The Enterprise Cloud Native Journey

By Keith Townsend

Executive Summary

There is a great deal of excitement around Kubernetes and PaaS. There is also some overlap. IT organizations have competing priorities. On one end Kubernetes offers the ultimate in developer flexibility. While the flexibility proves a powerful business capability, organizations find themselves frozen with the number of options to spin up a Kubernetes powered development environment.

Platform as a Service (PaaS) offers a rapid on-ramp to modern application design while providing the guardrails required to support modern applications in the enterprise. However, these guardrails may also prevent bringing legacy applications along. This eBook sheds light on when to select a PaaS vs. Kubernetes or something in-between.

This eBook focuses on the business and technical challenges of deploying, adopting and managing a cloud-native infrastructure that delivers on the promise of business agility. We set out to shed light on the following topics.

- The relationship between Platform as a Service (PaaS) and containers
- When to adopt pre-packaged PaaS solutions such as Cloud Foundry
- Deploying traditional applications in a cloud-native platform
- Approaches to multi-cloud
- Leveraging cloud-native infrastructure for serverless or Functions as a Service (FaaS)

About the Author

Keith Townsend

Keith Townsend, Founder of the CTO Advisor, is a management consultant and analyst with over 20-years of experience.

He has worked across a number of industries including but not limited to Finance, Pharmaceuticals, Manufacturing and the U.S. Federal Government.

As a consultant Keith has helped Fortune 500 companies redefine their global data center strategies and IT processes.

Keith is a sought-after speaker and industry influencer. He has written for
publications including ZDNet, TechRepublic, and TechTarget.

He is a frequent co-host of the analyst web show theCube, recorded live during many major industry conferences.

Introduction

I spoke with an architect working for a global professional services firm that was fresh off the heels of a private cloud deployment. The architect was lamenting how they'd invested so many resources in deploying a Windows Azure pack-based cloud that developers weren't consuming. Enterprise IT leaders solely want to provide cloud-native infrastructures that enable agility while avoiding an explosion in management overhead.

From early cloud management platforms (CMP) to OpenStack to now Kubernetes, the enterprise has been in pursuit of the perfect balance of form vs. function.

Much of the attention has moved away from CMP and all in one private cloud platforms such as OpenStack. Containers, orchestrated by Kubernetes, has become the latest trend in cloud-native infrastructure. However, Kubernetes hasn't solved the core business and technology challenges associated with adopting cloud-native architectures in brownfield environments.

Organizations can look to hyperscale companies such as Netflix for examples of how to deploy containers in brownfield environments that are adopted by developers. Looking at the case of Netflix, IT leaders will better understand the relationship of infrastructure to the development process. Also, these leaders gain a better comprehension of the value containers bring to monolithic applications and microservices-based applications.

In this eBook, we'll discuss the business and technical challenges of deploying, adopting and managing a cloud-native infrastructure that delivers on the promise of business agility. Containers prove an excellent abstraction for cloud-native development. Hopefully, this eBook proves a reference for creating container strategy within your organization.

Defining Cloud Native

There's the need to level set on definitions. What exactly is "cloud-native?" It's a term that the largest to smallest enterprise technology firms have adopted. Is an application deployed using native AWS products such as EC2, Elastic Load Balancers, and Auto Scaling cloud-native?

The term has married the concept of containers. Here's the Cloud Native Computing Foundation's (CNCF) definition for cloud-native.

Cloud native systems have the following properties:

- **Containerized**. Each part (applications, processes, etc.) is packaged in its container. The packaging facilitates reproducibility, transparency, and resource isolation.
- **Dynamically orchestrated**. Containers are actively scheduled and managed to optimize resource utilization.
- **Microservices oriented**. Applications subdivide into microservices. The architecture significantly increases the overall agility and maintainability of applications.

With the CNCF definition in mind, this eBook focuses on containers, container orchestration, and microservices in brownfield environments. As with any great technology discussion, we'll start this discussion with people.

Who does cloud-native impact?

There are several lenses from which to look at cloud-native. Most conversations begin with developers and speak to operators. However, these are two broad groups that include several sub-categories. Not every developer has the same needs or responsibilities. The same applies to operators. Cloud-native is yet another application architecture within an enterprise with several application architectures. Let's break down the personas in each group.

The majority of cloud-native documentation commonly refers to developers and operators when discussing the roles in application management. Developers commonly create and package applications for deployment within a cloud-native infrastructure. Operators deploy and manage these applications in production. Organizations such as Netflix combine the roles using DevOps as a management concept.

However, most enterprise IT shops organizational roles don't map cleanly to the two-well-defined cloud-native roles. The term developer isn't very straightforward. When referring to developers in typical enterprise IT, the term applies to three different functions within IT.

Developers is a broad term in traditional enterprise IT. When looking at it through the lens of an infrastructure IT team, the term developer maps to any team that's an internal customer. That group includes packaged application support, internal developers, and third-party developers.

Packaged Application Support – Commonly referred to as the "App Team," application support typically customizes and deploys packaged software such as SAP or EMR. The application support team works with the packaged software provider to determine the eligibility of a software product's deployment in cloud-native infrastructure.

Internal Developer – The internal developer writes business line software. Similar to Netflix' journey to containers, much of the existing software is monolithic in design. These developers require both incentive to re-platform their applications for cloud-native infrastructure or guarantees that existing software runs reliably in cloud-native infrastructure without much modification.

External Developer – The external developer is an external contractor or 3rd party application developer. While organizations realize the advantages of external help, organizations also realize the challenges of changing culture outside of their direct control.

Again, in the idealistic cloud-native deployment, developers hand off container packaged applications to operators, or operations systems, to deploy. Operators determine the attributes of the infrastructure needed to scale the packaged applications in production. Changes in the application update via new container images that are rolled into production by the operations team.

The cloud-native operator loosely translates to the IT infrastructure team. In most organization, IT infrastructure includes the following groups. I'll attempt to encapsulate these multiple groups into two major categories.

Datacenter Infrastructure – Datacenter infrastructure includes all of the physical and logical components needed to provide basic compute capability. Groups within this category include Network, virtualization, cloud infrastructure, storage, service management and security.

Platform Infrastructure – Platform infrastructure includes the layer above the operating system commonly referred to as middleware. These are services that developers can leverage to create applications. Database Administration (DBA), message bus services, and web services commonly full under this group's purview.

Application Deployment Workflow

One of the appeals of cloud-native is the ability for developers to create container images needed to deploy an application. The ability to package the application enables operations teams to manage a wide variety of applications on common cloud-native infrastructure. In practice, brown-field application deployment isn't as clean.

Due to the lack of a prescriptive method of packaging applications for deployment, many teams have adopted an ad-hoc process for application deployment and updates. The following workflow normally represents the process.

Developers receive an operating system instance from the operations team. Developers install and test their code on the pre-production system. After a series of check and integration testing, the application goes live. Developers handoff runbooks and support information to the operations teams. In case the application requires updates of break-fix, a similar process is used to deploy updates.

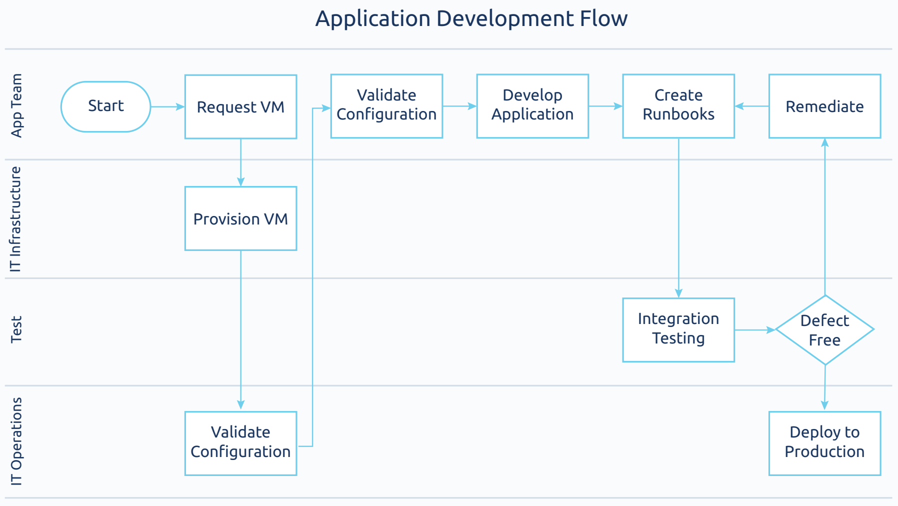
Not only is this a challenge for existing monolithic applications, but this workflow also makes adopting containers difficult. It requires organizations to document, train, and implement a new workflow for packaging, to deploy, and supporting cloud-native applications.

Kubernetes Early Adopters

As in interesting aside, lines of business (LoB) were early adopters of cloud-native and Kubernetes. The trend I've noticed is similar to the trend of cloud adoption.

While LoB may initially adopt Kubernetes, similar to the early trend in public cloud, the service becomes a centralized platform.

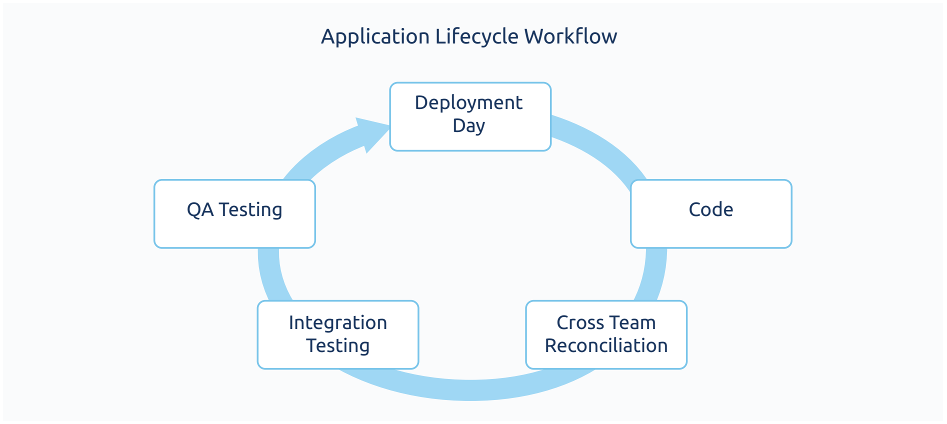
Greater interest from internal IP develops as the demand crosses multiple LoB and central services.



Process Before Technology

Give careful thought to how cloud-native fits into application development and management. In most cases, enterprise organizations find they must implement a continuous integration/continuous delivery (CI/CD) process before adopting cloud-native infrastructure.

The CTO Advisor recently engaged a customer that experienced the pains of implementing cloud-native technologies. The customer had an existing monolithic application with a distributed development team. The application deployment life cycle is very familiar to most organizations.



The infrastructure team implemented virtualized environments for each development team. Each team could now develop application features independent of a central system. The organization called the CTO Advisor because the development process slowed, and reduced application reliability.

It only took one phone call to determine the problem. After years of a refined systems development life cycle (SDLC), the organization introduced chaos by providing tooling without a new process to leverage the tooling.

With only a single day to implement code, each team had to coordinate features and integration testing. While it created a bureaucratic logjam, the existing process ensured proper integration testing. In the previous development environment, a single day a month was targeted for application updates. An undocumented natural workflow evolved that include robust integration testing,

By distributing code development without an updated integration testing process, the team introduced quality control issues into the development lifecycle. Cloud-native tools proved the wrong tool for the existing process.

If you haven't designed a CI/CD process rollout alongside your cloud-native implementation, stop and pull together your stakeholders. You don't have to have a CI/CD process with automated testing and controls, but you must ensure cloud-native infrastructure doesn't break your existing workflow. Taking the current process that typically exists between developers and operations as-is will ensure a bumpy transition to a cloud-native implementation.

NETFLIX

A Lesson from Netflix

Netflix is bottom-up organization. The Titus container team understood that it couldn't merely command each application group to adopt containers. The project team had to ensure that the platform adapted to existing application architectures.

It was critical that Titus supported existing APIs and development processes. The need to customize the container orchestration to Netflix' existing operations played a role in the decision to build Titus vs. adopt an existing open source project like Kubernetes.

Most organizations don't have the luxury or in-house talent to customize Kubernetes to existing processes. It's critical to identify critical points in your existing application support model that doesn't comply with you selected cloud-native tooling.

The What of Cloud-Native

A common question - what types of applications deploy in Cloud-Native infrastructure. It is a common myth that Cloud-Native infrastructure only supports greenfield deployment. While greenfield deployments of microservices based applications garner much of the headlines, there's a strong argument for deploying existing monolithic applications in container infrastructure.

Simply put, enterprise IT has little capacity for net new operations. As an enterprise architect for a Fortune 500, I've had to turn away projects that offered significant business value. The application architecture presented too much of an operational snowflake. In the end, application support costs more than the business opportunities provided by the application. The operations team needed to hire a dedicated team to support the single application. Spinning up a dedicated operations stuff or procedure for snowflakes proves a model that doesn't scale.

One could make the argument that it was an opportunity to develop a team where a cloud-native application support team gets a start. Without a pipeline of applications and more importantly a legitimate architectural framework for future microservices based applications, effort and resources go to waste in one-off implementations.

It's with these challenges organizations look to leverage existing support and technology infrastructure to support containers. It starts at the physical infrastructure. I spoke with the general manager of a leading virtualization software company that shared many of his clients want to leverage their existing virtualization platform for containers. While there are obvious financial incentives to keep existing infrastructure, the primary driver was the ability to support existing applications in a containerized model.

Netflix discovered that containers were the ideal platform to more efficiently utilize public cloud-based VM infrastructure. AWS EC2 instances are relatively rigid compared to private infrastructure. One of the advantages private infrastructure is the ability to over provision infrastructure. Take a VMware vSphere environment as an example. In VMware vSphere, the cost to assign 32GB of RAM and four vCPUs to a VM that only requires 8GB of RAM is relatively low.

While there is a utilization efficiency hit, organizations don't realize the financial impact of the inefficiency. Four virtual hosts have a fixed depreciation cost whether the host runs 32 VMs or 128 VMs. Public cloud infrastructure is priced based on the resources provisioned. If an organization provisions 32GB of resources, the public cloud bill with reflecting 32GB of resources. Containers allow a higher level of abstraction than the virtual machine.

In Netflix' use case, the company realized cost savings by placing legacy applications in containers and then deploying the containers into EC2 instances. Higher resource efficiency is only one example of the advantages of containerization of legacyapplications.

Other improvements include:

- Reduced downtime from consistent application deployment enabled by packaging
- Application portability across private and public cloud
- Stepping stone to application modernization

During DockerCon EU 2017, much of the discussion centered on the modernization of monolithic applications. Organizations such as GE shared their experiences modernizing applications via containerization. In the case of GE, the company looked to containerize stateless application. The company then used the experience to understand the management gap between virtual machines and containers.

Where is Cloud-Native Deployed

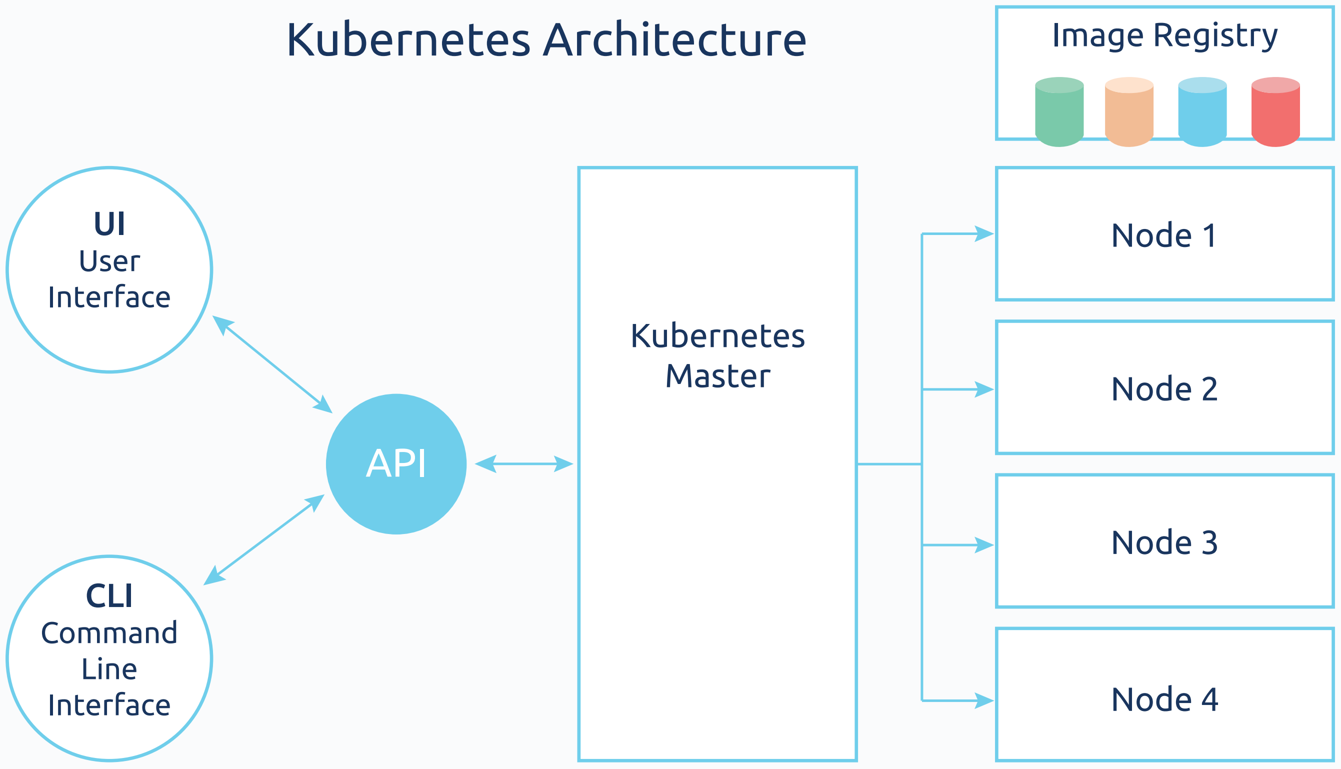
Where might cloud-native applications appear? The beauty of cloud-native infrastructure remains the abstraction. In the case of Kubernetes, a cloud-native infrastructure is a service contract between the operator and the consumer (i.e., the developer).

If a consumer requests a container, the cloud-native infrastructure provides a consistent experience regardless of the underlying infrastructure.

Since cloud-native infrastructure presents an abstraction, the underlying infrastructure, the options for infrastructure range from public cloud to private infrastructure. The point of integration is the cloud-native platform.

Network, storage, and compute drivers are all implemented at the platform level. The application has no insight into the underlay other than what's exposed by the platform.

Kubernetes Architecture



The relationship of containers and PaaS

Separating the concept of PaaS and Containers is difficult. Today, it seems to be a no-brainer to include containers as part of a PaaS deployment. Many container deployments resemble PaaS. It’s important to level-set and provide a reminder of the purpose of each solution.

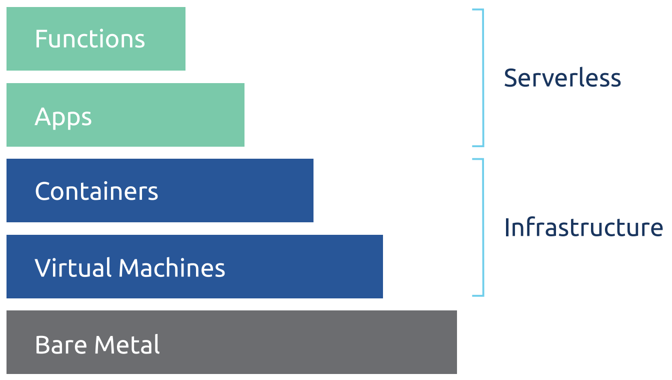
PaaS

Most PaaS solutions use some form of containers internally. Docker, then dotCloud, liberated the container from the platform.

PaaS as a development platform predates the container movement as a development platform. The core value of a PaaS is the ability to disaggregate the code from the infrastructure. Application developers shouldn’t concern themselves with the concept of a virtual machine, IP address subnetting and LUN storage provisioning. These concepts are required knowledge of the operations teams. However, application developers should focus on the business challenge, the data, and code needed to solve the problem.

An example of a massively successful PaaS is Salesforce.com. Salesforce hides the complexity of database administration, server provisioning, Operation System (OS) versions from the developer. The developer codes to the Salesforce PaaS and lets the Salesforce operations teams worry about the details of the infrastructure.

Bringing it closer to on-premises operations, solutions such as Cloud Foundry enable similar capability for a broader range of applications. Cloud Foundry is what’s called an opinionated platform. More on opinionated platforms later. The critical thing to note is that adopting Cloud Foundry allows infrastructure teams to provide all of the required components of a PaaS without a tremendous amount of engineering. For those familiar with hyperconverged infrastructure, it’s a similar concept but for software environments.



Some components of a PaaS include the following.

- Platform API
- Platform control plane
- Load balancing
- IaaS interface/plugins
- Logging
- Message Bus
- Authentication
- Access control
- Service catalog/discovery
- Container Service

Many of these components are consumer facing. Developers must understand how to consume the services comprising the PaaS. Infrastructure and development teams must come together and understand the purpose and therefore design of the PaaS infrastructure.

Containers

A container is a packaging of application components including operating system dependencies and application binaries in a single file. Containers are incredibly portable packages. Before the broad adoption of containers, an application team may package an application using .tar files or executable binaries.

When writing to a PaaS, a .tar file is sufficient. The code within the .tar has all of the metadata needed to run the application. However, today, most enterprises don't deploy to a PaaS. Operations teams must receive a set of dependencies from the developer such a specific version of MySQL or other required runtime.

Within the container image are the application and all of the operating system dependencies. As long as the target infrastructure runs the container runtime, let's say, Docker, the application will run without modification. A developer can compile an application on their laptop and have a reasonable assurance the operations team can deploy it in production without modification to the infrastructure.

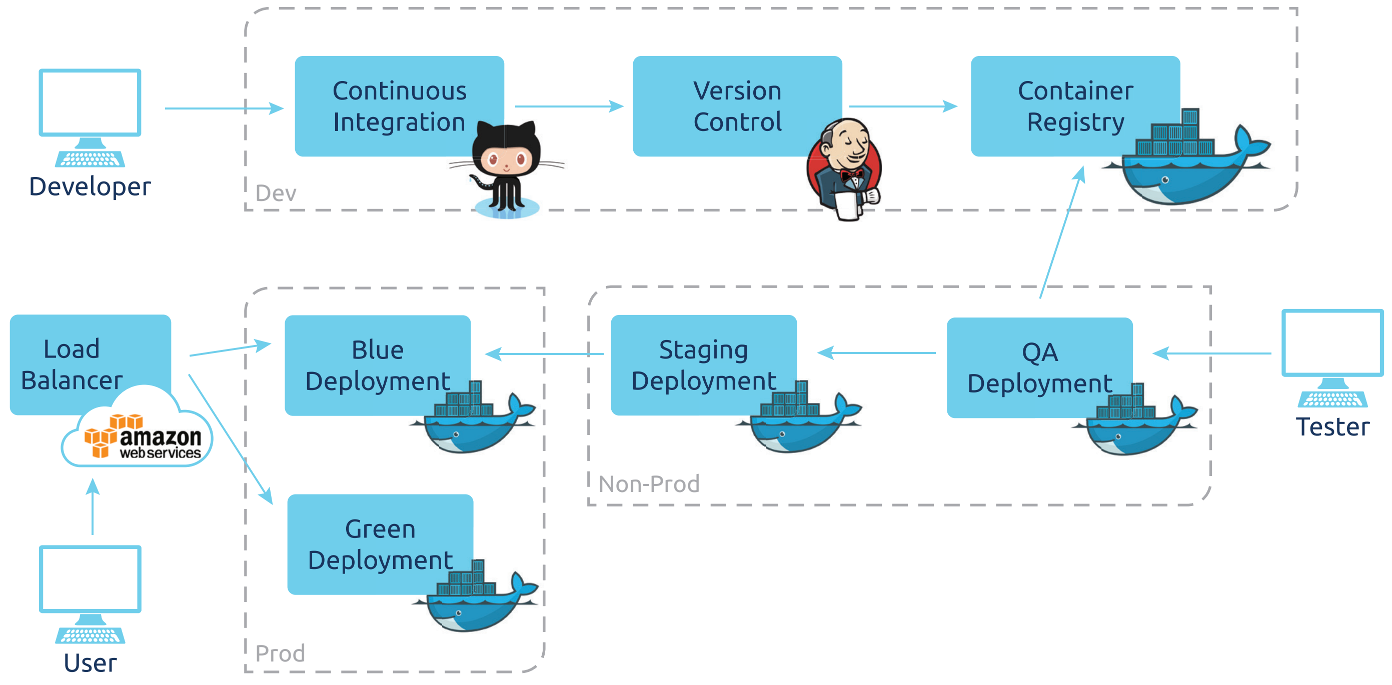
Packaging is an extremely important advantage of the container driven cloud-native infrastructure. Operators can best choose where to run a given workload. If a microservice such as global image encoding service requires massive scale, then the operations team may choose to run the service across several regional cloud providers.

Containers offer a better packaging when developing code to run on commodity infrastructure. However, containers alone don't offer a PaaS-like experience. Going back to the Netflix example, Netflix took traditional JAVA applications and converted the applications to containers.

By placing the Java binaries and dependencies in containers, Netflix was able to leverage the infrastructure better. During the publication of this eBook, Netflix announced the open source of their Titus platform. Some interesting stats include launching a half-million containers a day and 200K clusters a day. A varying degree of application types exists across the Netflix container ecosystem.

The team went out of the way to ensure developers had access to all of the tools and interfaces in the non-containerized environment. The development workflow remained uninterrupted. Developers didn't need to adopt new constructs of an opinionated PaaS for the organization to receive the advantages of cloud-native infrastructure.

As an infrastructure building block, organizations can build PaaS solutions using containers to run the underlying components. Container-driven solutions are sometimes referred to build your own PaaS.



Kubernetes

There are some natural questions for raw consumption of containers. One such question is management. It's easy to see an environment where containers become unwieldy and impossible to manage. How does an organization practically manage thousands of containers spread across the world?

Netflix obviously had this challenge in their container journey. The organization chooses to build a container orchestration platform on top of the powerful data center orchestration platform Mesosphere. The team considered leveraging the early work of the Kubernetes project but ultimately decided to build a custom orchestration platform in the form of Titus best fit the needs of the organization.

Some of the needs of Netflix mimic the needs of the enterprise. One of the largest challenges was the ability to support existing applications. Netflix is a bottom to up engineering culture. This means they let the needs of the application engineering team drive design decisions.

The rest of the industry settled on Kubernetes as the de-facto platform for container orchestration. Kubernetes originated out of Google. Within Google, the platform was responsible for global container orchestration. Google open sourced the basis of Kubernetes and donated the code to the Cloud Native Compute Foundation (CNCF). As of this writing, the CNCF recently promoted the project to production at version 1.1.

The appeal of Kubernetes is the vendor-neutral approach to governance and project direction. Members of the CNCF reads like the who's who of enterprise IT. All three major cloud providers are members of the CNCF with announced services provide container orchestration using Kubernetes.

Outside of the major cloud providers, Platinum members include Intel, Cisco, Dell Technologies, Docker, Alibaba Cloud, IBM Cloud, Mesosphere, Oracle, Pivotal, Red Hat, and VMware. The sponsor of this eBook Nirmata is a Silver member.

There is extensive and deep support for Kubernetes. So much so that the major competitive container orchestration solutions from Docker and Mesosphere now directly support Kubernetes.

Containers vs. PaaS

There's a distinct theme in the container vs. PaaS discussion. The conversation isn't an either/or option but both. Even projects such as Cloud Foundry recognize the value of containers. The question for the enterprise – Should you build your own PaaS using containers or deploy containers in an existing PaaS platform.

Simple question with a surprisingly easy answer. I'm a management consultant by trade so, I'll resist the urge to give a non-answer. Plainly put, it depends on your desired operational objectives and timeline. I've established that the goal is to build an infrastructure that application teams will consume in pursuit of building applications.

Adopting a PaaS platform such as Cloud Foundry (CF) means fast time to value. In CF, every significant part of the PaaS is predefined. CF has a project for each of the primary services required to deliver an application development environment. There is minimal engineering needed for the organization.

If you want a jump start on not only cloud-native infrastructure but cloud-native applications, I'd argue there are few better options than CF. On the flipside of the equation, you are married to CF as architecture and perspective. You will look at containers from the perspective of a PaaS. Think through what it means to redeploy all of your traditional applications in CF. For most applications, redeploying in CF requires re-writing the application.

A 12-factor application design remains the standard for re-writing applications for a cloud-native environment. Many executives find that existing applications don't lend themselves to a 12-factor design. Or executives find very little value in taking on the risk and cost of refactoring an application for 12-factor design.

Replatforming is something, like Netflix, most teams wished to avoid. They wanted a platform that would support both microservices applications and monolithic applications. Adopting CF brings with it speed but significant inflexibility while adoption of containers enables broader adoption of cloud-native best practices.

If the primary driver is to provide a cloud-native infrastructure that supports containerized monolithic applications a Kubernetes-first infrastructure proves a better solution. Similar to Netflix, organizations may first focus on bringing their monolithic applications to Kubernetes. After recognizing some of the benefits of containers for monolithic applications, organizations may choose to build microservices based applications on the same underlying infrastructure.

Kubernetes management doesn't change from running monolithic applications vs. microservices. The challenge is missing that opinionated PaaS layer. Some thought has to go into what components of the PaaS must Kubernetes replicate and what solution will provide these PaaS-like capability.

Kubernetes isn't without any consistent development environment options. There is a reason for a debate between using Kubernetes as a PaaS vs. Cloud Foundry. First, the appeal of Kubernetes is the ability to run workloads across environments. A container can run on a container host in Amazon or a container host on VM's in VMware vSphere.

Likewise, a Kubernetes Deployment object allows for portability of more complex application groupings. By using out of the box Kubernetes features, development and operation teams have a wide range of options for building portable applications.

When using a Kubernetes Deployment object to build an application, developers create a portable YAML-based description of the application. The contents of the YAML file will have details such as which container images to deploy, the number of pods required and the required network ports. Additional out of the box features include autoscaling pods. The project has published examples for deploying stateful and stateless applications of variable size.

Projects are looking to extend the basic out of the box PaaS capability of Kubernetes. Helm is an example of a project that provides management of applications built on Kubernetes. Helm may allow for better portability of Kubernetes-based applications across clusters and providers.

Taking the concept, a step further, Kubernetes Deployment objects enable more complex use cases. One such use case is to deploy a container-based PaaS. As highlighted in an earlier chapter, most PaaS platforms are deployed using containers. There's work to be performed in both Kubernetes and individual PaaS solutions to enable native Kubernetes as a platform for the underlying infrastructure.

The whole market is still developing. Kubernetes has the potential to be the foundation of future PaaS projects. Redhat OpenShift is an example of a platform deployed using Kubernetes.

CaaS vs. PaaS Cheat Sheet

Use Case	CaaS	PaaS
Rapid deployment of end-to-end environment		X
Multi-cloud	X	X
Granular service selection	X	
Containerizing legacy apps	X	
Microservice architecture adoption	X	X
Optimize VM/Cloud Instance utilization	X	

Serverless & Functions as a Service

An example of the potential of Kubernetes as a PaaS is the work being in-progress in the Serverless computing world. Within the CNCF rages a debate defining Serverless. The most famous example of a Serverless platform remains the AWS' Lambda Functions as a Service (FaaS).

Lambda eliminates the concept of an operating system in the service. The code resides within an S3 object. The code is executed based on an event trigger from an AWS event such as a file written to S3. Developers have no sense of the amount of CPU, network or storage required to execute the code. The only lever available to Lambda consumers is RAM.

The infrastructure operations exist outside of view of the service consumer. If something in the infrastructure were to break such as an OS kernel panic, there's no indication to the application. The function would just run on a different container.

The last point of the previous paragraph is critical. These Serverless functions run inside of containers. Containers offer the instant run capability required for functions at scale. There are a few open source projects that look to offer Serverless constructs in customer side infrastructure.

The debate within the CNCF revolves around the operations model. If a customer has to concern themselves with the container infrastructure running the FaaS platform, is it serverless?

Regardless of your side of the argument, the important concept to walk away with is that FaaS and Serverless run within the containerized infrastructure.

Here are a few open source projects that deploy on a Kubernetes infrastructure.

- OpenWhisk
- Open FaaS
- Fission

The progress of FaaS on containers indicates the direction PaaS points. It's a reasonable assumption that the opinionated PaaS options for Kubernetes will quickly grow.

Kubernetes Solution Checklist

You've gotten to the point where you wish to deploy a Kubernetes solution? Looking at the landscape, there is no shortage of Kubernetes solutions.

Here's a short list of Kubernetes distributions and providers.

- Nirmata (Sponsor of this eBook)
- RedHat OpenShift
- VMware PKS
- Docker Enterprise Edition
- Cisco
- Cloud Foundry
- CoreOS
- Google Cloud
- Microsoft Azure
- AWS (in-preview as of this writing)

For a full list, visit the Kubernetes project page.

With so many distributions where to start in selecting a solution. On the next page is a checklist for some criteria in selecting a Kubernetes solution.

Container as a Service vs Platform as a Service

Operations	
Feature	**Description**
Multi-cloud support	Ability to deploy and manage applications across multiple clouds including private and the major public cloud providers Consider if support is for individual VM (pods and control plan) or integration with cloud providers Kubernetes implementation
Legacy application support	Features and tools to manage stateful and stateless monolithic applications Tooling for containerization of existing applications
ITSM integration	Existing ITSM integration points including configuration management, change management, and out of the box integration with leading ITSM platforms such as Service Now and CA
Policy-based workload management	The ability for abstract the application from the underlying infrastructure. Based on policy, operations team's ability to deterministically place workloads on the underlay that best meets application requirements
Policy-based cluster management	Ability to manage Kubernetes cluster based on centralized policy such as autoscaling of Kubernetes and deployment of new pods based on deployment
Upgrade	The effort required to upgrade from one version of Kubernetes to the latest version
Delivery model	Solution delivered as SaaS, managed, or packaged product

Development	
Feature	**Description**
Deployment Object	Simplification of packaging Kubernetes Deployment objects to create PaaS-like capability and application portability across clusters
CI/CD Integration	Integration with testing and development tools such as Jenkins and Chef
Code management	Integration with code repositories and tools such as defect tracking
Image repository	Catalog and organization of services and microservices container images

Security	
Security console	Centralized security console enabling centralized policy management
Image scanning	Ensuring images in the repository meet organizational security standards
Enterprise integration	Integration with existing enterprise security products and features
Secrets management	Standalone or integrated solution for management of credentials and keys

Conclusion

The world of PaaS and CaaS are slowly evolving to merge. Organizations such as Netflix have shown how to make the change from monolithic applications to cloud-native architectures gradually. The immediate benefit includes better utilization of public cloud resources and the ability to deploy and manage cloud-native and monolithic applications using a single operations model.

If organizational priorities focus on application development and require a fast on-ramp to modern application architecture, a prebaked PaaS may prove more valuable.

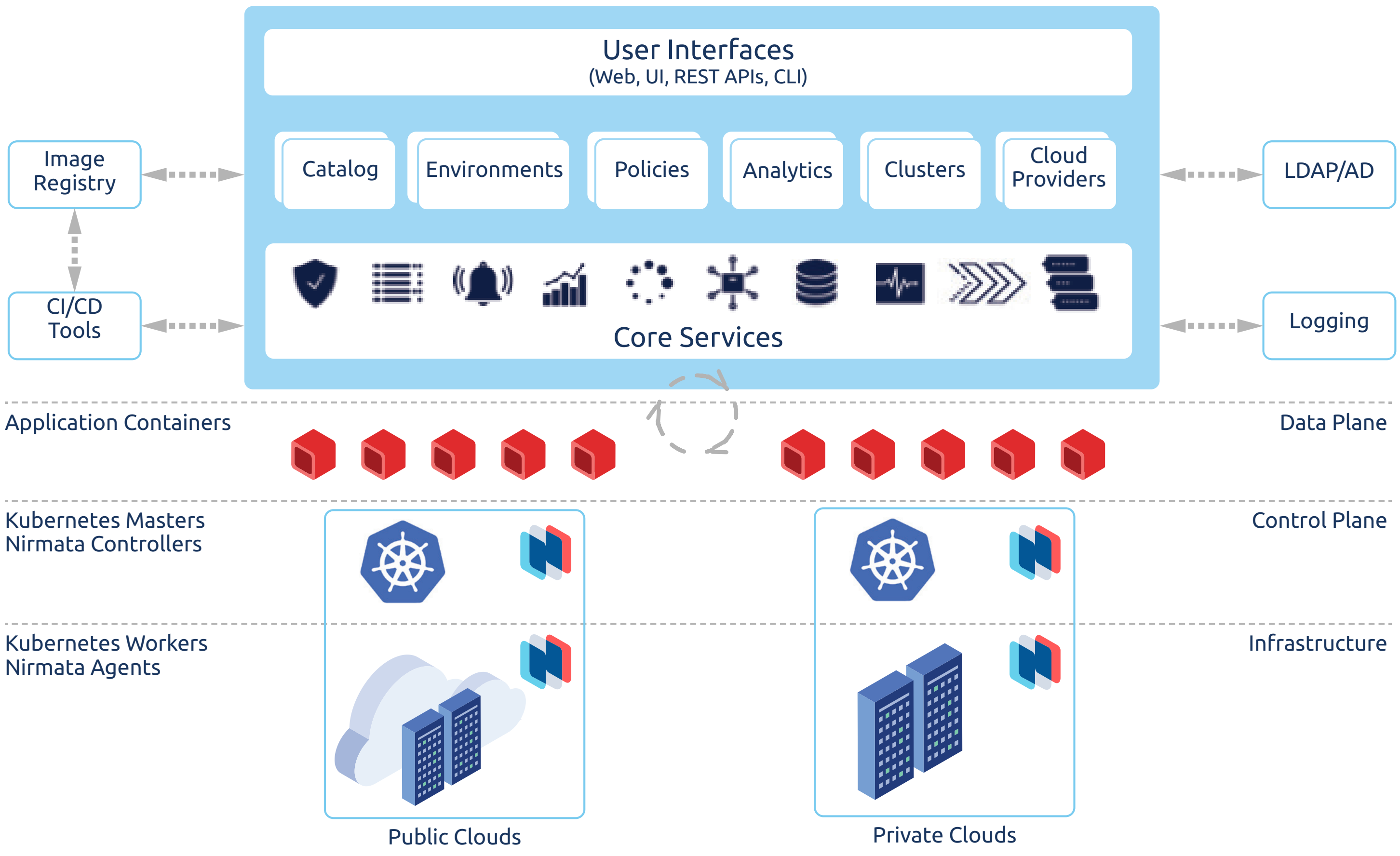
Organization needing both PaaS for application development and containers for application modernization may want to look at an hybrid solution. A hybrid approach involves adding PaaS-like capability to a container platform. The hybrid approach provides just enough PaaS capability for cloud-native application development while taking advantage of containers.

About Nirmata

Nirmata is a complete Kubernetes-based enterprise container platform that natively supports both traditional and microservices-style applications. With Nirmata your teams can easily manage multiple clusters and your entire application portfolio across all your environments from a single, intuitive interface.

Key features

- Simplified Kubernetes cluster deployments via self-service interfaces
- Automated Cluster Management, including upgrades and elastic sizing
- Governance of clusters and workloads
- Cross-cloud application portability
- Kubernetes application modeling, visibility, and management
- Continuous delivery with integrations to existing build tools
- On-demand application environments
- Fast troubleshooting, proactive assessment



Sign up for a free trial and
check out our blogs and demo videos

Learn More

info@nirmata.com
www.nirmata.com