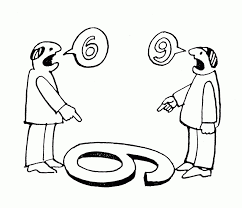
|  |  |
| --- | --- |
|  | Enterprise Cloud Control Plane Solution  An Enterprise Control Plane For the Cloud  November 2019  Version 0.1 (Draft) |

# Summary

In science and philosophy, a paradigm is a distinct set of concepts or thought patterns. How we think shapes what we do and it is in this context that I would like to propose that the Cloud represents a new paradigm for the enterprise that must shape a very different approach in order to participate in the economic opportunities have only recently become possible.

The previous decades of exponential progress in computing and network technologies have resulted in the conditions necessary to transform the economy through the application of economic benefits inherent in "software + the internet”.

The Enterprise Cloud Native concept outlined below is an attempt to articulate how the lens of the past is predominantly & inappropriately being applied to the opportunity that is in front of us. The purpose of this paper is to establish the conditions for a new mindset. One that is intent on creating the foundations that are most appropriate for the new kinds of business opportunities that are only possible to a Cloud Native Enterprise.

# The Problem - The Cloud Is Not Just Another Datacentre

## When The Computer Was Just A Server

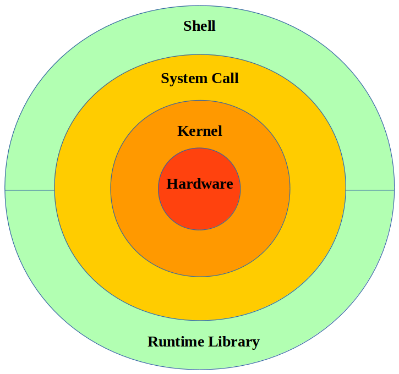
29th of October 1969 ARAPNET was born – as of November 2019 that makes it 50 years old. For most past 50 years - computing was dominated by the notion of a server and its operating system. Applications established a relationship with this computing construct and in turn operated within its constraints.

These constraints primarily related to the "illusion of process Isolation & dedicated resources". The Server was a physical\virtual unit that ultimately shared everything.

This gave rise to the need for an application that starts running when the machine finishes "powering on and self-test (POST)" and stops when you kill the power - this application is commonly referred to as the Kernel. The kernel's role is to isolate applications and their bugs\faults from each other; and presented an illusion of isolation and resource dedication through the concept of "system calls" that would perform 'interrupts" that effectively choregraphed a thread\process switching performance.

The vast majority of apps built over the era's of computing generations that have come and gone - mainframe, client-server, web and mobile have largely conceptually adopted the "Server as the computer" metaphor. Whilst more processing distributed from the server into the client over successive generations, the underlying app\server paradigm did not fundamentally change.

## The OS Kernel (SOE), Firewall Rules and the Private Network were the Primary Enterprise Controls

During the "Server" eras - the Operating System was the focal point of enterprise controls along with the networks that connected the servers that these operating systems were running on.

The permissions that were required to mutate the functioning of the Operating System Kernel were closely guarded as were the permissions for mutating the allowable networking communication flows.

Establishing an enterprise Governance and Risk posture that solved for the constraints\controls represented by the Operating system Kernel and the Network Firewall became the defacto substrate for securing the duty of care that enterprises had to uphold for the protection of data privacy and the integrity\availability of sensitive code. The physical origins of the Datacentre and the pre-internet era within which they emerged bestowed a level of implicit trust for the physical nature of private networks.

## Now That The Computer is a Datacentre

In the past 10+ years however, we have experienced the emergence of apps that have adopted a fundamentally different concept of computing - one that sees the datacentre as the computer - more accurately, very large datacentres as the computer ([warehouse scale computing](https://www.morganclaypool.com/doi/pdf/10.2200/S00193ED1V01Y200905CAC006)).

The apps that adopt the datacentre as the computer metaphor for computing operate from a quite different set of foundational principles to those of the "server" era. These apps are more appropriately described as "internet scale services".

 This perspective is a departure from a view of the computing problem that implicitly assumes a model where one program runs in a single machine. In warehouse-scale computing, the program is an Internet service, which may consist of tens or more individual programs that interact to implement complex end-user services

We define the computer to be architected as a datacentre despite the fact that Internet services may involve multiple datacentres located far apart - multiple datacentres are sometimes used as complete replicas of the same service - so the logical construct is the Datacentre as the computer with global replicas

The distinguishing features that separate client\server type apps from Internet services style apps are:

* **Massively More parallelism** - client server hundreds\thousands - internet service 10,000's to 1,000,000's
* **Increased Change Cadence** - by establishing an API abstraction - the implementation can change with no effect on the clients - so releases are much more frequent moving from quarterly to hourly\minute
* **Platform homogeneity** - the underlying hardware is cost effective from a supply chain and management perspective
* **Fault free operations** - the platform homogeneity creates an increased inherent risk of failure - so it is expected and incorporated into the design of the service - faults are considered a natural part of daily life. The cluster level infrastructure software creates an abstraction for the application that hides this complexity and in return the app must be able to gracefully survive a

○ Release,

○ Restart &

○ Reschedule

## The Kernel Has Fundamentally Changed as Isolation Became Real

The Warehouse Scale Computing paradigm created the conditions necessary for Cloud Computing to emerge. In warehouse-scale computing, the program is an Internet service, which may consist of tens or more individual programs that interact to implement complex end-user services. It is through this concept of the App as an Internet Services that cloud computing was born - Infrastructure, Platforms and Software became Internet Services that were distributed globally across Warehouse Scale Computing.

The IaaS, PaaS and SaaS providers needed to create a kernel abstraction for their users in much the same way that the Operating System kernel was required. This "Internet service kernel" had the responsibility of maintaining the "Security and Resource Isolation" for consumers of the shared warehouse scale computing construct. These Isolation constructs operated at the level of the Datacentre or region rather than at the traditional level of the Server\Operating-System. These new isolation constructs provided much greater levels of isolation - and the scheduling systems that the Internet Services were distributed across provided much greater levels of resource utilisation such that it considerably disrupted the cost structure.

Internet Services needed to provide hard multi-tenant isolation that created security and resource protection mechanism for tenants. Software defined networking, compute and storage provided the platform upon which the hardware within the datacentre became programmable - this created the ability to independently address control plane concerns and data plane concerns and to apply a "separation of concerns pattern" consistently across multiple tenants.

Tenants of the Internet Service (IaaS, PaaS & SaaS) became consumers of truly isolated constructs. The key dimensions that deliver this superior Kernel Isolation capability are:

1. **Identity** – who is attempting to access, where are they coming from and what is their contextual profile?
2. **Privilege (***RBAC, Authorization*) – The nature of privilege being granted (time, segregation, appropriateness)
3. **Scope (***Radius, Isolation***)** – the maximum level of impact that can result from the combination of Privilege and Identity

## By Adopting the Mindset of A Kernel Developer

Designers of automated systems administration tools can benefit from a certain mindset

***Think like a kernel developer, not an application programmer***

Traditionally a good multitasking operating system is designed to isolate applications (and their bugs) from each other and from the kernel and produce the illusion of independent execution. Systems administration is all about making sure that users continue to see that illusion.

As the Cloud Kernel Isolation constructs have changed, the previous illusion of independence can now be converted into reality and expanded to include not only app\services but entire commercial operating entities with varying levels of scope\isolation in between.

An opportunity presents itself to create an Enterprise Graph that represents the Identity or Blueprint of the operating entity. The enterprise graph provides the ability to see the entire enterprise, reason over it from multiple angles of interest and walk the relationships so as to zoom in and out.

This can only be achieved if the Enterprise Graph\Blueprint is treated as the back bone of State Management such that nodes in the graph are created and destroyed as demand fluctuates, and whilst the nodes exist the form the Space where in Values are changed over time. This provides the ability to record an immutable history of value changes relative to their space in time. This capability underwrites the ability to know everything as it happens and that ever has happened; Omniscience.

## An Opportunity for a New Kind of Enterprise Cloud Kernel Emerges

Within the context of the enterprise - cloud computing is increasingly becoming an attractive proposition. However, the adoption mindsets and approaches are typically following in the footsteps laid down by those of the "computer is a server" era.

The Cloud is an Internet Services that is exposed via a set of API's; meaning that Enterprise Cloud Infrastructure is now inherently programmable. The Enterprise Cloud Kernel can now become a Software Program, which is something that was very expensive to achieve previously.

By extension, this Enterprise Cloud Kernel has at its disposal, programmable Isolation constructs that are far superior to those of the traditional model. The Cloud internet service is designed to anticipate failure and remain stable even when the infrastructure it is running on is experiencing outages or is otherwise changing. This posture of "resilience through distributed isolation" is very much a shift in security and operational mindset from the traditional posture of "Centralised Stability". The Software defined nature of Distributed Resilience and Hard Isolation represented by the Cloud Internet service provides the Enterprise with the opportunity to create an Enterprise Cloud Kernel that elevates these security, isolation, resilience & cost structure benefits and scale them out across a global footprint of IaaS, PaaS and SaaS services.

# The Cloud Native Opportunity For the Enterprise

## Participating & Thriving in the API Economy

Traditional Enterprise Datacentre operating models that follow the "Heads out, Costs Down" trajectory of a predominantly "[SG&A](https://en.wikipedia.org/wiki/SG%26A)" governed business entity typically end up in an straight-jacketed outsourced contractual relationship that significantly limits the ability of the organisation to change and adapt to the operating reality.

Even if an enterprise manages to avoid the outsource "your mess for less" reality, the traditional operating model mindset is one of people managing many heterogeneous systems - requiring deep functional skills and cross functional processes that are exposed to the Enterprise Datacentre customers who are predominantly the Business Units that operate a P&L that generates revenue and profit for the enterprise. The defacto approach is to establish an IT Service Management Operations that creates "services" that can be requested by the business unit customers via a ticketing tool resulting in eventual fulfillment.

This experience rarely if ever provides a satisfactory experience to the Revenue generating business units - and is one of the primary reasons for the explosion of cloud related "Shadow IT" attempts. SaaS was the easiest place for Shadow-IT to emerge as it was very easy to open a browser, transact for a SaaS service - and no one in the Enterprise IT function was any the wiser.

SaaS, PaaS and IaaS represent a delivery model that supports the current expectation of the Revenue Generating Business Units and suits the current day competitive reality that necessitates access to a continuous stream of Cloud and Open Source Software (OSS) innovation.

The Enterprise Cloud Kernel provides an opportunity for a new kind of Enterprise IT service that can exist in parallel with the old paradigm yet remain sufficiently decoupled such that its superior operating benefits are not limited.

This new experience elevates the inherently Cloud Native benefits of:

1. Self-service & on demand API driven interaction model
2. Superior Isolation constructs
3. Globally distributed & resilient nature
4. Significantly optimised cost structure

Presenting the Consuming Business units with a Cloud Native experience specifically designed & codified for the Enterprise. One that is repeatable at low marginal cost and designed in such a way as to allow for continuous improvement and scaling beyond the point diminishing returns that is inherent in the traditional models.

# Enterprise Cloud Native Solution Consideration Set

The following concepts are considered core to understanding how the paradigm of computing has changed for the enterprise. The objective is to identify the customer, the value proposition, the shift in the computing dynamics and to outline some of the suggested solution aspects required.

## Who is The Customer in this Cloud Era: it's all about the Software Engineers

Cloud native engineers are an emerging tribe of economic actors that build & version API's that are continuously evolving instances of specialisation. They represent a source of both innovation and monetisation that creates the conditions for new kinds of business models to emerge that were previously impossible. Cloud native engineers collaborate around the building & versioning of these API's using RFC's. They change and improve these API's using PR's and Issues. They work in the open where their talk can transparently be measured against their walk and thus cultivates a culture of continuous learning by doing & sharing. The API's that they create can be used by authorised consumers who can be anywhere on the planet. The operational scale & rate of change that this creates dwarfs anything that has been experienced in previous computing paradigms. This represents a different challenge for the Enterprise - one that it has never experienced prior. This problem demands the Creation of a platform that has the group wide controls embedded whilst permitting Cloud native engineers to innovate; The existence of the cloud native engineer customer implies that an Enterprise Cloud Platform becomes a set of versioned API's that evolve through the use of RFC's, PR's and Issues. Honouring the cloud native engineering interaction model, these API's must permit authorised self-service interactions with the Enterprise Cloud Platform.

## The Potential Value of a Software Business: *Monetisation & Productivity at Global Scale*

Apps and Data are the primary objects of value within the realm of software. Apps possess the inherent property of being a potential source of monetisation or productivity. Whereas Data possesses the inherent property of being a potential source of insight that can lead to new or additional monetisation or productivity benefits. API's are the "authorised self-service" & OnDemand delivery mechanism for the instances of monetisation\productivity that are accessible over the internet.

## The Economic Promise of Software & The Internet: *Zero Margin Cost & Free Global Distribution*

The previous decades of exponential progress in computing and network technologies have resulted in the conditions necessary to transform the economy through the application of economic benefits of "software + the internet". Software is both Information and Machine. In its resting state it is information - and via the internet it can be copied & replicated at almost zero margin cost and transmitted globally, for free. However, once software enters its runtime state - it becomes a machine. It leverages the locally accessible computing power to "execute its function" & interact with data. The pervasiveness of powerful computing power and the global interconnectedness of people and things - increases the opportunity space for software to create new forms of monetisation and productivity that were previously impossible.

Now Cloud native software engineers can create “monetisation & productivity” assets that can be distributed globally at almost zero marginal cost. This moves the importance of this type of capability out of “[SG&A](https://en.wikipedia.org/wiki/SG%26A)” and in the “[COGS](https://en.wikipedia.org/wiki/Cost_of_goods_sold)” representing the most profitable opportunity for the business.

## Enterprise Software Architecture: Minimising the HR required in the building & Evolution of Software Systems

The speed to value property that Software possess - wherein a change is made to a text file, this text file is quickly converted into machine executable format and run - is an extremely seductive property. It focuses the mind on achieving the function. In the small, this is a fast path to function however it encourages short cuts that elevate the importance of "the function" at the expense of structure or design. Structure is the key to flexibility and by applying focus to structure within a context where there is continuous and distributed changes to different instances of function; efforts can be made that remove the diminishing returns that inherently exist when structure has been forsaken.

Change without attention being paid to structure creates monoliths.

A Structure that elevates the principles of

* "Single Responsibility",
* "Common Closure" and the
* "inversion of dependencies" where
* "dependencies flow in the direction of stability"

creates a structure wherein continuous refactoring and elastic expansion become possible.

Properly applied, software architecture will minimise the number of human resources required for the building and continuous evolution of systems. The incentives underpinning Software Architecture align with the "zero marginal cost" economic benefits embodied in "software & the internet". When combined it establishing a trend towards the zero-marginal cost of producing new versions of software. When you sufficiently peel back the layers of hype that typically encase Micro-services and SOA - the underpinning principle is software architecture.

## Enterprise Computing Complexity: Intentionally Avoid Drifting Into Complexity

Software architecture plays a critical role in avoiding the inherent “drift into complexity” that marks the trajectory of many enterprise cloud ventures.

A complex system is one wherein its components interact in multiple ways following local rules in the absence of a higher instruction set that defines the universe of possible interactions. This produces the property of emergence where the only credible sight is hindsight.

Within such a system - it is technically impossible to understand\define the current state, therefore there is no way to manage or apply controls. Complexity is the root cause of most problems within software today. Understanding how complexity causes problems is the first step towards architecting a structure that solves for this.

The single biggest cause of complexity in software systems results from the poor\mismanagement of state. If you let complexity creep into your system through the poor\mismanagement of state - complexity will breed more complexity and you are now out of control of your system.

The key to managing complexity is staying out of it. This can be achieved by explicitly managing how you identify state (value, space & time) and how you change state (different value, same space, different time). Anti-patterns to these two sources are - mutable changes to state where the history of previous value\time is lost via implicit control logic.

The key to staying out of complexity is through the "immutable (versioned) 'full-definition' of state" that is applied through a "declarative interface via an immutable (versioned) deterministic ordering" of control logic. Within such a state managed system - any divergence from the intended state is either a code bug or a breach in security. State is fully described and tightly controlled (*avoiding complexity*) within a structure that solves for the continuous refactoring and elastic expansion of software function. These functions are instances of state.

## Continuum of Computing Concerns: *Multi-Cloud & Multi-Era*

The utility of "Apps and Data" can outlive the era in which they were created. Each era is defined by the specific architectural dependencies that underpinned the value optimisation e.g. Mainframe, Client-Server, Web, Mobile & Cloud Native. The Enterprise Cloud Native Solution must cater for all apps & data, across all era's and focus on a logical solution that can be applied to all public clouds - therefore it must cater for both "system and process virtualisation"

# Primary Solution Objectives

## *Zero Trust @ Maximum Benefit*

Within todays distributed multi-cloud and multi-era compute environments - taking a dependency on a "place of trust" that is typically dependent on a physical space - is increasingly impossible - whilst its original implicit claim of "trust" was questionable in the first place. In today's globally interconnected reality - the primary assumption must be one of "***trust only if you can continuously verify integrity***".

* *To possess Integrity within a computing construct - is to be capable of identifying and enforcing proper modification to system state*
* *State originates from textual values at a point in time*
* *This state becomes subject to "control logic" that converts the desired textual state into runtime-operational state*
* *Integrity is the probability that the runtime operational state is exactly what was desired, and that the desired state was properly originated*

As the sophistication and innovation of the dark web continues to improve at pace - the risk-based application of cyber security becomes a first-class concern of any platform that exists to host apps that interact with private data. Additionally, when the economic opportunity represented by "software and the internet" are so existentially compelling - enterprises must embrace this opportunity whilst minimising the limiting effects that a zero-trust security model could have.

## *Fault tolerant @ Scale*

The cloud native enterprise solution must

1. Scale to many independent tenants, these tenants in turn host a number of API's, all of which are operating concurrently (*hosted in the Enterprise Graph*)
2. Be capable of distributing workloads across the cloud providers operating space
3. Offer a Zero-Downtime promise with respect to maintenance & operations

In essence - the solution must be Multi-Tenant & Fault Tolerant - that is to say that – the degree to which it can demonstrate “Resilience”, “Availability” and “Recoverability” is a function of its Tolerance to faults. Faults can be benign or malicious in nature - Failure vs Breach. The essential concern for a Fault Tolerant system is Fault Isolation which is best achieved through a graph of isolated nodes\partitions. The Multi-tenancy dimension of the solution demands a cost effective, scalable & Secure approach to delegated administration.

## Cloud Native App & Data Lifecycle Platform for the Enterprise

The primary purpose of the Cloud Native Solution within the context of the Enterprise is to provide a lifecycle platform that optimises for

1. The inherent "monetization" & "productivity" benefits of Apps and Data whilst at the same time
2. Preserving the economic promise of "software and the Internet", whilst also
3. Leveraging the economic benefits of a structure that optimises the cost of creating new versions of Apps & Data solutions

## *Delegation Of Administration @ Scale*

The fault tolerant problem space points strongly in the direction of a partitioning solution - the creation of zones of autonomy that are strongly isolated from each other with respect to both security and resource consumption

In order to achieve zones of autonomy, delegation of management must be possible, but not at the cost of the system wide privacy & integrity.

Attention now turns to the problem of scaling the administration of "many zones of delegated administration" optimising for

1. Marginal cost (lowest)
2. Security (highest)
3. Simplicity (usability)

The problem that we are attempting to solve for now is Access & Privilege Management

### Fine Grained Privilege: *substantially increase policy management cost & degrade security*

This approach to access & privilege management is inherently complicated. As the administration space grows in scale if the management method uses "manual activities" complexity emerges and by definition - becomes inherently unmanageable.

If managed via "automated activities" complexity can be avoided - however the following inherent issues remain present.

1. Privileges are modelled based on the "needs of the app\service" being hosted within the zone. The specific behavior profile needs to be mapped to the minimum privilege needed and then - a policy for the zone is created that exactly matches this profile. This introduces an amount of cognitive friction and delay for every "app\service to zone" combination. This complexity increases the administrative demand and due to the intricate nature of the mapping between the need of the app and the privilege definition it creates more opportunities for misconfiguration. This can impact security & productivity. This approach tends to result in an approach that simplifies of the privilege definition and in the worst cases - ends up allocating dangerously powerful privileges to app\services
2. A second issue that arises from this type of management approach is app\service incompatibility - as new enhancements to an app are required - the profile needs to be extended - and this introduces a continual loop on an app\service basis - that adds to the demand on the central administration function. Additionally, if two apps of similar profile share the same zone and require segregation - then both cannot exist within this zone and a new zone needs to be created
3. Transitive privilege can also emerge as a side effect of being resident on a platform where the underlying privilege infrastructure is constantly expanding and changing.

### Partitioning: *side steps the problems introduced by fine grained privilege*

The root cause of the complexity is twofold.

1. On the one hand there is the expressiveness of privilege. Coarse grained policy - like the UNIX style - Root or User - is less expressive and simpler to manage. Most enterprise implementations tend to move in the direction of such simplicity due to the effects that it can have on management cost
2. On the other hand, are the security implications of using a more simplified expression of policy tends to grant higher levels of privilege more broadly.

Within the context of a multi-tenant environment where it is advisable to adopt a low to zero trust model - this coarse grained approach creates a significant security challenge as the scope of privilege is not mutually exclusive - it over laps introducing the risk of malicious or accidental impacts.

Partitioning or compartmentalizing administrative functions into strongly isolated zones - removes the security problem and provides the conditions necessary for a multi-tenant solution that allows for inexpressive policy and mutually exclusivity.

This solves for the 3 inherent issues relating to Fine Grained Privilege Management

### *Scaling Expressiveness: what if we need different kinds of Policy Expression*

Partitioning provides an additional property whereby - Security Policy can be locally managed. As the scope of the zone is the extent to which the privilege is relevant - different zones can be decorated with policy definitions that are of different levels of expression.

This introduces a scaling challenge - however, this is largely offset by applying a ***Declarative Policy Overlay*** that instantiates a "class of expression" and binds this with a defined Zone definition through automation. Allowing for this "class of expression" to be overlaid against a zone - means that it can be applied to more than one zone and thus the scope of privilege can also be explicitly managed.

It is now necessary to enforce the mutual exclusivity at the level of the definition of Scope of the zone.

# Foundation Principles: *Cloud Native Enterprise*

It is within this context that the following principles are embedded into the foundations of the Enterprise Cloud Native Solution. The solution needs to elevate the following principles so that they emerge as a function of usage.

## Ownership & Accountability

The "right of possession" comes with the duty of responsibility, therefore the boundaries of ownership and accountability must be crystal clear and segregated from each other. The coordination problem in the enterprise is generally riddled with “a lack of clarity” regarding ownership & accountability. Friction, delay, budget overruns, and conflict prevail.

## Coupling & Cohesion:

The enemies of organisational scale and agility are:

1. Process friction,
2. Cross functional dependency and
3. The law of diminishing returns.

Team sizes of 6-8 people optimise for trust and productivity. A Team's delivery effectiveness & efficiency is optimised through autonomy. The ability to scale a team of teams is optimised through "human readable\Machine executable" inter team processes.

## Least\Appropriate Privilege

Access to and usage of "Info and resources" are subject to legitimacy, necessity & timeliness. When it is appropriate for a significant amount of powerful privilege to be granted - the risk function is a combination of Time and Impact Space. Whenever it is "appropriate" to grant significant privileges then is should be offset by a reduction in both "time" to offset probability, and scope to offset impact.

## Correctness of Critical State

The lifecycle management of critical state must be capable of supporting deductive reasoning so that the correctness of state can be associated with truth\fact (*a value at a point in time applied to a specific space*). For state to be considered critical - implies that is of the greatest importance. (*Immutable Values & Pure Functions where the "pure function" is also an immutable value are central concepts to adhering to this principle*)

## Modular Composability & Verification

Modules should do one thing and do it well. They should be independently verifiable and expect that its output becomes the input for another Module. As their outputs are to be expected to become inputs for other modules versioning, compatibility and support are essential properties of modules. The universal communication method between modules should establish a dependency on the stability and persistence of the OSI stack.

## Inherent Code Flexibility

As Software eats the world – an Enterprise Cloud Kernel must elevate the architectural flexibility property that optimises for ease of change over correctness of function. It is more important for code to be changeable than it work. Code that does not work but that is easy to change can be made to work with minimum effort. Code that works but is hard to change will soon not work and will be hard to get working again. Code needs to be flexible enough to easily add\remove\modify features. A system that creates to conditions that support this principle will de-risk the consumers against the predominant incentive models that reward the opposite behavior.

## Elevate Lifetime Value of Digital Assets

As the life time value of a digital asset has historically outlive the era of computing optimisation that it was created in; and as the current trajectory & speed of computing innovation increases; the platform must elevate the digital asset such that it can continue to deliver value beyond the lifetime of the originating computing era. It must therefore be decoupled from modules and encourage\facilitate the importance of "single responsibility" and "common change boundaries".

## Assume Breach\Failure

For any computing system, the probability of breach is always greater than 0%. The probability of failure is also always greater than 0%. The inevitability of change increases the probability of both breach and failure. This reality must be embraced and embodied into the operating reality of risk management.

## First Principles Threat Controls

Rather than reasoning by analogy or being fragmented by hype, we must boil the cyber threat model down to its most fundamental truths and reason up from there. When we do we will find that there are only two fundamental threat scenarios.

* A Breach COULD Occur 🡪 Probability
* A Breach HAS Occurred 🡪 Impact

## Defence in Depth & Scope

The Probability of "Breach\Failure" must be subject to a number of independent controls that is equal to or greater than 2 so as to optimise for probability. These controls must be capable of being applied to a partitioning solution to optimise for impact in the event that a "Breach\Failure" actually occurs

## Default to "Fail\Breach" Secure

It must be a conscious decision to increase the probability or impact of a "failure\breach" equation within the system. By default, the system must favour the lowest possible impact and probability. Any change in this default equation is governed by "*Correctness of Critical State*".

## Omniscience

Every Action, Interaction and resulting Outcome is to be known as it happens and never forgotten. The Omniscient substrate must preserve the spaces (partitions) within which values change over time - of critical interest are resource\policy state changes and identity control\data plane interactions

# References

Many of the insights from the following resources have been synthesized into the narrative above.

* [Why order matters](http://www.infrastructures.org/papers/turing/turing.html): Turing equivalence in automated systems management: November 2002
* [Making reliable distributed systems](http://erlang.org/download/armstrong_thesis_2003.pdf) in the presence of software errors: December 2003
* [Integrity considerations for secure computer systems](https://ban.ai/multics/doc/a039324.pdf): April 1977
* [The Datacentre as a Computer](https://ai.google/research/pubs/pub35290): Introduction to the Design of Warehouse-Scale Machines: 2009
* [Out of the Tar Pit](https://github.com/papers-we-love/papers-we-love/blob/master/design/out-of-the-tar-pit.pdf): Feb 2006
* [Piaget’s Theory of Cognitive Development](https://www.massey.ac.nz/~wwpapajl/evolution/assign2/DD/theory.html)
* [Software is eating away](https://a16z.com/2011/08/20/why-software-is-eating-the-world/)
* The [key economic change](https://stratechery.com/concept/aggregation-theory/distribution-and-transaction-costs/) introduced by the Internet is the effective elimination of marginal distribution and transaction costs