DeployHub

Understanding the New Kubernetes Pipeline

Changing the CI/CD Pipeline to support a microservice architecture

By Tracy Ragan, CEO DeployHub, CDF Board Member

For Application Development and Production Control Professionals.

Why Read this Whitepaper

Organizations are beginning to move from a monolithic development practice to a Kubernetes and microservice architecture. When this occurs, there are fundamental changes required in the CI/CD pipeline. This whitepaper covers those changes and provides insight into how DeployHub fits into this new Kubernetes Pipeline.

Key Takeaways

- Microservices will have their own repository and workflows. CI/CD tools will need to support workflow templates.
- Configuration management will be lost as large monolithic builds are replaced by small builds with little or no compile. Link decision making is done at runtime.
- Dependency Management and versioning changes to tracking services to applications.
- Organizing microservices into Domains becomes critical for sharing and re-use.

Kubernetes Pipelines are Different

Understanding the difference between Kubernetes pipelines and traditional CI/CD pipelines is not always clear. Yes, Kubernetes disrupts 'business as usual' when it comes to your CI/CD pipeline. Adapting to these changes will be required if you want to take full advantage of a Kubernetes architecture and move from a containerized application to one that is designed around microservices. Shifting from monolithic to microservices is a big deal. In this article, we will cover some of the basic differences between monolithic applications and microservices that are at the core of the pipeline disruption. We will cover four big Pipeline challenges:

- CI/CD Workflows - from a few to hundreds.
- Small builds with no links - a loss of basic configuration management.
- Configuration management is shifting.
- Inventory management for sharing.

We will also briefly cover the changing landscape of the Dev, Test and Prod environment structure common in waterfall. While you may not move away from this environment structure soon, eventually you will need to.

CI/CD Kubernetes Pipeline Challenges

Let's start with the first and most obvious difference between Kubernetes pipelines and monolithic CI/CD. Because microservices are independently deployed, most organizations moving to a microservice architecture tell us they use a single pipeline workflow for each microservice. Also, most companies tell us that they start with 6-10 microservices and grow to 20-30 microservices per traditional application.

Now if you add different versions for each microservice, you may quickly end up with thousands of workflows. Most enterprise companies that we talk to indicate that they believe their cluster will eventually manage 3,000 - 4,000 microservices. We like to call this a death star. In a monolith model you may have managed one workflow per release potentially creating hundreds of workflows, with most not being used. As you updated your application, you may have made corrections to the new workflow. This is not what you will be doing in microservices.

A Kubernetes Pipeline needs the ability to manage thousands of workflows. CI/CD tooling will need to make some adjustments to correct this problem. Workflow templates are critical in solving this part of the Kubernetes pipeline. To manage thousands of Kubernetes pipeline workflows they must be re-used, just like microservices.

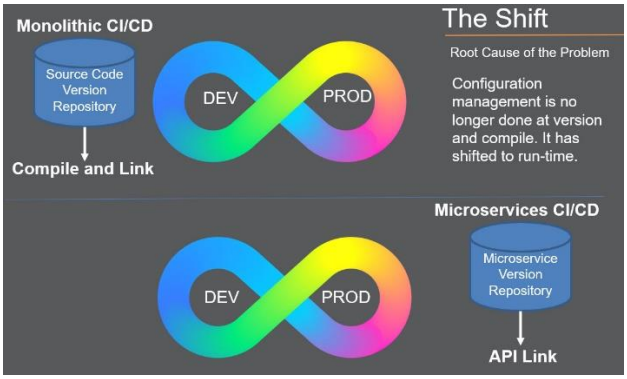
Solutions like JenkinsX are a great start for solving this problem. Other tools like CodeFresh also provide the ability to template your workflows. What you don't want to end up with is a custom workflow for every microservices. Your Kubernetes pipeline will need a solid way of assigning a template to a microservice. In this manner, fixing a template will update all of the workflows. Kubernetes pipeline workflows will be dynamic not static as we see in a monolithic approach.

The Kubernetes Build

We started this journey with the 'build' process, meaning check-out and compile/link. There is some irony that this part of the traditional CI/CD process will be going away.

Yes, there is still a build but now it is focused on updating content into a docker container and registering the container. With microservices you no longer create an application using several pieces of source code that are thousands of lines long. A microservice may only be 300 lines long at the most.

In addition, microservices are often written in a language such as Python which does not require a compile. Other languages such as Go are compiled but are tiny and fast. The big difference is that the Kubernetes pipeline build does not perform linking. As you may well know, microservices are loosely coupled and linked at run time. This shifts the version and build configuration management of monolithic CI practices to be resolved at your run-time environment. Now that is a huge change in how we manage our application code base. Even version control will be impacted. The practice of branching and merging will be less and less critical. Not too many developers will branch code of 100 lines long.



The Configuration Management Shift

Configuration Management is Shifting

The Kubernetes pipeline impacts both dependency management and versioning, the core of configuration management. Now that you have your head around the diminishing role of the build, you can start thinking about how configuration management will change. While you may still use a library management tool for bringing down open source code into a microservice 'build,' it will begin to look different.

In fact, some of those libraries may be distributed as microservices themselves. Open source distribution will change too. The focus of dependency management and versioning will shift from source code and library versioning to microservice and application versioning.

The ability to map the microservices that your API developers are creating to the applications your solution teams are writing will be an essential addition to the Kubernetes pipeline. And remember, a new version of a microservice creates a new version of your application. The ability to track microservices to applications is essential in order to understand impact, when to deprecate and which version of your application your end users are running.

In a monolith, developers control configuration management very tightly through the compile/line process. Because microservices are loosely coupled and shared across team, the developers have less control over the services their applications are consuming. They are not making those decisions at build by statically linking particular versions of shared objects. A Kubernetes pipeline takes dynamic linking to a new level. DeployHub does a particularly good job of solving this issue. It includes a back-end version control engine that integrates with the Kubernetes pipeline to auto increment microservice and therefore application versions. It continuously performs tracking of the

dependencies and creates maps of the cluster.

Inventory Management for Sharing

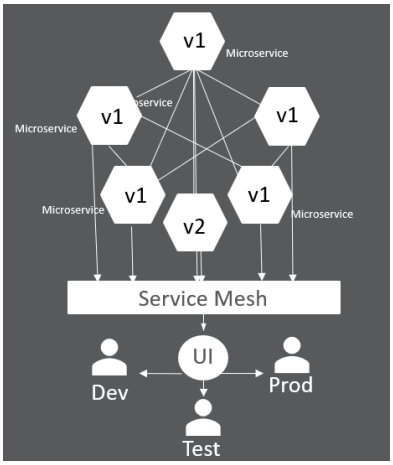
In order to be successful with microservices, they need to be shared. You don't want the situation where you have 10 different single sign-on services written by 10 different teams. The ability to organize microservices facilitates sharing and is achieved using a Domain Driven Design (DDD). DDD is the process of structuring microservices into 'sub-domains' or 'solution spaces.' Organizing microservices in this way is critical if you ever want to give multiple teams the visibility into which microservices are currently available, and which ones may need to be contributed back to a solution space. DeployHub was designed to support a Domain Driven structure. It includes Domains and Sub-domains (with security) allowing teams to catalog and share reusable microservices.

One more thing... No more Waterfall

It's not my intention to offend anyone, but you must agree that the concepts of separate Development, Test, and Production environments come from a waterfall practice, not agile. This waterfall practice is built into our continuous delivery pipeline with some of the same exact waterfall scripts driving the process under the continuous delivery orchestration engine.

With a true microservice architecture, all of this goes away. Service Mesh needs to be called as part of the pipeline to manage the access of new version of a single microservice to end users (creating a new version of the application.)

In essence Development, Test, and Production become defined by configurations of microservices and a service mesh manages the access which then defines if a developer, tester, or end user is using it. Some really advanced companies are starting to go down this road. Check out this presentation from Descartes Labs on their use of Spinnaker and Istio to manage a single cluster running multiple versions of a single solution. It is my prediction that we will be hearing a lot more about Istio and how it is used as part of a Kubernetes pipeline continuous deployment in the very near future.



Service Mesh and Kubernetes

DeployHub – a Single Source of Truth for Modern Software

DeployHub was designed around the concepts of microservices, their management, sharing, and organization. DeployHub allows you to catalog, publish, version and then deploy microservices. Let's look at how we address our four pipeline challenges:

DeployHub and the CI/CD Workflows

Every team will need to decide how they are going to manage microservices inside their workflow process. As discussed, it's most likely that microservices will have a separate repository and a separate workflow. But the path of least resistance might be to utilize your existing monolithic CI/CD pipeline. DeployHub can support either method:

Option 1 – Individual Workflows. DeployHub tracks which applications consume each microservice version. It aggregates the collection of microservices and their versions to the monolithic equivalent. It deploys the microservice individually but creates a new version of the monolithic application logically.

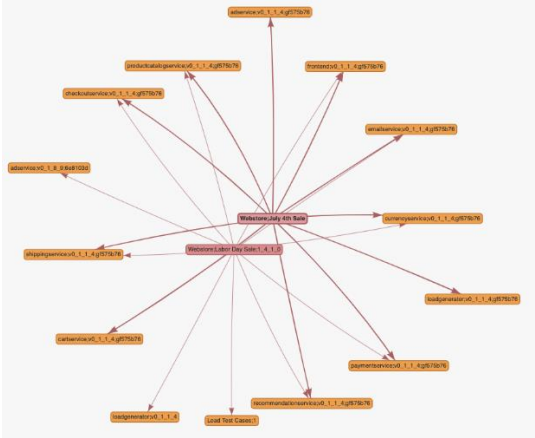
Option 2 – Monolithic workflows. DeployHub tracks each component of a monolithic application individually. When a new application release is pushed forward, it determines which microservice (component) needs to be released vs. re-releasing all of the microservices. DeployHub performs iterative releases on a monolithic application, supporting a process that allows a microservice to be independently deployed even though many microservices are part of the workflow.

The Kubernetes Pipeline Build

DeployHub provides a Blueprint process that allows developers to define their application base line as a package. DeployHub uses this base line to define the initial relationships between the application and the microservices it consumes. Once defined, DeployHub tracks the differences over time based on that base line. Then DeployHub interacts with the Kubernetes pipeline to auto increment microservice versions to application versions, tracking which versions of services each application version is consuming. By doing so, it creates dependency maps.

Automated Configuration Management

DeployHub performs automated configuration management. It tracks every microservice, the deployment meta data, and who is using it. For each microservice defined to DeployHub, the meta data needed for deploying becomes part of its version attributes. This means any team can re-use the microservice immediately - it is ready for release. If it has already been released to the cluster, DeployHub knows this and does not re-release it with the application package. It creates maps of your application versions over time so you always understand which version of a microservice your application version is using. This type of configuration management was previously done during the monolith build. But in a Kubernetes pipeline it is lost. DeployHub provides a way to re-create that level of information and insights.



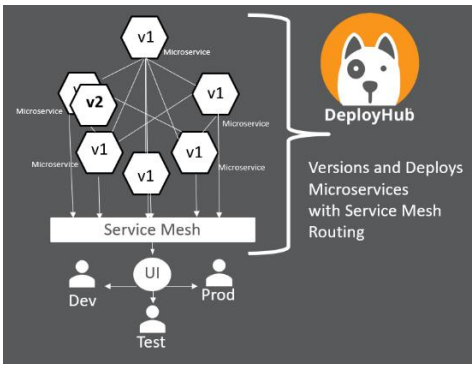
DeployHub Dependency Maps

Inventory Management for Sharing

DeployHub is a sharing platform for microservices. A Domain Driven Design defines catalogs for organizing microservices and managing inventory. Developers publish their microservices (or rather their pointers to where the microservices are being stored such as Quay, Docker Hub, etc.) to DeployHub under a specific domain. Domains can be secured or openly shared across teams, it's your choice.

What is coming - Single Cluster Release Management

DeployHub can interact with Service Mesh to enable certain user groups to access a particular version of a microservice. DeployHub releases your new version to the cluster and interacts with Service Mesh enabling developers to use the new version of the microservice. The deployment step is only done once. When the developers are ready, Service Mesh provides access to Testers. During Development and Test, the application is comprised of both production microservices and dev/test microservices. When QA testing is completed, Service Mesh is updated to allow production users access to the new microservice. A monolithic deployment to 3 separate environments is no longer needed. The deployment is done only once and users are routed to the appropriate combination of microservices. While you may not be ready to ditch the waterfall just yet, DeployHub is and will be ready for you when you get there.



DeployHub and Service Mesh Routing

Conclusion

Moving to a Kubernetes and microservice architecture will require that you tweak your existing CI/CD pipeline to support this new modern architecture. A Kubernetes pipeline requires some simple changes: template workflows, a new configuration management process to replace the build, a versioning scheme to track microservice versions and application versions, and a Domain structure for cataloging services. Eventually, you will become so proficient with microservices that you will see the benefit of moving from multiple Dev, Test, Pre-Release, Release clusters to one or two clusters using service mesh routing. The good news is that this new microservice architecture takes us to a new level of service for our customers, one that stabilizes 80% of the environment by requiring only incremental updates of small services, fault

tolerance and high availability. In other words, I think we finally achieved agile.

Get Started with DeployHub

To begin tracking your microservice configurations and versions, sign up with DeployHub Team. This option is based on the open source Ortelius Project. It is provided for free and hosted for high performing development teams who want to manage services in the new modern architecture.

DeployHub Team Sign-up: The hosted team version can be used to track and configure unlimited microservices and applications with unlimited end users and endpoints. https://www.deployhub.com/free-team-sign-up/

Get Involved in the OS Project: Help us create the best, open source microservice configuration platform available at ortelius.io.

About DeployHub

DeployHub empowers high performing software developers to simplify a microservice architecture through configuration and inventory management. DeployHub catalogs, publishes, shares and deploys microservices across the organization, quickly and safely on a hosted (SaaS) platform. DeployHub is based on the Open Source Project Ortelius

About the Author

Tracy Ragan is CEO of DeployHub and serves on the Continuous Delivery Foundation Board. She is a microservice evangelist with expertise in software configuration management, builds and release. Tracy was a consultant to Wall Street firms on build and release management for 7 years prior to co-founding OpenMake Software in 1995. She was a founding member of the Eclipse organization and served on the board for 5 years. She is a recognized leader and has been published in multiple industry publications as well as presenting to audiences at industry conferences. Tracy co-founded DeployHub in 2018 to serve the microservice development community.

Visit us at DeployHub.com