GoCD
BRAND STYLEGUIDE

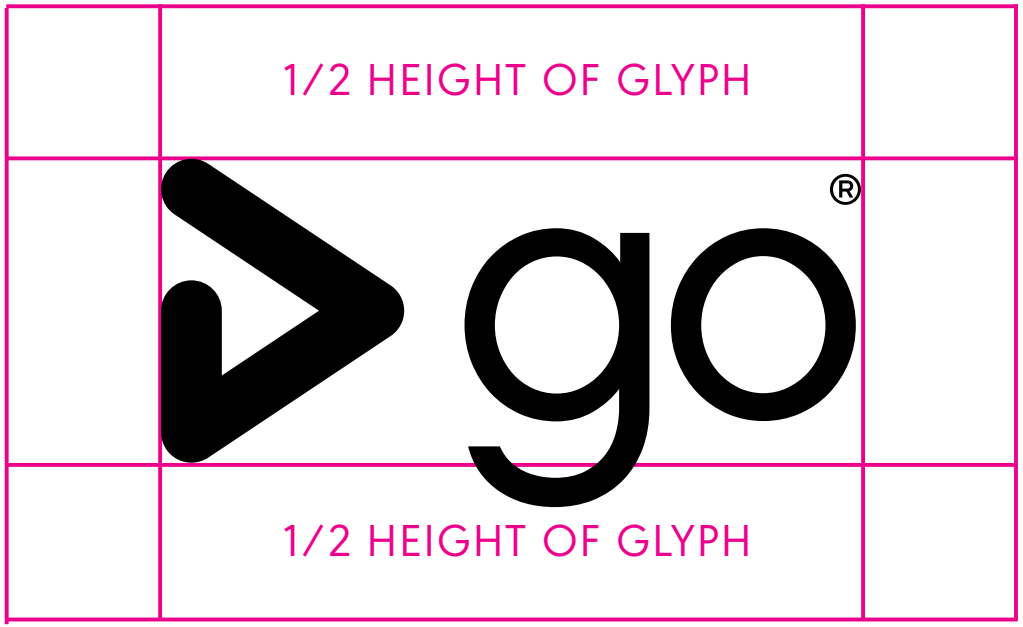
go

LOGOS

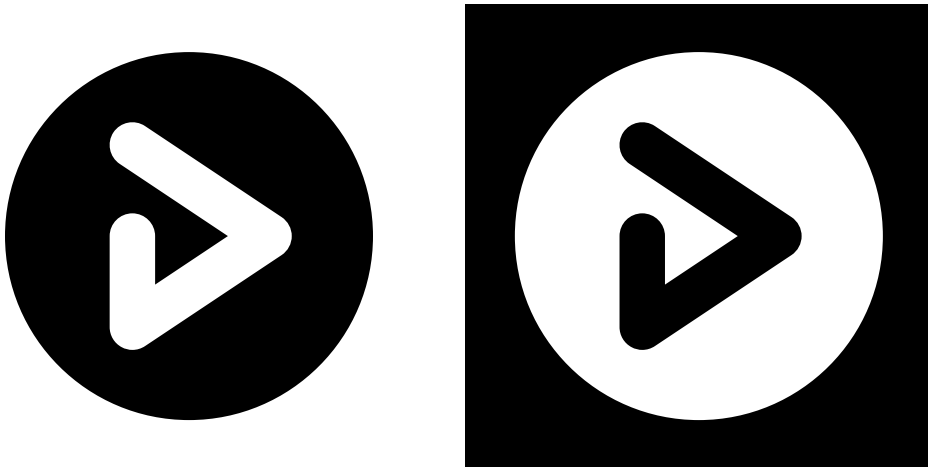
MAIN LOGO

MAIN LOGO SHOULD BE IN BLACK AND WHITE ONLY. THE WORDMARK SHOULD NOT BE DISPLAYED SEPARATELY FROM THE TRIANGLE GLYPH.

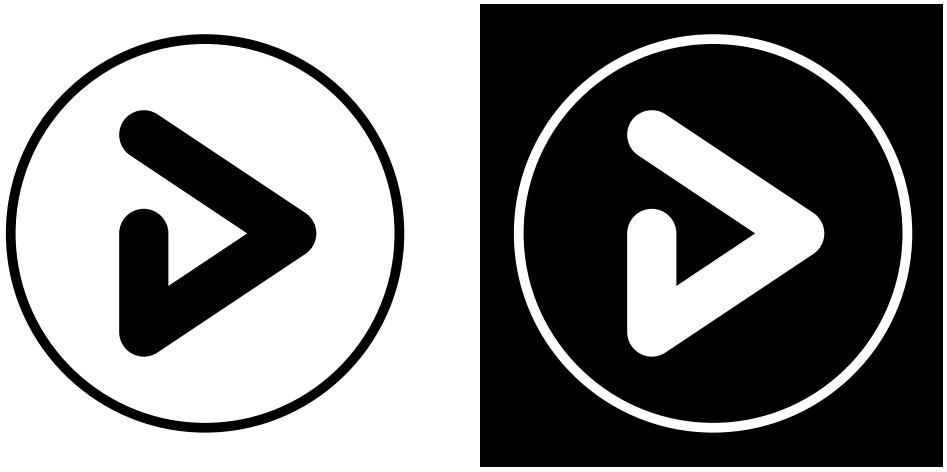
CLEAR SPACE



SUPPORTING
BRAND MARK



SUPPORTING
BRAND MARK
ALTERNATE



BRAND MARKS SHOULD BE IN BLACK AND WHITE ONLY

When typed out, ‘GoCD’ should always be displayed as:

Uppercase	**G**
Lowercase	**o**
Uppercase	**C**
Uppercase	**D**

FONT FAMILIES

URW Geometric

The use of URW Geometric is deprecated for use in communcation channels. Poppins Latin will be used as the primary typeface moving forward.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

32pt

Poppins Latin

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

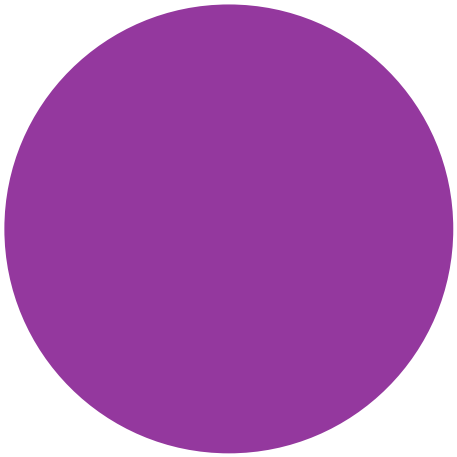
30pt

Poppins Latin source: https://github.com/itfoundry/Poppins

COLOR

BRAND COLORS

The GoCD color palette conveys a tone of thoughtful engagement, with a distinct edge. The primary brand colors are used for photo overlays and help to distinguish the brand in the majority of settings, while the supporting colors can be brought in to create variation and a fresh feel whenever needed. All tones play well against black, which is key to the treatment of brand photography.



#94399E
R148 / G56 / B158
C49 / M91 / Y0 / K0

#25282D
R38 / G41 / B46
C75 / M67 / Y59 / K64

SECONDARY BRAND COLORS

The GoCD color palette conveys a tone of thoughtful engagement, with a distinct edge. The primary brand colors are used for photo overlays and help to distinguish the brand in the majority of settings, while the supporting colors can be brought in to create variation and a fresh feel whenever needed. All tones play well against black, which is key to the treatment of brand photography.

KEY BRAND ART

IMAGES

USE CASES

WEB BANNER

EMAIL FOOTER

ONE SHEETS

WHY GoCD OVER JENKINS

APRIL 25, 2017 / DAVID RICE, MANAGING DIRECTOR THOUGHTWORKS PRODUCTS

We often get asked by potential GoCD users "Why GoCD?" or "Why GoCD over Jenkins?". Or even more often: "We use Jenkins. It's not great but it's set up, we are familiar with it and it's good enough. Why change?"

This blog post sets out to answer those questions. We are not going to tell you GoCD is perfect or even that it does all the same things as Jenkins. It wouldn't be true. Nor will we provide an exhaustive feature comparison. There is no doubt that this post will skip some Jenkins features that are loved by its users. We will share where GoCD shines when compared to Jenkins. We'll compare philosophy, getting started, continuous delivery, and plugins. **(Spoiler alert: Use GoCD if you want to do continuous delivery).**

PHILOSOPHY
GoCD is a best-of-breed tool for Continuous Delivery (CD). Full stop. GoCD aims to support the most common CD scenarios out of the box without any plugin installation. GoCD's model maps to the core concepts of CD deployment pipelines.

Jenkins is a general purpose automation tool. It is not a best-of-breed CI or CD tool. (Yes, Jenkins is a pretty good CI tool.) Pretty much any use case requires installation of a particular set of plugins. Near everything is pluggable and there are over 1000 community plugins. Plugins are fundamental to Jenkins.

Bottom Line: *If you want a single-purposed, focused tool use GoCD. If you aren't laser-focused on CD and prefer maximum tinker-ability, Jenkins might be for you.*

CONTINUOUS DELIVERY
CD is a priority for GoCD and for ThoughtWorks. GoCD exists to help its users implement CD. ThoughtWorks is a thought leader in CD and our people regularly write and speak about CD to the wider tech community.

GoCD encourages that there be only one way to implement the fundamental CD patterns. When you search for help on how to implement the various deployment pipeline patterns you will generally find a single, well-known, well-tested answer. These searches will turn up results for Jenkins as well but the results might show obsolete plugins or many solutions, without any definitive guidance.

If you are are new to CD and deployment pipelines, GoCD's getting started tutorial is a big help. It will teach you the key deployment pipeline patterns while you learn how GoCD works.

Jenkins 2.0 placed a big emphasis on its Pipeline feature. But there is little evidence that Jenkins has made CD its top priority. CD is still implemented by the installation of a variety of plugins. Many common CD patterns (build an artifact only once, full traceability up and down stream, and more) are either impossible to implement or can only be cobbled together with fragile combinations of plugins. The official Pipeline feature documentation does not reference CD or deployment pipeline concepts. You might struggle when implementing many of the core deployment pipeline patterns.

Bottom Line: *If you are serious about implementing CD, beginner or expert, GoCD is the right choice. It's easy to model deployment pipelines and the GoCD community truly cares about CD*

GETTING STARTED
GoCD has an easy to follow, definitive tutorial for getting started. This tutorial teaches the basic concepts of CD as well as the GoCD domain model. Someone new to GoCD will be up and running in 15 minutes with this tutorial. It teaches the app, the core concepts, and the preferred style of building deployment pipelines.

A large part of installing Jenkins is installing the right mix of plugins for your use case. It can be challenging to know what plugins you will need up front. Jenkins 2.0 has improved this experience a bit by providing a wizard to guide you through plugin installation. This wizard includes a default set of common plugins and prompting you to setup your first build. But this won't prevent you from needing an in depth understanding of the plugins to create the correct plugin mix for your use case.

Both products can present a tough getting started experience for someone who just wants to dive right in. GoCD's model can be difficult to grok if you are coming to it from a pure CI tool. Jenkins has hurdles around configuring required tools, uninitutive SCM configuration, and installing additional plugins.

Bottom Line: *If you are wanting to build deployment pipelines for CD, GoCD's tutorial will quickly get you to a very productive place. Both tools can be frustrating to use if you are new to them and you don't RTFM.*

PLUGINS
GoCD's plugin philosophy is that plugins should extend its ecosystem and not alter its core functionality. Nor does the team aim to use plugins to make GoCD ultimately flexible. The aim is to keep it as easy-as-possible to implement CD on GoCD but in as many environments as possible.

GoCD has a handful of extension points: SCM, task, notifications, authentication, authorization, configuration, elastic agents. The team designs endpoints such that all plugins will be interoperable. It doesn't matter whether the core team or the community has built the plugin.

Jenkins has a thriving plugin ecosystem. They should be proud. Jenkins is seemingly infinitely configurable, hackable, and extendable via plugins.

It's our opinion requiring just the right set of plugins to get Jenkins to support your use case is problematic. Upgrades are challenging. Support and maintenance are unclear. Not all plugins play well together. There's almost never one way to do something. Multiple ways of doing things is not bad in and of itself but the follow-on from that is that it's hard to find answers to questions around how to implement builds & pipelines. And we think that's a huge problem.

Bottom Line: *GoCD provides its core value out of the box. Maybe you will add a few integration plugins to make GoCD fit better in your environment. Jenkins will require many plugins to deliver value. You will need to understand the plugins, how they interoperate, and how to upgrade them. GoCD will feel more stable. Jenkins will feel more hackable. You will need to decide which is a better match to your needs and philosophy.*

THE FINAL WORD

If you are doing or want to do CD you should be using GoCD. GoCD will feel good to both beginners and those highly experienced in building deployment pipelines. If you want to utilize a vast plugin catalog to automate "all the things" then Jenkins might be a better fit for you.

The GoCD community continues to grow, is passionate about CD and is where you should be if you are serious about CD. We are happy to discuss individual needs in our user group so get in touch. ➤

WHY GoCD OVER JENKINS
gocd.org/blog @goforcd

WHAT'S NEW IN GoCD

OCTOBER 16, 2018 / ARAVIND S.V, PRODUCT MANAGER AT GOCD

The continuous delivery space is moving at the speed of light. Containers, infrastructure, and cloud, are all moving incredibly fast, and so is GoCD. In the past few months we have released some exciting new features.

GoCD IS CLOUD NATIVE
We've built two distinct sets of features to make GoCD cloud native. First, we've made it easier to operate GoCD on cloud infrastructure providers. Second, we have provided better support for container-based workflows.

OPERATING ON CLOUD INFRASTRUCTURE
People operating CI and CD servers on cloud providers want to perform as little administration of their build infrastructure as possible. Getting started should be easy. Build resources, or agents in GoCD parlance, should scale up and down transparently, as needed. GoCD addresses this with a suite of "elastic agent" plugins that provide scalable build infrastructure on Amazon ECS, Docker, Docker Swarm, OpenStack, and of course, Kubernetes.

Veteran users of GoCD likely have spent a fair bit of time managing their agent grids via scripts, puppet, chef and similar. An elastic agent plugin eliminates this work as GoCD installs, starts, and stops agents as needed on whatever cloud provider you point it at.
On Kubernetes, we've provided a helm chart to make it near trivial to install and operate GoCD in its entirety on your Kubernetes cluster.

There are two elements to our improved container based workflows:

1. Teams who are fully immersed in Docker want pretty much everything to be Docker-based. If you are utilizing our Docker, ECS Docker or Kubernetes support then all your build activity will now be Docker based. It just works.
2. GoCD now supports Docker images as native GoCD artifacts. GoCD abides by "build once and only once" and provides full traceability up and downstream for artifacts. You can now specify a Docker image repository as a GoCD artifact repository. This allows Docker images to be a part of GoCD's native artifact tracing, pushing, and fetching.

ANALYTICS PLUGIN FOR ENTERPRISE USERS
We have added a CD analytics plugin to our GoCD enterprise offering. The analytics plugin is intended to provide visualizations and actionable metrics to help you to optimize both value stream and the underlying build resources powering your pipeline.

One of the highlights, is when you're looking at your value stream, you can pick any two points and take a quick look at throughput or cycle time. This helps you answer questions like "How often do we deploy to production" and "How long will this take to reach the customer" more accurately. We have free trial available.

PIPELINES AS CODE
In case you missed it, another cool feature we have improved over the past year is "Pipelines as code". With this feature, you can store your pipeline configuration as YAML or JSON files in your own repository, so that you can modify, control and version it externally.

We also exposed this ability as a plugin endpoint so anyone can write a plugin for a config repository, to store the configuration in any manner you choose. For more in depth reading on GoCD's pipeline as code feature, please visit our documentation for more details.

Another supported use case is to view your pipeline trends and identify builds that are slowing down. GoCD allows you to drill down to different levels of your pipeline to find root causes, e.g. slow tests or a lack of resources.

The GoCD enterprise offering provides you add-on software and support from the core team to enable GoCD for larger enterprise environments.

IMPROVED THE MAIN DASHBOARD
We have also made a couple of dashboard improvements. First, we improved performance for big organizations with hundreds or thousands of pipelines. Second, we have added personalization to the dashboard. You can now filter the dashboard to show only specific pipelines and pipeline groups. And you can save those settings as custom dashboard tabs for future reference.

Brief caption goes here

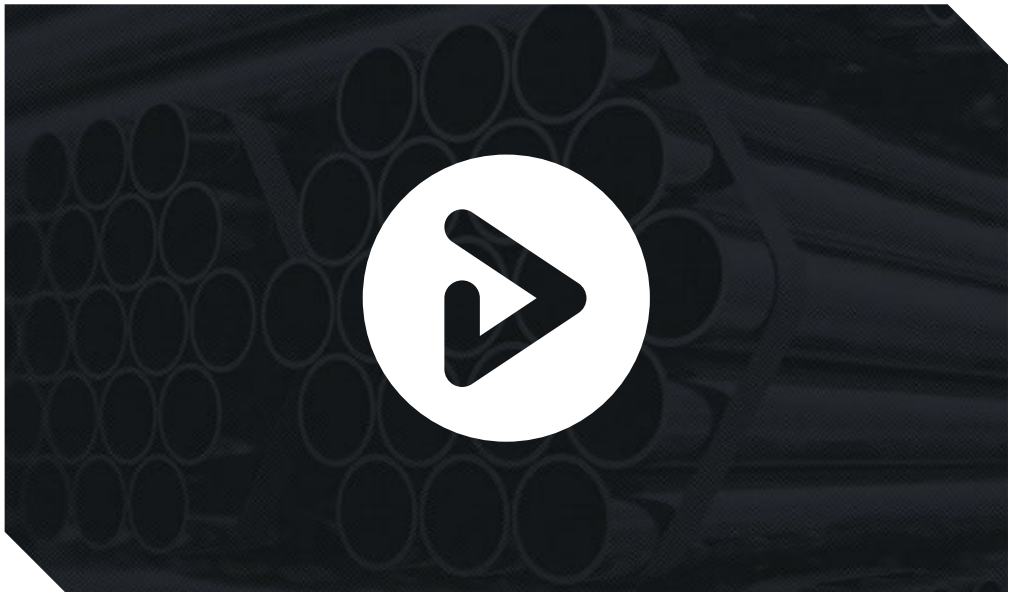
Brief caption goes here

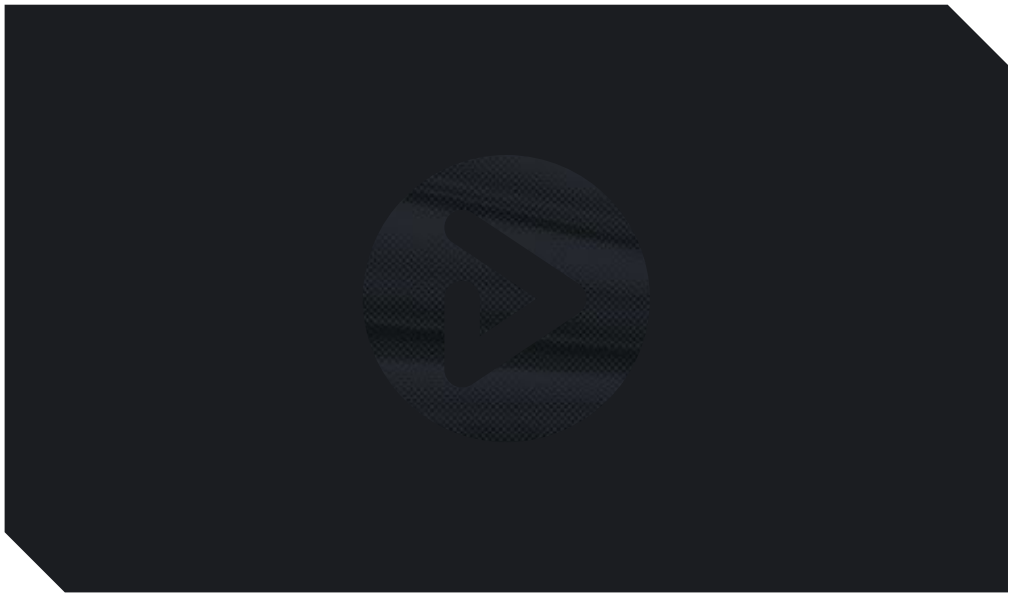
WHAT'S NEXT

Moving forward, we're going to be focusing on user experience for a few months. To start, we are going to make it much easier to utilize pipelines as code. This will include making it easier to add new pipelines as well as get feedback on whether your YAML or JSON specifies a valid pipeline. Beyond that we are likely to look at improving how we store secrets, including integration with popular 3rd party tools. If you are interested in learning more, subscribe to the GoCD Release Bulletin. ➤

WHAT'S NEW IN GoCD
gocd.org/blog @goforcd

SUPPORTING
BRAND MARK





PRINTED BANNER

POSTCARD


go
Apply this sticker to your webcam to guard against webcam spying
• Prevents webcam spying
• Sticks to any surface
• No sticky residue!
• Microfiber front for screen cleaning
gocd.org
@goforcd
Free & Open Source CI/CD Server
Easily model and visualize complex workflows with GoCD.

E-BOOK

CONTINUOUS DELIVERY 101

VOLUME 1

SUZIE PRINCE
ARAVIND S.V.
DAVID RICE

go

GoCD is an on premises, open source, continuous delivery tool with comprehensive pipeline modeling and value stream map to help you get better visibility into and control of your teams' deployments.

gocd.org

Contents

PART 1: THE BASICS

In this section, we'll look at what is 'DevOps' and how it has improved our way of working. We'll introduce key practices that aid a DevOps culture - Continuous Integration, Continuous Delivery and Continuous Deployment, and highlight the differences between them.

Continuous Integration (CI)

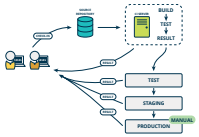
Continuous integration is a practice that puts the integration phase earlier in the development cycle so that building, testing and integrating code happens on a more regular basis.

CI means that a developer who writes code on his laptop at home, and another dev who codes on her desktop in the office, can both write software for the same product separately, and integrate their changes together in a place called the "source repository". They can then build the combined software from the bits they each wrote and test that it works the way they expect.

Developers generally use a tool called the "CI server" to do the building and the integration for them. CI requires that developers have self-testing code. This is code that tests itself to ensure that it is working as expected, and these tests are often called unit tests. When all the unit tests pass after the code is integrated, developers will get a green build. This indicates that they have verified that their changes are successfully integrated together, and the code is working as expected by the tests. However, while the integrated code is successfully working together, it not yet ready for production because it has not been tested and verified as working in production-like environments.

Continuous Delivery

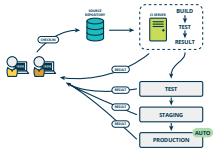
Continuous delivery means that each time the dev team makes changes to the code, integrates and builds the code, they also automatically test this code on environments that are very similar to production. We call this progression of deploying to - and testing on - different environments a "deployment pipeline". Often the deployment pipeline has a development environment, a test environment, and a staging environment, but these stages vary by the team, product and organization.



In each different environment, the code is tested differently. This gives more and more confidence to team that the code will work on the production environment when the code is deployed there. Crucially, the code is only promoted to (tested on) the next environment in the deployment pipeline if it passes the tests of the previous environment. This way the dev team can get new feedback from the tests in each environment and, if there is a failure, they can understand more easily where the issue might be and get it fixed before the code goes to the production environment.

Continuous Deployment

This is a practice where every change that developers make, assuming it passes all the test stages, automatically goes to production. You need to be practicing continuous delivery before you can achieve continuous deployment. Some may want to decide whether to do continuous delivery (versus continuous deployment) from the start, but that decision really can't be made until you're doing CD.

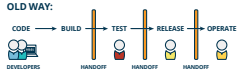


DevOps

The word "DevOps" comes from the combination of the words "development" and "operations". DevOps is a culture that promotes teams developing and operating software working together. Specifically, DevOps refers to cross-functional communication and collaboration of all roles, including things like security and compliance, during the software delivery and deployment process.The goal is the ability to release better quality software more quickly and more reliably.

Common traits of organizations who have a so-called DevOps culture are: autonomous poly-skilled teams, high levels of test and release automation and common goals between the poly-skilled members.

OLD WAY:



NEW WAY:

DevOps culture is commonly associated with continuous delivery because they both aim to increase collaboration between development and operations teams, and both use automatic processes to build, test and release software more quickly, frequently and reliably. All things that people like us want.

PART 2: ARE YOU READY FOR CONTINUOUS DELIVERY?

In this section, we will explore some core development practices that are prerequisites for continuous delivery. We'll present questions you need to answer honestly about your own people, teams, and organization to determine your readiness for continuous delivery.

Do you put everything into version control?

A foundation of CD is the ability to put a specific version of your application into a given environment at any point in time.

- Putting everything needed to make your application into a version control system
- Any time you change anything, push the changes to version control.
- Write an automated script that, given a version, checks out everything from version control and assembles your application

CD is impossible when software teams (or the people on a single team) work in isolation from each other. When development work happens in isolation, our preferred mechanism for doing this is called "trunk-based development":

- Everyone regularly pulls others' changes from version control
- Everyone regularly pushes their changes to version control
- Everyone works in the same place in version control typically called "trunk" or "master"

Do your developers practice Continuous Integration?

For CD to be successful, the entire organization must trust that your software is high quality and always in a working state. In terms of development team practices, CI is the fundamental building block to achieve this level of trust.

- Developers check code into trunk/master multiple times each day
- Developers maintain a suite of unit tests that verify the code works - before merge, locally, and post merge - on an integration machine or CI server

The end result is a development team that has high trust that the code in trunk/master actually works. This will leave the development team more willing to push code to testers, or even production, more regularly.

Do you automate relentlessly?

To practice CD, the entire team needs to get into the mindset of relentless automation of nearly everything.

Some components and aspects of your process that need automation:

- Tests at different levels, such as unit, integration, UI, regression,security and performance
- Database schema creation, data migration and rollback
- Installer creation and signing (if you have them)
- Generation of documentation for every release
- Last-mile deployment of your application to any environment
- Provisioning of infrastructure all the way from test environments to production
- Provisioning of developer workspaces

Relentless automation might seem daunting. The best approach is to figure out the manual processes you are already using, and then make a plan to gradually automate them. As you begin to achieve small successes, you will want to automate more and more.