Nexus of Evolutions and NextGen Application Delivery Approaches

(A PiA© White Paper)

1. Introduction

Digital Business - the ways of conducting business is changing at an immense pace. Emerging new business models not only blur the digital and physical worlds, but also disrupt the ways we engage with customers, take orders and exchange requests.

The Nexus of Forces [1] — the convergence of mobile, social, cloud and information — has already become the platform for digital business.

- *Really... who is the customer? a person? a virtual account with delegated access by Google? or an appliance?*
- *How does nexus of forces impact enterprise applications? information exchange patterns? encoding of information? application interfaces? transport? service orientation?*
- *Which layers of a typical enterprise application fracture and diminish by this earthquake? which new concepts flourish? Systems of Engagement? Systems of Aggregation? Systems of Records?*
- *Is expiry date reached for once upon cutting edge technology service oriented (SOA) and layered enterprise applications? Will multi-tiered applications survive? Or is the NEW enterprise application just merely suite of lightweight interacting services?*
- *What about operations? How can applications automatically scale and self-deploy on a massive infrastructure? How can precise measures and operational analytics feedback be provided to this new way of working? Dev into Ops, Ops into Dev and treatment of infrastructure as code?*
- *Is merely "keeping the bad guys out" security approach OK? Single-sign-on enough? How regulations (SOX, PCI and others) and fine-grained entitlements based authorization requirements impact application architectures?*

For enterprise solutions and products to thrive in an era of exponential change, design of enterprise applications must stay abreast of key trends across a broad spectrum of technologies with the right strategic choices made.

As a turnkey solution and BSS/OSS product vendor, what is PiA-TEAM stance amid this turmoil? What are the architectural key takeaways and guardrails for next-generation PiA-TEAM developed solutions and products?

In this regards, this white paper shares insights regarding "how information technology, development, operations and computing evolutions in the past decades" influence and change our ways of working - Nexus of Evolutions.

Finally, the document provides a summary of key takeaways and guardrails to be considered for planning, designing, building, deploying and operating next-generation enterprise application architectures - Key Takeaways and Architectural Guardrails.

Those applications; with utmost time-to-live resilience that avoid symptoms of a rotting design against functional, operational and infrastructural changes.

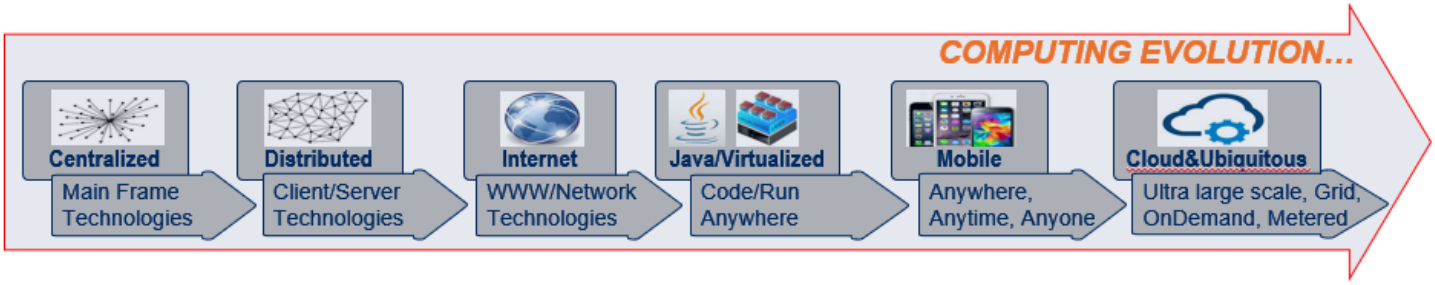
2. Nexus of Evolutions

How things evolve in the world - typically depicted by drawing circles. Even though conceptually sometimes it may seem "back to square one", the new state generally has many new benefits over the old as well as bringing its own new dilemmas to trigger a new cycle of change, such as; is cloud and its providers the new mainframe? are we back to centralized computing?

Enterprise applications have evolved dramatically in the past decades. The following **Nexus of Evolutions** need to be considered for designing, building, deploying and operating next-generation solutions and products. There may be many different ways of such a listing, but we wanted to follow the logical footsteps of cause-and-effect viewpoint to naturally discover "the architectural guardrails of next-generation PiA-TEAM developed solutions and products".

2.1. Computing and Cloud Evolution

The following diagram depicts the computing evolution.



In the mainframe era of Centralized computing; systems were expensive, maintenance and troubleshooting was cumbersome. The IT people to design, build and operate systems were also in short supply and thus utilization of mainframes via timesharing of a) access to these systems b) processes on these computing resources made sense.

As the scarcity of these resources diminished, businesses wanted more freedom and control. This led to mini-mainframes being owned by companies and finally to PC revolution.

Computers became cheaper with businesses providing PCs to tens, then hundreds and finally many thousands of employees as standalone systems. As a side effect, information sharing and exchange of information problems sprouted; transferring information using floppy disks was not practical. Common access to shared information was becoming a necessity more and more.

Networked PCs came into existence; token ring, ethernet – with new ways to connect all those computers together so information could be shared. Was peer to peer access enough? It did not make sense to have everybody keeping their copied version of truth - this led to putting data on one centralized file server so that everybody can access. As shared information grew, this time security concerns raised; businesses started looking for solutions to manage authentication and manage access to resources; Windows Active Directory, UNIX Kerberos realms and alike.

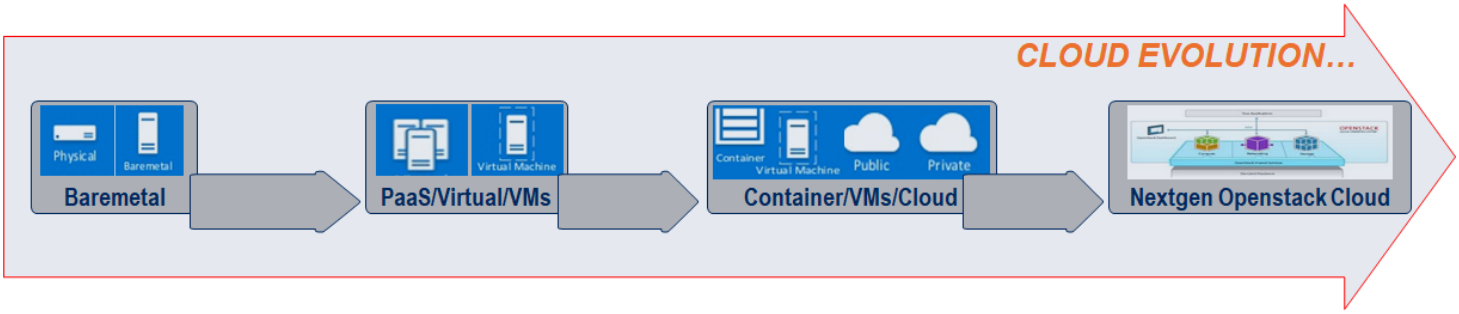
Following, Internet took over as the common interconnectivity backplane, among many other impacts, kicking off "The network is the computer" [2] era. With the introduction of Internet services, especially Netscape leading World Wide Web and world of internet browsing, the needs to access, visualize and process over-the-network-linked information irrelevant of desktop machine and operating system type were arising.

The effect? Virtual machines and virtualized languages; even though virtualization concept goes back to 1960s, it is not a coincidence that Java and JVM (Java Virtual Machine) and VMWare have been founded in years between 1994-1998.

Since then with the uptake of mobile devices and enhancements in mobile telecommunications, we observe ever increasing volume, velocity and variety of information processing over ubiquitous computing anywhere, anytime, always connected and more and more over mobile devices.

The consequence has been a new IT paradigm shift; **Cloud**, on one aspect a networked and distributed computing model; on another dimension a centrally managed model for enabling ubiquitous access to shared pools of virtual compute, network, storage and application resources.

For enterprise organizations, the adventure that once started as outsourcing bare metal servers and then infrastructure as services, turned into **client-cloud computing** based enterprise application model; the entire platform being offered as a service.



For individual users, working on many mobile small footprint devices as well as computers, switching between these devices require cloud for common storage as well as common capability services accessible anywhere any time.

Meanwhile, virtualization has come a long way. First it started virtualizing at machine/CPU level; type-1 hypervisors running directly on bare metal (such as VMWare ESX/ESXi), type-2 hypervisors virtualizing machine on a host OS (such as VirtualBox, QEMU etc.). Introduction of KVM in 2007 turning Linux kernel to a hypervisor blurred the distinction between type-1 and type-2, effectively converting the OS to a type-1 hypervisor.

On the other hand; new container approaches merged (Docker initial release 2013); why virtualize and decouple at machine level? instead decouple dependency at OS level resulting in a more lightweight isolation of application from its surroundings. All this has been creating a new perspectives.

In 2010 as a joint project of Rackspace Hosting and NASA lead to Openstack[3] foundation in 2016. A full blown open source platform for creating and managing private and public clouds with full freedom and control; dynamic control and virtualization of compute, storage and network resources through a datacenter. This still was not enough since still deploying, maintaining and automatic scaling of applications with service discovery mechanics changed from cloud vendor to vendor.

The result was Kubernetes[4] focusing on containerized applications rather than virtualized apps and providing a platform for automating deployment, scaling, and operations of application containers

across clusters of hosts. Kubernetes v1.0 was released on 2015.

That is more or less where we are today, with many companies already moved to the cloud and many more considering such a move. The client-cloud computing model among many other benefits offers cost savings and convenience with additional possibilities as well as new challenges; central management of all capability services within enterprise applications, secure access, authorization and management of this vast information base and on-the-fly dynamic processing scalability with automated and standardized deployment mechanisms.

So, even though totally a different playground, one can argue that we are back to centralized computing with cloud. What does that mean in terms of predicting future trends?

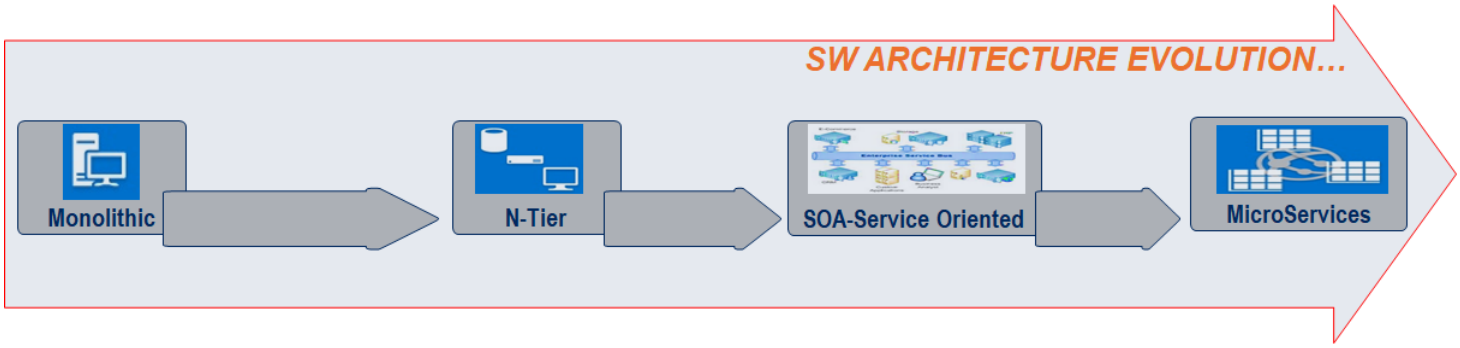
Like other trends, we believe "**client-cloud computing**" will run its course and culminate in a second cycle moving partially **AWAY** from centralized computing, a similar pattern at a different scale to bounce back from; mainframe to PC to networked to internetworked to cloud. We have already started observing some forerunning indicators;

- With evolution of standards, some Enterprise organizations have more tendency for setting up and managing their own virtualized environments and private clouds; Openstack and/or Openshift/Kubernetes. The drawbacks of cloud costs and external dependencies are a concern for enterprise. Just like PC revolution, organizations demanding more control of freedom, still keeping the capability to extend if needed on public cloud is leading to this.
- With the diversity of new gadgets on an Internet of Things and connected into our daily lives, we have started seeing more and more personalized in-home solutions boom; devices being sold like old times PCs. So people will want to control their own data, manage privacy with full control of information flow valves; what, when and how to back-up and question synch with public cloud. One such example is; My Cloud Personal Network Attached Storage servers. So eventually, all this may lead to new ways of computing creating a new balance between personalized versus centralized.

2.2. SW Architecture and SDLC Methodology Evolution

2.2.1 SW Architecture Evolution

Inline with Compute and Cloud Evolution detailed in previous section, the following diagram depicts the SW Architecture Evolution.



late 1970's and 1980's

For the sake of this document we can define a **Monolithic Application** as an application running in its single isolated process handling all behavioral logic as well as storage and retrieval of data.

Following typical engineering practices, and programming language constructs, for sure a certain level of modularity can be achieved by dividing the processing logic into sub routines, separating information access logic from business logic and packaging of reusable elements into reusable libraries. Processing scalability can also be achieved by running multiple instances/processes behind a load distribution mechanism if needed.

1990's

With the advent of networked computing and increased complexity of business applications, it did not take too long especially for enterprise applications to first split database layer from application. Then, this trend is followed for full **N-Tiered Application** style. Mostly 3 layers; a relational database as the persistence layer, server-side back-end applications that may communicate over a middleware layer and a separate UI layer.

2000's

The SW Architecture and IT industry were hit by the storm of Service Oriented Architectures (SOA) **SOA Application** style; a **distributed** processing architecture promoting self contained software components providing **services** to other applications. Dozens of best SOA implementation practices emerged as well as technologies and products that support SOA are still around and in-practice; Enterprise Service Bus(s), http-soap protocols etc. Unfortunately, companies learned the hard way via many failed projects that **unless implemented correctly** SOA was a massive, expensive, complicated architecture style.

Purpose and intention of following a concept often impacts the way how we do things resulting in a failure or success; the same was for **SOA**. The application of SOA concepts in real life to details of design whether be over lightweight modeled entities or coarse grained, whether be entity driven or functionalistic without any precautions on extra burdens of SOA; communication errors handling etc. resulted in chaos in huge projects.

> Service-oriented architecture is less about how to modularize an application, and more about how to compose an application by integration of distributed, separately-maintained and deployed software components [5]. It is enabled by technologies and standards that make it easier for components to communicate and cooperate over a network, especially an IP network.

2010's
In enterprise applications pain points are around modularity as well, and without putting in place a proper game plan, blindly applying SOA **will end up with further complexity without bringing in any modularity**. **The pain points originally aimed to be addressed by SOA are shifting towards achieving full blown distributed modularity rather than solely focusing on interconnectivity.**

Today after, reflecting on lessons learned, **Microservices Application** style is taking the IT industry by storm as the go-to style for developing highly scalable and modular applications.

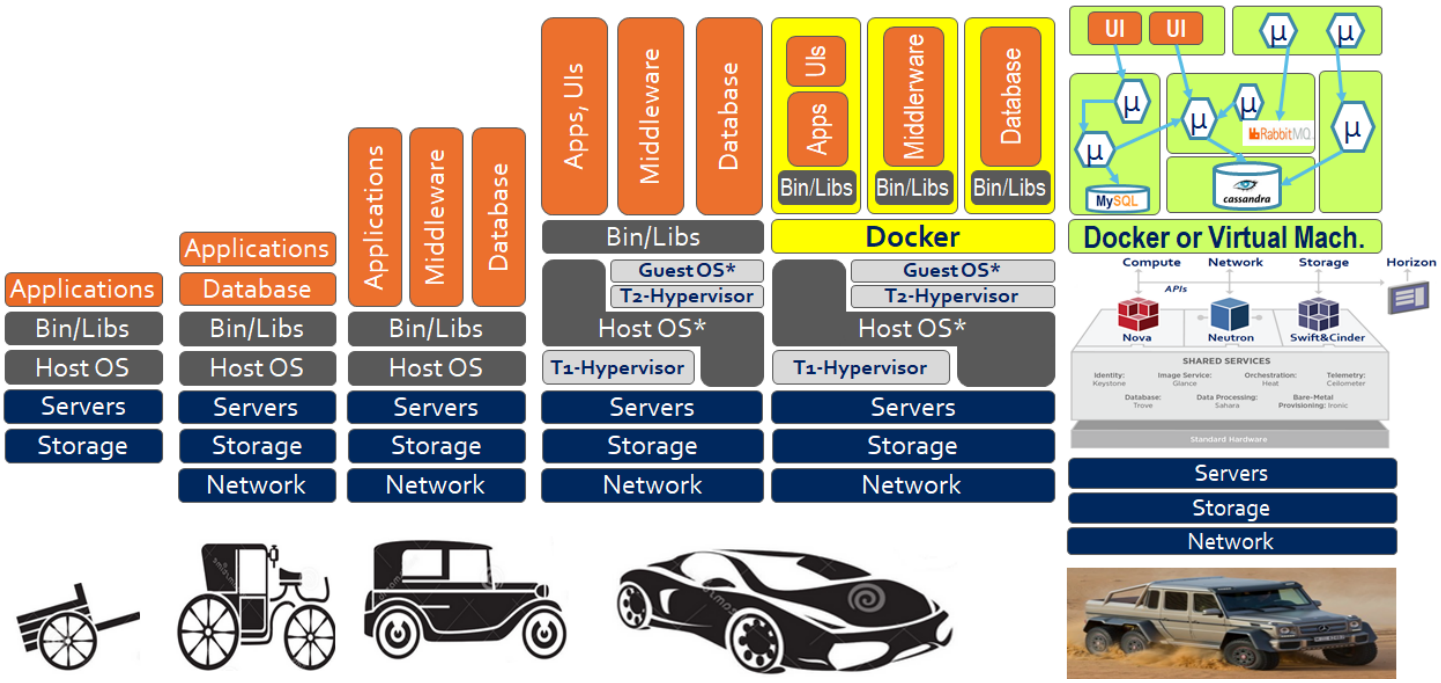
There is still NOT an easy formula or prescription to decide to WHAT degree of fine granularity a **Microservice** shall be modeled and HOW an enterprise application can be modeled as a "suite of small services, each running in its own process and communicating with lightweight mechanisms [6]". However, the foundational principles with enhancements based on mistakes of the past are there with Microservices approach.

The intuitive mystery around the art of software design is still around;

- no golden rule for smelling things that are prone to change and making those aspects of application configurable
- no magic methodology for foreseeing skeleton patterns and sensing things that are likely to remain same

2.2.2 SW Deployment Architecture Evolution

Inline with SW Architecture Evolution detailed in previous section, the following diagram depicts the SW Deployment related evolution.



Software packaging and deployment has come a long way since mainframe era; when hardware was the expensive part, in those days software was mostly bundled with the hardware.

Starting with 1980s, with PC boom and uptake of mass market PC software, software was packaged and shipped in various media types; floppy disks, CDROMs etc. with out-of-the-box installation scripts and configuration UIs, the installation was always on an OS (operating System) running on bare metal.

Starting with 1990s, with the advancements in network connected PCs and internet boom, several ways of distributing software through the network emerged. The initial release of debian package manager in 1998, is one of the successful examples as a game changer of managing complex package dependencies as well as library dependencies.

In 2000s, installation of software from centralized repositories through such package manager tools with auto update and patch mechanisms were already in practice in many mainstream OS distributions such as; Ubuntu Linux, Windows etc.

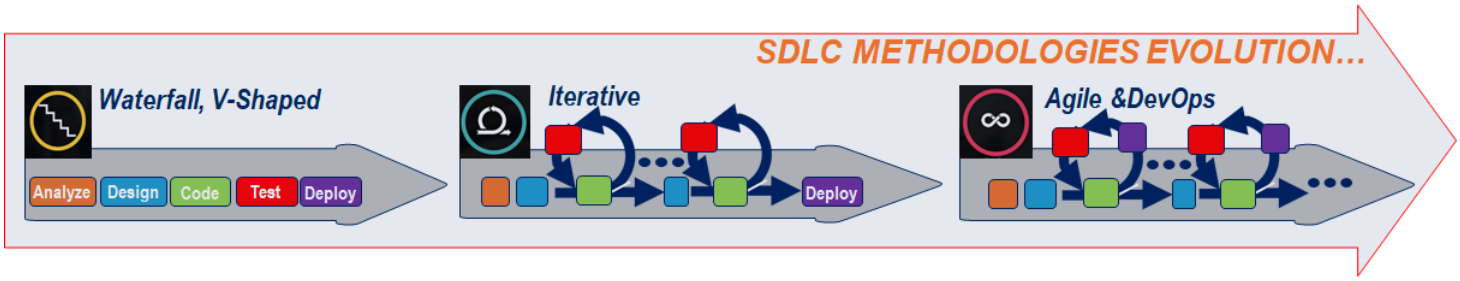
After 2010s, today, merely installing an enterprise application is NOT enough. Large enterprise organizations with application stacks running on cloud whether be private or public need automated ways and templates of deployment over 100s or 1000s of virtualized server environments or containers over a short period of time.

In other words, merely packaging of software inline with one of the industry package manager formats such as RPM (Redhat Package Manager) or DEB (Debian Package Format) etc. is NOT enough.

Out of the box software must now support relevant templates or automated deployment mechanisms utilizing tools such as; Ansible, Puppet etc. resulting in software provisioning, configuration management, and application deployment over the cloud; this is whole a lot more capability compared to merely automized installation of software to a single machine.

2.2.3 SDLC Methodology Evolution

The following diagram depicts the SDLC Methodoloy Evolution.



Since mainframes, **Waterfall** is the most straightforward and oldest of SDLC methodologies.
Analyze, then **Design**, then **Code/Build**, then **Test**, then **Deploy** and finally **Operate**.
Waterfall game plan is simple; finish one phase properly, then move on to the next. Do NOT go back. If you find an improper missing element from previous phase, fix it in current phase. If you happen to find issues with root causes from several phases back, Good Luck!!!
In short, Waterfall is easy to understand, simple to manage, but rigid.

V-Shaped "Verification and Validation" model, is a slight spin-off from Waterfall. A corresponding quality assurance contract is associated with each waterfall stage making the V; such as Analysis of Requirements yields and is associated to Acceptance Testing, Design related to Integration and System Testing and so forth. It's still difficult to go back and make changes like Waterfall, but quality assurance has a special focus to prevent big surprises.

With 1990s, the foundational concepts and practices encouraging **Iterative** approaches take place, such as Scrum. Instead of assuming fully known requirements and trying to complete a full blown analysis and design, a set of software requirements prioritized, analyzed, designed, coded and tested. A new version of semi-stable, but NOT production-deployed software is produced with each iteration. The iterations are repeated until the complete software is ready.

Since the degree of agility required by business is still not fully addressed with Iterative Approaches, after 2010s, today **Agile&DevOps** are the popular practices.

On the **Agile** dimension, the product or solution is broken into smaller cycles; on-going continuous releases produced, each with small, incremental changes from the previous release, but what is different than **Iterative** that it is also and deployed in production. In bare minimum, this requires continuous and automized testing, plus deployment.

Agile blended with **DevOps** on the other hand is a new approach to Software Engineering; unlike previous methodologies SW Development and Operations work together in the SAME team for the SAME common goal; considering only functionality from Dev perspective is NOT enough, but also operational perspectives of packaging, automated deployment and operational monitoring concerns must be addressed. This requires automation tools for continuous integration testing including deployment scripts and version management of also operational config files. These scripts and templates checked-in side by side with source code.

The good part is that SW engineering does not end with deployment, and **DevOps** force mixed teams of Dev and Ops assuring more complete **Operationally Manageable** design with enough paid attention to logging, system metrics, monitoring, package update(s) and patching aspects.

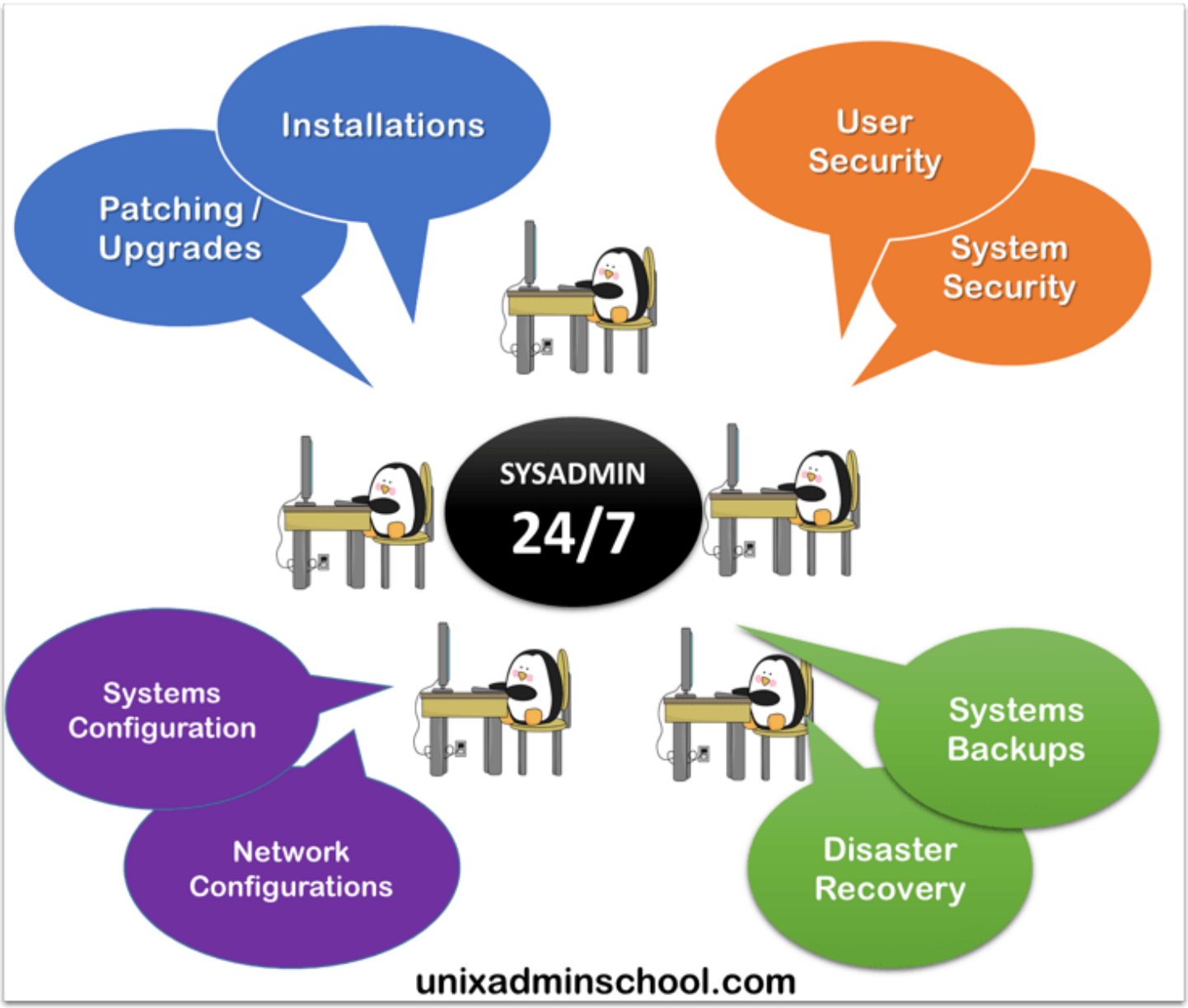
For sure **Agile&DevOps** better addresses today's concerns as detailed in previous SW Architecture Evolution sections and is still evolving with many helper tools and technologies.

2.3. Other Impacting Areas of Evolution

2.3.1. System Administration and Operational Analytics

This section draws attention to two key areas related to operations and system administration.

The diagrams below depict the Old School System Administration [7].

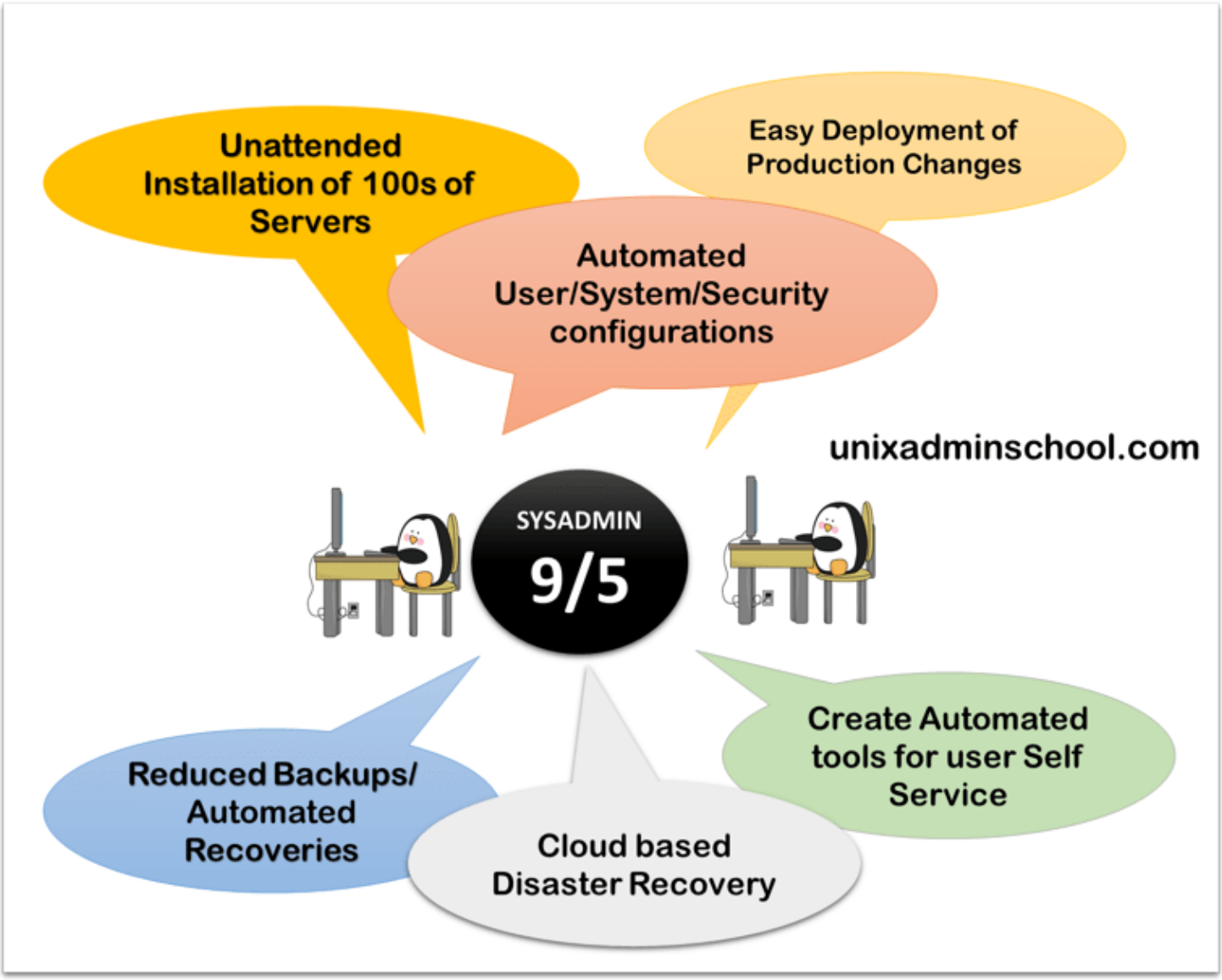


The olds days of dealing with the tasks below on a relatively low number of systems installed applications are over;

- Add/remove some user accounts and passwords
- Configure basic security settings
- Install some additional packages needed to run the applications
- Update OS kernel parameters

- Configure machine and tools for basic CPU, memory, IO usage monitoring.

The following diagram depict the School of System Administration now [7:1].



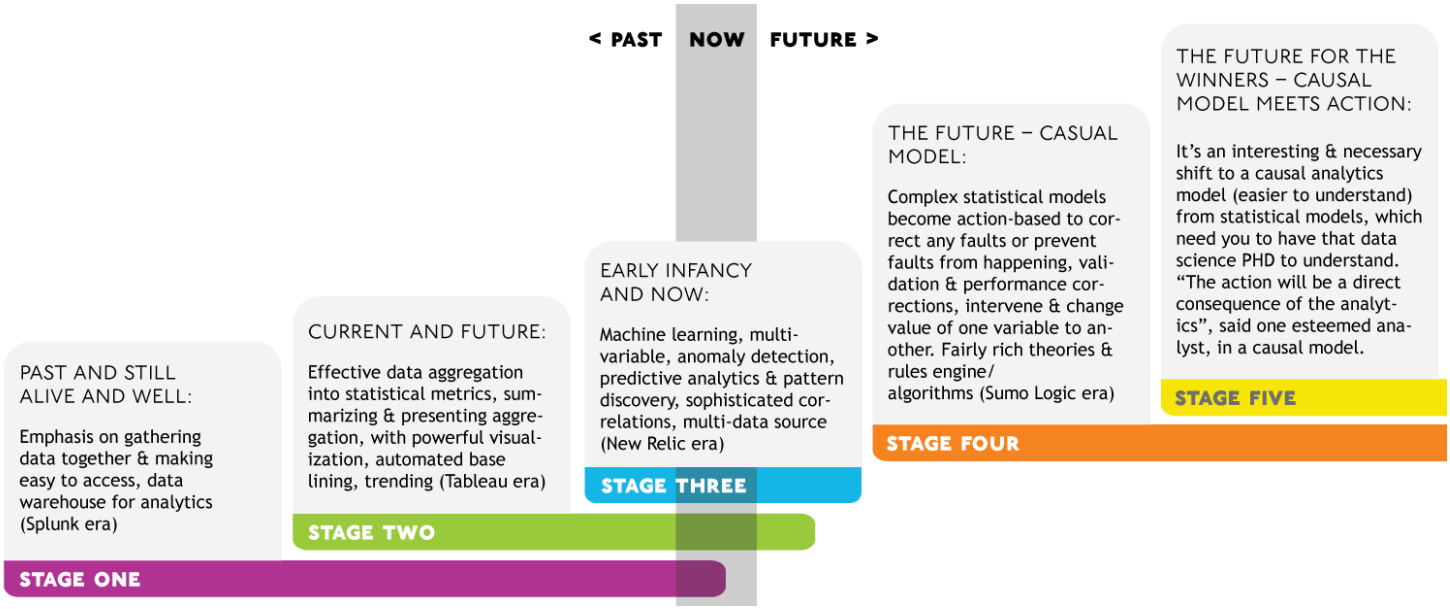
With DevOps the above are NOT merely problems of Operations any more, but instead considered part of the application. Thus, fulfillment of these requirements can be streamlined provided that applications conform to latest best practices and tools such as Ansible, Puppet etc.

Managing Sysadmin tasks and infrastructure as code with DevOps and Openstack cloud has already become a reality. All below is possible;

- Specify steps to create user once, and let tool do the same for each new user and for each machine
- Specify the templates to deploy application and propagate configurational changes and let the tools to apply these templates over 1000s of machines

As the number of deployed nodes over cloud or virtualized environments increase actionable insights is becoming a must. An application must make it possible by design root cause analysis that can only be achieved provided that metrics, counters and transactional logs are all in place and can be correlated.

In this regards the below diagram depicts the evolution in operational analytics [8].



At a bare minimum, new generation enterprise application must provide centralized logging over multiple nodes with relevant indexing engine capabilities and out-of-the-box dashboards utilizing technologies such as ELK-Elastic Search, Kibana and LogStash.

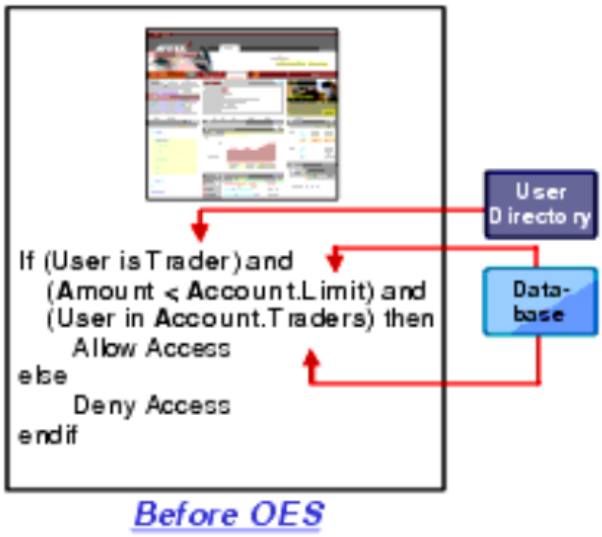
2.3.2. Security

The bare bone requirements for providing secure access to enterprise applications and authentication to various functionality in these applications have evolved dramatically in the past decade.

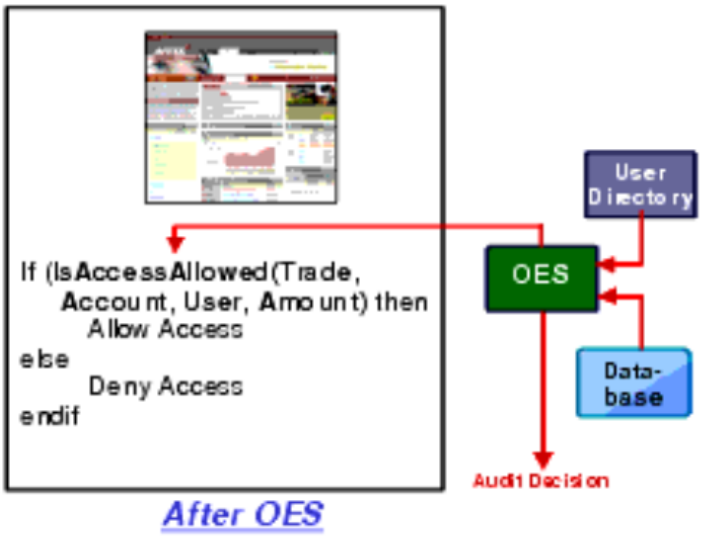
"Keeping the bad guys out" is not enough, secure web protocols only (ssl, https) are not enough; single sign-on solutions with deeper access control buried into these enterprise applications are minimum baseline.

Changes of regulations in enterprise application landscape; SOX compliancy, PCI compliancy and alike have been introducing changes in the enterprise application design. Sensitive Personal Identifying Information (PII), critical aspects such as; **data at rest**, **data at transit** must be considered during design.

Fine grainer access and authentication via user entitlements and the separation of application security logic from the application tier is a must. The below logic depicts the shift of change in design approaches in this regards[9].



Before OES



After OES

The new enterprise application architectures must be supporting with built-in and part of design mechanisms to support and be pre-integrated with external Entitlement servers considering PEP (Policy Enforcement Point), PDP (Policy Decision Point) concepts with clear strategies of RBAC (Role based Access Control) or ABAC (Attribute based Access Control). Choices must be made as part of design regarding usage of standards such as; NIST RBAC, XACML, Open AZ (PEP Decision API) and JAAS (Java Authentication and Authorization).

3. Key Takeaways and Architectural Guardrails

After shedding light to Nexus of Evolutions in previous sections and based on background information provided, the initial questions subject to this document can now gain more clarity in this section:

```
What is PiA-TEAM stance amid this turmoil?
What are the architectural key takeaways and recommended approaches
   for next-generation
      solutions and products?
```

This section provides a summary of key takeaways and guardrails that we consider at PiA for planning, designing, building, deploying and operating next-generation enterprise application architectures.

Many companies and teams understand (or claim to understand) "WHAT" the following Agile Manifesto means.

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The challenge is "HOW" to be truly agile. Just like a chain, a full design, development, test to shipment with continuous integration and deployment is only agile as its slowest link.

"HOW" to be agile on analysis&design is different than how the development team accomplishes the same goal. A whole project with very agile architectural, development and test approaches may still end up elephantine, if the next stages of Agile to Continuous Integration followed by Continuous Integration to Continuous Delivery have note been achieved.

We all know that technologies and the given names are prone to change, but the main engineering principles stay. At PiA, the below is our approach and manifesto shedding a light to the "HOW" in our approach to be truly agile. Note PiA's holistic top-down approach which has the reverse order compared to original Agile Manifesto items listed above.

- Responding to change over following a plan – Many projects that claim to be developed agile, with waterfall style strict project management boards will end up being non-agile. At PiA, we continuously aim to assess incoming use cases, their impacts on critical path considering functional as well as non-functional and CI/CD dependencies, evaluate their risks and potential impact on the current design. Only such a holistic approach can prevent a surprise, where the

relevant package is delivered in agile fashion, but blocked due to a missing Lab or integration preventing CI/CD and thus from the customer's perspective still not delivered.

- Customer collaboration over contract negotiation – Many projects blindly pursue the execution of a contract to fulfill a list of official requirements and percentage of completion. At PiA, we clearly collaborate with customers to understand and address the main pain-points or gain-points that trigger the relevant required capabilities. Only a clear understanding of these will assure a refined design, result in simple solution, simplified development that naturally results in less surprises and defects.
- Working software over comprehensive documentation – This motto of agile manifesto does not underestimate the importance of documentation as commonly interpreted by developers. It is the people that deliver and not the documentations or methodologies. In today's concurrent and collaborative engineering landscape where multiple teams in geographically disperse locations contribute to delivery, the purpose of the design and architecture team driving the solution is to support and assure delivery and the right documentation is the glue to prevent lack of communication. Therefore, PiA accepts the reality and is adaptive to the style and selection of a variety of design artifacts. A diagram is created only if it is beneficial for the solution design or development instead of a spreadsheet or a static word document. For sure, vice versa, for another case that requires a more detailed state analysis or data mapping, then for sure a different set of documents will be provided.
- Individuals and interactions over processes and tools – PiA design and architecture teams actively participate and evaluate the different aspects of agile approach. Typical approach of focusing on only development and test team agile communication is not enough. PiA approach is to blend functional user stories with non-functional dependencies and assuring operational and security constraints. This can only be achieved via continuous cross domain communication and individual interactions.

(A PiA© White Paper by Hudai Sami Asmaz)

References

1. https://www.gartner.com/en/documents/2049315/the-nexus-of-forces-social-mobile-cloud-and-information "Nexus of Forces" ↩
2. https://en.wikipedia.org/wiki/John_Gage "Sun Microsystems" ↩
3. https://www.openstack.org/ "Openstack" ↩
4. https://kubernetes.io/, "Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications" ↩
5. https://en.wikipedia.org/wiki/Service-oriented_architecture "Service Oriented Architecture" ↩
6. https://martinfowler.com/articles/microservices.html "Microservices" ↩
7. http://unixadminschool.com/blog/2016/06/evolution-of-system-administration-to-techopsdevops-culture/ "Evolution of Systems-Administration" ↩ ↩
8. https://dyn.com/blog/evolution-of-it-operations-analytics/ "Evolution of Operational Analytics" ↩
9. https://docs.oracle.com/cd/E12890_01/ales/docs32/secintro/entitlements.html "Oracle Entitlements Server -getting Security Out of Application" ↩