Streamlining the Cloud Native Developer Experience

An Intellyx Analyst Guide for Cloud Foundry Foundation

By Jason Bloomberg, Jason English and Eric Newcomer, Intellyx

Table of Contents




Introduction

If you were building applications for the cloud before Kubernetes came onto the scene, you already knew **Cloud Foundry** as 'the easiest way to do cloud.'

Now, with Platform Engineering practices making a strong comeback, the Cloud Foundry Foundation community is contributing a smoother development provisioning and deployment experience for cloud native developers.

This 4-part analyst guide from Intellyx will help developers and technical leaders understand how the open-source capabilities of Cloud Foundry offerings such as Korifi and Paketo can provide a stable, well-curated application delivery experience.

By Jason Bloomberg

Managing Director & Analyst
Intellyx

Cloud Foundry: Rethinking PaaS for the Cloud Native Era

Cloud Foundry has a colorful history that parallels the growth of the cloud. The organization pioneered the notion of Platform-as-a-Service (PaaS) – from its origins within VMware in 2009, to its rebirth within Pivotal in 2013, to the establishment of the open-source Cloud Foundry Foundation in 2015.

As containers became a predominant deployment mechanism leading to the establishment of Kubernetes as the most popular container orchestration platform, Cloud Foundry transformed itself into an application platform for Kubernetes, and eventually for cloud native development broadly.

As a development platform that users access as services (as with all other cloud native resources), Cloud Foundry can still technically qualify as a PaaS. That designation, however, doesn't do the platform justice, as so much of what makes a platform cloud native exceeds the original conception of PaaS.

Understanding the transformations Cloud Foundry has undertaken over the last decade, therefore, parallels the evolution of PaaS and the transition to the cloud native paradigm for application development and deployment generally.

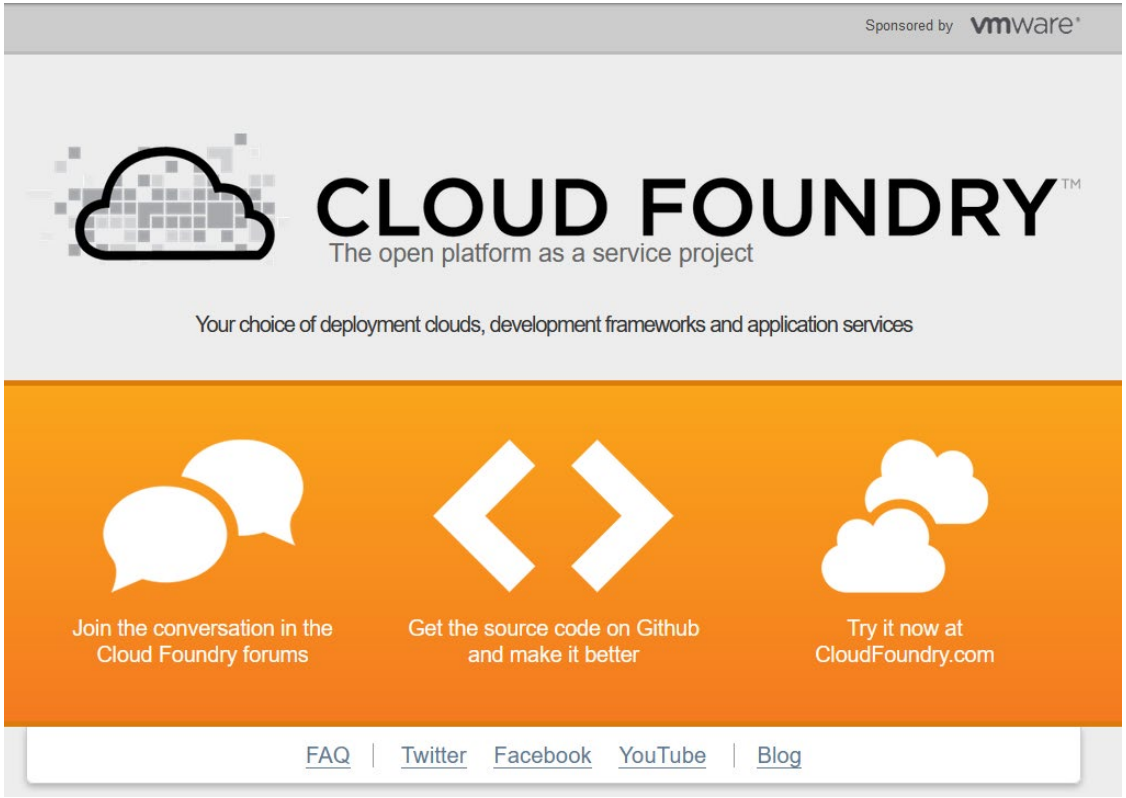
Running an Application Platform on VMs with BOSH

After VMware established itself as the virtual machine (VM) leader with its vSphere virtualization platform, Amazon Web Services (AWS) pioneered Infrastructure-as-a-Service (IaaS) with its Elastic Compute Cloud (EC2) in 2008. EC2 instances are little more than Linux-based VMs running in the cloud.

Early attempts at PaaS were essentially development tooling running in IaaS VM instances. In particular, VMware released BOSH in 2010 as an open-source toolchain for packaging, deploying and managing cloud software on IaaS or traditional (on-premises) VMs.

VMware's primary purpose for BOSH was to deploy early versions of the Cloud Foundry PaaS software, and by 2015, the project became a flagship effort for the Cloud Foundry Foundation, along with the Cloud Foundry Elastic Runtime (now Cloud Foundry Application Runtime).

In the early days before the establishment of the Foundation, however, VMware (and later Pivotal) positioned Cloud Foundry as an Open PaaS project (see its home page from 2011 below).



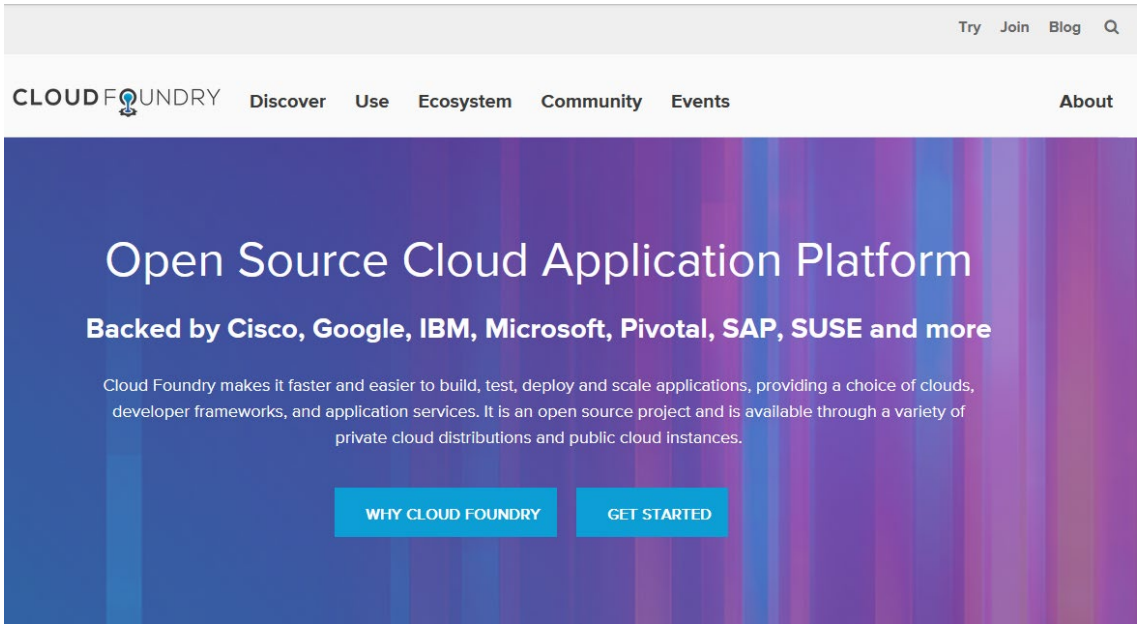
Cloud Foundry home page, 2011 (courtesy the Internet Archive)

These early days of Cloud Foundry aligned with the first-generation conception of the cloud as divided into IaaS, PaaS, as well as Software-as-a-Service (SaaS).

Cloud Foundry differentiated itself from commercial PaaS offerings as an open-source alternative platform that users could install and run in their own hosting environments (IaaS, VMs, or traditional servers) or access in the cloud as an 'as-a-Service' offering.

Moving Beyond PaaS

Between 2015 and 2018, the increasing popularity of Docker containers shifted Cloud Foundry's positioning away from PaaS to a cloud application platform, as its home page from 2018 shows.



Cloud Foundry home page, 2018 (courtesy the Internet Archive).

While Docker soon dominated the containers space, Cloud Foundry focused on interoperability among different container types. Containers transformed the context of application deployment from VM instances to containers. In essence, containers provide their own lightweight virtualization, running in any cloud or on-premises environment on the one hand while supporting any language on the other.

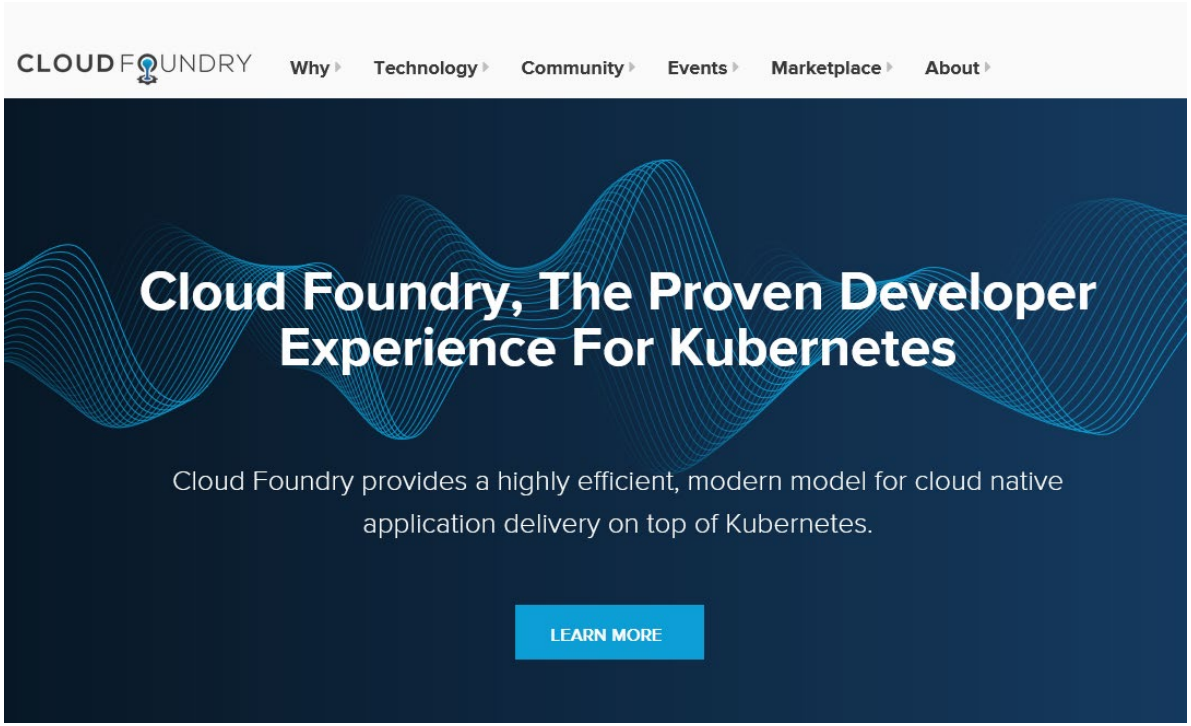
To help developers navigate this transition, Cloud Foundry increasingly focused on the developer experience, simplifying deployment of software with *cf-push*. cf-push gives developers one-click deployment, dramatically simplifying the deployment process.

The ease of use that cf-push brought to the table was especially important as containers became the dominant application infrastructure, as they bring additional complexity to deployment processes compared to software deployments directly into VMs.

Kubernetes Changes the Game

Containers promise better elasticity and scalability than IaaS alone, but to realize this benefit, they require an orchestration platform. By 2020, Kubernetes had established itself as the container orchestration platform of choice.

To respond to this shift in both application development and deployment infrastructure trends, Cloud Foundry rearchitected its platform for Kubernetes, as its home page from 2021 illustrates.



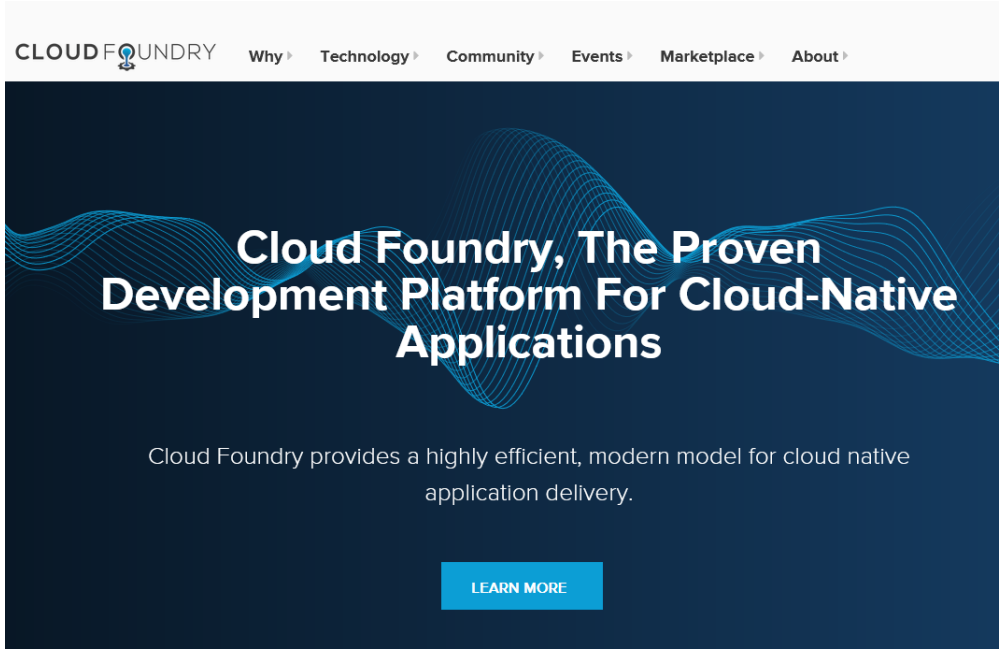
Cloud Foundry home page, 2021 (courtesy the Internet Archive)

As the screenshot above indicates, Cloud Foundry's core differentiator at this time was its 'proven developer experience' – which it built on the one-click deployment success of cf-push.

Kubernetes alone, however, only represents one part of the broader paradigm shift impacting enterprise IT and distributed computing in general. We call this paradigm shift *cloud native computing*.

Cloud native computing leverages the rapid deployment and horizontal scalability benefits that containers and Kubernetes bring to the table, but also comprises other modern trends in computing, including serverless computing, edge computing, CI/CD, DevOps, GitOps, platform engineering, and more.

Cloud Foundry is keeping up with these trends as it continues to mature its capabilities. Since 2022, it has positioned itself as the proven developer platform for cloud native applications, bringing together its platform maturity, ease of use, and cloud native architecture to differentiate itself as one of the most complete open-source cloud native application platforms on the market today.



Cloud Foundry home page, 2022 (courtesy the Internet Archive), largely unchanged in 2024.

The Intellyx Take

One might wonder whether Cloud Foundry is still PaaS after all these years.

The better question would be: what does PaaS mean in the cloud native world? Do we still need the term at all?

The notion of platforms is certainly here to stay. The idea of infrastructure software that enables developers to deploy applications is a powerful one. It is now finding its way into discussions of platform engineering and the internal developer platforms that many organizations are in the process of deploying.

As an application development platform, however, Kubernetes alone leaves much to be desired. Its creators never intended for developers to work directly with it in a hands-on fashion.

Instead, those creators architected it to support additional layers of tooling, either via open-source projects or commercial offerings – a role Cloud Foundry serves.

Given its historic roots, level of maturity, and unquestionable open-source credibility, Cloud Foundry has positioned itself as a leading cloud native application development platform – whether you call it PaaS or not.

By Jason English

Director & Principal Analyst
Intellyx

Paketo Buildpacks: Aligning container images with developer needs

Twenty years ago, software was eating the world. Then around a decade ago, containers started eating software, heralded by the arrival of open source OCI standards.

Suddenly, developers were able to package an application artifact in a container—sometimes all by themselves. And each container image could technically run anywhere—especially in cloud infrastructure. No more needing to buy VM licenses, looking for rackspace and spare servers, and no more contacting the IT Ops department to request provisioning.

Unfortunately, the continuing journey of deploying containers throughout all enterprise IT estates hasn't been all smooth sailing. Dev teams are confronted with an ever-increasing array of options for building and configuring multiple container images to support unique application requirements, and different underlying flavors of commercial and open-source platforms.

Even if a developer becomes an expert in docker build, and the team has enough daily time to keep track of changes across all components and dependencies, they are likely to see functional and security gaps appearing within their expanding container fleet.

Fortunately, we are seeing a bright spot in the evolution of Cloud Native Buildpacks, an open source implementation project pioneered at Heroku and adopted early at Pivotal, which is now under the wing of the CNCF.

Paketo Buildpacks is an open source implementation of Cloud Native Buildpacks currently owned by the Cloud Foundry Foundation. Paketo automatically compiles and encapsulates developer application code into containers. Here's how this latest iteration of buildpacks supports several important developer preferences and development team initiatives.

Open source interoperability

Modern developers appreciate the ability to build on open-source technology whenever they can, but it's not always that simple to decide between open source solutions when vendors and end user companies have already made architectural decisions and set standards. Even in an open-source-first shop, many aspects of the environment will be vendor-supported and offer opinionated stacks for specific delivery platforms.

Developers love to utilize buildpacks because they allow them to focus on coding business logic, rather than the infinite combinations of deployment details. Dealing with both source and deployment variability is where Paketo differentiates itself from previous containerization approaches.

So, it doesn't matter whether a developer codes in Java, Go, nodeJS or Python, Paketo can compile ready-to-run containers. And it doesn't matter which cloud IaaS resource or on-prem server it runs on.

"I think we're seeing a lot more developers who have a custom platform with custom stacks, but they keep coming back to Paketo Buildpacks because they can actually plug them into a modular system," said Forest Eckhardt, contributor and maintainer to the

Paketo project. *"I think that adoption is going well, a lot of the adopters that we see are DevOps or Operations leaders who are trying to deliver applications for their clients and external teams."*

Platform engineering with policy

Platform engineering practices give developers shared, self-service resources and environments for development work, reducing setup costs and time, and encouraging code, component, and configuration reuse.

These common platform engineering environments can be offered up within a self-service internal portal or an external partner development portal, sometimes accompanied by support from a platform team that curates and reviews all elements in the platform.

If the shared team space has too many random uploads, developers will not be able to distinguish the relative utility or safety of various unvalidated container definitions and packages. Proper governance means giving developers the ability to build to spec—without having to slog through huge policy checklists.

Buildpacks take much of the effort and risk out of the 'last mile' of platform engineering. Developers can simply bring their code, and Paketo Buildpacks detects the language, gathers dependencies, and builds a valid container image that fits within the chosen methodology and policies of the organization.

DevOps-speed automation

In addition to empowering developers with self-service resources, ***automating everything as much as possible*** is another core tenet of the DevOps movement.

DevOps is usually represented as a continuous infinity loop, where each change the team promotes in the design/development/build/deploy lifecycle should be executed by automated processes, including production monitoring and feedback to drive the next software delivery cycle.

Any manual intervention in the lifecycle should be looked at as the next potential constraint to be addressed. If developers are spending time setting up Dockerfiles and validating containers, that's less time spent creating new functionality or debugging critical issues.

Software supply chain assurance

Developers want to move fast, so they turn to existing code and infrastructure examples that are working for peers. Heaps of downloadable packages and source code snippets are ready to go on npm and StackOverflow and DockerHub – many with millions of downloads and lots of upvotes and review stars.

The advent of such public development resources and git-style repositories offers immense value for the software industry as a whole, but by nature it also provides an ideal entry point for software supply chain (or SSC) attacks. Bad actors can insert malware and irresponsible ones can leave behind vulnerabilities. Scanning an application once exploits are baked in can be difficult.

It's about time the software industry started taking a page from other discrete industries like high-tech manufacturing and pharmaceuticals that rely on tight governance of their supply chains to maximize customer value with reduced risk. For instance, an automotive brand would want to know the *provenance* of every part that goes into a car they manufacture, a complete bill-of-materials (or BOM) including both its supplier history and its source material composition.

Paketo Buildpacks automatically generates an ***SBOM (software bill-of-materials)*** during each build process, attached to the image, so there's no need to rely on external scanning tools. The SBOM documents information about every component in the packaged application, for instance, that it was written with Go version 1.22.3, even though that original code was compiled.

The Intellyx Take

Various forms of system encapsulation routines have been around for years, well before Docker appeared. Hey, containers even existed on mainframes. But there's something distinct about this current wave of containerization for a cloud native world.

Paketo Buildpacks provide application delivery teams with total flexibility in selecting their platforms and open source components of choice, with automation and reproducibility. Developers can successfully build the same app, the same way, thousands of times in a row, even if underlying components are updated.

That's why so many major development shops are moving toward modern buildpacks and removing the black box around containerization—no matter what deployment platform and methodology they espouse.

By Eric Newcomer

CTO & Principal Analyst
Intellyx

Why You Need an Opinionated IDP for Kubernetes

A big debate in platform engineering is how opinionated an internal developer platform (IDP) should be, and how much should be left to the developers' discretion.

A successful IDP draws a fine line between being too opinionated and being too open. Generally speaking, an organization tends toward being opinionated about standardizing on a small number of platform choices, while developers often prefer a platform to be more open to selecting their own tools.

Different reasons impact selection of an IDP:

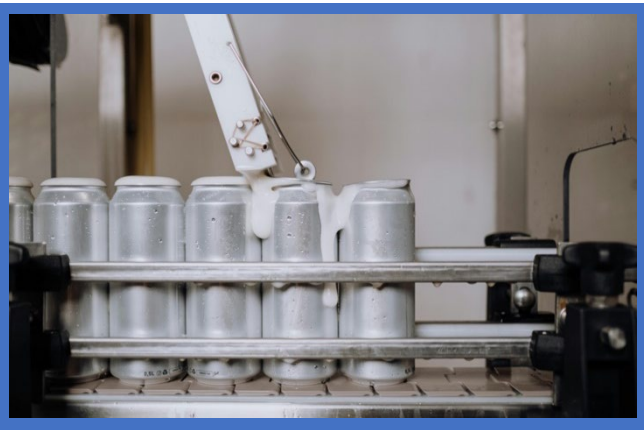
1. The organizational tendency toward being opinionated is typically due to the **cost** of buying and supporting too many software products, and because of the high cost of recruiting and training developers. Too many options increases the cost of both.
2. A big driver for choosing an opinionated IDP is ensuring application **compatibility** across the organization, especially when the application consists of hundreds or even thousands of microservices that need to work together.
3. But primarily, the choice of an opinionated IDP is about having someone with the right **skills and experience** and perspective choose the right technology to simplify the developer's job.

Organizations will often try to be flexible when different options produce the same result – such as using different IDEs of testing tools to produce compatible source code. But at the end of the day organizations will tip the balance toward opinionated solutions for cost efficiency.

The Role of Kubernetes IDP

As many people know by now, Kubernetes evolved out of an internal Google project called Borg, which was developed to automatically deploy programs across multiple VM clusters.

Working with Kubernetes involves two main aspects: configuring and setting up the clusters, and deploying programs into the configured clusters.



Many organizations, especially those with large volumes of Kubernetes clusters, set up Site Reliability Engineering (SRE) teams to configure and maintain the Kubernetes clusters for the developers to use.

Many open source projects and vendor products are available to complement Kubernetes for this reason – i.e. to reduce the overall effort of the SRE teams. Not to mention reducing the need for highly skilled Kubernetes experts, which are in short supply.

Other projects simplify the entire process of development and deployment for Kubernetes – this is the role of the Kubernetes IDP.

Effective Automation with a Kubernetes IDP

People often consider the moving assembly line the key enabler of mass production, or that the CI/CD pipeline is key to automated deployment. In fact it's the standardization of components that enables automation in both cases.

So an opinionated Kubernetes IDP needs to support a build pipeline that deploys standard components into clusters that can be assembled with other standard components, for example to create applications based on microservices that work seamlessly with each other.

Containers are an essential component of an opinionated Kubernetes IDP, because they allow developers the flexibility to use almost any development language.

However, applications typically have integrations with several services, such as working with persistent stores, secrets management, authentication and authorization policies, networking, microservice discovery mechanisms, and API frameworks, and so on.

The choice of OCI container formats and Kubernetes as standard components for a Kubernetes IDP are essential, but only the beginning of the choices for an opinionated Kubernetes IDP.

Buildpacks for Container Dependencies

When building a container for a microservice, there are typically multiple options to consider. Organizations often maintain an opinionated library of approved images to include for various standard functions, such as authentication and authorization.

As the previous blog in this series describes, using Paketo Buildpacks help standardize container builds. Buildpacks transform source code into container images, including all declared dependencies.

Buildpacks simplify automation, reduce build times, create more efficient workflows, and reduce overall cost.

The Korifi Kubernetes IDP

The Korifi Kubernetes IDP from Cloud Foundry supports the Cloud Foundry approach and implements the Cloud Foundry APIs on Kubernetes. Most core Cloud Foundry components are replaced by implementations using cloud native equivalents.

Korifi brings to Kubernetes the classic Cloud Foundry experience of deploying microservices written using any language or framework with a single cf push command. Developers don't have to worry about creating and maintaining their own complex YAML and Docker files.

Cloud Foundry was however originally developed before Docker and Kubernetes became standard choices. The Cloud Foundry community is now bringing its long-standing expertise in the area of a first-class developer experience to Kubernetes, however, offering an IDP that can run existing workloads on Kubernetes, and quickly create entirely new workloads for deployment to Kubernetes clusters.

Why Cloud Foundry for Kubernetes?

Cloud Foundry was among the pioneers of cloud native computing. Its implementation of the 12-factor approach was widely recognized as an essential approach to successful cloud native computing.

One of the great achievements of Cloud Foundry and its consulting practice was helping traditional developers grapple with the differences between the traditional "scale up" environment and the cloud native "scale out" environment.

Many developers find the Kubernetes part of the equation to be complex, and difficult to work with. Developer abstractions such as the Korfi IDP for Kubernetes significantly simplify the process of automated deployment on Kubernetes.

Cloud Foundry's history of cloud agnosticism helps to build solutions that work with any flavor of Kubernetes, and deploy on any cloud provider, data center, or hybrid infrastructure.

The Intellyx Take

The Korfi IDP with Paketo Buildpacks provide application delivery teams with total flexibility in selecting their platforms and open source components of choice, with automation and reproducibility.

That's why many development teams are adopting buildpacks, and removing the black box around containerization—no matter what deployment platform and methodology they use.

An opinionated Kubernetes IDP such as Korfi helps organizations navigate the complexities of deploying on Kubernetes by making some important technology choices for you, while leaving in place the flexibility for developers to choose their own language and dependencies.

By Jason Bloomberg

Managing Director & Analyst
Intellyx

Korifi and the Future of Cloud Foundry

In my last BrainBlog for Cloud Foundry, I reviewed the history of the organization from its birth within VMware in 2009 to its current incarnation as a development platform for cloud native applications.

In the intervening years, Cloud Foundry evolved from a virtual machine (VM)-centric platform to a Platform-as-a-Service (PaaS) offering that the Cloud Foundry Foundation re-architected for containers running on Kubernetes.

Re-architecting *for* Kubernetes, however, is not the same thing as being *Kubernetes-native*.

The Cloud Foundry team understands the difference. To this end, they rolled out **Korifi**, a Kubernetes-native version of its popular Cloud Foundry platform.

Korifi offers the same experience that Cloud Foundry developers have come to love, only on the emerging Kubernetes infrastructure.

The power of custom resources

To build Korifi, the Kubernetes-native version of its platform, Cloud Foundry implemented its APIs directly on Kubernetes by replacing existing Cloud Foundry components with Kubernetes-native equivalents.

Two Kubernetes technologies proved essential for this revamp: *role-based access control* (RBAC) and *Kubernetes custom resources*. RBAC and Kubernetes custom resources mimic the Cloud Foundry platform's *orgs* and *spaces,* respectively.

An org is a development account that one or more individuals can share. Spaces are shared locations that people can use for app development, deployment, and maintenance. Orgs consist of several spaces and map to identifiable Kubernetes pods.

Kubernetes custom resources extend the notion of a Kubernetes resource, which is a Kubernetes API endpoint that stores a collection of API objects. For example, Kubernetes' pod resource contains a collection of pod objects.

Custom resources empower teams to customize a particular Kubernetes deployment. They allow customized Kubernetes deployments without the code-branching and maintenance issues so familiar from customizing earlier generations of software infrastructure and application platforms.

Instead, cluster admins can go about their work as normal, updating custom resources independently of the clusters they're administering.

In addition, people can use standard Kubernetes tools like Kubectl to create and access custom resource objects the same way they would for built-in Kubernetes resources like pods.

Implementing Cloud Foundry APIs

Korifi leverages Kubernetes custom resources to implement Cloud Foundry APIs, thus supporting Cloud Foundry-compatible Kubernetes deployments that nevertheless run on standard Kubernetes.

For the Cloud Foundry developer, Korifi extends the classic Cloud Foundry experience. Developers can still write apps in their language or framework of choice and deploy with a single 'cf push' command as before.

In Korifi, in fact, a single cf push will call several separate API endpoints to orchestrate the push command, where custom resources specify the endpoints.

Developers are therefore able to orchestrate their own cf push commands if they prefer by manipulating the appropriate custom resources – without monkeying with the rest of the Kubernetes platform.

Working with Korifi

Many Cloud Foundry developers who are happy with the existing platform will continue to use it, leveraging Cloud Foundry's BOSH toolchain for packaging, deploying and managing cloud software on IaaS or traditional (on-premises) VMs.

Other developers, however, will prefer to leverage Cloud Foundry while following the idioms and best practices of Kubernetes. Korifi provides this opportunity.

Korifi can integrate with other components of the broader Kubernetes ecosystem following the standard declarative approach familiar to Kubernetes users. Operators, Helm charts, and the rest of the Kubernetes way of doing things are now available to Korifi users.

Korifi also works well with Kubernetes-native tools such as Envoy and Prometheus, while also allowing teams to extend their use of existing CI/CD and observability tooling.

Korifi uses Paketo buildpacks to deploy apps as Open Container Initiative (OCI)-compatible containers, so that developers don't have to deal with YAML or docker files directly (see my colleague Jason English's article on Paketo buildpacks).

In many ways, Korifi developers have the best of both worlds: the power and simplicity of Cloud Foundry combined with the automated networking, security, availability, elasticity, and horizontal scale of Kubernetes.

For organizations with platform engineering initiatives, Korifi also serves as an internal developer platform (IDP). IDPs provide a standard approach to building and deploying applications across an organization – and Korifi is well-suited to serve the role of such a platform.

The Intellyx Take

There are two core audiences for Korifi: the seasoned Cloud Foundry developer who wants a native Kubernetes platform, and the newcomer to Cloud Foundry who is looking for a mature IDP for their Kubernetes deployments.

For the first of these audiences, Cloud Foundry provides two basic options: continued support of BOSH on VMs as well as the Kubernetes-native Korifi.

But Korifi is more than a way for Cloud Foundry to keep its fans happy. It also serves as a viable competitor in the IDP marketplace, even for developers unfamiliar with the long history of Cloud Foundry.

About the Analysts

Jason Bloomberg is Managing Director and Analyst of enterprise IT industry analysis firm Intellyx. He is a leading IT industry analyst, author, keynote speaker, and globally recognized expert on multiple disruptive trends in enterprise technology and digital transformation.

Mr. Bloomberg is the author or coauthor of five books, including *Low-Code for Dummies*, published in October 2019.

Jason "JE" English is Director & Principal Analyst at Intellyx. Drawing on expertise in designing, marketing and selling enterprise software and services, he is focused on covering how agile collaboration between customers, partners and employees accelerates innovation.

A writer and community builder with more than 25 years of experience in software dev/test, cloud and supply chain companies, JE led marketing efforts for the development, testing and virtualization software company ITKO from its bootstrap startup days, through a successful acquisition by CA in 2011. Follow him on Twitter at @bluefug.

Eric Newcomer is CTO and Principal Analyst at Intellyx, a technology analysis firm focused on enterprise digital transformation. Eric is a well-known technology writer and industry thought leader, and previously held CTO roles at WSO2 and IONA Technologies, as well as chief architect and chief security architect roles at Citigroup and Credit Suisse.

About Intellyx

Intellyx is the first and only industry analysis, advisory, and training firm focused on customer-driven, technology-empowered digital transformation for the enterprise. Covering every angle of enterprise IT from mainframes to cloud, process automation to artificial intelligence, our broad focus across technologies allows business executives and IT professionals to connect the dots on disruptive trends. Read and learn more at https://intellyx.com or follow them on LinkedIn.

About Cloud Foundry

Cloud Foundry is a collection of open source technologies being used by leaders in manufacturing, telecommunications and financial services. Only Cloud Foundry delivers the velocity needed to continuously deliver apps at the speed of business. Cloud Foundry's container-based architecture helps deploy apps written in any language on a choice of cloud platforms — Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, OpenStack, and more. With a robust services ecosystem and simple integration with existing technologies, Cloud Foundry is the modern standard for deploying mission critical apps at global organizations.