Continuous Delivery

PATTERNS

for Design & Deployment

A Whitepaper by

TABLE OF CONTENTS

Introduction 3

Quick Reference..... 4

What is Continuous Delivery? 5

Benefits of Continuous Delivery 6

Layers of Continuous Delivery 7

Adoption of Continuous Delivery 9

Common Patterns 10

Blue Green Deployments 11

Canary Release 12

Microservices, Immutable Servers & Containers 14

Dark Launching & Dark Loading 19

Feature Flags 20

Holographic Hosting 21

Summary 23

INTRODUCTION

Companies and application development teams are continuously speeding up the process in which they deliver applications to customers by adopting new technologies and practices. Companies taking advantage of the cloud are now exploring techniques and practices that not only allow for them to scale quickly, but also to test and release products to customers quickly. In particular, there has been a practice that has been growing over the past 10 years which is called Continuous Delivery (CD).

Companies that were born in the cloud, such as Netflix and Etsy, have been implementing CD practices in their organizations allowing them to make positive changes to their application in days instead of months. There are many techniques and CD patterns that have been developed over the years by some of these early adopters. This white paper was written to serve as a central reference point of CD patterns and practices so that other startups can utilize the patterns in their own operations.

QUICK REFERENCE

Continuous Delivery – A process which allows the deployment of incremental software changes at any given moment and with little to no risk.

Continuous Integration – A process which allows developers to check-in code frequently in a way that minimizes issues with integrating code.

Continuous Deployment – A process which uses a fully automated system to deploy code to production environments with the click of a button.

Blue/Green Deployment – Identical infrastructures where one is used for production and the other for testing, which can be swapped instantaneously.

Canary Release – Like blue/green deployment, but where users can be moved incrementally to a new version instead of all at once.

Microservices – A design pattern which takes a monolithic application and breaks the application apart into isolated segments allowing for easier maintenance and higher fault tolerance.

Immutable Servers - Servers which are not allowed to change after initial configuration.

Phoenix Server Architecture – The use of immutable servers as a template to quickly and often automatically replace servers that have crashed or become unusable for any reason.

Containers – Isolation of an application plus its dependencies, libraries, config files, etc., from the rest of the system allowing for easier deployment.

Dark Launching / Loading – The process of having a back-end shadow the current system, processing requests for load testing and yet not visible to the user.

Feature Flags – The process of wrapping new features with some sort of toggle tag which turns on or off the feature for the user.

Holographic Hosting - A build process which builds the code in a way that allows for different versions of an application to exist on the same URL, allowing for easy implementation of canary release, blue/green deployment, and other CD patterns and tactics.

CONTINUOUS DELIVERY

What is Continuous Delivery?

Let's first lay down a foundation by defining CD and some of the key terms surrounding this practice. CD was originally born out of the Agile movement and derived its name from the first principle written in the Agile Manifesto.

"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

Jez Humble, one of the more established thought leaders on this approach, defines Continuous Delivery as "a set of principles and practices to reduce the cost, time, and risk of delivering incremental software changes to users."

> Continuous Delivery is delivering code safely, quickly, and sustainably.
>
> *– Jez Humble*

Martin Fowler, another thought leader within the realm of CD, provides his own perspective and definition which says "Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time."

The idea is that an organization should have the capability to deliver small software changes on an incremental or continuous basis in order to not only deliver improved software quickly, but also in order to reduce the risk associated with introducing software changes.

---

http://agilemanifesto.org/principles.html

http://www.thoughtworks.com/insights/blog/case-continuous-delivery

http://martinfowler.com/bliki/ContinuousDelivery.html

Benefits of Continuous Delivery

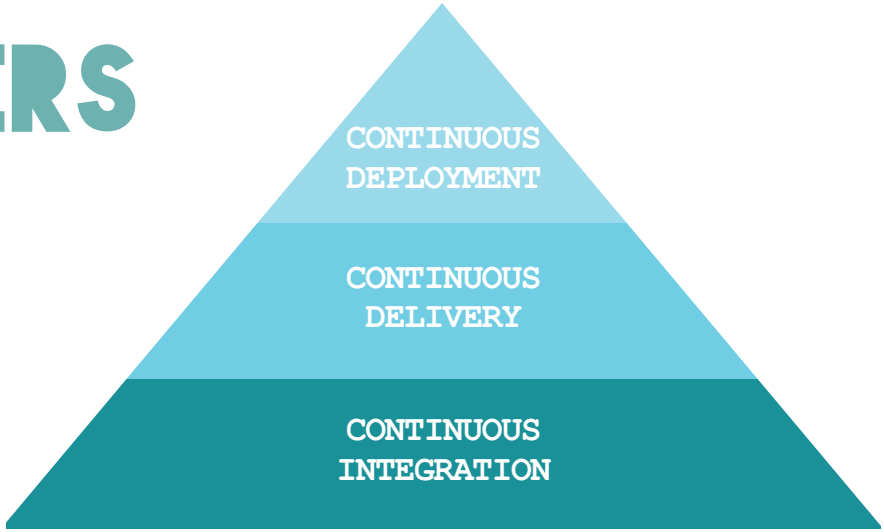
In most organizations, the idea of pushing new code to production is not generally thought of as a "low risk" event. In fact, deploying software changes is usually thought of as a strenuous and stressful event. Launching a change can result in a broken application and an influx of support requests or unhappy customers. The concept of deploying weekly or even daily in order to reduce risk may seem counterintuitive. However, if done correctly CD allows the organization to test small changes instead of deploying big batch code changes. The idea is to fail fast, and more importantly, fail small. Smaller code changes, often times, means smaller mistakes. Even if the code creates a major issue, a code change that is smaller makes it easier to find and identify the bug. Also, if the right systems are in place, it can be easier to roll back to previous versions of an application in the event that a new code deployment causes a major issue.

Automating both the deployment and rollback process is essential to a successful CD practice. Automation not only allows for on-demand deployment, but it also reduces the risk of human error. An easy way to start the automation process is to document every stage of the deployment process and then automate the steps that are constant and repeatable. Less can go wrong when steps are automated and human error is removed.

> Continuous Delivery aims to make releases boring, so we can deliver frequently and get fast feedback on what users care about.
>
> *–Thoughtworks*

Once processes are automated, CD allows for better quality control in the form of extensive testing. Automated tests can be performed, but more importantly the ability for your team or users to test the application before release can greatly improve quality. Whether your team practices Agile, Scrum, Lean methods, or any other industry practice, CD can become a tool that allows for a short cycled feedback loop.

LAYERS



Continuous Integration, Continuous Delivery, and Continuous Deployment: these terms all are similar in approach and philosophy, and yet have very different meanings. The best way to understand their differences is by visualizing them in layers.

Continuous Integration

Continuous Integration is a base layer or foundation to a continuous methodology. Continuous Integration is the practice where developers check-in code daily to a shared, master trunk which then results in an automated build. This allows for the seamless development of code from multiple developers. An automated test should be performed against the build to ensure functionality. As a result, Continuous Integration reduces the occurrence of two developers pushing conflicting code which would ultimately break the application. If the test comes back unsuccessful, the deployment team can easily track down and fix the issue.

---

http://www.thoughtworks.com/continuous-integration

Continuous Delivery

The next layer in continuous software development is Continuous Delivery. CD is the practice of ensuring that the organization's software is always in a state in which it can be deployed. This is achieved by implementing a successful continuous integration process, and then layering on top of this a well developed zero-downtime deployment strategy. Two of the more important practices in CD is to automate every step of the process and to build 1 click deployment scripts. This allows for organizations to quickly deploy application changes when needed. We have already defined and discussed the benefits of CD so let's move on to the final layer, Continuous Deployment.

Continuous Deployment

The final layer, Continuous Deployment, is a system that automatically deploys code when a developer commits to the master repository. As one can see, it is a prerequisite for the company to build both a working continuous integration system and a continuous delivery system. Once these two things are in place, then the final step of layering on the automated deployment system is achievable. Many companies may opt out of this last layer for one primary reason: quality control. With continuous deployment, the developer has the control on what is pushed to production. Code changes may occur daily or even multiple times a day. Conversely, with continuous delivery, the control can be left to the project manager as to what is pushed to final production. Code changes in a continuous delivery process generally occur more frequently, but release frequency is determined by the organization.

ADOPTION OF CD

Small Organization Friendly?

There are many case studies on the success of well-implemented CD methods. For example, it has been documented that Amazon is currently deploying new code to production every 11.6 seconds. However, it can be argued that CD is not simple for small organizations to achieve. In fact, it might even be the case that most CD success stories are overly romanticized and duplicating results is not plausible for smaller operations. In one study, it has been found that most small organizations don't effectively practice CD for a few key reasons. In a survey conducted by ThoughtWorks, it was found that only 15% of organizations have enough funds to implement CD with no prohibitions. Conversely, 82% of organizations believe that budget has the potential to be a barrier to implementing CD. Similarly, 88% of organizations were found to believe that a lack of technical knowledge or skill could be a prohibiting factor in implementing CD practices.

Even more telling is the fact that only 6% or organizations felt that they have enough time to implement a CD strategy. Conversely, 92% believed that a lack of time was a potential barrier to implementing CD. [1]

These statistics show us that small organizations which lack the budget and the ability to hire staff with the necessary skills are less likely to implement a CD strategy. Even if an organization has the resources to implement CD, they often won't because of time restraints.

While it is always impressive to read about how large companies such as Netflix, Etsy, Amazon.com, Nike and others can implement CD strategies, the reality is that it is less likely for smaller companies to successfully realize the benefits of CD methods.

http://info.thoughtworks.com/rs/thoughtworks2/images/Continuous%20Delivery%20_%20A%20Maturity%20Assessment%20ModelFINAL.pdf

COMMON PATTERNS

Blue/Green Deployment

There are varieties of CD patterns that are commonly adopted by large organizations. These concepts and practices are sometimes mixed and matched all in an effort to deliver high quality and well-received applications with zero-downtime. The two driving factors behind implementing these practices is to reduce risk or improve user experience.

FEATURES:

- Allows for "in the field" testing of the application in a production environment.
- Reduces risk associated with "big bang deployments."
- Allows for zero-downtime deployment.

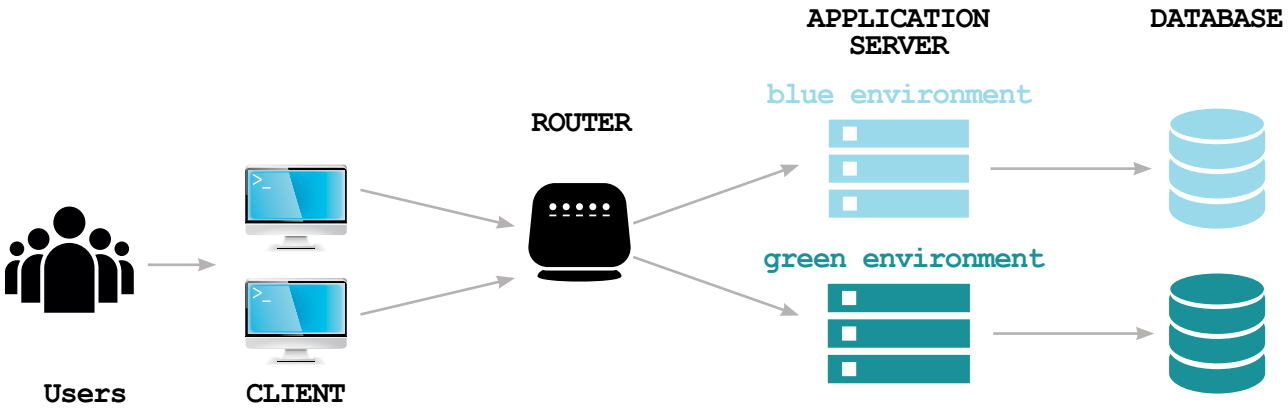
Blue/Green deployments is a pattern commonly used by organizations who practice CD. It is used to test the application before transferring users over to the latest version of the application, and also (if done right) to ensure zero-downtime deployments. The process involves creating another production environment which is the exact same as the current production environment. The current environment would be labeled "blue" and the exact replica would be labeled "green." The green environment would house the updated version of the application and would allow for proper application testing. Then, with some updates in the routing layer, users would be switched from the blue environment to the green environment. In the event that the application or environment is not working properly the organization can roll-back users to the old (blue) environment. Once users have been switched, and the application is successfully running then the old (blue) environment can be used for future blue/green deployments.

---

http://martinfowler.com/bliki/BlueGreenDeployment.html

http://www.thoughtworks.com/insights/blog/implementing-blue-green-deployments-aws

Blue/Green Process



Companies Using Blue/Green Deployment

Netflix is a popular use case of Blue/Green deployment. For branding purposes, they have changed it to Red/Black deployment, but the basic concepts are the same. They use custom built automation tools to help in managing their infrastructure, deployment, and rollbacks in the event that the new version is unsuccessful.

Localytics is an analytics and marketing platform for smartphone applications. One of their first steps in integrating Continuous Delivery in their organization was to adopt a blue/green deployment system in order to deploy new versions of their product.

Etsy is one of the more popular trail-blazers when it comes to cloud development and Continuous Delivery. In 2010 they revealed their Continuous Delivery and blue/green process which includes deploying to their staging environment nicknamed "Princess."

Canary Release

FEATURES:

- Allows for "in the field" testing in a cloud environment
- Reduces risk associated with "big bang deployments" through a gradual release
- Allows for easy A/B testing

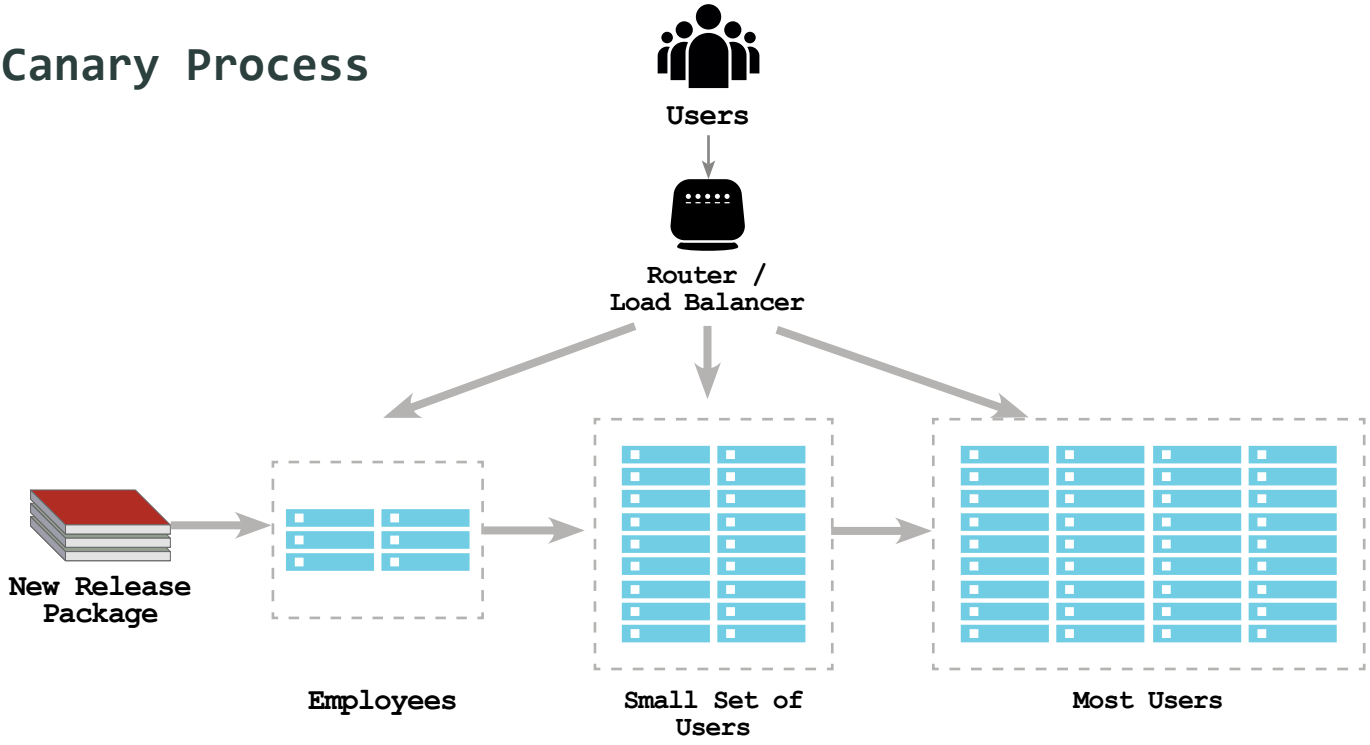
Canary Release is another popular CD practice and deployment pattern. The name originates from the old phrase "Canary in the Coal Mine," when miners would bring cages with canaries into mining tunnels to alert them should deadly gases exist serving as an early warning to exit. In Canary Release, should something start to go wrong with users on a new version, development teams realize this early on and can act quickly without having affected the entire user base. Canary Release is very similar to blue/green deployment, however, canary release allows for gradual, incremental release to users. Much like blue/green deployment, the organization would construct an identical production environment and deploy to that environment the latest version of an application. The difference between the two lies in the transfer of users from one environment to the next. Instead of transferring all of the users in one big batch, canary release allows for users to be transferred incrementally. The organization can reroute a few users to the new environment in order to test, or get feedback from users. Once the new application has been tested and the canary version is working appropriately the organization can route more users to the new version. Over the course of several transfers, all of the users will be using the newest version of the application. Common practice is for the initial launch of a new version to be released exclusively to internal staff for thorough testing. Once staff has tested the application the new version can then be slowly rolled out to users.

---

http://martinfowler.com/bliki/CanaryRelease.html

http://blog.heavybit.com/blog/2015/5/4/canary-releases-scaling-postgresql

Canary Process



Companies Using Canary Deployment

Facebook.com has built an extensive CD strategy and utilizes canary release for all feature changes. Most new Facebook features are built and then slowly rolled out to users over a few months. The first canary release is usually launched to 1% of their users and then increases over time.

Box employs 100+ engineers which are constantly churning out approximately 350 changes a week. In order to mitigate risk associated with deployment of their features, they implement canary release to better test their changes live and with current, real users.

Wix uses canary release among other CD practices in order to implement A/B testing of their new features. Their goal is to get quick feedback about their new features from real users in order to make better decisions on what features to implement in their production environment.

Microservices

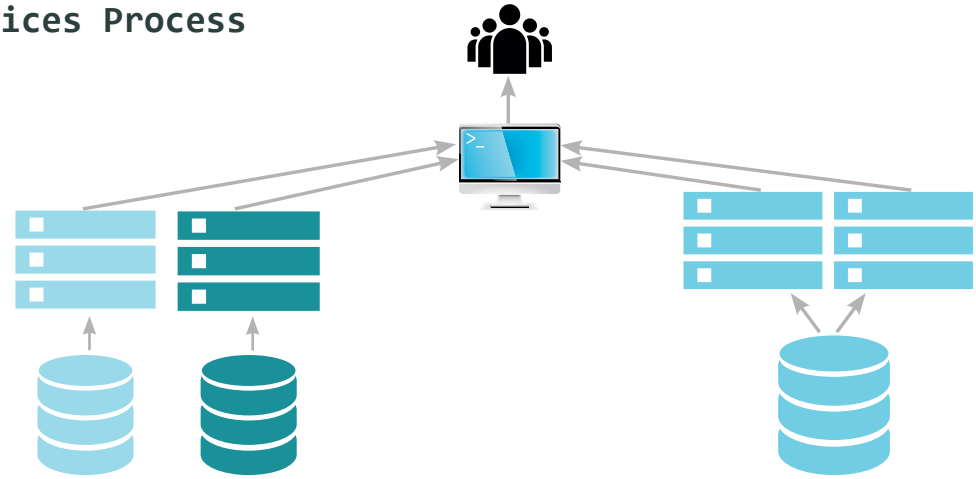
Next we will discuss a few concepts that are not exactly deployment patterns as much as they are infrastructure design patterns that are commonly used by teams implementing CD. Microservices, Immutable Servers, and Containers are grouped together due to the fact that they are all related in function. First, we will define each term. Then we will piece them together in order to show how they can be used in a CD strategy. .

Microservices is a relatively new concept that is still being defined, tested, and explored by many of the industry thought leaders in the world of cloud architecture. At a basic level, Microservices is a process of taking a monolithic application and architecture and then breaking it apart into segments that have the capability to operate (or fail) in solidarity. Each isolated service is then developed with its own unique parameters including using different languages and databases. Once each component is built, it is then wired together with the other services. When implementing organizations are primarily looking for components of the application that could stand alone. Netflix discusses an example of how they have implemented Microservices with their rating system. If, for some reason, the service that allows users to rate movie and show titles goes down, the rest of their application is not affected. Additionally, each "microservice" is managed by a small group comprised of both developers and operations professionals in order maximize efficiencies in both the development and release of each service to the greater whole.

> Conway's law says that the interface structure of a software system will reflect the social structure of the organization that produced it. So if you want to switch to a microservices architecture, you need to organize your staff into product teams and use DevOps methodology.
>
> *–Tony Mauro, NGINX*

http://www.infoq.com/news/2014/07/building-deploying-microservices

Microservices Process



Companies Using Microservices

Nike realized that their monolithic application was becoming much more difficult to manage because of how bloated their application had become. As a result, they moved to CD and implemented Microservices in order to help their team to get to production faster.

Karma discusses their approach to Microservices as "starting out with one big app" and then "splitting off pieces when it makes sense." As a result, they have been able to build their Microservices so that when something fails, it is less likely to affect their entire system.

Netflix not only adopted a Microservices design pattern, but also realized that their managing teams needed to reflect the infrastructure being implemented. As a result, Netflix divided their team into smaller units that managed each service.

Immutable Servers & Phoenix Servers

Immutable Severs and Phoenix servers are a match made in heaven. The central idea behind developing an immutable server architecture is to prevent configuration drift, and the goal of Phoenix Servers is to achieve the goal of a highly resilient architecture.

First, let's explore the concept of Configuration Drift. Configuration Drift, as Kief explains it, is the "phenomenon where running servers in an infrastructure become more and more different as time goes one, due to manual ad-hoc changes and updates, and general entropy." Often, teams will make changes to a server in order to fix a temporary problem without full understanding of the long-term effects. Also, it is common that changes are not documented appropriately causing future confusion as to the server's ever-changing configuration history.

A helpful solution is to prevent changes to the configuration of live servers (making them immutable), and along with that creating a well-documented organizational standard and template for the creation of future servers. This solution leads to uniform servers throughout the architecture. In addition to mitigating configuration drift, an immutable server architecture is beneficial in the fact that it makes it simple to create a resilient environment. If a server crashes, or goes down for any reason then replica servers can be easily and usually automatically spun up as replacements (similar to how a phoenix rises from its ashes). Netflix popularized this concept with the introduction of their Chaos Monkey which randomly destroys instances in order to test their application's resiliency.

---

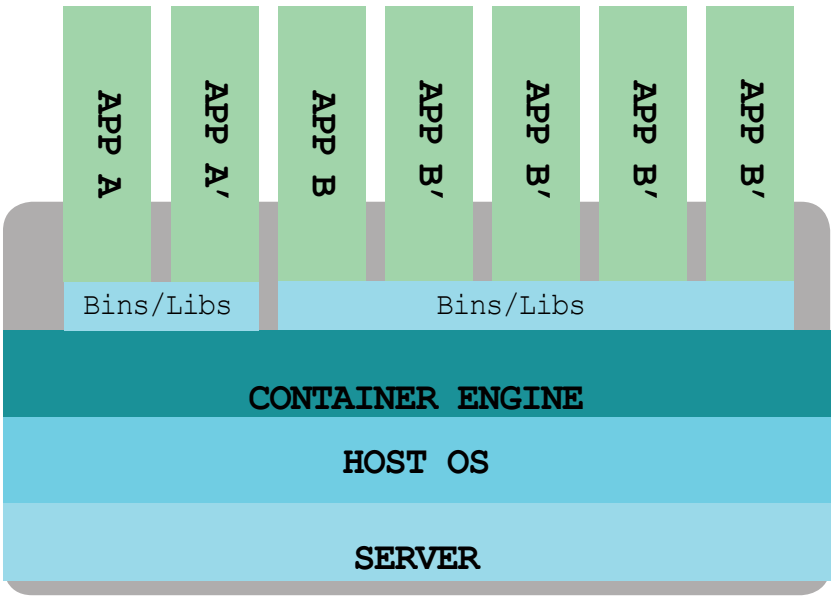
http://kief.com/configuration-drift.html

http://martinfowler.com/bliki/ImmutableServer.html

https://github.com/Netflix/SimianArmy

Containers

Container based technology, or containerization, is the idea of virtualization at the operating system level. Traditionally, virtualization happens with the help of virtual machines controlled by a hypervisor each containing their own operating systems. Applications would be run within the virtual machines allowing the application to be abstracted away from the physical hardware. Containers takes this a step further by "containerizing" the application plus its dependencies, libraries, config files, and anything else needed to run. This is achieved because the containers use features of the operating system's kernel to run completely isolated from the rest of the system. Running multiple containers on a machine allow for fewer resources to be consumed than running VMs, allowing for easier deployment of applications from one place to the next.



Design Strategy

Now that each term has been defined, let's piece them together into a CD strategy. When dividing up a monolithic application into microservices it is a requirement to spin up multiple virtual environments to handle, and yet isolate, the various services. Organizations have turned to containers in order to provide adequate isolation and easy deployment and management. Additionally, an immutable server, or even immutable containers could be applied allowing for easier creation and management of the infrastructure. Also, the containers would be more easily replaced in the event that a server becomes inoperable. Ultimately, creating separated microservices with immutable servers/containers should allow for easier deployment of new features and a much more fault tolerant system. Features can be delivered via an isolated microservice, isolating it from the rest of the system and allowing for a more "continuous" pipeline.

FEATURES:

- Fault tolerant system that could potentially be easier to maintain
- Isolation of services that allows for new features to be implemented without effecting the rest of the system

---

http://www.cio.com/article/2924995/enterprise-software/what-are-containers-and-why-do-you-need-them.html

http://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualization

http://ariya.ofilabs.com/2014/12/docker-and-phoenix-how-to-make-your-continuous-integration-more-awesome.html

Dark Launching & Dark Loading

Dark launching was first introduced and made popular by Facebook back in 2008 when they are working on the development of the Facebook chat feature. Essentially, Dark Launching is the practice of developing the needed infrastructure and server-side resources for the backend of a new feature. Once the backend is complete, the organization can then launch the backend resources to production (unknowingly to the users; hence "dark launching.") testing the infrastructure for load and functionality. Once the testing has been done appropriately, the UI elements of the new feature can then be deployed and made accessible to the users. For example, Facebook wanted to simulate the impact of several million users hitting the servers for their new feature so they allowed real users to soak test the machines without a the UI actually being accessible to the users. As a result, they were able to discover impairing bugs, make needed changes, and then "turn on" the UI when they were ready.

Dark Loading is another name for Dark Launching but may be a better representation of the process. In particular, the term Dark Loading has been used specifically for the process of load testing new database changes "in the dark." In the event that the database needs major changes, or even switching databases systems entirely, Dark Loading can make the process seamless without causing any downtime for the user. Once the database has been load tested, changes to the rest of the application can be made in order to start using the changed or new database.

FEATURES:

- Proactively load test new features without the user even knowing about it
- Ability to test and then update or completely change technologies
- Users experience new features that are pre-tested for performance

https://en.wikipedia.org/wiki/Soak_testing

http://tech.finn.no/2013/06/20/dark-launching-and-feature-toggles/

https://labs.spotify.com/2015/06/23/user-database-switch/

Feature Flags

One way to incrementally make available new UI features is through Feature Flags. Feature Flags (also known as config flags, feature toggles, flippers, and probably a few other similar names) is written into the code itself by wrapping the new feature with some sort of toggle tag. Feature toggles are used for two primary reasons. The first is for testing new features with users that fit certain requirements which would be set by the logic placed in the toggle. The second reason an organization might use this method is in the event that a new feature will take an extraordinarily long time to develop. The new feature can be "turned off" to users until the feature has been completed.

Companies Using Dark Launching, Dark Loading, and Feature Flags

Hootsuite has been using "feature flagging" for years and has found that this method gives them a great deal of control over their application. They simply set conditions at runtime for a certain feature and as a result they are able to control what specified users can or can't see.

Spotify made the decision to switch from a Postgres database to a Cassandra database. In writing about their experience ,they felt that the switch was analogous to "changing engines on a running car." However, with the use of Dark Loading they were able to make the deployment "one of the most boring deploys" they have every done.

Flickr was one of the first companies to utilize "flags and flippers" in their development process. This was done in order to work on new features that take months to complete. While they admit that it takes diligent maintenance to achieve positive results, they have been able to speed up development with fewer bugs.

Holographic Hosting

At Cycligent, we have recently invented and introduced an emerging technology called Holographic Hosting which helps organizations implement CD methods. Holographic Hosting is the core technology utilized in our cloud platform called Cycligent Cloud. Holographic Hosting and Cycligent Cloud were both developed by their sister company Improvement Interactive in order to simplify deployment and CD methods within their organization. As a result, Cycligent Cloud was released and is now available for developers to utilize in their own organizations.

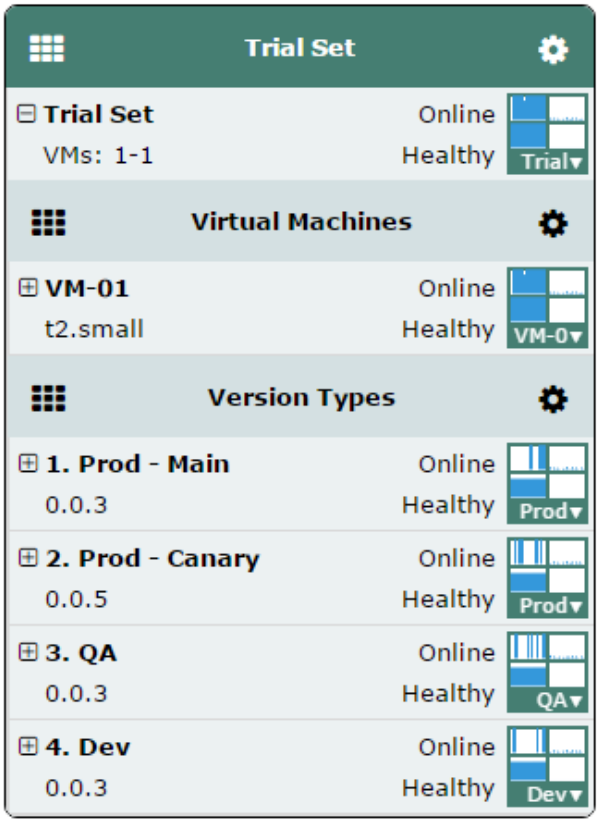
FEATURES:

- Multiple simultaneous versions on same URL allowing for Canary Release, Blue/Green Deployment or other successful CD patterns
- No need for code modification or infrastructure configuration
- Allows both small and large organizations to utilize and practice CD

Holographic Hosting utilizes a build process which builds the client side code and server side code in a way that allows for different versions of an application to exist on the same URL. A Holographic Router directs traffic to the various static client side resources and, as a result, the proper server side resources are accessed. Simply put, once multiple versions of an app are deployed the web router routes users to the various application versions within the cloud environment.

The process of Holographic Hosting allows for organizations to deploy multiple versions of an application simultaneously while still utilizing the same URL or URL path. Organizations are able to deploy canary, test, and dev environments simultaneously allowing for simple testing and rollouts. More importantly, users can have any number of environments and can name them to the specifications of the application.

Holographic Dashboard



In addition to simple multiple versioning, the Holographic Hosting technology allows for simple deployment, management, increased performance, and a more fault tolerant infrastructure. When an application is deployed, the source code is infused with additional functionality such as long workers, load balancing, auto-scaling capabilities, message buses, and other performance enhancing capabilities. Developers don't need to learn how to code for, or even modify their existing application's code in order to deploy multiple versions of their application with added functionality.

SUMMARY

We have listed several CD patterns that, if used effectively, could alter the way an organization works and delivers applications to their users. We predict that these best practices will evolve and new methods will be discovered furthering CD as an imperative methodology for organizations. In particular, we believe new methods such as Holographic Hosting will pave the way for smaller organizations with limited resources more easily adopt CD and, as a result, deliver better applications faster.

In the end, we hope that his white paper serves as a resource for organizations looking for ways to better their application development process.

Find us on:

www.cycligent.com

Call us: (866) 865-9292

support@cycligent.com