2017

The Expert View

HP CSA: The 9 Step Survival Guide

By Alex Evans

Automation Logic

Cloud Service Automation, (CSA), is a service that has been positioned at the heart of many private datacentres. Customers using such cloud management software, (typically those serviced by Hewlett Packard Enterprise), would try to achieve a streamlined and standardized approach to their IT Infrastructure delivery.

This Service Orchestrator approach would enable convenient, on-demand network access to a shared pool of configurable computing resources, (such as networks, servers, storage, applications and services). The inherent promise has been to provide infinite flexibility along with proven implementations to match almost any customer use case scenario.

Just a few years ago, it could easily be argued that CSA was years ahead of the next best alternatives. The core advantages for a company struggling with technical debt were compelling; single pane solution, graphical drag and drop service designer, 1000s of out of the box integrations with other ‘proprietary' enterprise software, simple to use End-user interfaces etc.

Unfortunately, the reality has been that many clients have been left both overwhelmed with the additional technical debt and with a sour taste in their mouths, from a tool-set that has never really delivered on its advertised potential and promise of infinite flexibility.

For the most part, clients seem to expend vast amounts of effort to achieve what I would refer to as, ‘the generic IaaS CSA offerings’, also known as ‘Server as a Service’. Although such an approach would typically consist only of a Server OS deployed onto some virtualisation technology.

The Service automation itself would, however, involve many layers of complexity. Such complexity would include approval, change management, billing, capacity planning, networking, IP/DNS address management, compute and storage configuration, software deployment, server patching, HA and DR setup, AD or Access control management. The promise of encapsulating these processes into a, ‘Single Click One Stop Shop', solution, more often than not becomes the first major milestone that most customers aim to achieve with the tool.

Many clients choose CSA to avoid vendor lock-in through its Hybrid heterogeneous cloud capabilities. What they don't realise is they will likely be locking themselves into a relationship with HPE, as the software is not something which is easily adopted. Also, without in-house competencies in the toolset, this would typically lead to spiralling technical debt.

I have worked with Hewlett Packard's Automation Product Suite for many years and have worked in both a technical and architectural capacity. This has been done on a multitude of datacentre automation solutions across Europe, and as one might expect from a good consultancy / professional services company, the guys harbouring the greatest experience are typically sent in to aid the engagements in the worst scenario's. Yes, that's correct, much of my time working with the tool was trying to bring projects out of an escalation scenario!

In this survival guide, I try to outline some of the biggest and most common fallacies encountered in CSA implementations… and possibly some tips about how to circumnavigate them.

1. There's only one way to skin a cat

Understanding the product capabilities and limitations of CSA, OO, SA, DMA are paramount to building good service designs. I've seen too many clients reinventing functionality which was available natively from the tools available, often simply because they are not fully informed or even held some unfounded prejudice against understanding how particular features worked. This could be due to learning culture issues or general resistance to change?

Classic examples specific to CSA where this can be witnessed:

- Not using Resource Offerings due to a lack of appreciation for Provider Selection mechanics.
- Using dedicated Components for Resource offerings to control execution order due to a lack of understanding of how the Lifecycle Engine mechanics.
- Creating Operations Orchestrations operations to set variables, which might have been defined on the flows.
- Using OO to implement something because OO can do everything.
- Using SA Ad-hoc Scripts and OGFS hacks when parametrized stored scripts would work much better.
- Building custom external DB to store Parameters/CI in external Database when CSA can hold service topological information.
- Defining user inputs when those values can be derived programmatically.

It makes a lot of sense to study the concepts guides from each of the tools, to figure out what the common use cases are for them. I have been at countless customers who have been sold HPSA as an OS Provisioning tool, and them not realizing that not just the deployment but the entire Server Lifecycle is manageable through the tool.

Many deliveries typically follow on from a Proof of Concept (PoC) phase, and many customers are tempted to shim that into production for a quick Return on Investment, but besides the obvious risks of stability and scale, they are missing opportunity to explore functionality and do things the best way rather than the fastest way.

2. Datacentre Automation is Cloud Automation

Align your ‘business to the cloud' or align the ‘cloud to your business’? That’s a common conundrum in IT organizations looking to optimize their datacentre.

CSA is used, more often than not, for private datacentre implementations, and trying to introduce a single solution to service the varying delivery requirements for the entire organisation will require a core shift in thinking. Processes which have been bent to meet edge cases, with rules which were commonly broken to suit urgent deliveries, all need to be aligned to conform. If you don’t make standardisation a core competency of your IT delivery, then CSA is ultimately going to bring you little value!

CSA through OO offers possibilities to automate practically any IT process and in today’s modern organisations’ more and more processes involve IT. With new processes being introduced at an exponential rate, legacy processes inevitably become redundant faster. It’s crucial to not fall into the trap of investing time to automate processes that could be avoided if the business operated differently.

As mentioned previously, the tool can automate practically anything, but that doesn’t mean it automates everything easily. There are some trivial tasks which can quickly evolve into complex challenges to automate effectively. Choosing to not support these on a business level can instead save a lot of solution customization and development pain.

Examples include:

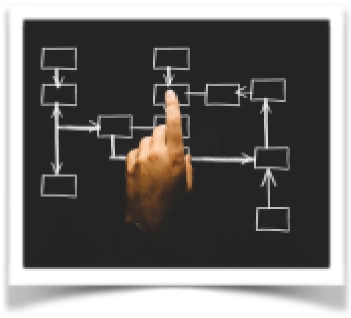
- Migrating Servers from one network to another, instead look upstream at the business processes that you can redeploy application and contented onto new servers.
- Upgrading OS on a Service, instead order new Service and redeploy applications and content using existing processes. (Invest time into Continuous delivery rather than Patch management!)
- Instead of the user opening a change ticket before ordering from CSA, allow the user to order in CSA and let the approval mechanism in CSA open their change request.
- Let the user pick a Server T-shirt size rather than specific CPU/RAM… that way the placement of VM’s and capacity management becomes much easier.

3. Out of the Box Content is King

In the world of CSA, out-of-the-box content is definitely **<u>not</u>** KING. Unlike many other Enterprise platforms, the OOTB content from CSA is likely to hinder you than give you a head start. I'd go as far as recommending CSA designs/flows as fantastic examples of how not to build a solution. Here are a few arguments to support such reasoning:

- The components which are shipped with CSA in the component palette have various properties defined on them to supplement their definitions. These properties have the most comprehensive mixture of naming conventions I've ever witnessed in a single product… even on the same component, you can find properties defined with 5 different syntactical patterns applied. A better solution is to build your component palette from the ground up and ensure everyone who is responsible for creating artefacts in the toolset are familiar with the naming conventions agreed upon before being granted access.
- Operations Orchestration has some superb out-of-the-box content and some great freely available content which can be taken from the community. The Operations Orchestration content pack shipped with CSA is, however, some of the worst developed content available. The Integrations flows typically consist of between 10 and 20 steps where they should be single operations. Merely investing time to develop a more efficient CSA integration library will reduce both the footprint and complexity of every flow invoked by CSA. Additionally, it typically makes sense to have a common CSA wrapper flow which is preconfigured to do standard CSA operations in a repeatable fashion.
- The software is pre-populated with a lot of provider types, component types and category types, 95% of which you're likely never to use. One could argue effort expensed whilst trying to locate your artefacts amongst this unnecessary bloat more than outweighs the benefits gained from saving a couple of minutes to define these from scratch. To compound the annoyances CSA, in most scenarios the tool prevents the deletion of out of the box bloat citing they are 'Critical Objects'… which begs the question why does the guys at CSA R&D think ***Amazon AWS*** is a critical provider type for a private cloud?

4. You wouldn't publish a book with typos in it.



The CSA R&D department really went the wrong way here. Consider the following:

Before CSA 4.7 it wasn't possible to deploy any service instances without a published offering, additionally, you couldn't create an offering unless a design is published… however, when you have published a design you can no longer edit it.

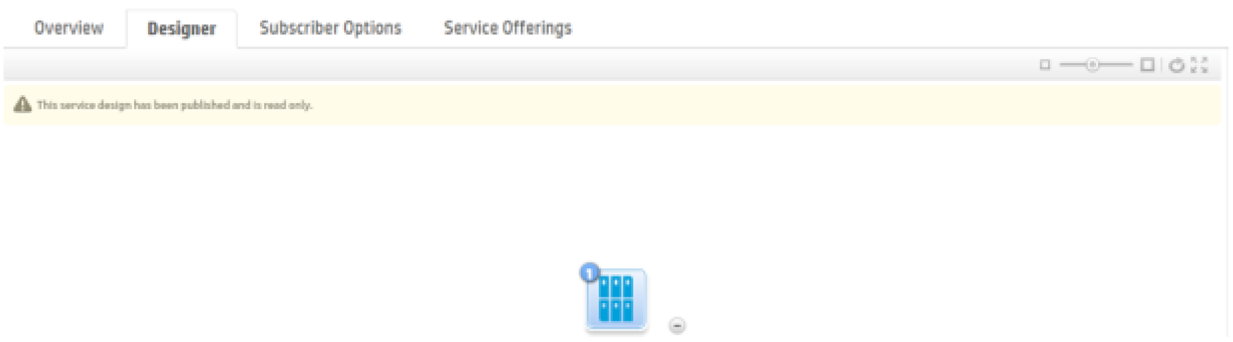
The core issue here is that as a service designer you need to test/dry-run the offerings in an unpublished state, often these designs may require small tweaks, e.g. To reorder resource offerings, to rename/change parameters, to change subscriber options or to update provider selection inputs etc. None of this is possible without first un-publishing the design, which itself isn't possible without deleting the offerings, which is impossible without destroying active/pending subscriptions! Now the real kicker, there's a good possibility you have pending instances as you're still in the development phase and you wouldn't be editing your design if it worked already!

The truth of the situation is that ***most*** customers have a completely isolated Production environment, and what publishing means to them is not setting some ***published flag*** in a CSA design, but instead exporting it and pushing it into a completely segregated CSA instance. Such an instance would typically follow a DTAP process (Develop, Test, Acceptance, Production), and changes in the service design would only happen in the Development instance of CSA. So, the entire concept of design publishing is not only a huge hindrance to developers, but it also brings zero value to many customers.

Today there are a few workarounds that you can be aware of to manipulate designs in a published state:

- Resource offerings are not versioned. This means you can change them as much as needed and any new service instances ordered which reference them will inherit those changes… it's not necessary to un-publish the design to make those changes.
- Dynamic Query Options are also not versioned, so if a subscriber's option requirements change frequently or it's not behaving correctly, then abstracting it into a Dynamic query can allow testing of the behaviour in changes to the script without republishing service designs.
- Offering creation and customization is likely to be a fixed repetitive task, but it can be very time-consuming. If you have to generate a lot of customised service offerings from a design, then the process can be fully automated through the API. This includes creating tags and assigning the correct approval policy. Clients who are able to run build scripts to regenerate Service offerings or destroy all offerings can save a lot of time and effort rather than doing it manually through the UI.

5. You wouldn't publish a book you can't read!



This service design has been published and is read only.

Similar to the previous point about editing a published design, this is another annoyance for those familiar with the tool, and it's a case of CSA trying to be intuitive yet failing miserably.

When a design is published, you may find a banner along the top saying 'This design has been published and is read-only', as in the screen above.

A more accurate phrase might have been '*The design is published and is not editable, also half the contents are now hidden so you also can't see what was published either'*

The things you won't be able to see in a published design are the following;

- Subscriber option property bindings. (Which is great if the design your testing is missing some bindings)
- Subscriber option property names.
- Dynamic Query input parameters
- CSA Component Properties settings.
- CSA Component Lifecycle Action Properties.
- Resource Binding Provider/Pool selection.
- Resource Binding Lifecycle

In credit to the CSA developers they have addressed many of these limitations in CSA 4.7-4.8 with a Sequence Designer UI overhaul. However, for those tied into earlier releases of the software the only real viable workaround is to create a draft version of the same design.

Now is probably a good time to make sure your organisation has included a naming convention for Service design which supports, identifying 'Published/Draft' versioning. Alternatively, if you're the sole designer working on a service design, then you might merely create a new version and increment the version number in order to read the design.

6. Subscription Management is a breeze, said no-one ever!

This is an issue that CSA has not addressed well for a long time. Until the elastic search functionality was added, there was no simple way to make a global search for a Service Instance or subscription and even if the Elastic search helps an MPP user locate their subscriptions or those shared with them, it's still a real challenge for a consumer admin or a Service operations manager to manage the large estates.

What typically happens is that clients each define their own way to help locate subscriptions. In the Service Operations page the out of the box experience is to click through the organisations, then identify the right users, then only being able to sort/filter on either name, design, status or date, which typically isn't sufficient, especially when you have 1000's of active services, with perhaps many using similar names.

So here are the workarounds to consider:

- Define a naming convention for the subscriptions…. You can use approval mechanism to reject requests which don't conform to convention. Standardising the name of the subscription might make identification much easier.
- You can use an Action Flow to submit a modification request to Append/Prepend a unique identifier to each subscription name. For a simple *Server as A Service* this might be a hostname, for a cluster it could be a cluster name etc. The first caveat to this workaround is that subscription names are only modified through service requests, meaning the name can only be updated after the service becomes active. The other caveat is that subscription name is a modifiable field on which users may unintentionally modify without notice. (but again, when a modification request is submitted, approval flows may reject it if the user is not the service account).
- Use a community plugin for CSA such as this: (https://github.com/alexevansigg/CSA-Enhanced-Operations) Which could expose subscriptions and component properties in a more user-friendly approach. (Disclaimer: I developed this plugin).
- Use a custom mail from the service deployment which contains all the searchable keys you need and include hyperlinks to the correct consumer/administration pages for the tool. (Yup … searching for a subscription in a Mail client will be quicker than in CSA OOTB!)

7. We understand the Lifecycle Engine that's why we don't use it.

The Lifecycle Execution Engine could have been made simpler, and it's been a complaint from many customers for a long time. The main issues are that it's misleading, and that specific sub-transitions (e.g. PRE & POST) impact the execution sequence in different ways, depending on the context they are triggered from.

'Having worked with many ***specialists/consultants*** dubbed as 'well versed' in CSA, I have come to the conclusion that fewer than 10% have a strong grasp of the sequence of which lifecycle actions will be executed. This figure drops even lower if you consider ***'solution architects'***, many of which, will have not have invested sufficient time to evaluate the intrinsic mechanics of the engine.

In CSA 4.7+ the CSA team wanted to address concerns from the clients that the lifecycle execution engine was too complex, the core changes introduced were that they renamed '*PRE*' and '*POST*' transitions to '*Before*' and '*After*' phases respectively, introduced a default view in the lifecycle editor which would hide *Before* and *After* phases and abstracted the deployed phase to a new view called *User Operations.*

Those were all welcomed changes but the lifecycle engine itself still applies the same logic when building the order of execution as in earlier versions of the product and below is an extract of a white paper from CSA41 which still today, is the most thorough explanation I know of describing how it all work.

Source: HP Cloud Service Lifecycle Actions in CSA 4.10

CSA's lifecycle engine references the service design to determine the execution sequence when realizing a service instance based on the design and maintains the lifecycle of the instance through its deployment, modification, and retirement stages.

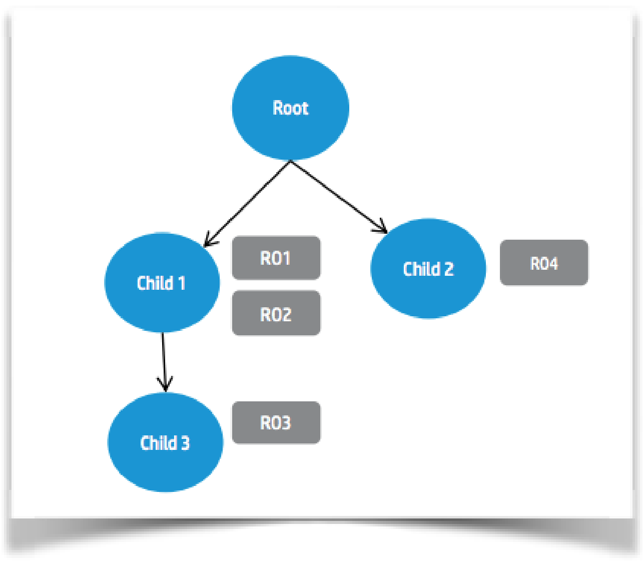
Every element (service component, resource subscription, or resource offering) in a service hierarchy transits each state and sub-state in a sequential manner. In other words, after all the elements transition to a stable state, the lifecycle engine initiates the next state transition until the desired target state is reached.

Multiple factors influence the lifecycle action execution order:

- *Parent/child relationships between service components*
- *Execution order of child service components (sequential or parallel execution)*
- *Execution order of resource bindings within a service component (sequential or parallel execution)*
- *Execution order of actions within a lifecycle state or sub-state (sequential or parallel execution)*

- *The current lifecycle phase*

Consider this simple component hierarchy with a root, three children, and at least one bound resource offering for each child. Child 1 has execution order 1 and child 2 has execution order 2.



Influenced by the factors listed above, the lifecycle engine will follow this execution sequence during each of the major Deployment transition states of Initializing, Reserving, and Deploying for this component hierarchy:

- PRE-TRANSITION Root
  - PRE-TRANSITION Child 1
    - PRE-TRANSITION Resource Offering 1 (RO1)
    - PRE-TRANSITION RO2
  - PRE-TRANSITION Child 3
    - PRE-TRANSITION RO3
  - TRANSITION Child 3
    - TRANSITION RO3
  - POST-TRANSITION Child 3
    - POST-TRANSITION RO3
  - TRANSITION Child 1
    - TRANSITION RO1
      - TRANSITION RO2
    - POST-TRANSITION Child 1
      - POST-TRANSITION RO1
      - POST-TRANSITION RO2
    - PRE-TRANSITION Child 2
      - PRE-TRANSITION RO4
    - TRANSITION Child 2
      - TRANSITION RO4

- POST-TRANSITION Child 2
  - POST-TRANSITION RO4
- TRANSITION Root
- POST-TRANSITION Root

The Key takeaways from the above extract are as follows:

- Component Actions are executed before Resource Offering Actions in the same sub-transition (sub-phase).
- PRE-Transitions executed on Parent before Child.
- Transitions executed on Child before Parent.
- POST-Transition executed on Child before Parent.
- Component Siblings with a lower order execute entire phases (PRE-IN-POST) executed before those with higher order.
- Resource Offerings with lower order execute sub-phase only before those with a higher order.
- Initialising and Reserving Phase occur directly after Approval Process.
- Deploying phase occurs on the start date of the Subscription.

The lifecycle engine will follow this execution sequence during Retirement of component and the following is the transition states of Un-Deploying, Un-Reserving, and Un-Initializing for this component hierarchy

- PRE-TRANSITION Child 2
  - PRE-TRANSITION RO4
- TRANSITION Child 2
  - TRANSITION RO4
- POST-TRANSITION Child 2
  - POST-TRANSITION RO4
- PRE-TRANSITION Child 3
  - PRE-TRANSITION RO3
- TRANSITION Child 3
  - TRANSITION RO3
- POST-TRANSITION Child 3
  - POST-TRANSITION RO3
- PRE-TRANSITION Child 1
  - PRE-TRANSITION Resource Offering 1 (RO1)
  - PRE-TRANSITION RO2
- TRANSITION Child 1
  - TRANSITION Resource Offering 1 (RO1)
  - TRANSITION RO2
- POST-TRANSITION Child 1
  - POST-TRANSITION Resource Offering 1 (RO1)
  - POST -TRANSITION RO2

- PRE-TRANSITION Root
- TRANSITION Root
- POST-TRANSITION Root

Now there are some notable differences in how the retirement execution order works in comparison to the provisioning phases with the key takeaways as follows:

- Component Actions Executed before Resource offering actions in same sub-phase.
- Entire Phase (PRE-IN-POST) executed on child before parent.
- Component Siblings with higher execution order (e.g. 2) execute entire phase (PRE-IN-POST) before those with lower execution order (e.g. 1).
- Resource Offerings with lower execution order (e.g. 1) execute sub-phase only before those with a higher execution order.

Given all these rules it can be very confusing to get your head around. My recommendation is to do the following when approaching a sequenced design:

- Define your components based on tangible objects, such as a Server, Network Card, Disk or Application.
- Define your resource offerings, and place them on the components which make the most sense. For example, an Offering to Manage Disk partitioning would go on a disk component.
- Create a dummy action which simply returns success or error based on input to dry run your resource offerings to check they execute in the correct order, and perform the correct actions when cancelling a failed build.

8. Pause on Provisioning… but we don't want anything to pause!

'Pause on Provisioning' was added in the CSA 4.2 and it's a very powerful tool when put to use whilst developing CSA Services. The way it works is that if any Process action encounters an error during any of the provisioning phases (Initialising, reserving and deploying) then instead of triggering corresponding error sub-transition on the failed lifecycle it simply pauses.

Next, a notification (via SMTP) will be sent from CSA to both the Consumer who ordered the subscription and to any Service Operations Managers who have been identified in the notifier list for that organisation. The email informs them that the subscription is paused and that somebody is investigating the issue.

Now when the Service Operations Manager receives their notification they can look into CSA and inspect the subscription in it's halted state, perform a diagnosis through the event log or OO flow logs, then make an informed decision on the following three choices:

1. Perform a manual intervention, typical use cases might be releasing new flow content because of a bug, or extending capacity to a resource pool which was exhausted.
2. Resume the Subscription. This will trigger the lifecycle engine to retry the failed process action, and if it succeeds continue to the next action, otherwise the subscription will pause again for a second round of intervention.
3. Cancel the subscription. This will then trigger the Failure Transition respective to the failed lifecycle action. Then it will automatically begin to execute the un-provisioning phases. The use case here would be the Service Operations Manager has determined that the subscription is non-recoverable even with intervention and must be rolled back.

Now, many would argue that pausing for manual intervention is not within ***'The Cloud Mantra'*** and would just build another service from scratch straight away when something goes wrong.

The fundamental problem here is lots of stuff goes wrong all the time when your '***Developing***' a service, and depending on the complexity of the design and how many '***Legacy***' systems you have to work with, maybe to get back to the point in the same build could mean 1hr-2hr lead time plus 10-30mins of clean up activity.

The second issue with not using the Pause on Provisioning is that OOTB failed subscriptions do not get rolled back automatically by the tool. So, if you have a busy platform and eager users who keep ordering more when their subscription fails, you quickly build a mountain of half-baked orphaned subscriptions.

There are a few caveats which one needs to also be aware of when deciding whether to use the pause feature:

1. It's a global setting for an entire organisation. It would have been much more practical to have it defined as default on organisation level, then be able to override the pause behaviour for individual catalogues and then again for individual offerings… not the case today.
2. Approval flows don't pause if they go wrong, these flows are outside of subscription lifecycle… so instead you need to ensure you have a timeout for delegated approval flows configured, and notifications built into those flows.
3. As the name suggests '***Pause on Provisioning***' only applies to Provisioning. Perhaps in an upcoming release, they might apply the same logic to un-provisioning phase… after all who wants to release an IP address if the server couldn't be decommissioned.

Having considered the above limitations, I still believe it's a great feature to enable for development and test environments. In Production, where realistically stuff shouldn't be going wrong so often, pausing isn't something you want to enable.

9. Let's just use System Properties for everything.

The possibilities to introduce/store properties inside a CSA solution are vast, and it's not surprising how many times a property or variable is defined/declared in an illogical location.

It could be in a flow when it should be read from a component, or it's defined on a Server Component when its more appropriate on an NIC component, or it's relative to a specific provider, but instead it's derived based on subscriber options.

The below table attempts to address some common scopes of variables and when they should be used.

Property Location	**Scope**	**Usage**
OO System Property	Global	Use when a property might be needed by all/any flow regardless of the design it is incorporated into.
OO Flow Input Property	Single Flow	Use when the property is specific to an OO flow (typically CSA flows pass UNIQUE IDs to CSA artefacts as flow inputs when invoking flows).
CSA Component Property	CSA Component	Use when a property is relative to a component. e.g. hostname on a server component, disk size on a disk component.
CSA Provider Property	CSA Provider	Use when a property is relative to a provider. E.g. A vCenter provider might require a property which describes the maximum VM version it supports.
OO flow Output Property	Single Flow	When properties need to be manipulated/merged its simple to do the translation in OO. Every step in OO supports generic inputs and outputs so purpose build operations are not necessary. E.g. storing

		Hostname and Domain in CSA component are easily merged to a FQDN component in OO.
Option Property	Option Model	These properties are defined in the subscriber option model and typically bound to one or more Component Properties. Best to keep life simple and use the same name for component and subscriber properties.
External Properties	External	There is always the possibility that properties consumed in the Service delivery are sourced from foreign CMDBs or databases. Typically, an assessment should be made where the best source of truth lies for a property.

It's quite important in CSA to understand where is the source of truth for data. When you are integrating various management systems together and you have complex dependencies of state, then a well thought out CMDB to represent the entire estate can become paramount. OO flows on the other hand should be made atomic with as many variables parametrised as possible making them both portable and reusable.

Summary

So, the HPE (Microfocus) CSA suite has been around for several years now and has slowly matured into a capable automation suite. Unfortunately, in its earlier years a plethora of caveats, gotchas and *Easter eggs* have left customers customising solutions beyond reason and loosing site of the true value delivered through automation.

What is most apparent to me now is that the majority of issues I see are with regards to the content which is created on the platform rather than the platform itself. If I look back to several years ago most of my time was spent chasing enhancement requests or product defects with the CSA core development team or Support. Whereas now I'm shaking my head trying to understand why client specific content has been developed in such a bad way. This article was written with the intention of getting people to think about how they design content in CSA, and while there are many reasons to migrate away from the platform, there are many ways to squeeze just that little bit more usability out of it until that day comes.

My Bio.
I'm a Senior Consulting Engineer at Automation Logic since May 2017 having previously spent 7 years working for Hewlett Packard Enterprise championing the Business Service Automation and their CSA Suite servicing clients across many different sectors. Additional to Automation, I have experience core strengths Software Development, Public Cloud Architecture and Business Intelligence.