Docker and the Three Ways of DevOps

Author: John Willis, Docker

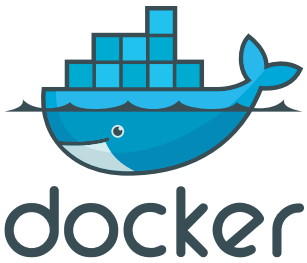


Table of Contents

Forward

I had the privilege of meeting John Wills at DevOpsDays Mountain View in 2010, and it was an interaction that I'll never forget. I was still trying to figure out what this whole DevOps movement was all about, and it popped into focus when he told me, "IT operations has been lost at sea for thirty years — DevOps is how we can finally find our way back."

In that moment, I knew he was a kindred spirit who had seen the same problems that I had, and that DevOps principles and patterns allow IT organizations to escape the almost inevitable downward spiral that occurs when Dev and Ops don't work together towards common goals.

I'm privileged that I've been able to work with John Willis on so many projects, including the upcoming DevOps Cookbook [http://itrevolution.com/books/devops-cookbook/], where we further describe the Three Ways, which are the set of principles from which we can derive all the observed DevOps behaviors that allow the sustained fast flow from Dev through Ops to the customer, while preserving world-class reliability, stability, and security. By doing this, organizations are able to maximize developer productivity, enable organizational learning,and the ability to win in the marketplace.

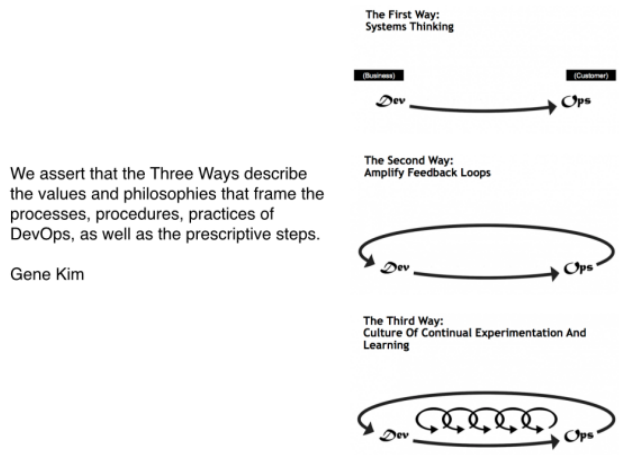
In this paper, John extends the Three Ways into the emerging age of containers and Docker. I trust you'll find it as provocative and useful as I did!

Gene Kim, July 2015

Executive Summary

DevOps is a mindset, a way of thinking, versus a set of processes implemented in a specific way. The goal of all DevOps organizations are the same: to improve the quality and speed at which innovation is delivered. What began an experiment has transformed into a movement to bring together software development and the operations to run the software closer together. Over the years, industry leaders and DevOps practitioners have come together to identify patterns and best practices so that successful methods are shared to the next wave of DevOps practitioners.

Three principles known as "The Three Ways of DevOps" have been identified and these are principles that all other Devops patterns can be derived from. The Three Ways of DevOps include: systems thinking, amplifying and shortening feedback loops and continuous learning.



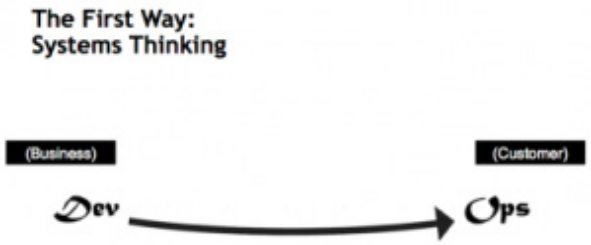
These principles describe the values and philosophies that frame the processes, procedures, practices of DevOps, as well as the prescriptive steps. The capabilities of the Docker platform with containerized compute, storage and networking for distributed applications provide promising results when applied to the Three Ways of DevOps principles. From exponential speed to and velocity to streamlining the feedback loop and collaboration across teams.

Key takeaways from this paper include:

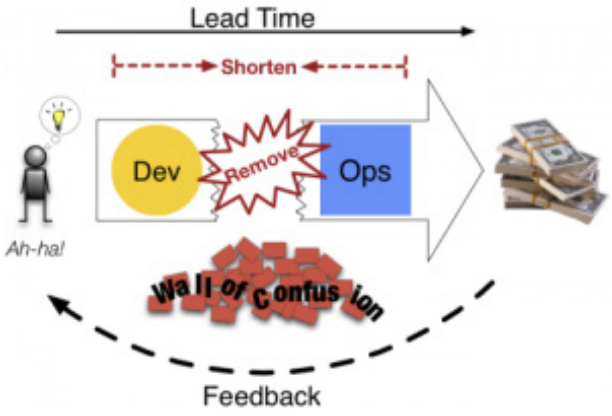
- A deeper understanding of Three Ways of DevOps principles and their purpose
- How to apply these principles with Docker into your organization
- Examples of results and benefits from other practitioners' experiences

The First Way: Systems Thinking

Systems thinking is defined as the "First Way" of DevOps. This is defined as flow and direction from left to right, sometimes referred to as a pipeline. It is important to understand the system as a complete value stream.



Managing this flow is also referred to as global optimization or bottleneck reduction. In DevOps jargon, this is known as the time it takes to get from "A-Ha to the Cha-Ching". In Lean, it is referred to as "Lead Time". This is synonymous with the time it takes to get from a whiteboard diagram to a paying customer feature or the time it takes to transform the things that people make into the things that people buy.



To be effective at the ""First Way"", you need to be able to apply increased velocity to all of the local processes while not slowing down the global flow. This requires attention to three main principles:

1. Increase "velocity" by accelerating each of the process components in the pipeline.
2. Decrease "variation" by eliminating wasteful or time consuming sub processes in the pipeline.
3. Elevate the processes by bounded context (isolating the functionality) therefore better visualizing and understanding the global flow (e.g. seeing the system).

What does Docker have to do with all of this?

Docker and the First Way

Velocity

Developer Flow

Developers who use Docker typically create a Docker environment on their laptop to locally develop and test applications in containers. In this environment, they can test a service stack made up of multiple Docker containers. Container instances spin up at an average of around 500 milliseconds. Multiple Docker containers converged as a single service stack, like a LAMP stack, can take seconds to instantiate. Convergence is the interconnection information that needs to be configured between the instances, for example a database connection info or load balancer IP's. Contrast this to non-container virtual instances running as multiple hosts. An alternative non-containerized environment can take anywhere from 2 to 10 minutes to spin up and converge depending on the complexity. The end result with Docker is that developers experience less context switch time for testing and retesting and resulting in significantly increased velocity.

Integration Flow

Docker can streamline a continuous integration (CI) with the use of Dockerized build slaves. A CI system can be designed such that multiple virtual instances each run as individual Docker hosts as multiple build slaves, for example. Some environments run a Docker host inside of a Docker host (Docker-in-Docker) for their build environments. This creates a clean isolation of the build out and breakdown environments for the services being tested. In this case, the original virtual instances are never corrupted because the embedded Docker host can be recreated for each CI slave instance. The inner Docker host is just a Docker image and instantiates at the same speed as any other Docker instance.

Just like the developers' laptop, the integrated services can run as Docker containers inside these build slaves. Not only are there exponential efficiencies in the spin up times of the tested service, there are also the benefits of rebasing multiple testing scenarios. Multiple tests starting from the same baseline can be run over and over in a matter of seconds. The sub-second Docker container spin-up times allow scenarios where thousands of integration tests occur in minutes that would otherwise take days to complete.

Docker also increases velocity for CI pipelines with Union FileSystems and Copy on Write (COW). Docker images are created using a layered file system approach. Typically only the current (top) layer is a writable layer (COW). Advanced usage of baselining and rebasing between these layers can also increase the lead time for getting software through the pipeline. For example, a specific MySQL table could be initialized at a certain state and could be rebased back to the original state in the container image for each test therefore providing multiple tests with accelerated efficiencies.

Deployment Flow

To achieve increased velocity for Continuous Delivery (CD) of software, there are a number of techniques that can be impacted by the use of Docker. A popular CD process called "Blue Green deploys" is often used to seamlessly migrate applications into production. One of the challenges of production deployments is ensuring seamless and timely changeover times (moving from one version to another). A Blue Green deploy is a technique where one node of a cluster is updated at a time (i.e., the green node) while the other nodes are still untouched (the blue nodes). This technique requires a rolling process where one node is updated and tested at a time. The two key takeaways here are: 1) the total speed to update all the nodes needs to be timely and 2) if the cluster needs to be rolled back this also has to happen in a timely fashion. Docker containers make the roll forward and roll back process more efficient. Also, because the application is isolated in a container, this process is much cleaner with far less moving parts involved during the changeover. Other deployment techniques like dark launches and canarying can benefit from Docker container isolation and speed, all for the same reasons described earlier.

Variation

A key benefits of using Docker images in a software delivery pipeline is that both the infrastructure and the application can both be included in the container image. One of the core tenants of Java was the promise of "write once run anywhere". However, since the Java artifact (typically a JAR, WAR or EAR) only included the application there was always a wide range of variation, depending on the Java runtime and the specific operating environment of the deployment environment. With Docker, a developer can bundle the actual infrastructure (i.e. the base OS, middleware, runtime and the application) in the same image. This converged isolation lowers the potential variation at every stage of the delivery pipeline (dev, integration and production deployment). If a developer tests a set of Docker images as a service on a laptop, those same services can be exactly the same during the integration testing and the production deployment. The image (i.e., the converged artifact) is a binary. There should be little or no variation of a service stack at any of the stages of the pipeline when Docker containers are used as the primary deployment mechanism. Contrast this to environments that have to be built at each stage of the pipeline. Alternatively, configuration management and release engineer scripting are often used at each stage of the pipeline to build out the service. Although most automation mechanisms are far more efficient than a checklist built service, they still run the risk of higher variation than binay Docker service stacks. These alternatives to Docker containers can yield wider variances of stability therefore increasing variation. A Dockerized pipeline approach delivers converged artifacts as binaries and therefore are immutable starting from the commit.

As an example, Gilt Group uses Docker as a primary deployment mechanism. Gilt makes use of a microservices architecture in a delivery pattern of what they call an immutable infrastructure. In other words, artifacts in their production environments are not updated. Infrastructure is always replaced. Meaning they either roll forward or roll backwards. Gilt demonstrated how a developer packages a set of container binaries and then provides one meta file (a Docker run description file) to the pipeline. Everything in the bundle is self contained. Prior to their Dockerized deployment process, Gilt had a repo of over 1000 release engineering build scripts managing over 1000 different repos of software and with 25 different deployment models. This old process created a wide range of variation in the pipeline. Discovery and ownership of the scripts often created bottlenecks in the pipeline. It was unclear what to do with break fix situations because of the variation in the process. A classic "DevOps" adage is having the developers get paged when an issue occurs whenfor production application issues. In the case of Gilt, not only do the developers wear pagers for escalations, they also have ownership of the complete embedded infrastructure.

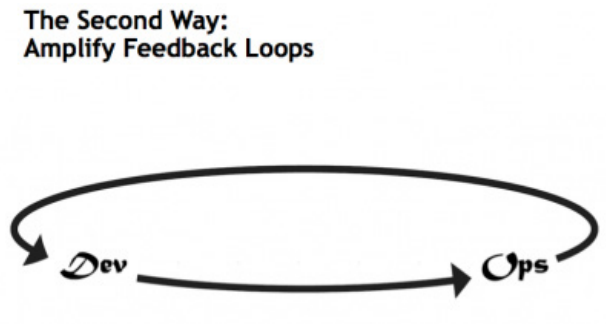
Visualization

A new model of disruption in our industry is called Containerized Microservices. In a microservices architecture "services" are defined as bounded context. These are services that model real world domains. There might be a service domain called finance or warehouse in a microservices architecture. When these bounded services are coupled as Docker containers and used as part of the delivery pipeline, they are immediately visible as real word domains. From an "DevOps" operational aspect, one of the keys to success is an organization MTTR (Mean time to Repair/Restore) metric. When services are bounded by their business context and then isolated as Dockerized containers they become elevated (i.e., visual) in the pipeline. This increased visibility can help an organization isolate, discover and determine proper ownership faster; therefore decreasing their overall MTTR.

The "First Way"" and Docker can provide global optimization around software Velocity, Variation and Visualization. Dockerizing the development pipeline, organizations can reduce the cost and risk of software delivery while increasing the rate of change.

The Second Way: Amplify Feedback Loops

The "Second Way" is defined as amplifying and shortening feedback loops such that corrections can be made fast and continuously. This is sometimes referred to as the right to left flow.



A defect is not a defect unless it hits the customer. Lean principles teach us that the earlier a potential downstream defect is discovered, the less costly it will be to the overall cost of the service delivery. Therefore the three V's also apply in the "Second Way". Velocity of a correction in the flow is essential. Variation also plays a role with the complexity of the infrastructure, where the defect has been identified needs to be simpler so that it requires less time to detect. Lastly, software artifacts need to be elevated and bounded (i.e., visual) to their source (e.g., source code, source code repository) in order to decrease the overall Lead Time of a service delivered.

Docker and the Second Way

Velocity

Much like velocity in the ""First Way"", in the "Second Way" it is about speed with direction. The important thing to remember about flow is that it's not always going in the same direction. There are interrupts in the flow due to defects and the potential changeover time related to the defect. To be effective at this "Second Way", DevOps need to have velocity in both directions (i.e., the complete feedback loop). How fast can the changeover occur? How adaptive is the process for not only quick defect detection but how fast can the system be reimplemented and rebased to the original baseline conditions? In Lean, there is something called the Andon Cord that was used to stop the production line if a defect was discovered in the production process. Because the Andon Cord would actually stop the line, this was a metaphor for the strength of the process.

It is the idea that pulling the cord could actually fix the defect and make a difference. Under this premise, it was more likely that a line worker would actually stop the line, even for minor defects, because they knew the process to fix the defect was actually streamlined. Docker's streamlining of packaging, provisioning and immutable delivery of artifacts allow an organization to take advantage of shortened changeover times due to defects and make it easier to stop the line if a defect is detected.

Variation

Here again the advantages of using Docker in the "Second Way" are similar to the advantaged outlined in the ""First Way"." In this example, it's about the complexity of the infrastructure that is created of where the defect is detected. A complex set of software artifacts at scale can be fragile. Software-based services can be made up of thousands of classes and libraries with many different integration points. A slight delivery variation like how the full stack was built, can be just enough to trigger a defect that can be very difficult to detect. Good service delivery hygiene mandates that all artifacts start as source in a version control system; however rebuilding everything from source at every stage of the pipeline might be just enough variation to trigger defect variants. A Docker delivery and the use of immutable artifacts through the pipeline reduces variation and therefore, reduces the risk of defect variants later in the delivery pipeline.

Visualization

One of the advantages of an immutable delivery process is that most of the artifacts are delivered throughout the pipeline as binaries. This allows a service delivery team to create metadata from the source that is maintained and can be visualized at any stage of the pipeline. It is not uncommon to see developers embed GIT SHA hashes related to the GIT Commit for a particular section of code in the Docker image. Other techniques for including additional metadata about the software artifact can also be embedded in the Docker image. R.I.Pienaar has a related blog post on his devco.net site about embedding metadata inside every one of his Docker images along with a couple of useful inspection scripts. Below is a list of some of the operational metadata R.I. includes in all of of his container images:

- Where and when was it built and why
- What was its ancestor images
- How do I start, validate, monitor and update it
- What git repo is being built, what hash of that git repo was built
- What are all the tags this specific container is known as at time of build
- What's the project name this belongs to
- Have the ability to have arbitrary user supplied rich metadata

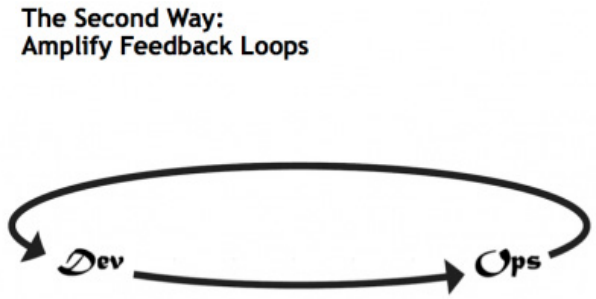
All of this is another form of a "Second Way" feedback loop. When troubleshooting a discovered or potential defect, visualizing embedded metadata can speed up the time required to correct the defect and therefore, reduce the overall Lead Time of the service being delivered.

The Third Way: Continuous Learning

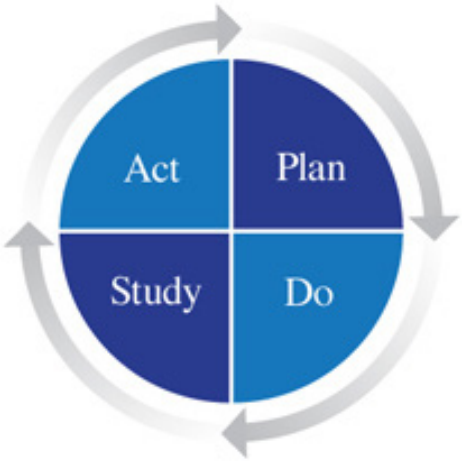
The "Third Way" of DevOps completes the full cycle. It has also been referred to simply as "Continuous Learning".

In DevOps speak, word Kaizen is often viewed as a method of continuous improvement in an organization. This "Kaizen" being achieved through a culture of continuous experimentation and learning throughout all activities in an organization.

The "First Way" is about a left to right flow and systems thinking. While the "Second Way" defines right to left feedback loops with quick turnaround.



In this "Third Way", the symbol of a complete loop is used because it ties the first two ways together through a rigorous implementation of a learning process.



source: The Deming Institute

Organizations that follow the principle of the "Third Way" employ a form of experiential learning. Edward Deming, a famous American management thought leader; calls this the Plan Do Study Act Cycle (PDSA). PDSA is rooted in principles of scientific method in that every thing you do is a small experiment.

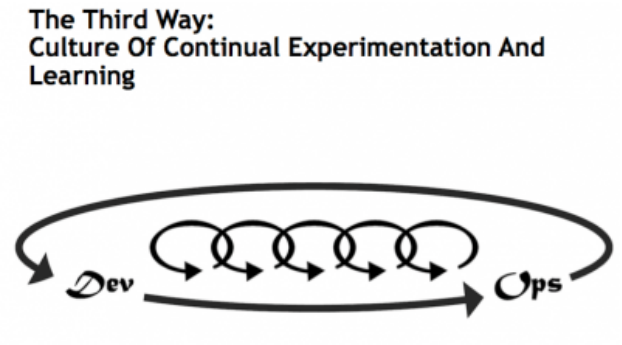
In 1999, Steven Spear and Kent Brown published an article in the Harvard Business Review called "Decoding the DNA of the Toyota Production System (TPS)". TPS is well know as the birthplace of Lean and it has been well established that DevOps inherits an abundance of Lean principles. In the article, Spear and Brown explained that TPS applied scientific method to everything that happened in the plant. One of my favorite quotes from the article is: "TPS is a community of scientists continuously experimenting".

Mike Rother, author of Toyota Kata, takes this idea a step further. Rother suggests that TPS leadership used PDSA cycles combined with a vision (i.e., a true-north) as a set of coaching tools to impose a sort of memory muscle through repetition (Kata). The term Kata is a Japanese word used to describe choreographed patterns of movements. Kata is used in the traditional Japanese art of Kabuki as well as in martial arts.

Rother suggests that TPS would align all of their activities as target conditions bound by a PDSA cycle always studying the results of each experiment before proceeding with the next action (the "A" in PDSA). They would ask the question in the "Study" step of the cycle – Did the experiment produce results that moved in the direction of the vision? These concepts of this "Third Way" become the tools for implementing DevOps in an organization.

Docker and the Third Way

In order for an organization to behave like a community of scientists, it has to have reliable lab equipment. In the classical sense, a scientists has a common set of equipment used while experimenting (test tubes, beakers, a microscope and of course safety goggles). Docker is a great adjunct to the "Third Way".



In the business of delivering software and services, speed to market is essential. It's not just how fast an organization can deliver services, it's how fast they can react and reproduce to a customer's validation of the delivered service.

In the previous sections , Docker is highlighted as a great solution for achieving exponential speed. For example, a large financial institution that is using Docker as a sort of "Container as a Service" vehicle for their internal customers. They have over 100 data scientists in their organization that have to constantly analyze large sets of data with disparate context. These data scientists have to match the right analysis tool to the right data. If they misjudge the fit, a lot of time is wasted or worse, sub-optimum results can be produced after a long running job.

Prior to implementing a Docker-based "Container as a Service" solution, it was extremely hard for these scientists to match an analysis tool to the data. Some analysis tool and data combinations perform well with a tool like Hadoop while others data sets are better suited for a tool like Spark while other sets work just fine with something like R. The point being there are a lot of tools out there for data analysis with new ones get added every other day.

This organization has created a sandbox environment of prebuilt container images (i.e., Docker images) that encapsulate all of the ingredients required to run the data analysis tool. The resulting solution is that any data scientist in this organization can instantiate a containerized set of data with a specific analysis tool (i.e., in a container) in minutes and can confirm or reject the experiment results in a two-order of magnitude less time.

Prior to implementing this Docker-based "Container as a Service" offering, the lead time for experimenting the data-to-tool process could take over two days of setup and request time to to deploy the test. Now they can run through a set of experiments testing multiple tools in a few hours.

Imagine the second order effects of this solution. Imagine that in this organization, a data scientist might actually be able to try out three or four different analysis tools before landing on a best-fit solution for the larger data analysis project. While prior to the "Container as a Service" offering, a data scientist might have just settled on the first "just good enough" solution.

This model can be used as a pervasive solution for any process in an organization that requires a similar selection process.

While the DevOps movement is a mindset at first, the major principles all point to business outcomes like faster innovation, higher quality and a feedback loop of continual learning. With the Three Ways of Devops creating a pragmatic model, new practitioners will have a higher rate of success. The Docker platform uniquely allows organizations to apply tools into their application environment to accelerate the rate of change, reduce friction and improve efficiencies.

www.docker.com