

Key factors to evade immediately when conducting a health check




Raphael Ranieri

Technical Lead + OutSystems MVP
PhoenixDX, Australia

1 – Avoid monolithic Apps on O11

If for some reason, while reading about the main differences between O11 and ODC you got the idea that **no more modules** meant everything in a single application, you might consider stopping using the OutSystems Architecture Canvas (aka 3 Layer Canvas), which recommends organizing your application into End User, Core, and Library layers.

By doing so, you may make your migration even more difficult and compromise your current O11 applications.



The Architecture canvas

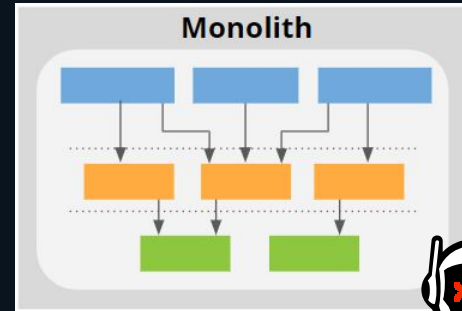
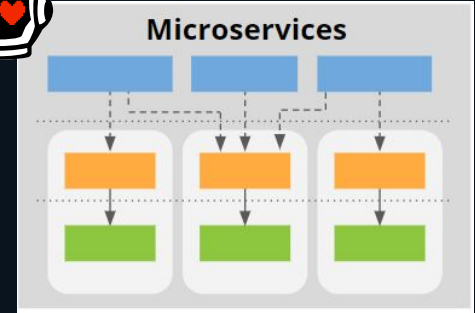
Layers

*Applicable to versions prior to OutSystems 11

<i>Orchestration Modules*</i>	No	Access Portals Cross-application processes and dashboards
<i>End-user Modules</i>	Services	UI and processes That provide functionality to the end users
<i>Core Modules</i>	Reusable	Business services Services around business concepts
<i>Foundation Modules</i>	Services	Non-functional requirements Services to connect to external systems or to extend your framework

2 – Avoid tightly coupled architectures

Instead, start using Domain Driven Architecture across the different O11 applications. The concept brought to O11 by this type of architecture is very much in line with the ODC architecture. In ODC, different applications communicate with each other using 'Service Actions'. This means that in ODC you will need to handle your reusable services practically as microservices, and If you can successfully define your different domains still in O11, based on different sponsors and life cycles, this means they will likely map to your new applications when using ODC.



3 – Avoid SOAP integrations

If you still consume or expose SOAP APIs, consider using REST or different protocols.

ODC will still allow you to consume SOAP APIs. It won't be as easy as O11, as you'll need to use C# extensions with high code, but you'll still have an option if absolutely necessary.

Exposing a SOAP API, on the other hand, is not a top priority for ODC going forward, so if you hope to migrate sooner rather than later, focus on changing your SOAP APIs as quickly as possible.



4 – Check your advanced SQL queries

In O11, you can choose to use SQL Server, Azure SQL Database, Oracle, or DB2 as the database for your applications. On ODC, your applications will be required to use AWS Aurora PostgreSQL which is an object-relational database.

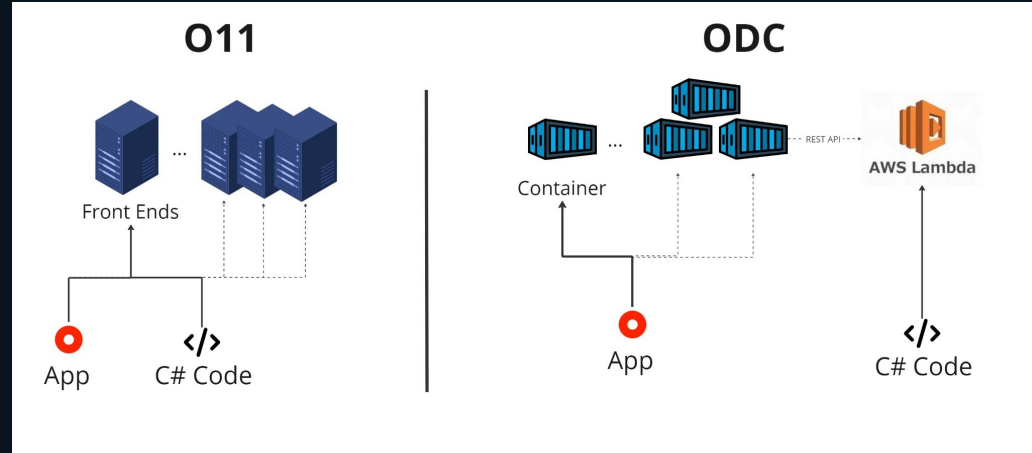
This means that in ODC you can build SQL queries with complex data types and object inheritance and will inevitably use different syntax.

Topic	OutSystems 11 example (SQL Server or Azure SQL Database)	ODC equivalent (Aurora PostgreSQL)
Comparison of Time data type	[...] WHERE {UseCase_Time}.[Time] > '11:01:41'	[...] WHERE {UseCase_Time}.[Time]::time > '11:01:41'
Concatenate text string	[...] WHERE {Org}.[Name] = 'First' + 'Last'	[...] WHERE {Org}.[Name] = 'First' 'Last'
INSERT	INSERT INTO {Product} ({Product}.[Name]) VALUES ('abc')	INSERT INTO {Product}([Name]) VALUES ('abc')
LIKE (Case and accent insensitive)	[...] WHERE {Org}.[Name] LIKE '%asd%'	[...] WHERE caseaccent_normalize({Org}.[Name] collate "default") LIKE caseaccent_normalize('%asd%') See this section for further examples.
Limiting records	SELECT TOP 10 * FROM {Organization}	SELECT {Organization}.* FROM {Organization} LIMIT 10
Random records	SELECT TOP 1 * FROM {Org} ORDER BY NEWID()	SELECT {Org}.* FROM {Org} ORDER BY RANDOM() LIMIT 1;
Selecting attributes	SELECT * FROM {Organization}	SELECT {Organization}.* FROM {Organization}
UPDATE	UPDATE {Products} SET {Products}.[Name] = 'abc' WHERE {Products}.[Id] = 2	UPDATE {Products} SET [Name] = 'abc' WHERE {Products}.[Id] = 2

5 – Check your Extensions

Another big difference from O11 to ODC, is the way custom code extensions (.xif) will work. In ODC, all your custom code extensions (like C#) will run using AWS Lambda.

This means that your code will run in a serverless service, which will be accessed through a REST API, and not in your application container. In other words, your custom code will follow the principle of FaaS (Function as a Service).



6 – Special attention to your Core Widgets

In ODC the way you reuse elements will be a little different from what we are used to.

We currently have only two types of applications that can be created in ODC. App or Library:



app

Responsive interface that can run on all browsers and devices.



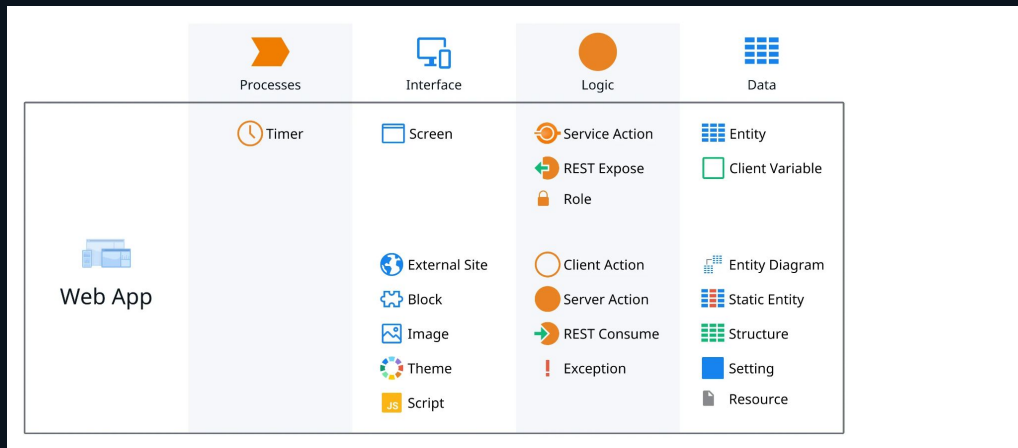
Library

Set of UI and logic elements that can be reused.

6 – Special attention to your Core Widgets

The important information about Apps is:

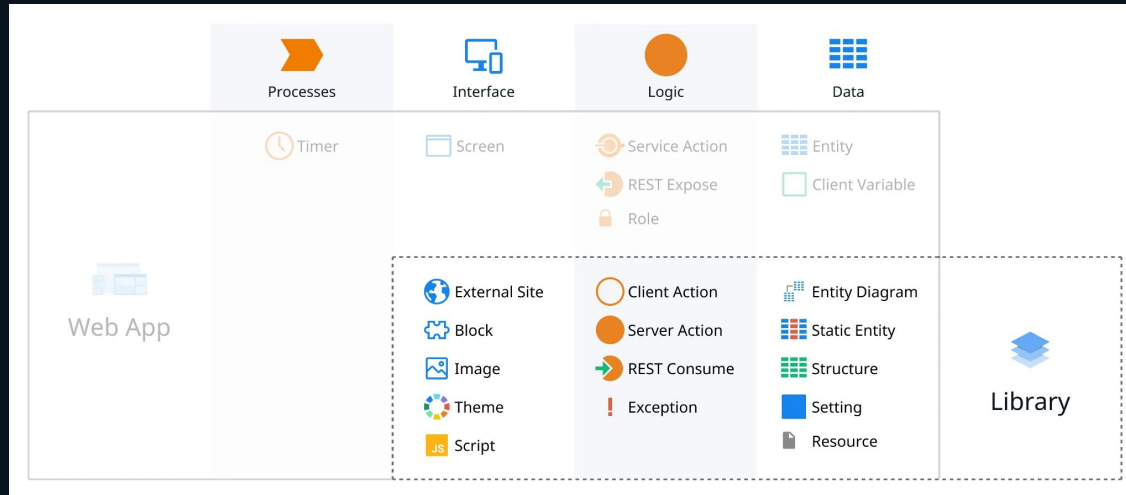
- It can have all kinds of different elements, including Entities.
- Entities can only be exposed as read-only and if you need to update a record in an entity you will need to do so by exposing a Service Action with the logic to update your record.
- Although Apps can have web blocks, they cannot be exposed as public.



6 – Special attention to your Core Widgets

A Library is a little different, what you need to know about this type of application is:

- It can contain only some types of elements.
- One of the element types that it can't contain are Entities.
- A Library can also only consume elements from other Libraries or consume external REST APIs.



6 – Special attention to your Core Widgets

A Library cannot interact directly with DB entities. It can't have a server action that updates a record, for example.

Libraries should only have reusable UI elements and actions that do not interact directly with the database. Therefore, if your Core Widgets modules (which are supposed to be reusable) have actions that use database entities, you will not be able to easily turn them into Libraries in ODC, and you will not be able to reuse the web blocks in different applications.

