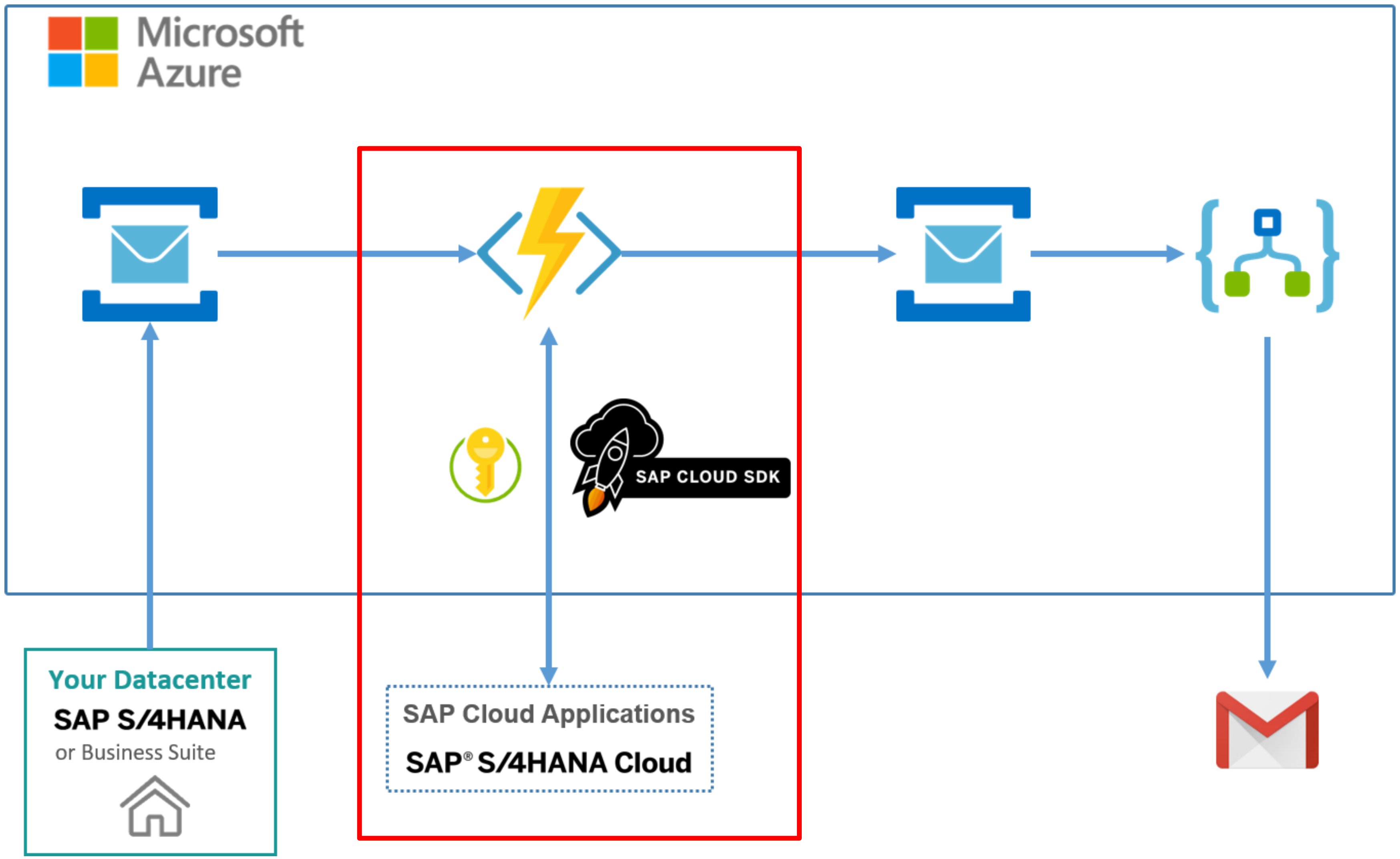
Lift
Enabling Data First Execution


Driving INNOVATION together
EMBRACING Partners and Customers
The Intelligent Enterprise and the Intelligent Cloud
SAP
Microsoft
Microsoft | SAP
© Microsoft 2019

Reshape
Enhancing Processes with Intelligence and Automation

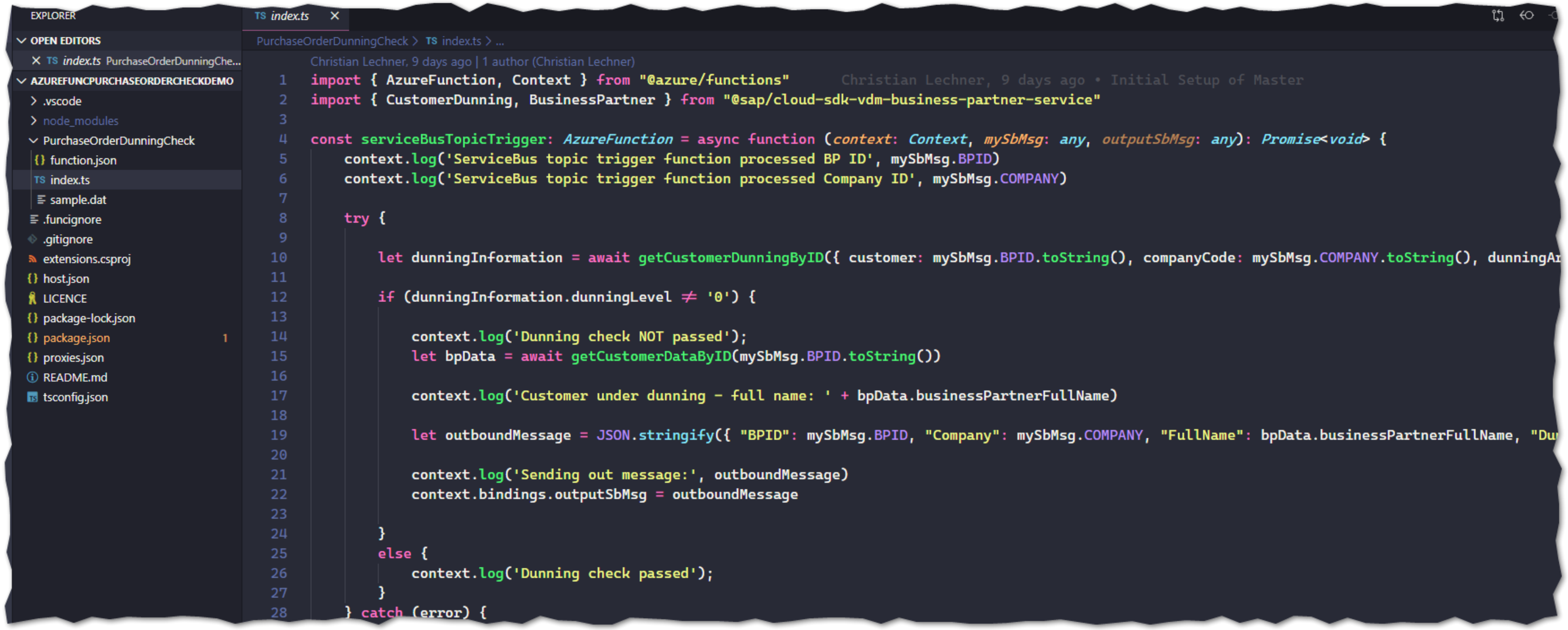
Create
Creating Experiences with New Business Models

Scenario – Serverless Extension of SAP



https://blogs.sap.com/2019/12/09/a-serverless-extension-story-from-abap-to-azure/

Azure Functions 101


EXPLORER
OPEN EDITORS
index.ts PurchaseOrderDunningChe...
AZUREFUNCPURCHASEORDERCHECKDEMO
.vscode
node_modules
PurchaseOrderDunningCheck
function.json
index.ts
sample.dat
.funcignore
.gitignore
extensions.csproj
host.json
LICENCE
package-lock.json
package.json
proxies.json
README.md
tsconfig.json
index.ts
PurchaseOrderDunningCheck > index.ts > ...
Christian Lechner, 9 days ago | 1 author (Christian Lechner)
Christian Lechner, 9 days ago • Initial Setup of Master

```typescript
import { AzureFunction, Context } from "@azure/functions"
import { CustomerDunning, BusinessPartner } from "@sap/cloud-sdk-vdm-business-partner-service"

const serviceBusTopicTrigger: AzureFunction = async function (context: Context, mySbMsg: any, outputSbMsg: any): Promise<void> {
    context.log('ServiceBus topic trigger function processed BP ID', mySbMsg.BPID)
    context.log('ServiceBus topic trigger function processed Company ID', mySbMsg.COMPANY)

    try {

        let dunningInformation = await getCustomerDunningByID({ customer: mySbMsg.BPID.toString(), companyCode: mySbMsg.COMPANY.toString(), dunningAr

        if (dunningInformation.dunningLevel ≠ '0') {

            context.log('Dunning check NOT passed');
            let bpData = await getCustomerDataByID(mySbMsg.BPID.toString())

            context.log('Customer under dunning - full name: ' + bpData.businessPartnerFullName)

            let outboundMessage = JSON.stringify({ "BPID": mySbMsg.BPID, "Company": mySbMsg.COMPANY, "FullName": bpData.businessPartnerFullName, "Du

            context.log('Sending out message:', outboundMessage)
            context.bindings.outputSbMsg = outboundMessage

        }
        else {
            context.log('Dunning check passed');
        }
    } catch (error) {
```

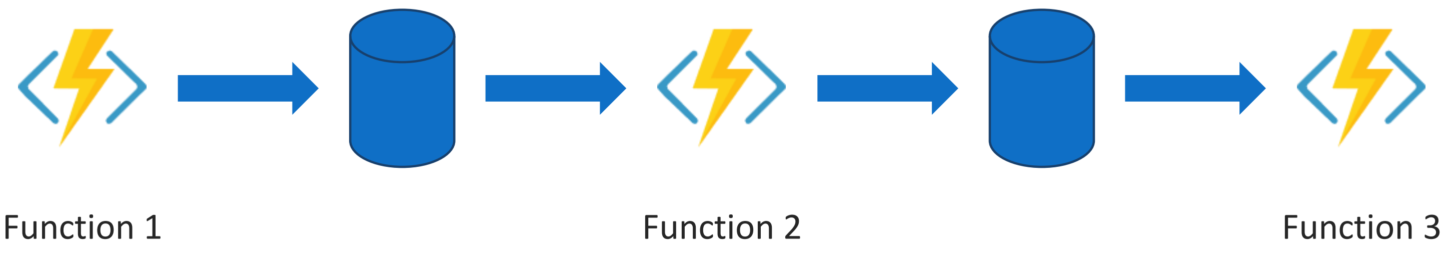
Serverless is great … but Functions as a Service come with “drawbacks”

Principles and Best Practices

- Functions must be stateless
- Functions must not call other functions
- Functions should do only one thing

How do we model “workflows” in a FaaS world?

Function Chaining to achieve State



Problems

- Unclear relation between functions
- Queues are a necessary evil
- Context must be stored in a DB
- Error handling becomes very complex

Durable Functions for the Rescue

- **Extension** to the Azure Functions Framework
- Preserves local state via **Event Sourcing**
- **Heavy work happens** behind the curtain
- Supports you in front of the curtain with **additional features**

How does this work?

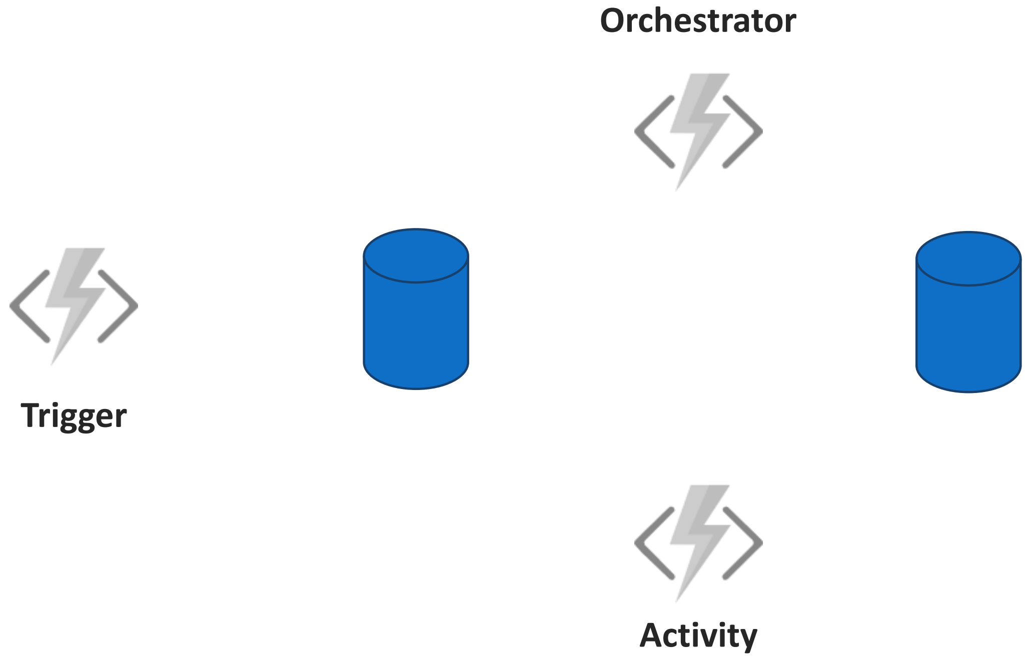
Tasks in orchestrator

1. `let x = await ctx.CallActivityAsync(“F1”)`
2. `let y = await ctx.CallActivityAsync(“F2”, x)`
3. `return await ctx.CallActivityAsync(“F3”, y)`

How does this work?

Tasks in orchestrator

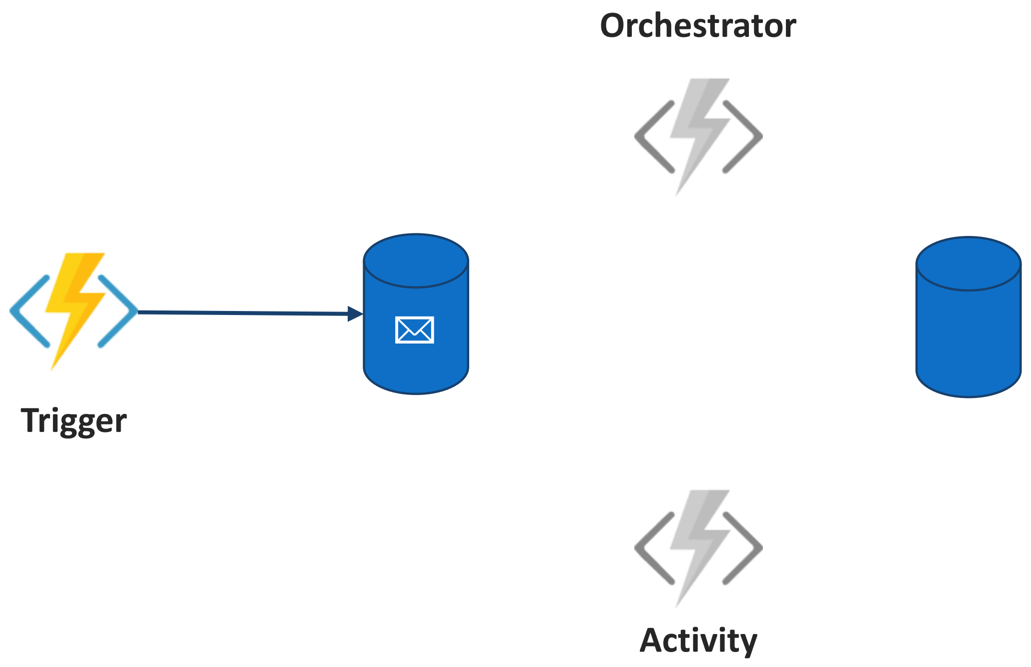
```
1.  let x = await ctx.CallActivityAsync("F1")
2.  let y = await ctx.CallActivityAsync("F2", x)
3.  return await ctx.CallActivityAsync("F3", y)
```



Step 1 - Trigger

Tasks in orchestrator

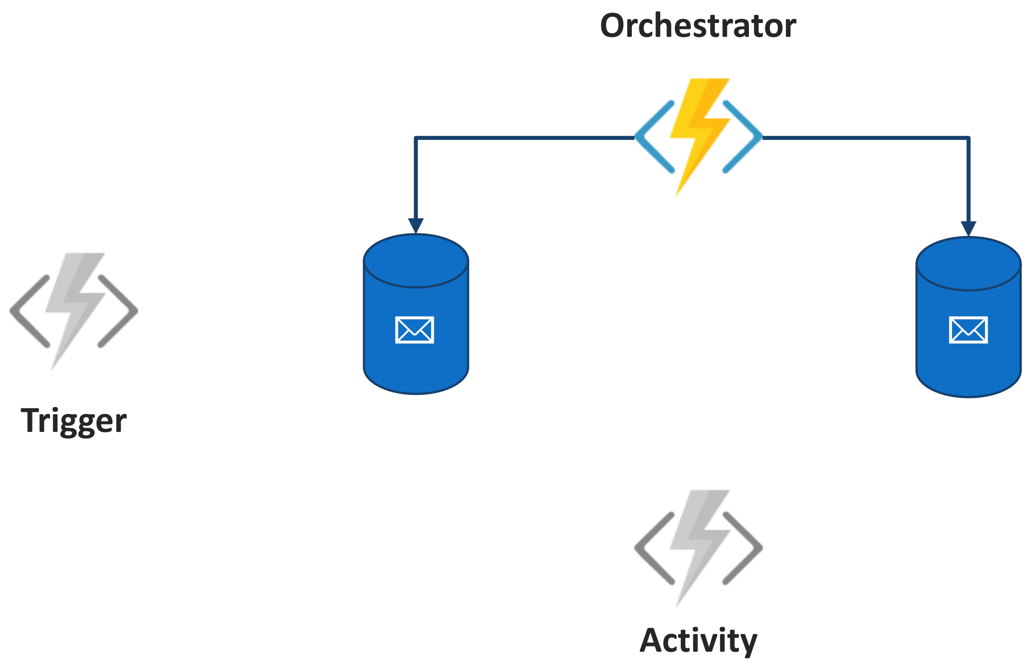
```
1.  let x = await ctx.CallActivityAsync("F1")
2.  let y = await ctx.CallActivityAsync("F2", x)
3.  return await ctx.CallActivityAsync("F3", y)
```



Step 2 – Orchestrator fetches event & schedules task for F1

Tasks in orchestrator

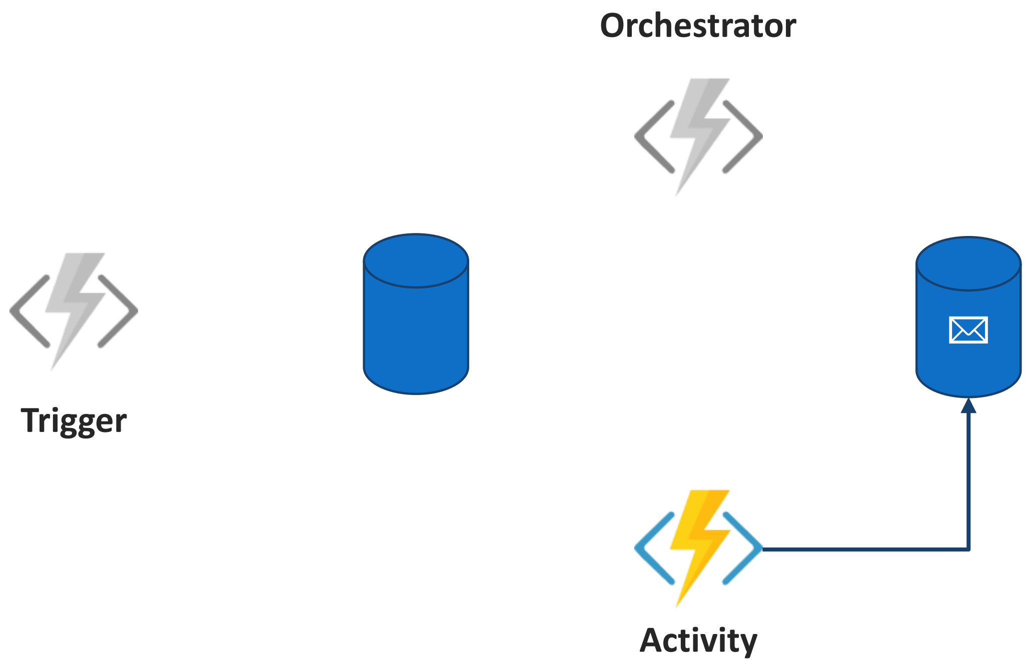
```
1.  let x = await ctx.CallActivityAsync(“F1”)
2.  let y = await ctx.CallActivityAsync(“F2”, x)
3.  return await ctx.CallActivityAsync(“F3”, y)
```



Step 3 – F1 executes task & returns result

Tasks in orchestrator

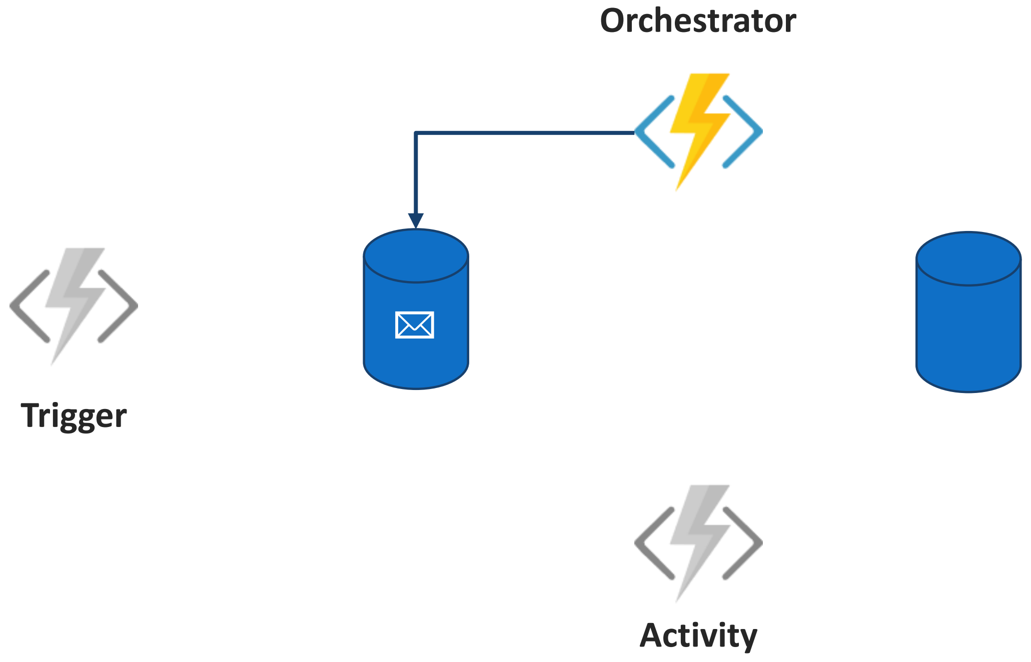
```
1.  let x = await ctx.CallActivityAsync("F1")
2.  let y = await ctx.CallActivityAsync("F2", x)
3.  return await ctx.CallActivityAsync("F3", y)
```



Step 4a – Orchestrator updates Event History

Tasks in orchestrator

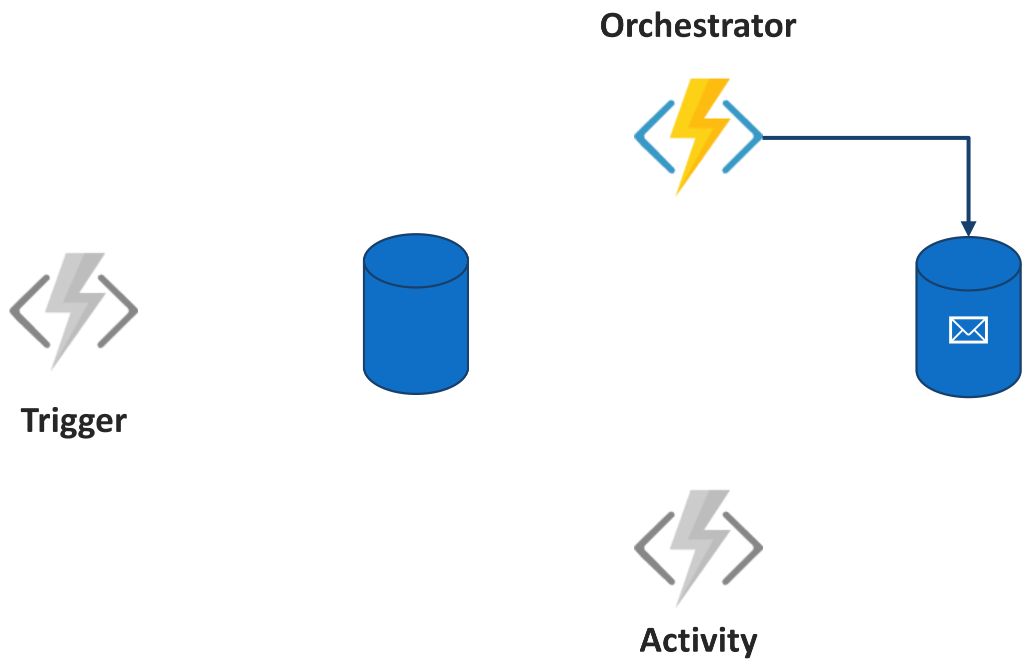
1. ✓ `let x = await ctx.CallActivityAsync("F1")`
2. `let y = await ctx.CallActivityAsync("F2", x)`
3. `return await ctx.CallActivityAsync("F3", y)`



Step 4b – Orchestrator schedules next task Event History

Tasks in orchestrator

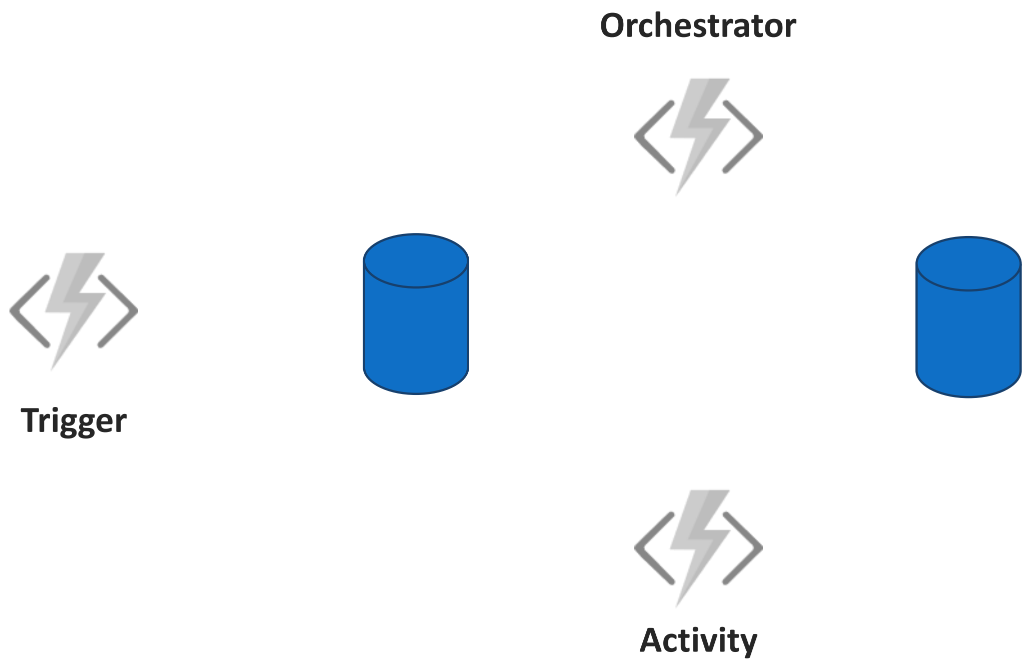
1. ✓ `let x = await ctx.CallActivityAsync(“F1”)`
2. `let y = await ctx.CallActivityAsync(“F2”, x)`
3. `return await ctx.CallActivityAsync(“F3”, y)`



Step 4b – Orchestrator schedules next task Event History

Tasks in orchestrator

```
✓ 1. let x = await ctx.CallActivityAsync(“F1”)
✓ 2. let y = await ctx.CallActivityAsync(“F2”, x)
✓ 3. return await ctx.CallActivityAsync(“F3”, y)
```



Summing up: Durable Functions are like …

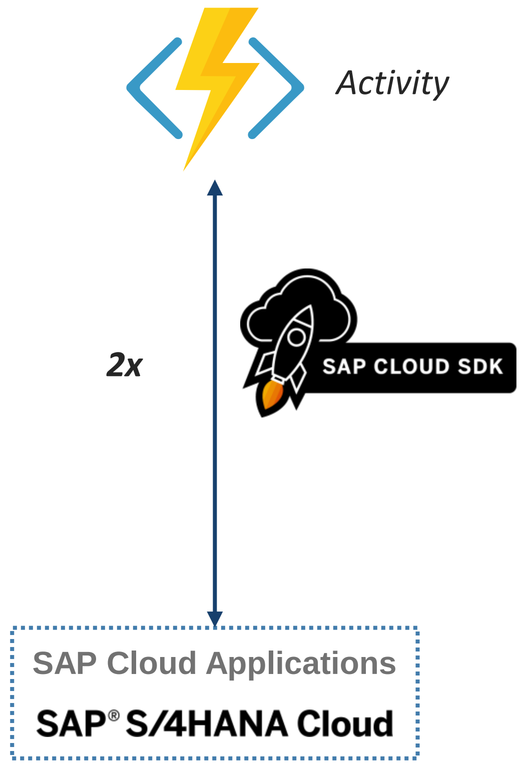
Let’s get local

- Azure Functions Runtime (https://docs.microsoft.com/en-US/azure/azure-functions/functions-run-local?tabs=windows)
- Azure Durable Functions Extension (npm install durable-functions)
- Microsoft SQL Server Express (https://www.microsoft.com/de-de/sql-server/sql-server-downloads)
- Microsoft Azure Storage Emulator (https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/storage/common/storage-use-emulator.md)
- Optional: Microsoft Azure Storage Explorer (https://azure.microsoft.com/en-us/features/storage-explorer/)

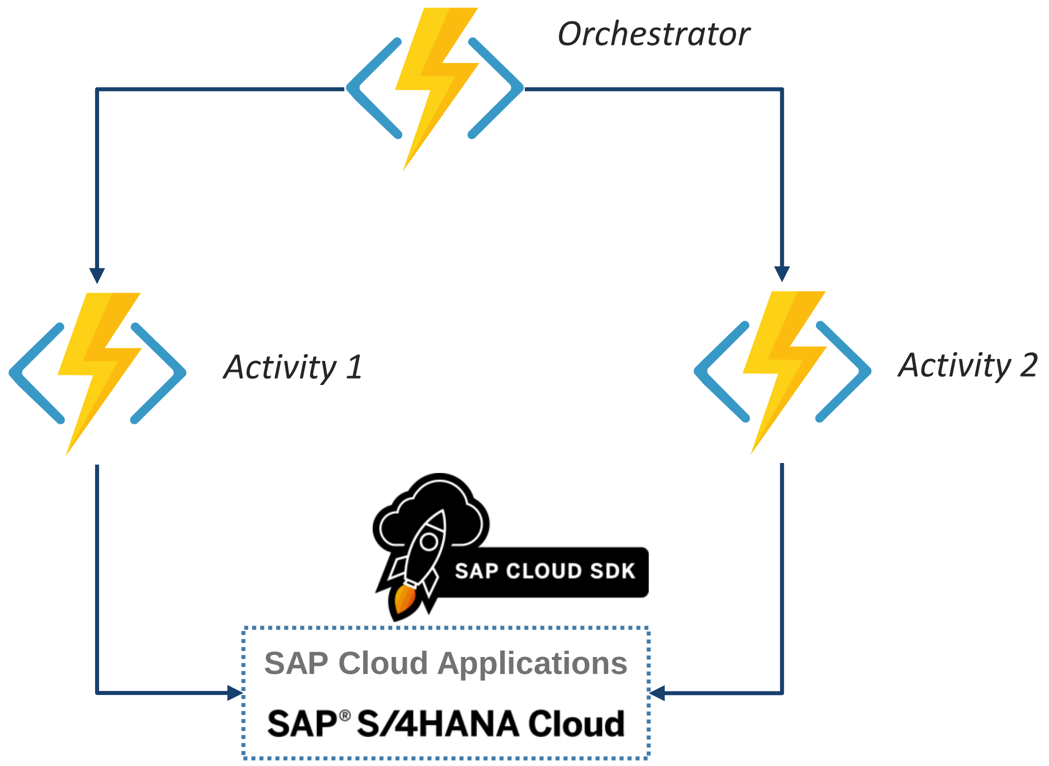
Walkthrough: https://youtu.be/HdhPC_K6cLo

Challenge 1 – Decompose Single Function into Orchestrator and Activities

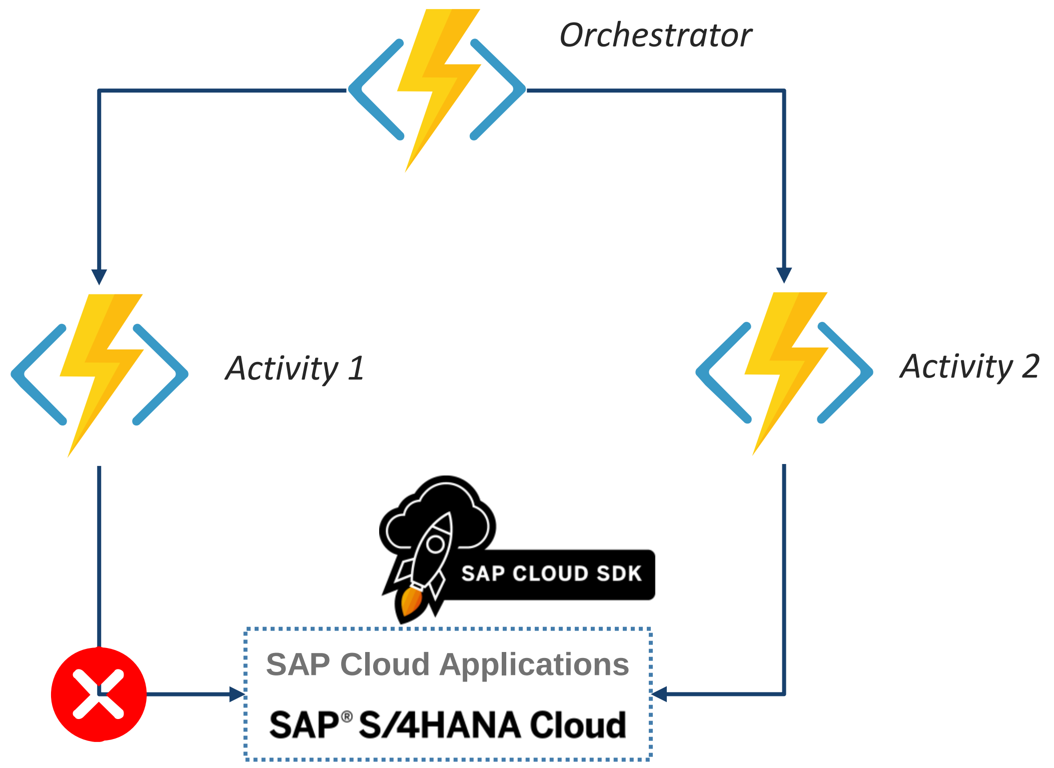
Current State



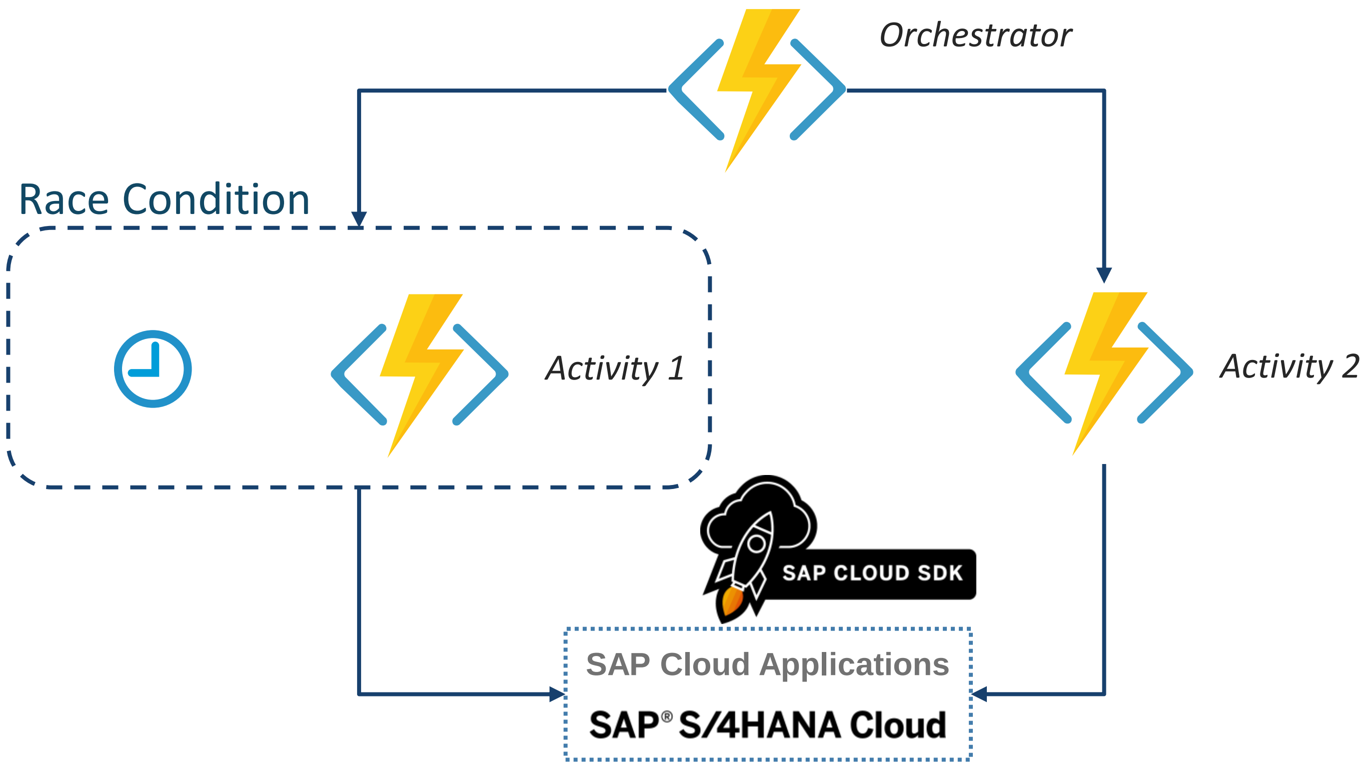
Target State



Challenge 2 – Handle Errors in Activity Calls (Retry)

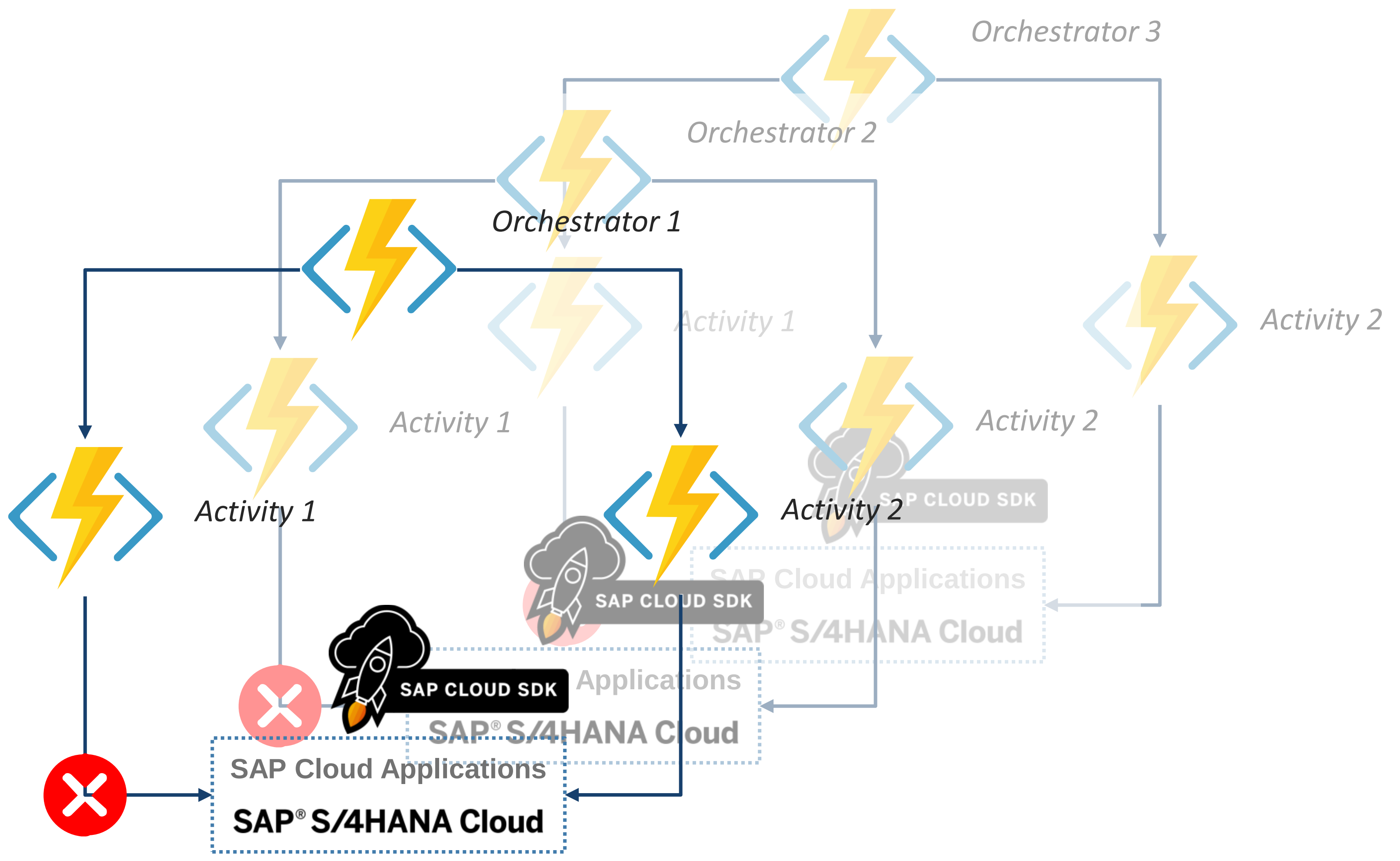


Challenge 3 – Handle Timeouts in Activity Calls



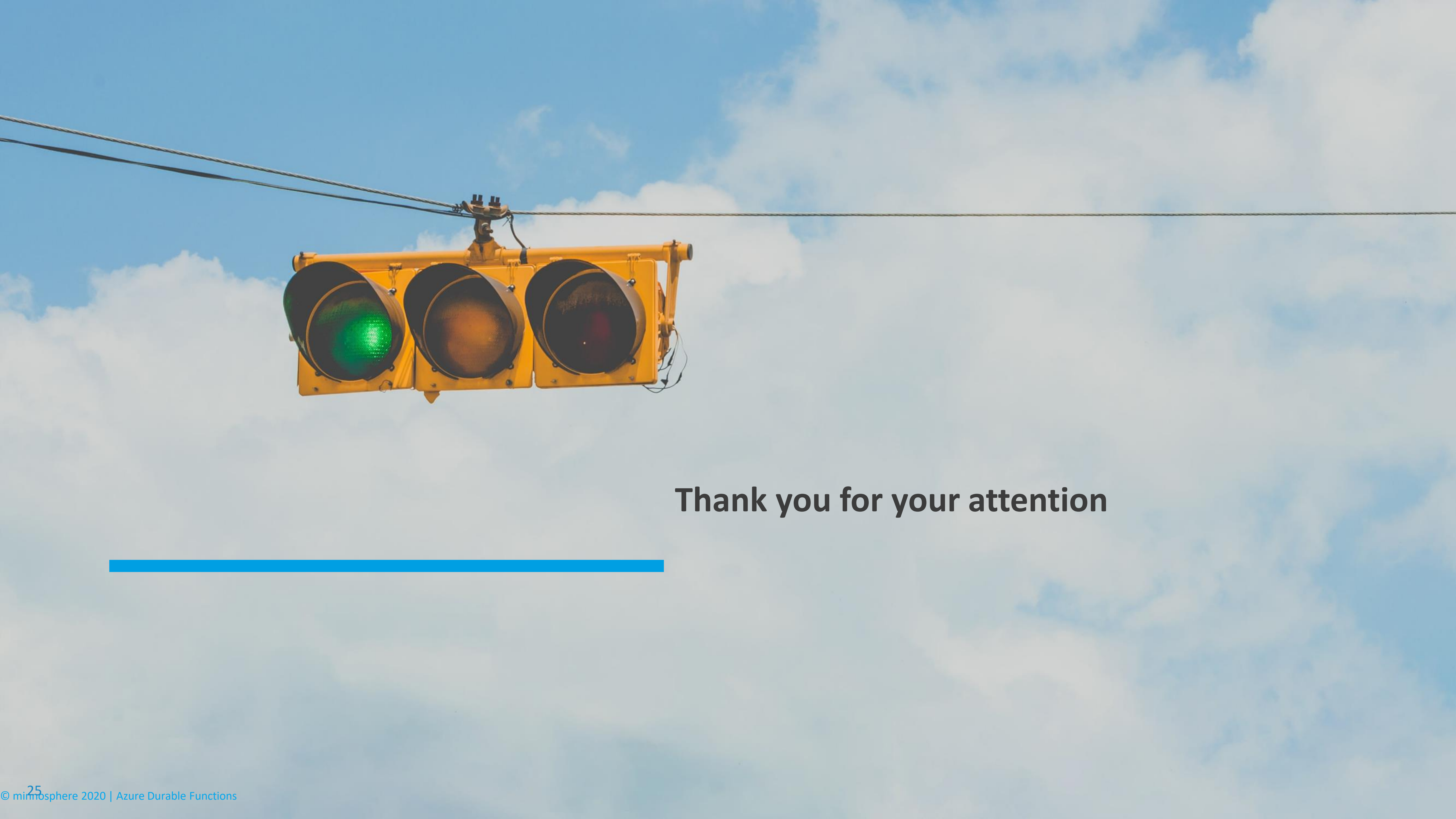
Challenge 4 – Scale Out … Do we have an external State?

Circuit Breaker



Summary - Durable Functions

- … are a great AddOn to Azure Functions
- … allow the modelling of complex scenarios without losing the benefits of FaaS
- … manage state for you
- … low entry barrier due to support of local development
- … open up new options in the context of side-by-side extensibility



Thank you for your attention

Dr. Christian Lechner
christian.lechner@minnosphere.com

minnosphere GmbH
Robert-Buerkle-Str. 1,
85737 Ismaning/Munich
Germany

www.minnosphere.com

minnosphere
company of .msg

Additional Resources

SAP Embrace – my 5 cent

- https://blogs.sap.com/2020/01/29/my-thoughts-on-sap-embrace-part-1/
- https://blogs.sap.com/2020/01/29/my-thoughts-on-sap-embrace-part-2/

Serverless Extensions with Microsoft Azure

- https://blogs.sap.com/2019/12/09/a-serverless-extension-story-from-abap-to-azure/
- https://blogs.sap.com/2020/02/17/a-serverless-extension-story-ii-bringing-state-to-the-stateless/

GitHub Repo

- https://github.com/lechnerc77/AzureFuncPurchaseOrderCheckDemo

Walk Through Local Development with Azure Durable Functions:

- https://youtu.be/HdhPC_K6cLo