Seeing through the clouds: Oversight at multiple levels

White Paper

Contents

List of figures

List of tables

White Paper

Purpose

To help the reader understand the facets that combine to make 'Hybrid IT', and what is required from an organisational perspective to *manage* Hybrid IT. And in so doing realise how counterintuitive the name is, since the IT management element never *actually* manages anything truly hybrid.

This paper does not address data governance in terms of ownership and regulatory control. Although clearly important from a business perspective, it does not require IT 'management'. It covers just the technical security of data within the business service.

The paper is cast according to a 3-fold business problem, each of which in turn affects IT;

Problem 1 - Speed

A CEO demands a Digital Enterprise strategy that has pace. The demand for faster time to market from IT is unavoidable

Problem 2 - Cost

The CFO demands a radical reduction in IT CapEx (for hardware and software)

Problem 3 - Quality

Traditional home-grown or first generation applications are slow and error prone during both deployment and management

Key Takeaways

SaaS-first is preferable to cloud-first for the business. In order to compete, the democratisation of non-value adding capability must be achieved with ruthless efficiency.

Although it is possible to be a provider of services using traditional IT methods, it is neither efficient to operate in this way, nor is it in line with increasingly prevalent corporate goals.

A review of the applicability of 'rigid' ITIL processes needs to be undertaken in light of changing execution practices.

Previously 'hidden' costs need to be exposed to consumers, and inefficiencies in the IT organisation that may have always been present, but remain unaddressed need to be dealt with as a matter of urgency.

Is 'Hybrid IT' a thing?

In short, YES! by definition; it is simply a mix of providers, on- or off-premise, SaaS, PaaS, IaaS, public- and private cloud that provide services to support your business in performing its [various] business functions.

But, as will be seen within this paper Hybrid IT only really exists at the enterprise level, comprising IT as a whole, where suppliers may exist in-cloud or on-prem. Whilst a specific business service *may* be a subset of this, the provision of a single application *cannot* be seen this way

Although Hybrid IT initially sounds like an interpretation of [Gartners] 'bi-modal' IT, it is the goal of any IT organisation to be fastest that their customers need them to be whilst operating at a sustainable cost and not accepting any risk beyond the appetite of the business.

But to *manage* all of these things as one amorphous mass of technology, applications and suppliers would be enormously difficult (see sidebar). In this paper there is an assertion that there are 3 layers of management. Although the body of the paper will focus on layer 2, it requires information to be made available from layer 3, and likewise passes information up to layer 1.

LoB IT

It is certainly not impossible to do this, as it has been, and is being, achieved by many IT organisations and indeed lines of business for quite some years. But achievement does not necessarily mean ease, or effectiveness.

Rogue, or shadow IT, or 'citizen developed' applications are testament to this, since the LoBs going down this route have handed back difficult decisions and architectural problems back to IT when faced with the mire of scale and governance

3 layer Model

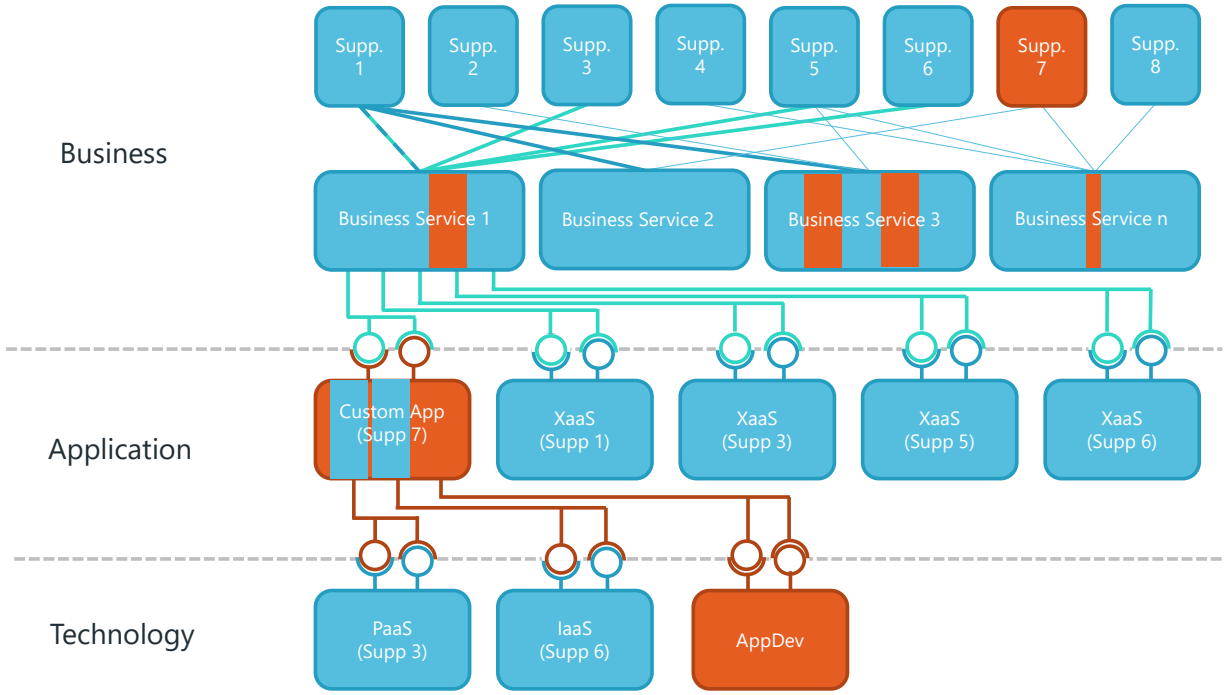

Figure 1 - 3 Layer Structure

Layer	Description
Business	Macro-scope --- the entire technology estate from ALL suppliers and ALL business service perspective. Multiple service components may be provided by the same supplier for different services.--- Blue lines
	Service owner scope --- the orchestration of specific suppliers within the confines of a specific business service.--- Turquoise lines
	Interested in SaaS offerings only as these represent the best way of moving entirely to OpEx, require little or no IT knowledge and leverages no/lo-code capabilities or an Integration Platform as a Service (IPaaS) provider for integrations.
	Data security is of primary importance as the only part of the service that remains 'owned'.
	Demanding of transparency from suppliers about reliability, cost and security. Plus control(ability) when any of the aforementioned is impacted
	Have zero control over the configuration of the application layer as provided by SaaS suppliers
Application	The suppliers of SaaS offerings demanded by the business layer, to fit the speed, cost and security concerns.
	Search for democratised 'as a Service offerings' from PaaS and IaaS providers that best support the custom value-add for the application
	Reliant on 'Infrastructure as code' for automated deployment as a mechanism to reduce cost (which can be passed on to consumers) and improve control
	Required to provide reliability, cost and security information to their consumer for analysis and optimisation.
	They are demanding of transparency of information from their suppliers about reliability, cost and security.
	Has no control over the configuration of the technology layer as provided by PaaS/IaaS/FaaS suppliers
Technology	Suppliers of technology offerings that serve as base building blocks for the application layer.
	Use automation as a matter of course to improve speed and control of delivery.
	Required to provide reliability, cost and security information to their consumers for analysis and optimisation

Table 1 - 3 Layer Descriptions

The keen reader will notice that although there are 3 labels attributed to the layers, but the picture suggests 4 layers. The reason for that is that the management of Hybrid IT happens ***between*** the 4 layers in the picture.

- L1 – The supplier/service interface
- L2 – The service/application interface
- L3 – the application/technology interface

Throughout this paper the focus is on L2 – The service/application interface, addressing the other layers as it touches them, and also addressing each summarily where there are notable differences between them and L2.

Layer 1 – Supplier Management

To be completed

Layer 2 – Service Owner

Acknowledging the evolving goals of most organisations as they embrace the technological adaptations required for the digital transformation of the business;

'Reduce cost in general, but removing as much capital expenditure as possible whilst increasing responsiveness to the business (ultimately customer) demand, whilst maintaining or improving quality AND all within the bounds of acceptable risk'…a tall order.

Business services can be viewed as loosely coupled groups of applications that work together to provide some business outcome. They are dependent on each other when framed in the context of the data that must pass between them, and from a technical level may be described as an 'application group'

Let's use Uber as an example of Service Owner scope (in Table 1 - 3 Layer Descriptions). It is a relatively simple example since Uber only provides one business service, Passenger transportation. However, they do consume applications provided by multiple suppliers, as well as creating some custom content themselves, and as we shall see they democratise the platforms on which that 'internal' application runs also.

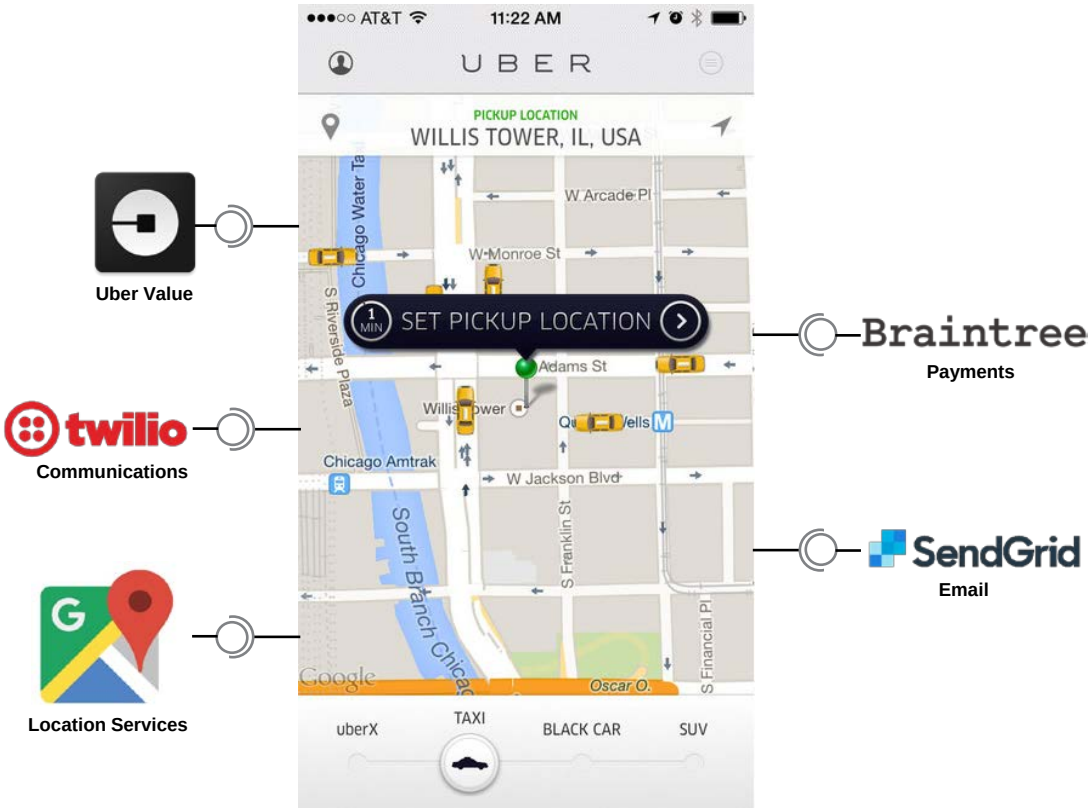

Figure 2 - Uber Service

As such they look like many modern businesses and nicely fit into the 3-layer model. The figure below shows the same conceptual model to describe Uber [at the L2 – The service/application interface].

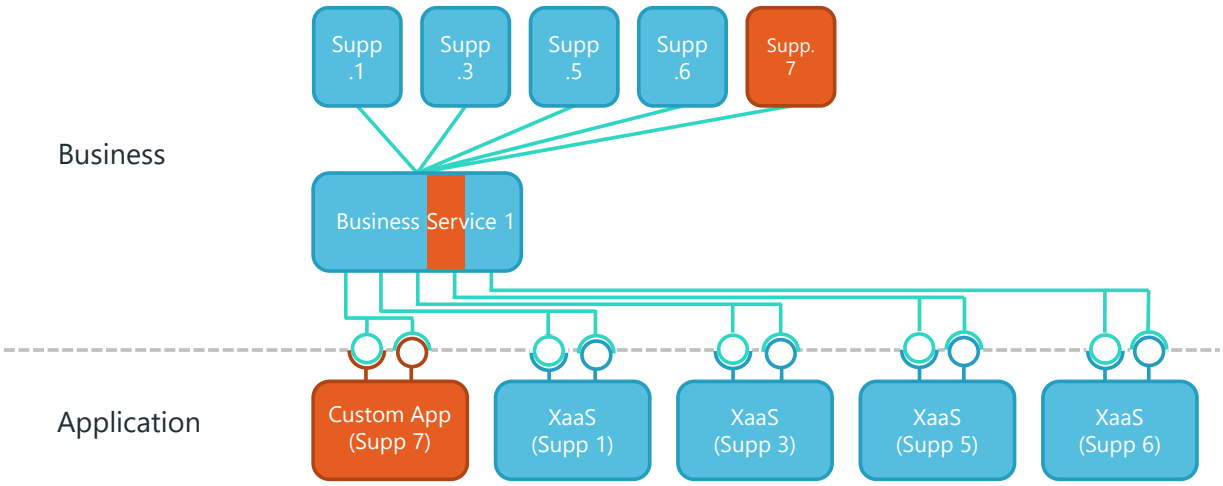

Figure 3 - XaaS

Service Owners act on behalf of the business, regardless of whether their role exist as part of IT (Business Relationship Manager) or the LoB (Service Delivery Manager), as such their goals are identical. They own the optimisation of the service along the competing lines of speed, cost, quality and risk. They have as per Problem 2 - Cost, a very clear mandate, reduction of CapEx. And as such the approach is not 'cloud-first'[1] as IaaS and PaaS providers are fond of saying, but ***'SaaS-first'***. Uber has used this approach in choosing suppliers Braintree and Twillio etc, to provide services that are required, but not part of the core Uber value proposition.

In a traditional world this might be construed as, 'buy-don't-build' as was common when software was only ever installed on premise. COTS software purchased and installed as such was touted as 'better' from a supportability perspective, which is still the case. However, it failed to allow for the pain of upgrades and its integration into the wider business service topology and its associated cost. It also has a 'bought but seldom used' tag associated with it in many cases. SaaS offerings greatly improve on this.

> Pause for Thought: *Is this Hybrid IT Management?*
>
> *Yes, if you consider Hybrid IT to be the use of central IT services and 3P SaaS providers.*
>
> *No, if you consider that we are managing only SaaS providers, regardless of whether they are internal or external*

If you are a young enterprise wanting to craft, or consume a new business service, a SaaS only scenario is going to work to your advantage immediately. In Ubers case, for it to be able to deliver on its [at the time] unique offering it would need to develop some in-house capability ('Uber value' in Figure 2 - Uber Service) alongside these SaaS applications, in much the same way as an existing enterprise on their 'journey to cloud' has mature business processes and services that have been customised [over many years] to fit the specific needs of their business (magenta item in Figure 3 – XaaS service). This is provided by internal IT, only when made possible by the ability of IT to provide the service on level terms with its service components siblings, ***as a service***[2]

Consumption Bias

It is very easy therefore to measure progress on the goal of becoming OpEx oriented (Problem 2 - Cost). Simply measure the proportion of your business service that is consumed rather than owned.

---

1 Those seeking a cloud first strategy are application developers whose likely intent is that their developed application will be provided using a SaaS model, thereby completing the circle.
2 Although it is possible to provide applications using a traditional (manual) IT environment. To be part of a business service with the speed and reliability attributes necessary to solve the problems highlighted in Is 'Hybrid IT' a thing? this means processes and controls that are available by a high degree of automation and transparency

Whilst a proportional measure is sufficient in service terms, as with most business metrics a financial measure is preferred, so a better to quantify the ratio of CapEx to OpEx costs in currency terms for any given business service.

As a key and substantial metric towards a business goal there is no other single indicator that shows progress as well as this one. Only when internal IT is used in a traditional manner (without cost attribution) or managed services on owned infrastructure would this value be anything but a 100% OpEx output.

As you can probably guess consumption driven IT has a significant implication on the IT operating model as will be described throughout this paper.

On ownership and control

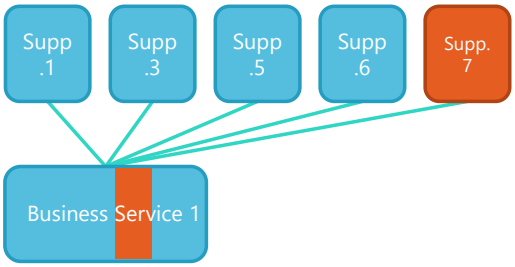

Figure 4 - L2 Supplier Management

2 Hats

Elements of managing provision, consumption (and integration) of service components exists in new structures as it has always has [albeit unrecognised] in traditional IT.

The goal should be to push commoditis-able elements down [and eventually out of] your organisation to eliminate CapEx, and minimise OpEx. Creating the ability to adapt service and application configurations quickly, through exceptional design and supplier choices.

Unlike L1 – The supplier/service interface, a business service owner, managing suppliers at L2 – The service/application interface is only interested in the management of the suppliers of the applications that make up a ***single*** business service, regardless of whether they are provided in cloud or on premise (turquoise lines in

Figure 4 - L2 Supplier Management). This is wholly the responsibility of the consuming service owner, but can be and is typically delegated to technical owners in IT. Whilst the business service owner is responsible for the output of the business service they do not care how it operates see Jason Bloombergs [Forbes] post 'In search of the elusive citizen integrator'.

Since each application is provided as a service, neither IT, acting as the owner of the service composition, nor the business, has the ability or the right to dig deeper into how the service is provided and which technology and/or suppliers are used for any lower level application component. This is different from traditional central IT provided applications that are 'owned' (CapEx). The consumer[3] must ***trust*** the supplier to provide;

A simple rule of thumb is that no layer [in the 3 layer model] needs to see further than the top of the layer directly beneath it i.e that which is explicitly exposed by the lower layer

- The information necessary for them to make decisions, and
- The ability for them to take action based on those decisions, ideally in an autonomous way

Importantly, this is also true for the providers of the custom application. The in-house developers of the application must provide the same level of information as do SaaS providers[fn1]

What consumers must realise, is that the purchase of on-demand, consumable services is not the same as ownership, there is no control or authority over the supply of that service, and you must manage them as such.

---

[3] Business service owner or IT organisation acting on their behalf

Layer 2 management is the gathering of information from the application layer is a way that is helpful to the business layer, and that may, or may not cause the service owner to take action based on that information. But what is this useful information?

In short, it is only that information that allows them to 1) select a supplier (decision criteria in Table 2), and 2) optimise the application (optimisation capability in Table 2) that they receive from that supplier (through configuration).

Decisions to work with a specific supplier are made infrequently, so can be made by humans, on human timescales, of the order of days, weeks or months. But once the supplier has been onboarded then future interactions can, and should, be made by machine, with one notable exception…enhancements[4]

But at this management layer we must also consider the interaction ***between*** the application 'siblings'. Like a parent of children, your actions for one child may, or may not be affected by the behaviour of another. This means the management layer must take some responsibility for the configuration of a single application ***and*** the service as a whole, and this means deciding what to do with the information provided.

	Decision criteria examples	Optimising capability examples
Security	SDC • Levels of assurance for data recovery and integrity • Security criteria certification	SOC • Authentication failures • Logging data access requests/sources • Customised endpoint security • Authentication and authorisation
Reliability	RDC • Levels of assurance for availability and recovery • Point of presence locations • Capacity	ROC • Ability to design tolerance and scaling • points of presence traffic/change
Quality	QRC • Systemic information access • Systemic configuration and control capability • Partnerships	QOC • General activity logs (when did something change), consumer or provider initiated • [Re-]Configuration by API • Performance metrics (of the provided application per consumer) against upper bound • Thresholding
Cost	CRC • [Multiple] attractive cost bands	COC • On-going consumption • Thresholding • Suggestions

---

[4] Feature creation or extension specific to the need of a particular consumer. These would be raised as backlog items in the SaaS providers development backlog, where, if an increasing proportion of end consumers request the same or similar functionality would raise its priority in the backlog, and would result in either an enhancement to an existing product, or a new product entirely. It may be assumed that defects may be raised in a similar manner, however, this process is wholly within the realm of the application developer/curator to 'catch' and resolve as part of their operational practices, as such is described in Incident and Event Management.

Table 2 - Supplier provided information

But what about the information that SaaS suppliers need to provide ***during the consumption*** of their application. These 'metrics' allow the service owner to manipulate the application in order that they can optimise it, in the context of the business service they own. Let's call these 'designed capabilities', that are the responsibility of the service owner using supplier provided information or capability.

They are *management* level decisions at the level of the business service that can be made with no user interaction either retroactively, or by design (preferred). All information and actions therefore must be accessible systemically (via APIs or web service perhaps) and not require human interaction[5]. This means leveraging the abilities of [Enterprise] Architects, Data Managers and API Managers throughout the operational life of the business service, not just during a 'design' phase.

The implication being hardest hitting for those traditional IT organisations wanting to participate in this ecosystem: They must be capable of delivering the optimising capabilities described in Table 2 ***on-demand*** (not retrospectively). As such, this means that they must be reliant on automation[6], and enable systems, rather than processes (and humans) to provide the information as necessary.

	Designed capability examples
Security	• Integrity and security of data in transit. Along the data flow paths between SaaS entities • Using the available capabilities to ensure the attack surface of the business service is optimised for the data. Authentication mechanisms, encryption and access controls.
Reliability	• Ensuring that the consumed application is available at the point of consumption, within acceptable performance within cost boundaries. • Automatic scaling within and between offering bands • Point of consumption health checks • Automated sibling reconfiguration after configuration change. e.g. failover.
Quality	• Rulesets – creation of triggering points e.g. for failover, or scaling. Bounds of operation before human intervention is required • Algorithms – Workload analysis patterns. Is data demand coming from where it is expected (security). Do the SaaS point of presence need to be reconfigured (quality) • Integrations – Optimising data flows between applications acceptable within the security configuration bounds based on gathered endpoint traffic analysis. • Automation - What are the [automatic] responses when the (un-)expected happens?
Cost	• Optimisation of costs through assessment of where the workload is running and data low paths. Perhaps through automated switching within and between applications, or by changing business processes • Influencing the demand profile of the application

Table 3 - Consumer designed capabilities

Data Access Patterns

Data pattern analysis of workloads is a key capability, not just as part of security, but in terms of quality of the application from a consumer perspective, and where trade-offs need to be made against budgetary constraints. It will allow the determination of;

- Security and Quality --- Expected or unexpected data traffic flow between applications and between consumers and applications.

---

[5] A consumer of a business service should never have a need to request access to a 'specific' application that is part of the service. It should be handled at the business layer (via a request catalogue), that is provided by the service owner, or by proxy, at the management layer.

[6] They will use the capabilities of democratised technology providers operating at the Technology provider to make this possible leveraging Infrastructure as code to make and define the request. This is the level of private cloud to provide the automation. Traditional IT structures are typically too reactive and people oriented to be able to cope with the provision of information in this manner.

- Cost and Quality --- Data volumes transferred.

Whether you are designing a new business service from scratch, or are in the process of migrating your services (application groups) to SaaS providers, how, and from where, data is consumed by end users, how it flows between various applications within your business service (application group) are crucial factors to be aware of.

Data Pattern Control-ability

Data access is controllable, semi-controllable and uncontrollable, this is related to the fact that *systems* can be controlled, *internal users* are reasonably controllable, whilst *external users* are 'in the wild', their choices about from where, and when they interact with services are beyond control.

As has been the case for decades, systems running chatty applications should be placed together in a single datacentre to improve the end user experience for the service as a whole. However, traditional datacentres may have evolved from a computer room in the same location as the internal users that have accessed those systems. As the computer room grew they have been relocated to dedicated datacentres that may or may not be close to the users. Little consideration was given to the reduction of quality in terms of user experience that may have followed, as it was potentially offset by increasing data transfer rates offered by technology. Often this has been in tandem with the globalisation of the business and associated relocation of internal users so a 'net-zero' effect may have been assumed, but never validated. There are always trade-offs that need to be made between location (quality of experience) and cost. Only now, as business move to cloud[7], and to a lesser extent SaaS solutions are capabilities being introduced to quantify and qualify the 'chattiness' of application groups in order to mitigate any costly oversight.

Whilst the above considers applications suitable for internal users, where we can move either systems (easy) or personnel (harder) to optimise the user experience, where the service is made available publicly, the location of the accessing users could potentially be anywhere around the globe. This provides additional complexity into the designed responses of the service.

The ability to know how data flows, between where, and at what volumes is important both during design and in operation of your environment. The ability to recognise and influence that same data and your consumers is a fundamental part of optimising any business service on SaaS.

> *Do not congratulate yourself for measurement alone! The ability and frequency which with you can measure data and demand patterns is secondary to being able to respond effectively [through design] to these variables.*

The key question for architects and data managers to ask is; 'How can I ***design*** operational response(s) into my service when data access patterns differ from those expected?' I have asked all my suppliers to provide me the right information, and they have granted me the ability to systemically take actions, why would there be a need to continue to be so reliant on humans to tell me that things are not working as they should. All I need to know is that something ***did*** happen and when the design [of the service] worked around it, automatically. Designers (service developers) can then figure out ***WHY*** this happened at a later time.

- How should the service respond if it detects that it is being accessed from an unauthorised source, either human or system?
- What happens if there is a spike in data volumes, how should the service validate that the spike is as a result of business demand or an attack?
- Certain data flows only exist at specific frequencies, how can I optimise for that?

Consider putting in place an analysis regime that can answer these questions, both at time of design ***and*** during operation [ideally using the same techniques].

---

[7] In public cloud data egress has a cost. Often even between locations within the same public cloud provider

Data security

Data remains as important in SaaS structures as is in a confined datacentre environment, probably even more so, since it is almost all that is *'owned'* of the service. Providers must be trusted to act responsibly with your data.

Data exists in 2 states;

- At rest – Red circles in the below figure, and
- In transit – Yellow lines in the below figure terminated with yellow circles as accessible end-points

Both must be secured, but each can be considered separately. As with everything related to security (and operations in general), prevention is better than cure. This, by implication, is in the remit of designers not operators.

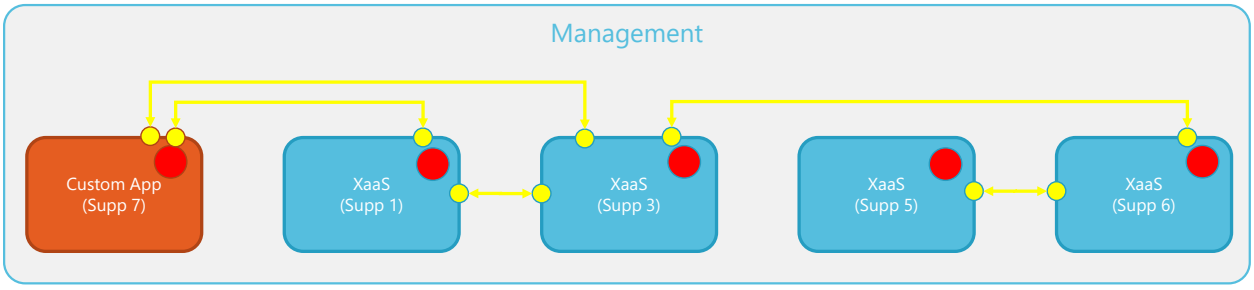

Figure 5 - Data Entities and Flows

The endpoints (yellow circles) in combination will be evident to any security aware IT staffer as the attack surface. They must be secured as a point of entry into your data[8].

Penetration testing and vulnerability scanning is a hurdle that traditional projects have undergone at time of application instantiation prior to moving into production, once this gate has passed it is seen as a problem for operations, these 'one and done' practices can no longer be the case as the SaaS applications being consumed are evolving continuously as part of the development methods employed by the SaaS providers.

Some suppliers may be particularly sensitive to this as they are on the hook for segregating data for many organisations, but on the whole they should welcome it as it only serves to improve their security capabilities too.

Providers must assure that their services are as secure as intended[9]. But it would be remiss to take guarantee on trust alone and must ensure that suppliers are doing their job well. However, targeting suppliers with scanning and penetration on a whim is ill advised, as their response may simply be to take down [your part of] their SaaS application (or even their entire their service) as they see it as an attack. There must be an agreement in place on how this can happen periodically, and systematically, using tools or more non-conventional techniques such as hackathons or game days where challenges are set to access your data.

As well as endpoint protection there is the data whilst in transit between endpoints. Architectural decisions need to be made as how this should be achieved. This ***must*** be a design choice of L2 – The service/application interface since it is unknown to the providers how a specific service might want to accomplish this[10]. Plus, the service owner (as a representative of the LoB) must provide some guidance on the importance of the service data.

We can make the assumption that it does indeed need to be secured since it passes between SaaS services via the internet and the internet is inherently 'open' by default, and as a result insecure.

---

[8] Not systems, as at layer 2 there is no responsibility for the systems that provide the application, this is within the remit of the application provider in the layer beneath

[9] Note that this does not have to mean highly secure in all cases, since consumers have differencing requirements for security and some providers may only address part of the market

[10] Providers are on the hook for providing an agnostic mechanism of achieving this

Which brings us to data at rest. It cannot be avoided that data *must* reside on the storage systems of suppliers at some point, and for varying lengths of time, but you must decide how, and by what/whom it can be accessed. In most cases it will only be a 'what' (the application) that should need access to the data, so why provide human access? Data should be obfuscated from administrators of SaaS providers, and most will provide this capability by default, either by encryption or by advanced role based access, including multi-tenancy (included as part of the cost). Users that require access to applications should be authenticated using a trusted system and authorised using some role-based access control system (RBAC).

Encryption, for example, creates latency as it needs to be decrypted at time of use, so brings with it a trade-off decision

Only when combined have you considered all of the above have you addressed all the concerns of data security. But this does not mean that have all possible options covered. Again being proactive is key, use of defensive or offensive hackathons will validate the operational capability for data integrity. Defensive methods are about the improvement of existing measures, whereas offensive hackathons are about forcing your way into a system through potential weak points and known, or unknown vulnerabilities.

But what happens if you *do* discover some new vulnerability, or hole, or access risk? Is this an incident? No! It is merely a discovery, a *defect* of the service, to be handled (prioritised) by the service owner. In the worst case, on discovery the link with the errant provider is severed, and either switched to an alternate (if the system was designed for that) or wait until the provider to resolves the issue. There has been many as case where breached systems have been suspended whilst the severity of any attack or flaw has been validated and verified. And whilst the naysayers say there will be significant brand damage, this has never been catastrophic [for the biggest players] and tendencies to bounce back are remarkable. [Ref: Target, Yahoo!]

Reliability

IT operations exists because unknown things happen, and those things need a response. This response is also unknown (at least the first time) so takes time to define and complete. This has been as a result in the separation of development and operations functions. The historical figure of the programmer analyst, where the responsibility was combined no longer exists, and has resulted in a 'throw it over the wall' mentality that allows the developers to be measured on speed of development and operations to on uptime, and never the twain shall meet. In this mix, ***quality*** is often imbued on operations and measured in terms of availability and performance. Using reactive measures such as these as the key metrics of quality is the epitome of denial within IT. Fooled into believing that since these metrics are gathered ***close to*** the consumer (and easy to measure) that they must be correct. However, they have relatively low importance if we improve the reliability of the service during design.

What we need to get much better at is;

- Recognising failure modes during design and build, and embedding responses into the design of services (and applications)
- Forcing the failure modes, through being proactive, therefore making the unknown more known.

The above are self-reinforcing when used together, where forced failures improve the design of the service, and embedded responses reduce the amount of failures. Whilst there are still going to be unknown scenarios, the availability of SaaS and other elastic environments allow for a seminal shift in perspective, through deliberately testing failure modes ***and*** their associated Recovery Practices.

Testing and Production

As 'operational' metrics are gathered from non-production environments it blurs the line between the environments themselves.

If the changes required are small enough then why not test them in production? E.g. A/B or canary testing

In SaaS environments the service is 'always' in production

Cost and commoditisation

Where all other factors are the same, the only factor to compete on is price. Cost comparison is required by similar providers, especially where the service is commoditised. A close to real-time analysis might be needed to choose the optimum provider at any specific time, similar to those provided by CDN broker Cedexis; content streaming capabilities are pretty much identical between providers, but at different times, locations, or based on demand, the quality of service may be worth paying a higher price for, rather than the defaulting to the lowest cost option. Whilst this is may be most appropriate for those commoditised platform service at Hybrid IT Layer 3 – Application Owner it may also be relevant for Hybrid IT Management Layer 2 – Business Service Owner.

On Switching suppliers

It has been noted in this paper that one of the possible responses to threat or issue is that we can switch provider. Whilst this is indeed possible it does have its downsides.

If you are consuming SaaS services only then it is a matter of re-integrat-ability. The mechanism where you must be able to make supplier applications 'talk' to each other by facilitating the required data flow.

This explains the increasing presence of Integration Platform as a Service providers to facilitate these connections (and switching of connections) in companies such as MuleSoft. Plus the increasing use of low-code/no-code techniques that simply allow the connection of applications with no coding skills in the recognition that Hybrid IT Management Layer 2 – Business Service Owner is within the reach of non-IT personnel with little, or no IT knowledge the so-called 'citizen integrator'

But at Hybrid IT Management Layer 2 – Business Service Owner, there is also 'data lock-in' since the format of data for a specific provider is seldom the same as another provider. This technical 'barriers to exit' must be overcome during any switch of providers. The perceived benefits, cost or otherwise, must be greater than the cost of getting over the barrier. This necessarily means that switching providers is not a decision that can be taken [by IT] without the decision passing by the business service owner, or architect, who is aware of budgetary constraints and other principles in play. As such it can only ever be a strategic decision rather than an operational one.

Capability Comparison

One must also be aware of developments of the service of your chosen provider, as this may enhance the ability to make decisions and/or take action in a particular scenario.

Also, developments in the competitive landscape of your provider. Whilst we have talked about the fact that there are barriers to exit (or lock-in) to specific providers. Analysis of the market may reveal that the benefits of such a move outweigh those pitfalls.

A similar process is in play in Hybrid IT Layer 3 – Application Owner systems at the IaaS/PaaS layer as the data transfer costs between PaaS providers form part of the barriers to exit, the other part is the application architecture, if it is reliant on specific functions provided by the PaaS provider, leading to 2 distinct possibilities;

- Applications must be re-factored at time of switch, or
- The capability to switch is built into the application from the start.

Whilst the former allows a degree of flexibility, it is a zero-value activity with associated cost. The latter adds complexity to any application and requires additional maintenance, increasing the overall price of the product. And further, relies on the original designer knowing the future possibilities of *'the best'* alternative even before it might even exist. There are, however, certain circumstances in which these issues are lessened. Consider the use of containers [running on Kubernetes orchestration]; these products are an almost ***perfect commodity*** where the application can be transferred between providers for only the cost of the data transfer with no refactoring.

Scenario 1

An initial analysis of data flow within the existing datacentre has discovered a single application that provides reporting services. The single server on which the software is installed runs 24x7x365 but only generates reports once per month. The application requires about 4 hours to perform the report generation using data from

applications in the same datacentre. The data is accessed by HR users in multiple locations around the globe from a website (part of the reporting application) as static content.

A simplistic look at the solution is to move the existing server to an IaaS/PaaS provider (public cloud) such as AWS or Azure, this would allow for the server to be spun up only at the time that it is required, thereby saving money on redundant energy costs. However, application management and ownership still resides with the business, and only in part achieves a desired goal of moving CapEx to OpEx. Data ingress from applications still residing in a local datacentre is free. However, access to the reports by end users results in data egress costs, since the website is supplied with the application. Additionally, if redundancy is required this too will be additional cost that needs to be borne in data transfer.

As has been stated the preference of most business consumers should be 'SaaS-first' and the movement should not be of technology, rather capability. Look for a SaaS solution that provides reporting capability similar to what the installed application provides. This shifts both report generation and application maintenance to the provider (true OpEx) and is provided on a consumption basis. The tariffication of the SaaS offering hides any data egress charges since they are bundled in a pay-per-use consumption wrapper. As a consumer we should not care what technology is used to produce the reports, but just the fact that they ARE produced.

Is it required?

Considering the technological aspect of the business service may not be all that is required. It may be the correct time to address the business process in use.

Do all the users accessing the data need to be spread globally or is there an option of optimising the human elements of the process?

Are the reports REALLY required? What decisions are made using the report data? And can then be addressed using more automated processes?

The questions that the consumer needs to ask (as relates to management) are;

1. Is the service provided as reliable as it needs to be in the context of the service it supports?
2. Is the data transfer and storage as secure as it needs to be for the type of data involved?
3. How are my consumers going to be affected by locational performance issues?
4. How much does it cost?

The responses to these questions should be built into the design of the service, rather than into an operational response. Operational design is not a new concept, however, current IT operations practice only exists because sub-par development and design practices within the enterprise community and the 'throw it over the wall' philosophy that prevails. DevOps practices are beginning to redress this (see Reliability).

There are potential additional benefits in practice of SaaS. Reporting is able to be made fault tolerant easily, depending on the capabilities of the provider, at potentially at little incremental cost, since the existing scale of the SaaS provider means that they are not creating new capability, merely leveraging existing capacity. Choosing the right provider allows you to reach your intended consumers at the best location using provider points of presence, (or a SaaS CDN provider).

Scenario 2

In a traditional environment the amount of consumption is predefined as part of the requirements gathering phase and often at peak load, the providers will be chosen based on their capacity to achieve this, as well as on other measures. This has a defined cost which needs to be approved. And this defined capacity approach leads to a static design.

In Hybrid IT Management Layer 2 – Business Service Owner, budget is still approved but asynchronously from the design at L1. The design is variable and is based on a least load capacity with the required redundancy for data and computational power, this redundancy provides the required availability throughout the points of

presence where required. Changes to demand are catered for in the design up to an upper bound defined by the budget. Changes within these upper and lower bounds are enabled automatically through designed elasticity. A trigger is created when the growth of the system is such that it will breach the upper bound (and therefore budget), only then will a change be initiated and is cost driven. Any change implementation therefore is much less frequent and no longer a response to something happening, but a prediction of something that *may* happen.

Whilst at Hybrid IT Layer 3 – Application Owner, mechanisms are in place that allow this to happen within a single application the service as a whole relies on the fact that changes in one application may require that changes in a sibling application may also need to happen, an orchestration system needs to be in place that picks up an event from an API and is capable of triggering action into another API. Perhaps an application provider has failed over to its alternate location, which means that data flowing flow it would not be allowed to enter a sibling application due to security rules designed into the application, the ability to change these rules without human intervention is key to the autonomic nature of the application and to avoid potential downtime in the service.

Layer 2 – Application Provider

In Hybrid IT Management Layer 2 – Business Service Owner there is a huge dependence on the ability of the provider to make the consumer aware of what is happening for the consumed part of the service, [NOT the SaaS application as a whole] particularly where it affects cost, security or reliability;

When combined with similar information from all suppliers and passed through an analytics engine with appropriate algorithms, there may be insights gleaned about areas in which there are service optimisation possibilities. Coupled with a ruleset codified during service design, or through service operation this can trigger automated action being taken [invoked by a 'receiving' API] to autonomously reconfigure the service.

These autonomous actions are also very likely to be affected by factors outside of IT. Consider marketing running a sales campaign for a product available on an e-commerce site, may require proactive action to be taken on the sizing of the applications. The cost of which is borne by the product category owners rather than through an IT budget. In the ideal scenario the design would accommodate these peak loads but by exception this may still be true.

Figure 6 - Application Provider

Drawing on the information from Hybrid IT Management Layer 2 – Business Service Owner, and with the knowledge that the Business Service Owner sees just as *'a.n.other application provider'* (supplier #7) we have an obligation to provide certain information (outbound), and certain capabilities (inbound). Let's summarise some examples that have been mentioned to this point.

Area	Information	Reference
Security	Access and Authentication Traffic Logs – source and destination	Data Access Patterns Data security
Cost	Cost per consumed application	Cost and commoditisation
Reliability	Application availability Optionally internal performance	Reliability
Quality	Activity Logs Traffic Logs – end user demand	Data Access Patterns
Action	Configuration capabilities	Actions

Table 4 - Provided Information (X-ref)

Cost

The total cost of consuming the application for the current billing period. Optionally, but ideally, with current rate of consumption information and burndown.

The information is easier for an internal application owner to provide since the created application is 'custom' for the business and has only a single consumer. However, traditional IT has historically had trouble identifying all costs associated with the provision of service, hence the lack of internal chargeback regimen in most organisations, so may find this difficult to achieve.

Also, this structure is inherently expensive as is not built for scale. If the IT organisation could provide the same application to multiple LoB users, or external consumers (multi-tenant) then costs can be spread accordingly and becomes more competitive against potential SaaS players, but this adds another complexity layer to already difficult cost identification.

Security

The provision of log information that is related to security. This is not to expose ALL security related information to the consumer, just from where and by whom/what data is being accessed and any data access attempts, either failed or successful.

This requires that adequate capability exists and is in place within the provider, both at the network and application layers and can be separated from the rest of the IT estate.

Quality

Activity logs from the application as relates to configuration that applies to the consumer. It merely requires that the application is developed in such a way that it tracks its own configuration changes.

Network traffic and data demand patterns for end-point consumption need to be available such that decisions can be made on the most appropriate placement of the application relative to demand. Subject to security, data and cost constraints

Did the application require to be failed over to an alternate site? Since there may be actions required to be taken to handle this failover on sibling applications.

If a request to scale the application was received or if an additional point of presence was requested, did this complete successfully and within what timeframe.

Reliability

The exposure of application availability information. Is the application running? And available for consumption for ALL customers at ALL application locations, this is not the same as the ability of 'sibling' applications to be able to connect to the application (service component).

This is a relatively simple health-check that should always correspond to the consumers monitoring of the same measure for a single application consumed by a single consumer at the same location, any difference is addressable by the service owner. For multi-tenanted applications then this is the baseline by which consumer monitored availability is measured.

There are no promises for performance, since this is within the remit of the consumers' configuration. For this not to be the case it would have to be consumed as a managed service with a separate and agreed SLA

Actions

Providing information is OK, but the consumer also wishes to configure the application without interaction with an intermediate person, since this saves time.

A minimal solution would be to provide a self-service portal for reconfiguration of the application, but although this would remove supplier side human interaction it does rely on consumer side interaction, and as has been said throughout the first section, the responses should be automated to address Problem 1 - Speed, and due to the removal of human error Problem 3 - Quality.

With this in mind the solution can only be to provide configuration capabilities systemically;

- Move point of presence location
- Scale up the application
- Shut down the application

- Limit access; by role, and by destination

Failure to provide any of the above, may result in IT being marginalised as a provider

Information	Possible Action	Area
Data transfer to/from SaaS application	Scale application up/out	Cost
Sources of demand	Move SaaS Points of presence	Quality
Authentication errors, invalid connection attempts	Quarantine	Security
New users added/old users removed	None (compliance only)	Security
Computational/transactional load decrease	Scale application down, reduce points of presence	Cost
Computational/transactional load increase	Multiple points of presence or CDN	Quality

Table 5 – Possible Actions

Layer 3 – Application Owner

This section describes the differences between a 'traditional' IT perspective and the growing requirements of an IT organisation assisting a business driving towards digital transformation and the problems described in Is 'Hybrid IT' a thing? It looks from the perspective of the owner of an 'internally' provided application, where the owner of the application is part of the business that consumes it. It can be assumed that all other SaaS application providers work in a similar way, within [or as] their own organisations. The IT organisation of which the application development team is a part knows that the business needs them to provide and maintain an application that can be consumed easily as part of a bigger business service.

In Figure 7 - Application Owner View (and other figures throughout this paper) each 'box' is a discrete entity, the 'internal' methods and practices used to provide the various components are not important when viewed from the consumer perspective, and therefore ANY mechanism can be used, although some are more likely than others to be more able to support the solution of Problem 1 - Speed and Problem 3 - Quality.

In the previous 2 sections it has been noted that each layer consumes some information, and is also the provider of information, sometimes this can be simply passed through the system, other times it needs to be generated, or transformed on the way. The figure below has a similar structure of one we visited in Figure 3 – XaaS service.

The areas of concern remain the same as for the business service.

- Is the application compliant from a security perspective?
- Is the application resilient?
- Does it provide sufficient assurance over quality, performance and availability
- And can it be provided at acceptable cost

Operational Excellence

Within each provider the mechanisms by which applications (also services) are operationalised within IT is not relevant to the provision of the service itself.

Whilst it may be good practice to measure improvement in terms of operational excellence. None of these metrics are important to the consumers of the applications provided

- How available and performant your systems are.
- How many incidents have been created/resolved and in what time, and
- How much automation you have.

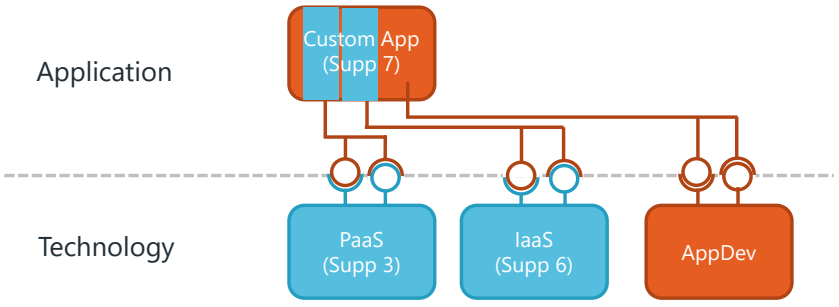

Figure 7 - Application Owner View

Similarly to the business service, and because of Problem 2 - Cost we would like to reduce the amount of work done, both in development and operation by internal teams, the onus is on the application owner to pass on the responsibility of most of the low level functions to suppliers, and consume them as a service.

At this level [public] 'cloud first' offerings are very appealing. As they do away with ownership of operational maintenance of various elements. For instance, in IaaS we no longer have to worry about hardware, for PaaS services such as databases, we no longer need to worry about patching, clearly a drain on the resources of internal IT.

The use of IaaS services therefore is less appealing than those of the higher level services like PaaS and FaaS (function as a service), the so-called serverless capabilities of public cloud providers. (see below)

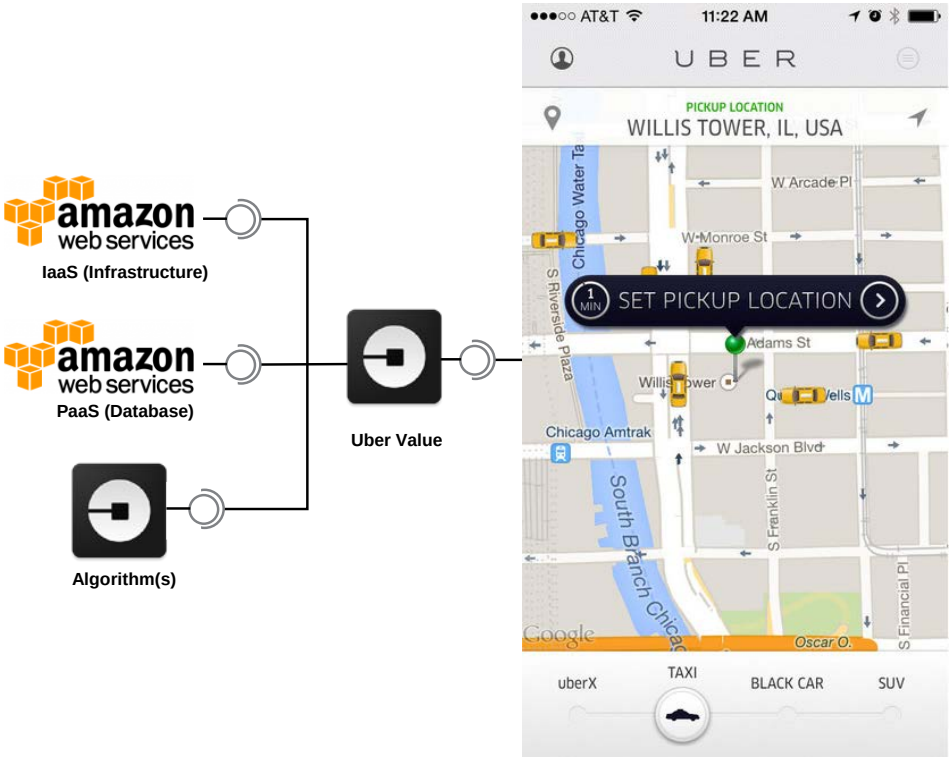

Figure 8 - Uber Layer 3

In Figure 8 - Uber Layer 3, the Uber example is taken to the next level[11]. The application, *Uber Value* is distilled into its various components, where commoditisable offerings are passed to low-level technology (public-cloud) vendors, allowing Uber to concentrate on its value; creating algorithms to get drivers to passengers for the right price.

IaaS Boundary services.

	IaaS	IaaS Boundary	PaaS
Example	Storage	VM	Database
Operational Maintenance	None	Security – compliance and patch management OS management	None
Development/Deployment	As Code as part of the application	As Code as part of the application	As Code as part of the application

---

[11] This is for example only and is not intended to represent Ubers 'actual' topology. In fact it is highly likely not to use any IaaS services, but use Functions as a service (Lambda) and/or higher level services for the machine learning platform, although for this example this simplified structure is sufficient

Table 6 - Public cloud aaS

There is much confusion amongst those with a traditional view about what public cloud (IaaS) providers actually provide. This stems from the fact that they see the 'cloud' as an extension to the on-premise virtual [machine] infrastructure that they know and love, it is not! Amazon Web Services attempted to prevent this confusion by calling its service Elastic Compute Cloud, but since EC2 is a nice abbreviation the nuance is lost.

Virtual Machines *would* be a PaaS service as they contain an operating system. EC2 and its peer equivalents from Azure and GCP however, are not, they are just a bunch of compute resources from various 'blocks' of infrastructure in a datacentre that only come together as a server at time of request[12]. Cloud providers in their benevolence have attempted to make life easy for consumers, by allowing them to choose an operating system to put on top (to make the compute useful), the cost of which is included in the pricing. Though this may appear like what they are providing is a VM [with an operating system], they are not. The operating system is the choice of the requestor and remains the responsibility of the requestor during operation. So whilst consumed SaaS applications at L2 – The service/application interface leaves only security (and some quality) concerns to address operationally, PaaS and IaaS leave us with much more operational responsibility, and not truly broaching the OpEx criterion of Problem 2 - Cost.

It is here therefore that we must remind ourselves that the strategy should not be 'cloud –first' but SaaS first for the business as a whole. Cloud-first strategies should be by exception only, and are [largely] restricted to the developer domain (see Reliability below), i.e. within the scope of IT. IT needs to understand this to be a partner in any digital transformation of the business, this may be a hard pill to swallow and people are very protective of the status quo as it affects their jobs.

Having said this, within the scope of this paper it is considered that the application required is not available through SaaS, and is to be developed internally, and created on a cloud provided service to minimise (though not remove) operational cost.

Data Access Patterns

The principles that apply at L3 – the application/technology interface are largely the same as those at L2 – The service/application interface. Although unlike at L2 where applications can either run on-prem or in-cloud for a specific service, components of an application likely to be co-located, all on cloud, or all on prem. Since there is imperative to reduce CapEx and the associated choice of cloud-bases IaaS and PaaS providers is it extremely likely that the application [code] will run on public cloud, for reasons we shall see.

Traffic analysis and logging between application components now also becomes important from a cost perspective, as well as for security and quality. Since IaaS/PaaS providers charge by data egress, between cloud regions[13], and to end consumers[14].

The design of the application should be 'responsive', i.e it uses this information directly from the suppliers of the information, or via an intermediary that has transformed it into usable insight, in close to real time, so that action can be taken as and when appropriate.

Data Security

Application Owners are custodians of consumer data; data must be secure between the application code as it travels between IaaS/PaaS elements and when resident on them, e.g database datastores, message queuing systems, or plain old storage, as required by the business service, noting that some business services do not require security of the n^{th} degree. As with everything related to security (and operations in general), prevention is better than cure.

---

[12] Put together by software through a declarative definition as part of the request process (the software defined datacentre)
[13] Also between cloud and on-prem components, although mixed environments will be rare due to cost constraints
[14] End consumers may be people using the application or other applications within the context of a business service.

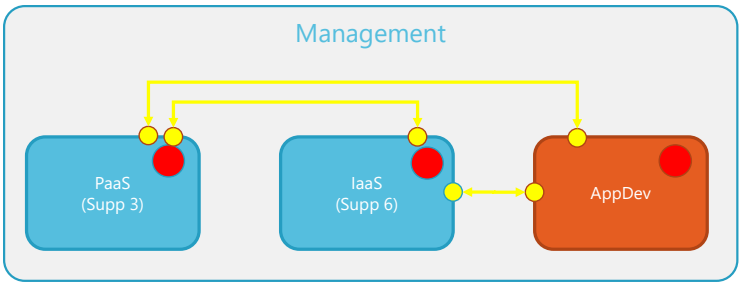

Figure 9 - Data Entities and Flows

This by definition is in the remit of designers not operators, and ***must*** be a design choice of the Hybrid IT Layer 3 – Application Owner though it is

1. Invisible to application consumers (business service owners), and
2. Unnecessary information for IaaS/PaaS service providers themselves

Again, similarly to L2 – The service/application interface, appropriate access controls, proactive penetration testing and vulnerability scanning is required continuously. Operating systems still need to be patched, and application owner responsibility in this model, as an IaaS service. Databases, however, as an example of a PaaS service may need to be disconnected, if the vulnerability is so severe (e.g SQL injection)

Reliability

As has been noted, at L3 – the application/technology interface, data egress is a major concern. This was not a problem for L2 with SaaS providers as data movement costs are absorbed into the cost of the service. It has implications on the design of application resilience; data egress between cloud regions (or to another on-prem or cloud service) implies a cost (in most cases).

L3 – the application/technology interface is the 'true' playground of DevOps as a quality and automation driven practice. At this layer the application code is joined with any provided IaaS and PaaS services to form a complete application. Since the application may call the IaaS and PaaS elements directly this appears to blur the lines between siblings (at the application component level). DevOps is the return of the programmer analyst, albeit as a team rather than an individual; 'if YOU create it, YOU run it!' The application owner at this level should ask 'How do things break? And, what information can be provided so that potential breakages can be 1) detected, and 2) counteracted? Both, when combined, should be viewed as a defect of design, not of operation. Any activities to work around the detected problem during operation should be built into the service design. See The changing role of ITIL in elastic environments.

Whilst DevOps may be seen as something that applies to custom developed applications at L3, when applied to SaaS providers at L2 the same principle is seen; the supplier providing the application is the same one that is responsible for operating it.

As indicated in Figure 7 - Application Owner View there is continuous ownership between the application itself and the application code provided by internal application developers (the same providing organisation). This is DevOps! As such, unlike L2 – The service/application interface, the AppDev provider is VERY likely to know a lot about their siblings and how they are consumed, but not their operation. They, as well as their application component siblings are on the hook to provide auditability during release in the same way as described in Design vs change vs release vs audit.

This is a key change to the metrics that are required for quality (see Reliability). They are related to problem resolution in terms of cycle time for defects for identified problems. However, this cycle time should not only be for defects caused by problems in production, it should include those that have been generated in pre-production phases [that are related to quality].

Cost

Where all other factors are the same, the only factor to compete on is price. Cost comparison is required by similar providers, especially where the service is commoditised, for IaaS providers the obvious commodity is containers, especially those orchestrated through Kubernetes, rapidly becoming the de facto option for container orchestration. A close to real-time analysis might be needed to choose the optimum provider at any specific time, similar to those provided by CloudGenera. These costs must be accommodated in the price of the provided application, and be designed with these considerations in mind.

These costs need to be minimised within trade-off boundaries, since provision of an application in the cheapest cloud region (see Reliability) may not always offer the desired performance of the application. Thus, knowledge of the use of the application is important (see Data Access Patterns).

This is possible where we are also part of the same business as the service owner, (and where custom application providers have an advantage over pure SaaS providers) but has obvious implications over placement during design.

The changing role of ITIL in elastic environments

In highly automated, elastic environment the general principles alter the bias with which some ITIL practices are used. They affect how CIs are viewed, and services are designed, and its implications for configuration management. How automation affects change management and other IT Operations practices such as change management, incident management and event management.

A note about CIs

With the assumption that we do not need to know any detail below one layer beneath where we exist and 'own' relatively little of our provided application, in terms of the actions that can be taken to resolve issues, this changes how CIs (Configuration Items) need to be represented, and the detail that any specific layer needs to see. For simplicity, two major subsets can be defined;

Inward facing (provided systems)

A low-level or detailed view of specific building blocks;

- Application Layer – IaaS/PaaS providers and application code required to build applications
- Technology Layer - servers, routers, disks, database instances required to run application code or provide PaaS, IaaS, FaaS services.

Outward facing (consumed services)

A high-level, logical view of collections of application or technology instances as a consumable service

Each has its own lifecycle in IT Service Management but both must be treated differently. Since any 'management' needs to happen only for the Outward-facing CIs, as they are exposed by the provider, and available to consumers, outside the box in Figure 12. Inward-facing CIs are visible only to IT personnel, inside the box in Figure 12 as providers of a 'service' are subject to whatever operational controls and processes are [currently] in place, be they;

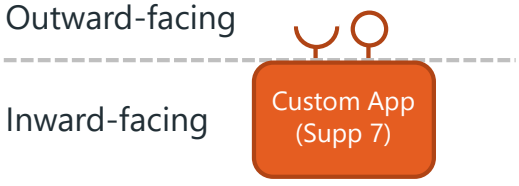

Figure 10 - In/Outward facing CIs

- Manual – often seen in traditional IT environments, or
- Automated – As part of SDDC, or internal or hosted private cloud

This is a key point, since it need to be recognised that 'management' at either L2 or L3 is agnostic of the processes that take place within individual suppliers.

If CIs have changed then what does this mean for configuration management? In short, configuration management is no longer about the management of individual instances, but *management of configuration*. Where configuration could be any number of possible combinations of instances within the bounds of a pre-determined (declared) configuration at design.

CI Class	Typical Scope	Perspective		Example(s)
		Provider	Consumer	
Inward Facing	Instance	Facet of design, does not care about end-use	None. (Knows that it must exist, but does not care in what specific form)	An infrastructure service --- a number of servers, logically grouped together in a cluster. A database service --- a TCP port made available via a load balancer on a single IP (perhaps using the infrastructure service). An application service --- an exposed URL that links to some software, (perhaps this is also installed on an infrastructure service), but maybe consumed as a service
Outward Facing	Logical	None (provided there is sufficient capacity)	A facet of deploy-ability. Requires summary endpoint information that describes the consumable service	

Table 7 - CI Classes

This means that no longer does a post-hoc operational 'discovery' of components need to take place. The configuration is held as part of a document, written during design, but ***as code.*** During deployment the designed configuration is made available to a configuration management system, but is merely a placeholder for what might actually exist in reality (or virtuality as the case may be, see sidebar).

Change Management

External facing CIs are consumed from the providers [as black boxes], and typically are amalgamated into another, larger, business service for consumption by end users. What would constitute a change in this structure?

- The introduction or removal of a provider service – a change to design
- An alteration in 'interfacing' between provider services – also, a change to design

Therefore the CI's that are relevant are design CIs, documents, actually configuration 'files', that change in current practice would change infrequently. But, as changes tend to releases (see below) may actually increase in frequency.

In either case, the ability of configuration management to ingest 'configuration' rather than discover CIs becomes increasingly relevant. (See sidebar)

However, the *total* changes to a service are greatly affected by the change practices of the providing suppliers. Even though an architecture may be fully responsive and resilient, your choices of suppliers may not.

Configuration Management

The traditional view is that each server that is part of a cluster is a CI, and the cluster itself is also a CI. When a cluster member fails we raise a ticket to fix it, or issue a change request to replace it.

Consider now the AWS auto-scaling group; we define the (max and min) number of instances that form the group and the launch configuration and we leave the provision of the system to provider health checks.

The only level we are actually interested is the group (or cluster) level that provides the service

Design vs change vs release vs audit

The ability of the service to self-heal *must* be built-into the service itself…automatically. Traditional change management can be 'designed out' of a well-architected system as resilience, in the form of fault-tolerance and elasticity, included as part of the design.

Consider simple A/B or multi variant testing and deployments to test resilience in production[15]. Setting up tolerance for a SaaS providers is simple and recovery practices can be tested continuously, prior to services being affected in a controlled way, leading to more confidence at the business level. If there is the confidence of the business that any alternations made to the design can be accommodated as a matter of course what is the need for traditional IT change processes? Approval is automatic since the risk is always low, since the service is tolerant of change. This means changes can be automated. Once changes can be automated then releases can be automated and the difference between a change and a release is not obvious. Once things are automated, then you do not need to know that they are going to happen, you just need to know that they have, leaving what used to be a cumbersome process as merely an auditable trail of activity (that can be provided to consumers).

To achieve this, suppliers must be transparent in their change practices, as these need to be distinguishable from downtime events. Although both types will impact your service, one affects operational SLA whilst the other impacts architectural choice and an upper bound to the service operation, defining the SLA.

Releases should have 'zero' impact on the operational service, regardless of their 'size', in practice this means that releases tend towards 'small' or even 'very small' (see types of change above). And the duration, as such, should be minimised.

Recovery Practices

Each period where a service is unavailable is a cost to the business. This is particularly true when suppliers are down, as downtime of a preferred supplier may mean using a more expensive alternative.

With this in mind; how often are the responses defined as part of actions to events [either security or system] tested? How do you detect unknown failure modes? In traditional environments this is seldom done, simply because of the inflexibility of the incumbent systems. The cost of this negative testing would be just too great. However, with as-a-service components, this becomes more applicable.

With consumable services we no longer have to wait for specific, infrequent events to happen again in order that we can test the response to any particular scenario, it is possible to spin up an exact replica of the environment as it existed at the time of failure (or just before), and test the solution for repeatability, or better still improve the design so that it no longer fails. Any common failure modes can then become part of standard, automated testing as part of general practice during development.

However, this still relies on the problem affecting service at least once, how do we detect those potential failures, prior to them happening, thereby proactively solving the problem? Ensure that application teams are writing enough traps into their code that can be captured, likewise with state data via metrics.

But, how do you proactively discover security flaws or unintentional holes in your architecture? You run hackathons and days when you specifically target the failure of your application whether it be production, some preproduction environment or a mirror of production. Netflix takes this to the nth level with its simian army, (Chaos Monkey available for use here)

The above is also true for workload placement and capacity, the ability of elastic environments to be (re-)configured simply and easily means that multi variant testing can be used to determine the correct configuration of resources in terms of scale or location, with reference to performance and cost.

---

[15] This can also be used to test new cost patterns as well as more traditional feature testing. Consider a SaaS provider with multi levels of the same offering (Basic, Premium, Complete) with each tier being rated on a data throughput per time increment basis, perhaps consuming the basic tier for 3 days is more costly that consuming the full tier for 12 hours with the same result.

Incident and Event Management

From an operational detection of faults perspective there are some attributes of highly elastic environments that affect these processes;

1. Elastic environments are *absolutely fault tolerant*, any availability concern that would affect service can immediately be routed around without impact to any consuming user.
2. Elastic environments are *scalable without limit*, so performance concerns are immediately mitigated by scaling horizontally or vertically without impact to the consumer

If there is no impact to the consuming user in either case then there is no service outage (or degradation), if there is no impact to service it follows that there is no incident. Through appropriate architectural practice, and the ability of the service to 'self-heal'. There is however, a problem to be solved.

Questions like the following still remain;

- Why was there a need to route round a potential availability issue?
- Why was there a need to scale [a part of] the service?
- What was happening within the service at the time of the intervention?

This is a significant differences in the bias that Incident management has within service management. Consider the current [simplified] processes in place in many organisations.

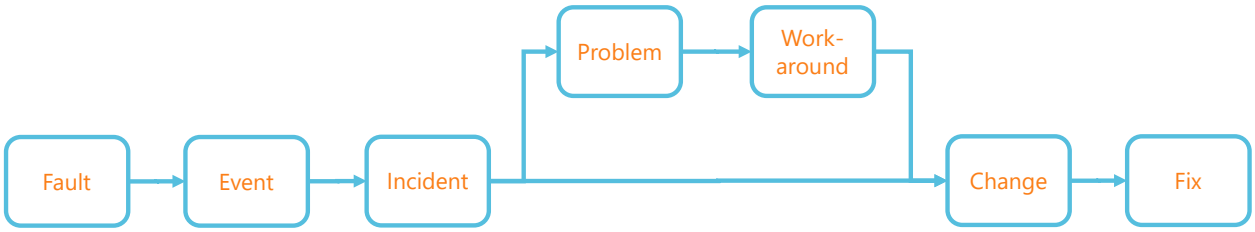

Figure 11 - Traditional Fault to Fix flow

As has been said though, since the known failure modes are covered via automatic responses built into the design this flow gets concatenated;

Event Management is not required in the traditional sense since there is no need for an operator (human) to take ownership and/or action. If external detection capabilities are required then there may be some thin layer of fault identification required, sufficient to call the autonomic response.

With and automatic response there is no service impact, this is preventative design. If there is no service impact then there is no need for an incident. Lucky since without event management there is nothing to systemically generate the incident. This relegates incident management to interactions with humans calling the helpdesk function. And the relevance of traditional metrics such as availability, performance and MTBF as quantified through the post-hoc reporting via incident resolution times is doubtful.

Since automated responses are standard changes, or releases, see above then change also disappears in favour of release management and control

For known failure modes proactively identified using Recovery Practices, no one would argue that the below isn't a more streamline process, and requires less human interaction;

Figure 12 - Streamlined Fault to Fix process

But what about those the 3 remaining questions that were mentioned previously? Something had to happen that resulted in a response (albeit automatic) by the designed system (service or application). In order for the same thing not to happen again we need to take some mitigating action. This takes us down the missing branch

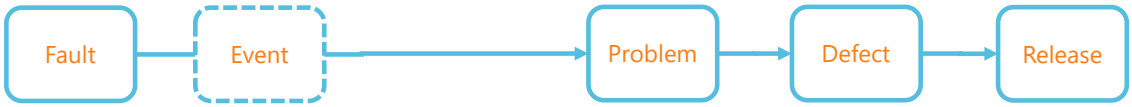

Figure 13 - Problem Diagnosis

In this process the response, and fault, are analysed for both correctness, and cause using tools that do not require any prior knowledge of the expected state of the system. Gathering logs, metrics and other potential influencing factors and combining them to determine why what happened did happen.

But what would *change*?

- The response to the known scenario, improving the quality of the response
- A change to the deployed system. An alteration to the design, or configuration file
- A net-new feature to the system. A new supplier, application, or feature. All facets of design, not operation

Returning to the use of incident based measurements of quality as raised in Reliability. These should then be replaced by the 'cycle time' for defects raised via problems using the route in Figure 13 - Problem Diagnosis