From Legos to Transformers – How the Software Life Cycle will change with Microservices

A whitepaper on shifting your lifecycle to support Kubernetes and Microservices.

By Tracy Ragan, CEO, DeployHub

Table of Contents

Summary

Historically our software lifecycle has resembled the process of putting together a Lego set. We assemble our super cool Lego dragon, and then it is pushed to a shelf for viewing. Don't touch it or it might break. This is the same with the software build and release process and the reason why to this day some teams still take several weeks to get a release out the door, even though the code was ready weeks before. We just don't like touching production. Leave it on the shelf to look at.

Agile tried to change these habits, but the ugly reality is that the build and release process has always been baked in the traditional waterfall processes. Yes, we update code faster and we execute a build script, but how often is the build moved up the pipeline to production? Not very often. These old habits are being hugely disrupted by Kubernetes and microservices. Our Lego set is turning into a Transformer, an ever-changing state of production. This might frighten some, but it is the direction we are finally moving. In other words, goodbye waterfall. This whitepaper explores the transformation in the software lifecycle that is being driven by Kubernetes and microservices.

The End of Waterfall, and the Beginning of a New Era

If you are reading this whitepaper, then I certainly do not need to reiterate what a waterfall methodology looks like. But what I will do is point out how waterfall has locked us into a 'Lego' mind set. One word describes the problem – monolithic. Monolithic means that a full 'application' is re-compiled and re-released every time a new update is required. While Agile taught us to make small coding changes, we continued our waterfall practices around re-building and re-releasing the entire application code base. This is not agile. We have been held hostage by our belief that incremental builds and releases were high-risk and therefore not done. Developers execute their full code build and deploy scripts for dev environments, but test and production do something different. Something else between the stages has blocked the pipeline and reinforced the waterfall practices. Releases are held back and the goal of agile is not achieved. With Kubernetes and microservices, all this goes away.

With Kubernetes, we move from a static monolithic application to a highly dynamic, ever changing 'Production' state. Production 'transforms' on a high frequency basis to deliver innovation and meet customer service level demands. The concept of a waterfall practice with unique development, testing and production stages goes away. In addition, the concept of software builds, code versioning and overall continuous integration will also change.

The Versioning and Configuration Management Shift

Version Control and configuration management is tightly coupled with the concept of software 'builds.' The process of pulling code from a version control tool, running a compile and link script to configure binaries for a monolithic application is a late 20th century concept. This process, which led to the birth of continuous integration, was required with monolithic style programming, but not so much with microservices. The need for 'fixing' builds or checking in code for the build goes away to a large degree. Developers will work more independent. Source code will be managed in many repositories instead of a few large repositories. A build will involve creating a docker image that is stored in a docker repository. Containers with their microservices will be built and pushed to production independently. The versioning and configuration management problem will now shift to run-time, not build-time.

Tracking Run-time Configurations

With microservices you must think ‘functions.’ AWS Lambda Functions, small snippets of code written in PHP or JavaSscript make up a microservice. Unlike the days of C, C++ or even Java where source code files could be hundreds of lines long and edited by multiple developers, microservices include a small number of lines of code, not hundreds. While developers will continue to use a source repository to check-in their code, the process of ‘building’ the code will look different. More importantly, linking is no longer done at build-time, instead linking is done through RESTful APIs at runtime.

In a service-based architecture, Application teams will write ‘private’ microservices and re-use microservices from ‘common’ teams. As teams learn to collaborate on microservices, there will be hundreds of relationships between Applications and microservices. A single microservice will most definitely be used by multiple Applications.

For this reason, the need for versioning and configuration management is still required, and in some cases even more critical. The change is that this information needs to be gathered just prior to a deployment instead of just prior to a software build (compile and link).

The Kubernetes Pipeline and Continuous Delivery

There is no doubt that continuous delivery orchestration will be part of the Kubernetes story. It will just orchestrate different steps. Version control will be used to pull a small set of code files and a ‘build’ will become a ‘docker build’ that creates a container image. In some cases, a compile will not be needed. Javascript, PHP and Python are interpreted, not compiled. The container image will be stored in a container repository where it will be referenced for a deployment. The deployment step will become more critical as this is where the versioning and configuration management will occur. The deployment step will have the visibility into which services were used to create the application, the SHA (secure hash algorithm) that references the container image, and the script used to update Kubernetes with the new container image SHA, and the cluster to where it was installed.

This build and deploy process is completely different from our old way of doing things. It will be important to keep this in mind as you define your continuous delivery pipeline that may need to support a hybrid approach for some years to come.

Things to Ponder

Here are some items to begin thinking about as you look toward your future in a service-based architecture:

1) Organizing Services into Domains

With your microservices now broken into independently deployable objects that talk via APIs, the process of organizing them into domains becomes super critical. Domain Driven Design is a keystone of a successful modern architecture implementation. For teams to share services and components they must be able to find them. And a microservice that is consumed by no one is a useless microservice. To achieve success in a service-based architecture, organization and management of these services is critical.

Microservices should be cataloged and shared in a method that best represents the logical functions of an organization. Take time on this step. It is critical to your success and will save you big dollars later down the road. The last thing you want is duplicated microservices across your organization. You must have a method for developers on the 8th floor to share their work with the developers on the 4th floor. If not, they will all write similar services – a costly mistake.

3) A Logical View of Your Application

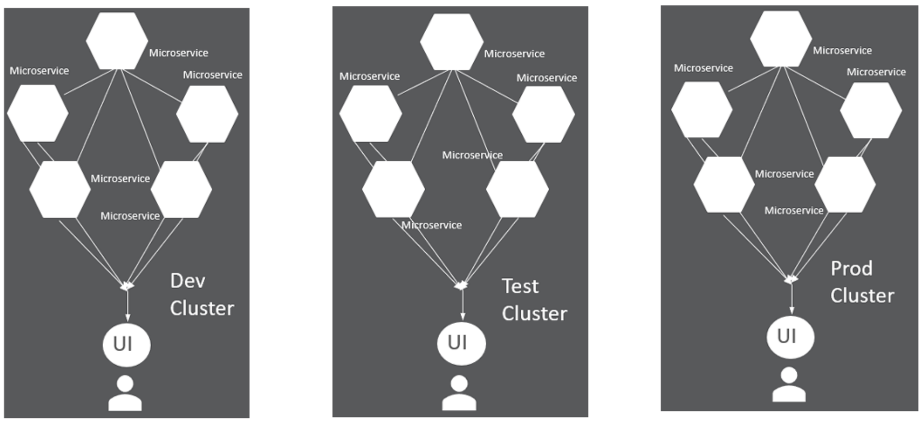
As you progress down this path, you will begin thinking less of an 'application' and more about services. Regardless, you will still be creating a software application in the same way as you did in your monolithic days. The only difference is that you will be tracking a logical view of your monolithic equivalent. This step is also critical. Each version of your logical application will be comprised of different versions of microservices and components. A new version of your microservice creates a new version of your application. And you will need to keep track of what applications are consuming which microservices. If a new version of a microservice is deployed, applications that consume them should be notified.

3) Microservice Configuration Management and Versioning

With a runtime environment that is in a constant state of change, configuration management and versioning become essential. They are the breadcrumbs of what occurred and how to get back or move forward to a healthy state when something goes wrong, and it will. While Kubernetes and microservices is a huge leap forward, it is not a utopia. Things will break.

Deploy Once and Route Many

If our waterfall methods disappear, will our environment stages also go away? This question is creating quite the conversation amongst Kubernetes and microservice enthusiasts. The answer seems to be evolving into 'yes.' Most organizations are beginning with a staged approach creating separate clusters for each environment:



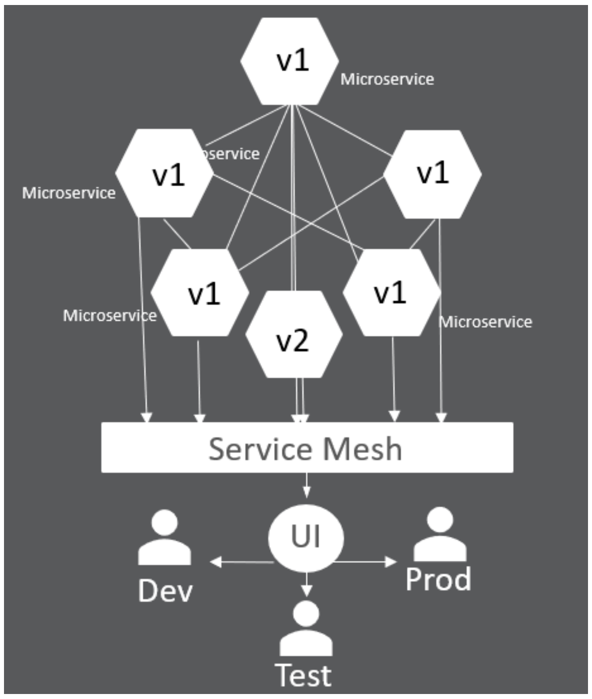
Staged Kubernetes

This approach seems to provide teams a level of familiarity that improves confidence in moving to this strange new architecture. But as organizations gain experience with Kubernetes and microservices, they begin seeing these staged clusters as too complex.

This familiar model will shift as the team becomes more comfortable and experienced in managing Kubernetes and begin using Service Mesh.

Kubernetes Service Mesh

Service mesh provides the ability to deploy a container to a cluster and then route specific personas to use the container, i.e. dev group, test group, or default to a production group. While database components may need some updates to point to different databases, other components and microservices can be treated more like feature flags. When a new container is released, service mesh can be told to route the specific users to the new container. In addition, service mesh can manage a structured rollout, routing to a percentage of users and expanding overtime. What this means is that the process of deploying over and over to different environments goes away. While at first glance this might appear to add risk, it reduces risk. Your deployment is tested and validated once and reused by many. With Kubernetes, every deployment is a Blue/Green deployment – service mesh is the layer that manages the routing forward or backward. And remember, when an update to a microservice is needed, a new version is pushed to production, but the old one remains until it is deprecated. So again, a rollback becomes a routing task.



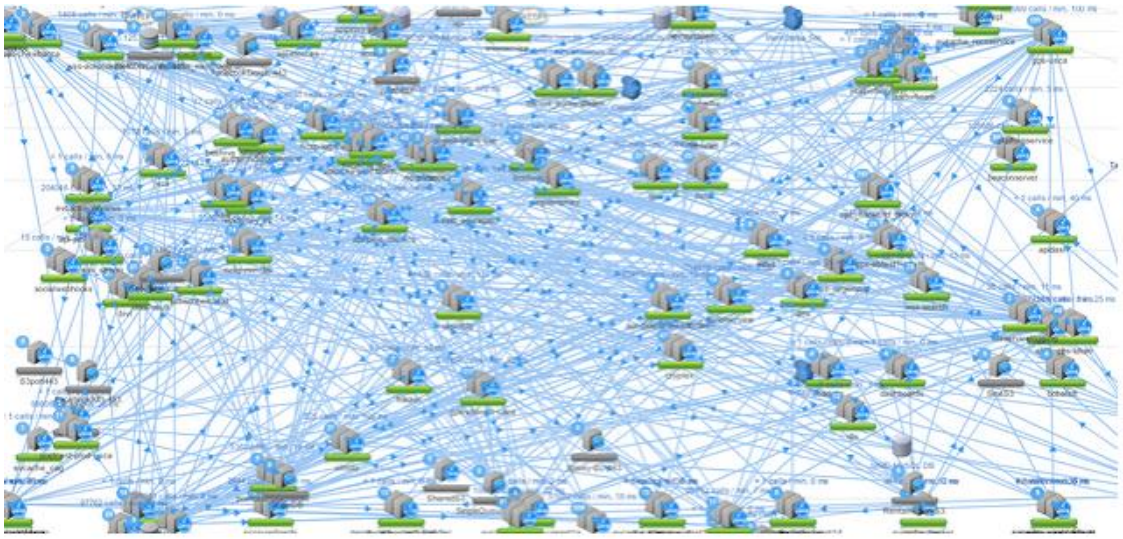
Service Mesh Routing

It is important to note that currently service mesh is still a new addition to the Kubernetes family. While this type of routing is not ready for prime time, it ultimately will be used to manage the routing of features and replacing the need for multiple clusters to support a waterfall like data center (separate Dev, Test, and Production clusters).

Seeing the Map

As with any advance in IT, there are always challenges. As you begin to deploy microservices that are being used across the organization they will grow in numbers very quickly. Tracking the relationships of the microservices and components to one another will also grow in complexity. The terms 'Frankenstein clusters,' 'haunted graveyards,' and 'death stars' are becoming common terms among Kubernetes users. In other words, if you have microservices running, but you don't know who is using them, or if anyone at all is using them, what do you do? Bring down the cluster and see who complains? That is one technique. The other is to track the dependencies as you deploy. Being able to know the dependencies in a useful view will be critical.

It’s absolutely critical to know when a shared microservice has been updated and who needs to be notified. These and other complexities must be addressed early in your service implementation to achieve long term success. Your dependencies will continue to become more complex. The more data about these dependencies, the better off you will be in the long run. Take a look at the Netflix microservice map and realize that your microservice environment will look like this in no time.



Netflix microservice map

Netflix Tech Blog – Allen Wang and Sudhir Tonse

Conclusion

Kubernetes is changing the way we write, manage, and deliver software. Our old way of moving monolithic applications to static build, test, and deploy environments will eventually go away. The replacement will be a continuously transforming collection of independently deployable microservices and components running in a collection of clusters, organized by domains. Our software applications will still exist, but we will see them as a logical collection of services and components. Software builds will be minimized as a no link process will be required to create a monolithic application. The link will be replaced with runtime APIs that allow microservices to exchange information. Our deployments will be greatly simplified while our deployment tooling will track the dependencies and locations of microservices running in Kubernetes. Deployment solutions will become a central key to tracking the services, relationships and locations with versioning and configuration management included in the deploy. In addition, the deployment step will integrate with service mesh to perform feature routing to users who are in dev, test and production groups not separate clusters.

In general, our new world is going to become much easier. But first, there will be some complex years as the community re-tools to automate and streamline the management of hundreds of microservices running across the organization.

About DeployHub

DeployHub empowers high performing software developers to catalog, publish, share and deploy microservices across the organization, quickly and safely on a hosted (SaaS) platform. Learn more at www.DeployHub.com. DeployHub is based on the Open Source Project Ortelius.

Tracy Ragan – CEO and Co-Founder, DeployHub

Tracy has had extensive experience in configuration management from both a build and release perspective. She is an industry leader who currently serves as a Board Member on the CD.Foundation. She also served on the Eclipse board as a founding member for 5 years. She can be reached on Twitter @TracyRagan

Getting Started with a Kubernetes & Microservices

Check out these open source, low cost or ‘free’ offerings to get started.

RANCHER

Kubernetes Cluster Management

QUAY

Container Builds and Repository

DeployHub

Continuous Deployment with Microservice Management

HELM

Infrastructure & Installation

Rancher

Use Rancher OS to help you get started on your Kubernetes journey.

DeployHub

DeployHub will support continuous deployments and the configuration management of the microservices, with a built-in domain design for services sharing.

Quay

DeployHub uses Quay to build a container image and store the results in the Quay repository. DeployHub references Quay for deploying containers.

Helm

Helm Charts are called as part of the container install process. DeployHub integrates with Helm Charts as “actions” that are called as part of the deployment of a microservice in each cluster environment.