Architectural Considerations for Open-Source PaaS and Container Platforms

by Brian Gracely | Saturday, July 2nd, 2016-Wikibon.com

Wikibon.com

Architectural Considerations for Open-Source PaaS and Container Platforms

by Brian Gracely | Saturday, July 2nd, 2016

http://wikibon.com/architectural-considerations-for-open-source-paas-and-container-platforms/

Architectural Considerations for Open-Source PaaS and Container Platforms

by Brian Gracely | 2 July 2016 | Analysis, Cloud

Premise: The market for open source PaaS (Platform-as-a-Service) and Container platforms is rapidly evolving, both in terms of technologies and the breadth of offerings being brought to market to accelerate application development. Many IT organizations and developers are now mandating that any new software usage (on-demand consumption) or purchase must be based on open source software so that they have greater control over the evaluation process. In addition, many organizations want the option to choose whether their deployments are on-premises or use a public cloud service. As the pace of change accelerates for many open source technologies, IT organizations and developers are evaluating the architectural trade-offs that will impact new cloud-native applications as well as integrations with existing applications and data.

The focus of PaaS and Container platforms has always been to simplify the developer experience in deploying applications into production. But there has been great debate over which elements of this simplification were the most important. Over the past 3-4 years, the following topics have been areas of platform differentiation and debate:

- **Polyglot Language support.** Early platforms were built for specific development languages (e.g. Java, Ruby, .NET), frameworks or runtimes, but this has evolved to the point where every platform can now support a wide variety of languages. This support may be native platform functionality, or be available through an add-on service or via support through containers.
- **Multi-Cloud support.** By definition, PaaS and Container platforms should be able to abstract any underlying cloud platform (e.g. AWS, Azure, Google Cloud, OpenStack, VMware, etc.). This has evolved to where most platforms will run on any platforms, with deployment packages, API tools and marketplaces to support seamless migration of applications between those cloud platforms.
- **Docker support (for on-boarding applications)**. Early platforms supported many ways to upload applications, packages and dependencies into the system. But as more developers have adopted Docker containers as their standard for packaging and shipping applications, support for Docker became standard on all PaaS and Container platforms.
- **Schedulers**. Inside of every PaaS or Container platform is a complex, distributed system which is responsible for managing the underlying resources that run applications. These system are called “orchestrators” or “schedulers” and their job is to coordinate the creation and availability of the underlying containers or VMs which provide the infrastructure for applications on the platform. Schedulers are currently an area of much debate, as various schedulers (e.g. Cloud Foundry Diego, Kubernetes, Apache Mesos, Docker Swarm) have different scalability characteristics and different levels of

integration with containers and other elements of the platform.

- **"Serverless" or Functions-as-a-Service support.** While PaaS and Container platforms deliver a simplified deployment model for developers, an even simpler version is emerging. This model, called "Serverless" or "Functions-as-a-Service" (FaaS), removes even more operational concerns from the developer and is beginning to gather some buzz from developers building single-page web, mobile and IoT applications. This is a capability that will eventually become an embedded service within any viable PaaS or Container platform. An excellent introduction to Serverless can be found here and here.

Less than a year ago, Wikibon published a series of research focused on Structured and Unstructured platforms, with a focus on how these platforms were designed to help developers build cloud-native applications. The evolution of PaaS and Container platforms has significantly evolved over the past 9-12 months. While some platforms are still highly Structured, the growing trend has been for the previously Unstructured platforms to become more "composable" or even Structured. Wikibon defines "composable" as a packaged offering that leverages a set of modular open source projects, but is more tightly integrated as a set of services that accelerate developer productivity and application deployments. Composable platforms are becoming more "opinionated" in their architectural choices, but they still allow architects, developers and operators some amount of architectural flexibility that may not be present in Structured platforms.

NOTE: Some vendors prefer the term "Containers-as-a-Service". That designation can to be aligned to the vendor technology being sold, but Wikibon views the overall platform as being about more than just containers – including continuous integration/continuous deployment (CI/CD), authentication services, application services, data services, security services, cloud infrastructure and many other elements to enable application deployments.

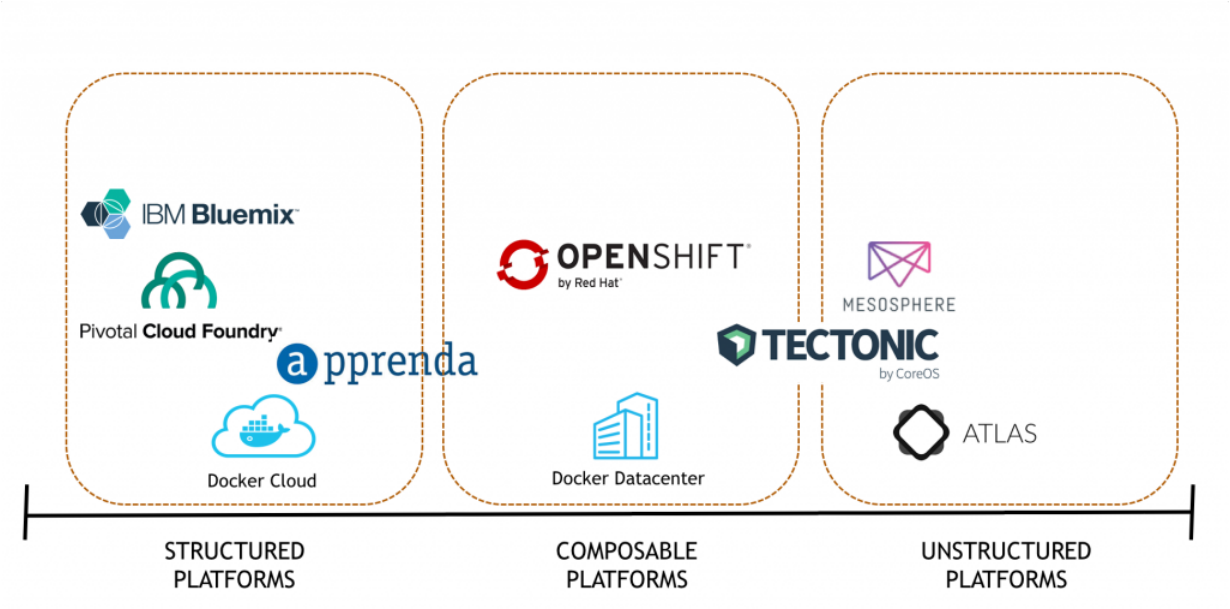

Figure 1: Structured, Composable, Unstructured Platforms (Source: Wikibon (c) 2016)

These following elements were the criteria for inclusion in this PaaS and Container platform research:

- **The core platform must be based on open-source technology (platforms under consideration – Cloud Foundry, Docker, Kubernetes, Mesos).** This allows end-customers to have transparency of architectures, the benefit of packaged/supported commercial offerings, as well as broad communities of developers to provide input into architectures.
- **The offerings should include compatible on-premises and public cloud offerings.** This provides end-customers with the ability to choose the consumption (purchasing) and operational model that best fits their business needs.
- **The offerings should have flexible architectures which allow integration with 3rd-party tools and external integrations (e.g. new open source projects).** This provider end-customers with the ability to determine where they want to integrate tools that may be limited or not native to the platform (e.g. identity service, logging, monitoring, functional processing, data analytics services, etc.)

The following offerings were evaluated for this research analysis:

- **Cloud Foundry:** IBM Bluemix (Public, Local) – IBM Bluemix is a broad set of cloud application and infrastructure services that can be consumed as a public cloud ("Public", via IBM Softlayer data centers) or managed as a ("Local") local service on a customer's premises.
- **Cloud Foundry:** Pivotal Cloud Foundry, Pivotal Web Services – Pivotal Cloud Foundry is a cloud-native application platform that can be operated on-premises, operated on a public cloud, or consumed as a public cloud service ("Pivotal Web Services").
- **Docker:** Docker Datacenter, Docker Cloud – Docker Datacenter is a Container-as-a-Service solution that can be operated on-premises or operated in a public cloud. Docker Cloud provides a universal control plane to operate Docker containers across multiple cloud infrastructure.
- **Kubernetes:** Red Hat OpenShift (Dedicated,Online, Container Platform) – Red Hat OpenShift is a container application platform that can be operated on-premises ("Enterprise") or consumed via the cloud as a ('Dedicated") or public ("Online") service.
- **Mesos:** Mesosphere DCOS – Mesosphere DCOS is a data center operating system for running container applications and stateful services. It can be operated on-premises or in a public cloud.

NOTE: These platforms were selected to provide a cross-section of the marketplace. This research is not intended to be an exhaustive list of the entire market, or to choose a "winner", as different platforms have different characteristics that align to application and business needs.

NOTE: The features of any given platform are constantly evolving, both in the commercial and open source versions. Specific (updated) details can be found on vendor websites, GitHub repositories and tracking sites such as Passifyit.

Two Critical Questions to Ask about PaaS and Container Platforms

At the core of every PaaS or Container platform, the underlying system elements were

primarily designed for newer, distributed applications and patterns (e.g. microservices, 12-factor apps, etc.). While this is fine for startups, it limits a platform's viability for customers with existing (legacy) applications . For any CIO, Enterprise Architect or DevOps team looking into PaaS or Container platforms, there are two initial questions to ask any vendor or open community:

1. **New Applications** – How can new applications be on-boarded to the platform? How are they supported in the platform?
2. **Existing Applications** – How can they be on-boarded to the platform, supported in the platform, and how much change is needed to run the application on the platform?

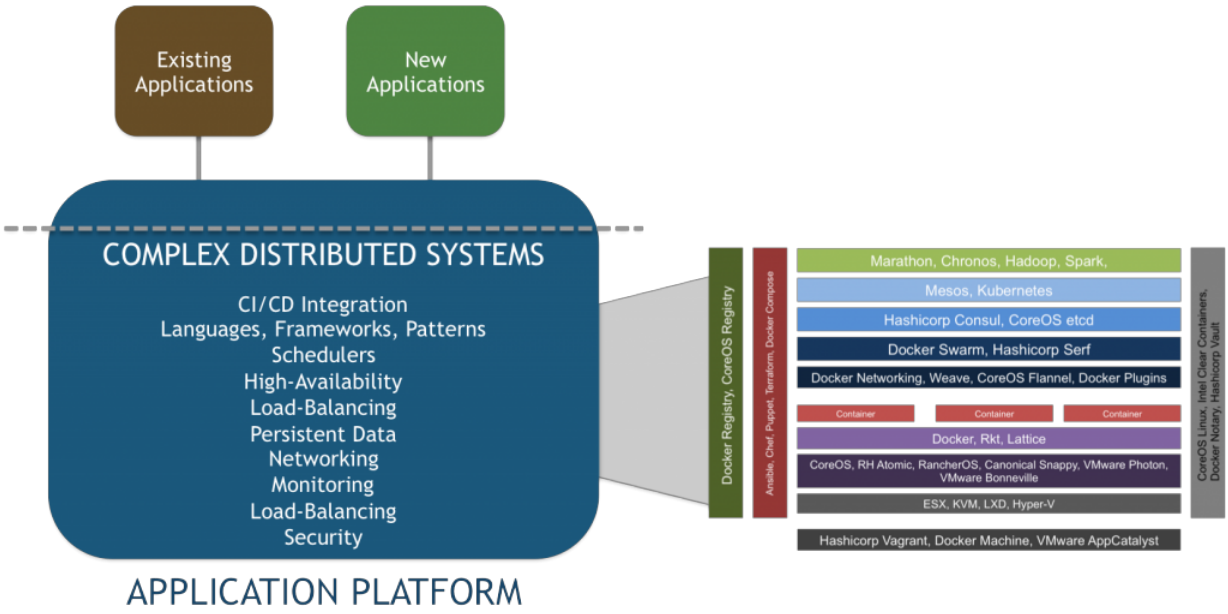

Figure 2: Deploying New and Existing Applications onto PaaS and Container Platforms (Source: Wikibon (c) 2016)

For new applications, the paths to on-boarding are fairly well defined. Applications (or code) can either be natively pushed to the platform, or the platform can accept containers or functions. In either case, the input can either come directly from a developer, or come as part of an automated system input (e.g. CI/CD artifacts, applications bundle with a container scheduler, etc.)

For existing applications, there is still much debate about the right way to on-board applications onto the PaaS or Container platform. Some platforms recommend that existing applications can be placed into containers (without any modification), and then run within the platform. Other platforms recommend that existing applications remain outside the platform and only be accessed through service brokers, edge gateways or API gateways.

	On-Boarding Existing Applications	On-Boarding New Applications
Docker Cloud / Datacenter	Docker Container	Docker Container
IBM Bluemix	CF Push, Docker Container	CF Push, Docker Container
Mesosphere DCOS	Container	Container
Pivotal Cloud Foundry	CF Push, Docker Container	CF Push, Docker Container
Red Hat OpenShift Container Platform	Source-2-Image, Docker Container	Source-2-Image, Docker Container

Figure 3: On-Boarding Existing and New Applications (Source: Wikibon, (c) 2016)

On the application side of the equation, developers would prefer to have as many options as possible to allow them to solve business challenges. But once applications, new or existing, get on-boarded into the platform, the operations teams are striving to create stability and consistency. The ability to create that stability and consistency will significantly reduce costs for all applications, which is why many companies are eager to on-board as much of their application portfolio as possible. With the increased efficiency that can be created through automated systems, Wikibon research shows that as much as $300B worth of IT operational costs will be removed from overall IT spending by 2026 (Source: Wikibon "True Private Cloud" research, Feb.2016).

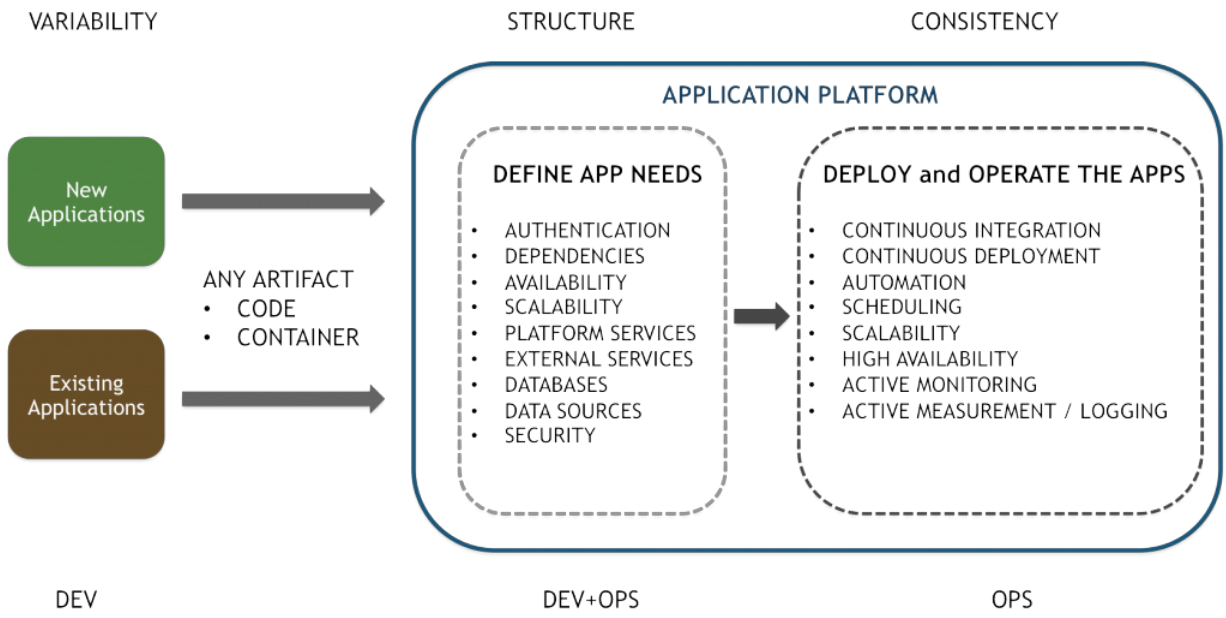

Figure 4: Understanding how New and Existing Applications interact with PaaS and Container Platforms (Source, Wikibon, (c) 2016)

There are several aspects to consider when on-boarding an existing application onto the PaaS or Container platform:

1. **Can the existing application be on-boarded without any modifications?** This is an approach that is possible for both Linux and Windows applications, when natively placed into Docker containers. The containers would then be run natively within the PaaS or Container platform, using the native scheduler within the platform. (e.g. Kubernetes or Swarm).
2. **Can the existing application be on-boarded with minor modifications?** This is an approach that is language or framework specific (e.g. Spring or Microprofile for Java, .NET Core or ASP.NET for .NET applications) and can leverage additional services within the framework to improve how the application runs within the platform.
3. **Can the existing application be accessed through a service-broker, edge gateway or API service?** Some existing applications, such as legacy databases or services running on a mainframe, should typically remain unchanged and only be accessed through a service-broker within the platform.

Criteria for Evaluating PaaS and Container Platforms

When evaluating any PaaS or Container platform, it is important to look at four key characteristics, which are focused on finding the proper balance between improving developer's ability to create and deliver software, and the operations teams ability to ensure that the applications run properly in expected and unexpected situations. These characteristics deliver a mix of Speed, Agility, Technical Flexibility and Business Flexibility.

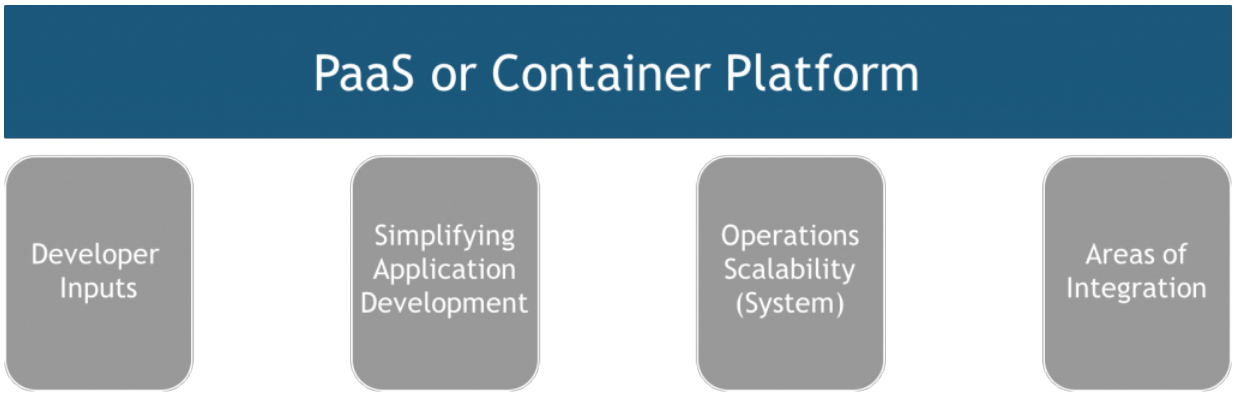

Figure 4: Core Elements of PaaS or Container Platforms (Source: Wikibon, (c) 2016)

1. **Developer Inputs** – How is an application on-boarded (and updated) into the platform?
2. **Simplifying Application Development** – What services within the platform make it easier for developers to build, scale and manage applications?
3. **Operations Scalability** – As more applications are added to the system, which elements of the platform will help it scale and be operationally efficient?
4. **Areas of Integration** – Platforms can not natively provide every possible service to an applications, so how can the platform integrate 3rd-party services to assist applications?

Inputs for Developers (Languages, Frameworks, Pipelines)

At the core of any PaaS or Container platform is the goal to provide application developers with functionality that will simplify their ability to build application faster and in ways that will scale as demand grows.

The platform should support a wide variety of development languages, as well as popular frameworks which enable additional services (databases, queuing, caching, notifications, proxies, etc.) to assist those languages. While languages such as Java and node.js and frameworks such as .NET are popular in many enterprises, PaaS and Container platforms should also provide support for emerging languages such as Ruby, Python, Go, Rust, Scala, etc.

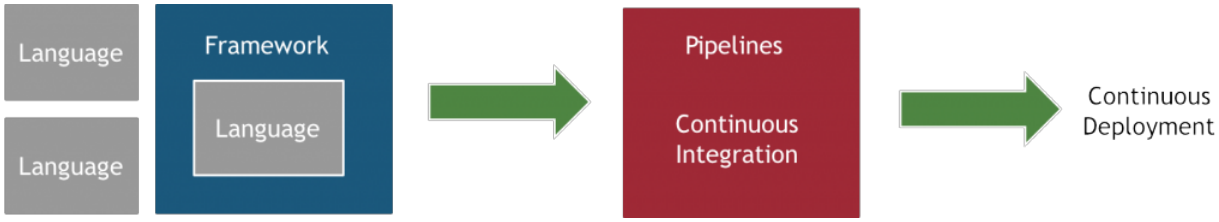

Figure 5: Languages, Frameworks and Pipelines (Source: Wikibon, (c) 2016)

While language support is critical to provide developers with flexibility in how they create application services, the business goal is not just to develop software but to deploy software in ways that will solve business challenges. This goal mandates that flexible workflows and build/test/deployment pipelines can be integrated into the platform. In the past, Continuous Integration and Continuous Deployment (CI/CD) systems were often external elements to PaaS and Container platforms. But as more companies evolve their internal organizational model to support DevOps culture and best-practices, the CI/CD frameworks are becoming more tightly integrated elements of the platforms. While some PaaS and Container platforms have an integrated CI/CD service, they should also be able to integrate with popular CI/CD services (e.g. Jenkins, CloudBees, CircleCI, Gitlab, TravisCI, XebiaLabs, Shippable, etc.). The Container or PaaS platform should not only be able to integrate with CI/CD pipelines, but popular tools and repositories for code inspection (e.g. Sonar) and code repositories (e.g. Artifactory, Git, etc.).

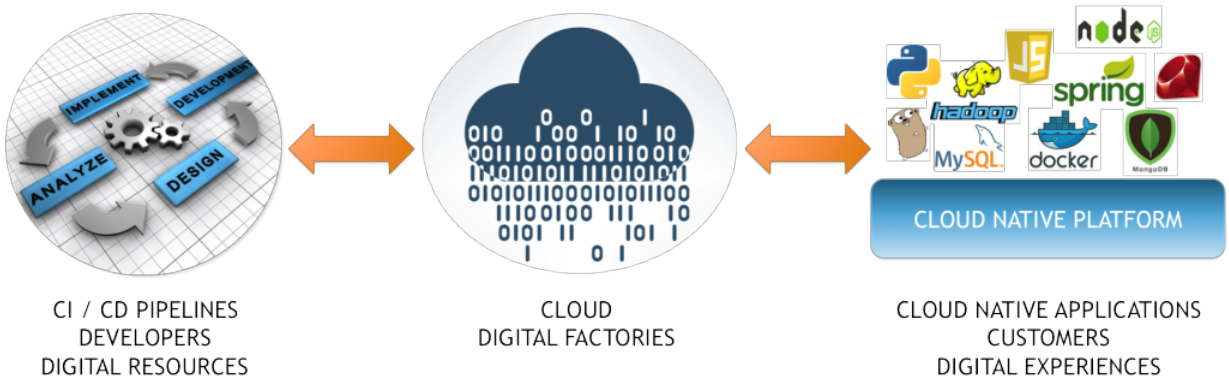

Figure 6: Digital Business Platforms – CI/CD, Cloud, and Cloud Native Platform (Source: (c) Wikibon, 2015)

Simplifying Application Development

As more companies move towards building cloud-native applications, or re-platforming existing applications, one of the critical areas of focus is how to simplify the application development process for developers. Container and PaaS platforms should deliver a set of "platform native" services and frameworks to help developers offload important capabilities from the application to the platform. Functionality such as Service Discovery, Messaging, Queueing, Routing and advanced middleware services should be embedded within the platform. These services not only allow the developer to increase their development velocity by reducing the need to build common functionality, but it also allows groups of developers to engage common application patterns.

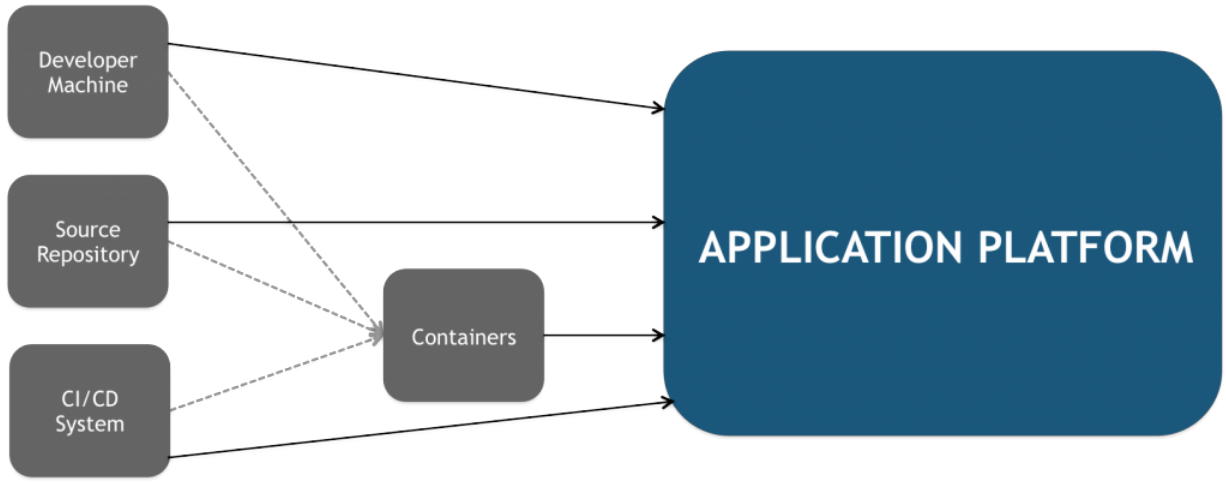

Figure 7: Application Inputs into the Platform (Source: Wikibon, (c) 2016)

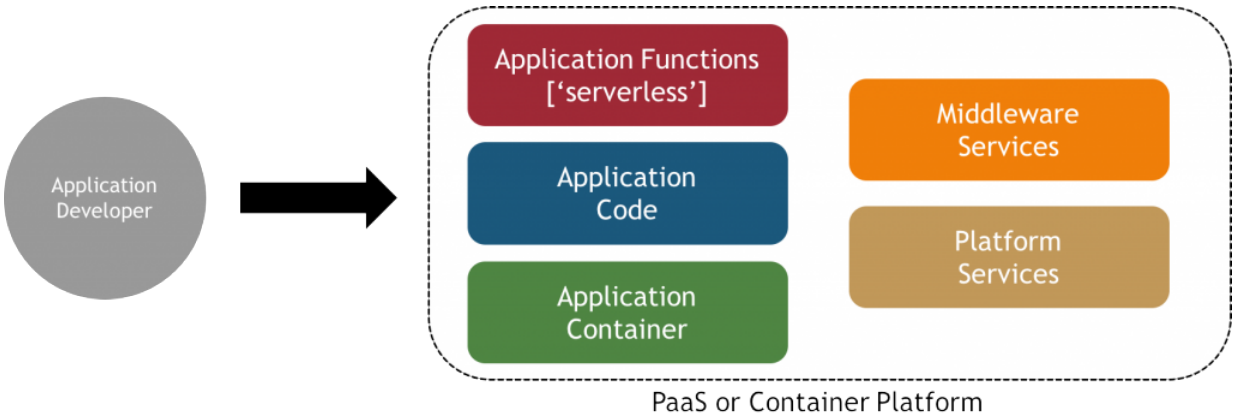

Figure 8: Application Functions, Code, Containers, and Services (Wikibon, (c) 2016)

While any framework could be run on a PaaS or Container platform as a local set of services within containers that are deployed by the development team, some services are natively enabled by the PaaS or Container platform. These services allow the developer to focus less on the operational aspects of the application and more on the patterns, user-context, user-experience, and data models.

	Platform - Application Frameworks
Docker Cloud / Datacenter	Any (in containers); None Native
IBM Bluemix	Any (in containers); Websphere
Mesosphere DCOS	Any (in containers); Marathon
Pivotal Cloud Foundry	Any (in containers); .NET, SpringBoot, SpringCloud
Red Hat Openshift Container Platform	Any (in containers), JBoss, .NET Core

Figure 9: Platform – Application Frameworks (Source: Wikibon, (c) 2016)

Operations and Scalability of the System (# of Containers, Networking, Staged Deployments, Load-Balancing, Multiple Clouds, Managed Versions)

While the development and deployment of applications is critical to any business, the ability for the PaaS or Container platform to also simplify operations is equally as important as factor to consider for any business. They must be able to scale to current and future workloads without having to make massive changes to the underlying architecture, and they should be able to run on top of a broad variety of cloud infrastructures (public and private).

One area of recent discussion in the PaaS and Container platform space is the underlying container schedulers. While schedulers are an infrastructure element of the platform, the scalability and ease of operations are critical factors to consider for customer's making architectural evaluations.

	Platform Container Scheduler
Docker Cloud / Datacenter	Swarm
IBM Bluemix	Diego (CF), Swarm (Docker)
Mesosphere DCOS	Mesos
Pivotal Cloud Foundry	Diego (CF Internal)
Red Hat OpenShift Container Platform	Kubernetes

Figure 10: Platform Container Schedulers (Source: Wikibon, (c) 2016)

Any interesting aspect of this discussion is around scalability and the rate of adoption of containers by customers. Below is some published scaling data by the various schedulers. **NOTE:** These numbers will change over time.

- Cloud Foundry Diego scaling – https://youtu.be/7APZD0me1nU?t=5m30s
- Docker Swarm scaling – https://blog.docker.com/2015/11/scale-testing-docker-swarm-30000-containers/
- Kubernetes scaling – http://blog.kubernetes.io/2016/07/kubernetes-updates-to-performance-and-scalability-in-1.3.html
- Mesosphere scaling – https://docs.mesosphere.com/1.7/usage/tutorials/autoscaling/requests-second/

What is interesting about the scaling numbers is to look at them in the context of early-adopters and data from live customers. This study from New Relic highlights an interesting trend about the rapid increase in container usage by customers as they become more comfortable with using containers for their applications. While these numbers are based on a small sample size, they do highlight the importance of looking at the scalability characteristics of the PaaS or Container platform, as the low-friction environments that containers create will lead to very high rates of adoption and usage by developers.

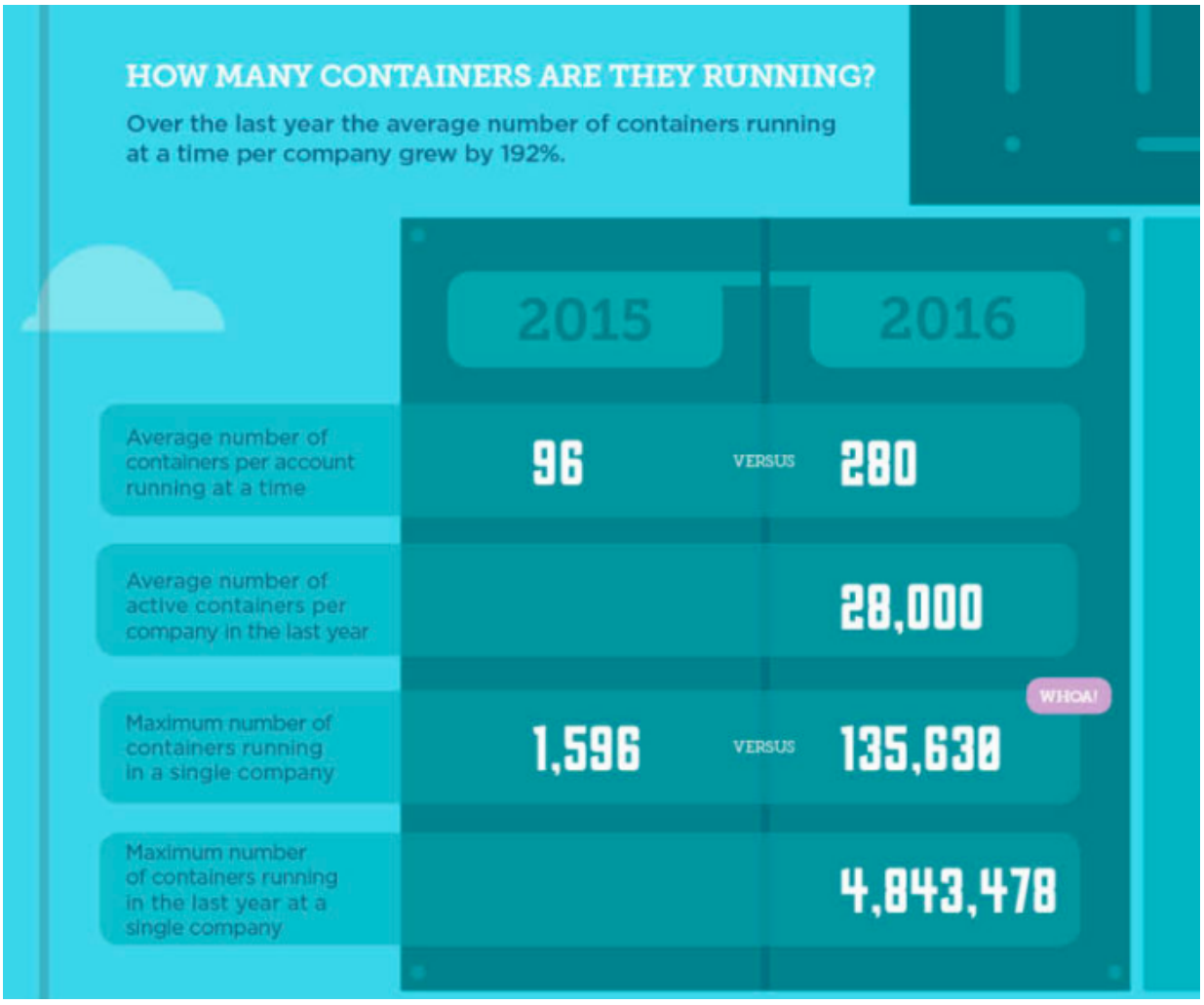

Figure 11: Container Usage by Customers (Source: New Relic "The Life and Times of a Docker Container [Infographic")

Another aspect that has evolved over the past 12-18 months is where various PaaS

and Container platforms can be deployed. Not only are the platforms now supporting both private and public cloud consumption models, but their communities have created installers to allow them to run in any cloud environment. In some cases these installers are available in the public cloud marketplaces, and in other cases the platforms are run as SaaS services that can be directly consumed by businesses.

	Private Cloud Infrastructure	Public Cloud Infrastructure
Docker Cloud / Datacenter	Linux, Windows	Any (Linux), Docker for AWS, Docker for Azure
IBM Bluemix	IBM Local (OpenStack)	IBM Bluemix
Mesosphere DCOS	Linux	Any (Linux), AWS, Azure, Azure Container Service
Pivotal Cloud Foundry	VMware, OpenStack	Any (Linux) Pivotal Web Services, AWS, Azure, GCP, OpenStack
Red Hat OpenShift Container Platform	OpenStack	Any (Linux), OpenShift Online, AWS, Azure, GCP, OpenStack

Figure 12: Cloud Infrastructure support for Platforms (Source: Wikibon, (c) 2016)

Areas of Integration (for Applications, Data Services, 3rd-party Services, etc.)

While PaaS and Container platforms are capable of running both new and existing applications, many of those applications will need to be augmented with additional services in order to meet critical business needs. These augmenting services could be native to the PaaS or Container platform, or accessible via APIs, API Gateways, Routing services or Service Brokers.

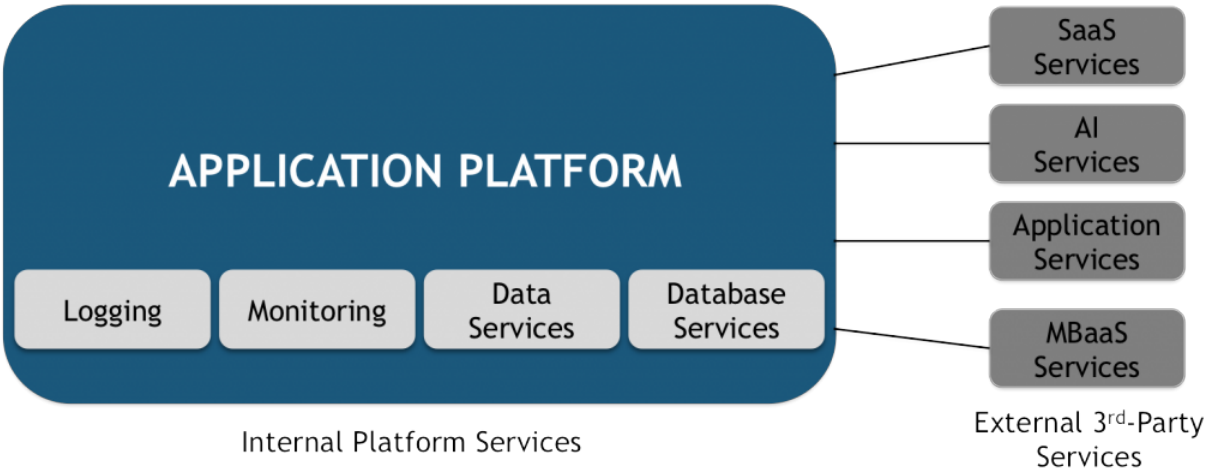

Figure 13: Internal and External Application Services (Source: Wikibon, (c) 2016)

Architects, Developers and Operations teams should consider the flexibility of PaaS and Container platforms to natively deliver those services today, as well as add external services in the future. As new usage models are being driven by IoT, Machine Learning and Serverless applications, the need for platform flexibility will be critical

over the next 3-5 years.

Conclusion

Over the past 2-3 years, there has been an enormous level of maturity delivered by the various PaaS and Container platforms in the market. While the underlying technology will still force architects, developers and operators to make choices which align to their business requirements, the breadth of choices for technology and deployment has rapidly accelerated. IT organizations and digital product groups should be considering how new and existing applications can be on-boarded into these cloud-native platforms, as they will assist them in not only delivering applications faster, but also reduce their overall IT costs by automating many areas of IT operations.