**CI/CD چیست؟**

در حوزه دوآپس CI/CD دو سرواژه‌ای هستند به کرات استفاده می‌شوند و در این مقاله قصد داریم تا به بررسی تفاوت‌های آن‌ها بپردازیم  **C**ontinuous **I**ntegration یا به اختصار **CI** به طور خلاصه به پروسه‌ای اشاره دارد که از آن طریق فیچرهای جدید به صورت خودکار با ریپازیتوری اصلی ادغام می‌شوند اما **CD**  هم مخفف واژگان **C**ontinuous **D**elivery است و هم به **C**ontinuous **D**eployment اشاره دارد به طوری که اصطلاح اول به فرآیندی اشاره می‌کند که از آن طریق نرم‌افزار دائماً آماده دیپلوی است اما اصطلاح دوم سازوکاری است که به صورت خودکار کدهای آماده را روی سرور/سرورهای اصلی منتشر می‌کند (به طور کلی، این مفاهیم ارتباط تنگاتنگی با فرهنگ اجایل دارند که در آن تمرکز روی بهبود مستمر و انتشار قابلیت‌های جدید در بازه‌های زمانی بسیار کوتاه است.)

**Continuous Integration چیست؟**

[Continuous Integration](https://sokanacademy.com/glossary/4633/continuous-integration/%D8%A7%D8%AF%D8%BA%D8%A7%D9%85-%D9%85%D8%AF%D8%A7%D9%88%D9%85) به تیم‌های مهندسی اجایل و دوآپس کمک می‌کند تا تغییرات اِعمال‌شده برر روی سورس‌کد روی تک‌تک سیستم‌های دولوپرها دائماً با سورس‌کد (ریپازیتوری) اصلی ادغام شوند که این کار ابتدا به ساکن با پیاده‌سازی یکسری تست به منظور اطمینان حاصل کردن از عدم ناسازگاری مابین کامپوننت‌ها/کدهای جدید با کامپوننت‌ها/کدهای قدیمی صورت می‌گیرد و نیاز به توضیح نیست که این استراتژی اساساً بروز کانفیلیکت (تداخل) مابین فیچرهای جدیدی که توسط برنامه‌نویسان مختلف افزوده می‌شوند را به حداقل می‌رساند و این تضمین را می‌دهد که ایشان همواره به آخرین نسخه از تغییرات دسترسی دارند (در همین راستا، توصیه می‌کنیم به مقاله [معرفی برخی از ابزارهای اپن‌سورس Continuous Integration](https://sokanacademy.com/blog/9968/%D9%85%D8%B9%D8%B1%D9%81%DB%8C-%D8%A8%D8%B1%D8%AE%DB%8C-%D8%A7%D8%B2-%D8%A7%D8%A8%D8%B2%D8%A7%D8%B1%D9%87%D8%A7%DB%8C-%D8%A7%D9%BE%D9%86%E2%80%8C%D8%B3%D9%88%D8%B1%D8%B3-%D8%A7%D8%AF%D8%BA%D8%A7%D9%85-%D9%85%D8%AF%D8%A7%D9%88%D9%85) مراجعه نمایید.)

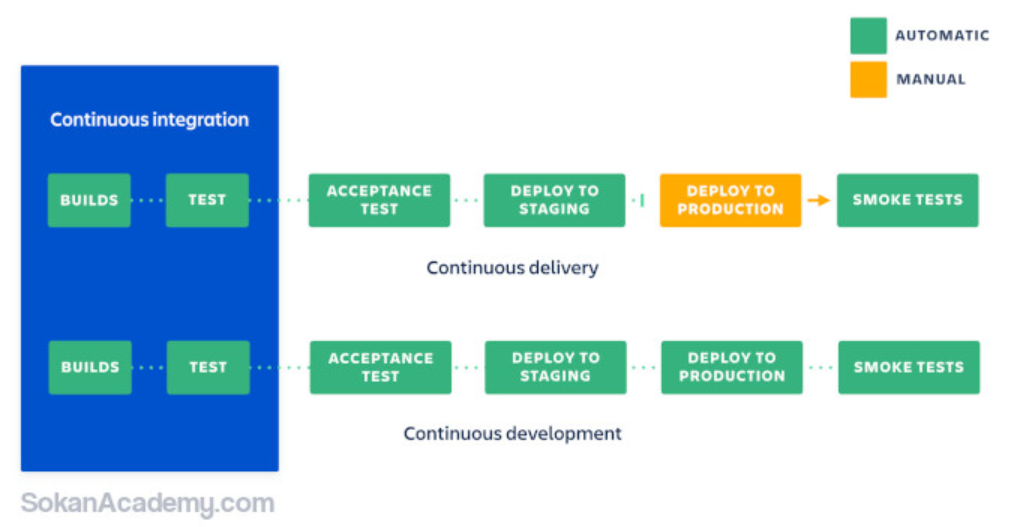
**Continuous Delivery چیست؟**

[Continuous Delivery](https://sokanacademy.com/glossary/10003/continuous-delivery/%D8%AA%D8%AD%D9%88%DB%8C%D9%84-%D9%85%D8%AF%D8%A7%D9%88%D9%85)  که گام بعدی پس از مرحله فوق است این تضمین را ایجاد می‌کند که تیم‌های توسعه نرم‌افزار می‌توانند با خیال راحت هر موقع که مشتری تمایل داشته باشد کدهای جدید را روی سرور دیپلوی کنند چرا که قبلاً تست‌های مختلفی روی سورس‌کد اِعمال شده و هیچ‌گونه مشکلی در ارتباط با انتشار نسخه جدید نرم‌افزار وجود نخواهد داشت. در این فرآیند، اقداماتی همچون بلید، پَکجینگ و دیگر کارهایی که باید برای کارا بودن یک نرم‌افزار صورت گیرند نیز تکمیل شده به طوری که CD این امکان را به توسعه‌دهندگان می‌دهد تا به صورت روزانه،‌ هفتگی و یا ماهانه نسخه جدید نرم‌افزار خود را در دسترس کاربران واقعی‌اش قرار دهند.

**Continuous Deployment چیست؟**

[Continuous Deployment](https://sokanacademy.com/glossary/10041/continuous-deployment/%D8%A7%D9%86%D8%AA%D8%B4%D8%A7%D8%B1-%D9%85%D8%AF%D8%A7%D9%88%D9%85)  پس از مرحله قبل صورت می‌گیرد بدین شکل که هر تغییری که در سورس‌کد اِعمال می‌شود و با موفقیت تست‌های مختلف را پشت سر بگذارد به صورت **خودکار** توسط سیستم روی سرور دیپلوی می‌شود و هرگز برای این کار نیازی به دخالت عاملِ انسانی نخواهد بود و تنها در صورتی این کار متوقف می‌گردد که سورس‌کد نتواند از پسِ تست‌ها به صورت موفقیت‌آمیزی برآید. روی هم رفته می‌توان گفت که از مزایای CD این است که توسعه‌دهندگان می‌توانند تمرکز خود را روی کدنویسی گذاشته و انتشار خلاقیت‌هایی که به خرج می‌دهند را به سیستم‌های کامپیوتری واگذار کنند و نیاز به توضیح نیست که این کار منجر بدان خواهد شد که سریع‌تر هم بتوانند فیدبک‌های مرتبط با تغییرات مد نظر خود را از کاربران بگیرند.

به همان ترتیبی که به توضیح این مفاهیم پرداختیم، آن‌ها یکی پس از دیگری می‌آیند و ارتباط‌شان از بالا به پایین است بدین شکل که CI بخش زیربناییCD است و Continuous Deployment نیز فرآیندی همچون Continuous Delivery است با این تفاوت که اجرای آن به صورت **خودکار** صورت می‌گیرد به طوری که تصویر زیر این ارتباط را به خوبی ترسیم کرده است:



همان‌طور که در تصویر فوق ملاحظه می‌شود، پروسه CI در هر دو صورت یکسان است و در Continuous Deployment پس موفقیت‌آمیز بودن انجام یونیت تست‌ها، کدهای جدید به صورت اتوماتیک روی سرور دیپلوی می‌شوند اما در Continuous Delivery بخش نهایی کار یا به عبارتی انتشار نسخه جدید نرم‌افزار بسته به صلاحدید تیم فنی به صورت دستی انجام می‌شود.

بدانید که Jenkins یکی از محبوب‌ترین سرورهای خودکار استقرار به صورت کدباز است. و ابزارهای معادل دیگری نیز دارد که یکی از موارد دیگر GitLab CI است. اگر سازمان شما بر روی GitLab فعالیت می‌کند، بهتر است که از همین مورد کارتان را شروع کنید، چرا که به خوبی با سایر بخش‌های GitLab ادغام شده است.

تیم برنامه نویسی به CI و CD احتیاج دارد تا بتواند به صورت یکپارچه فعالیت کند  و GitLab CI / CD رتبه 1 در Forrester CI Wave rated را دارا می باشد.

gitlab runner یک برنامه باینری به زبان Go است و نیازی ندارد که کتابخانه ی اضافی روی سیستم نصب شود (مگر اینکه از سورس کامپایل شود) cross platform است و اگر در کنار آن می خواهید از docker استفاده کنید آخرین ورژن آن را نصب کند زیرا gitlab runner  حداقل به ورژن 1.13.0 داکر نیاز دارد.

**آشنایی با مزایای  Continuous Integration**

پیش از آنکه به مزایای CI بپردازیم لازم به یادآوری است که برای این کار دولوپرها باید برای هر فیچر جدیدی که به نرم‌افزار یا اپلیکیشن اضافه می‌کنند و یا باگی را فیکس می‌کنند یک تست جداگانه بنویسند مضاف بر اینکه نیاز به یک سرور مجزا برای اجرای این فرآیند خواهیم داشت تا بتوان از آن طریق به محض کامیت کردن فیچرهای جدید، تست‌ها را به صورت خودکار اجرا کرده و همچنین ریپازیتوری اصلی را مانیتور کرد (در این فرآیند توصیه می‌شود که تمامی اعضای تیم مهندسی تغییراتی که روی محیط لوکال خود اِعمال کرده‌اند را حداقل روزی یک بار با ریپازیتوری اصلی ادغام نمایند(

با این تفاسیر، نیاز به توضیح نیست که اگر هم باگی در سورس‌کد وجود داشته باشد، در این فرآیند و با حضور تست‌های مختلف تا حد ممکن جلوی آن گرفته خواهد شد و کلیه مشکلات احتمالی در مراحل اولیه شناسایی شده و خیلی سریع توسط فردی که آن را ایجاد شده رفع می‌شوند به علاوه اینکه دیگر نیازی به تست دستی سورس‌کد نیست چرا که این کار به صورت خودکار توسط سروری که برایش در نظر گرفته شده است انجام خواهد شد و تیم تضمین کیفیت نیز زمان کمتری را معطوف به تست کردن قابلیت‌های بدیهی کرده و می‌تواند تمرکز خود را روی بخش‌های کلیدی‌تر بگذارد. در این فرآیند تست‌های مختلفی به کار گرفته می‌شوند که مهم‌ترین آن‌ها عبارتند از:

**- Unit Test** : در این نوع تست یکسری فانکشن‌هایی وجود دارند که این وظیفه را دارند تا عملکرد سایر فانکشن‌ها را تست کنند.  
**- Integration Test** : این تست اطمینان حاصل می‌کند که کامپوننت‌های مختلف به خوبی با یکدیگر سازگار هستند.  
**- Acceptance Test:** این تست همچون مورد قبل است با این تفاوت که بیشتر از بُعد تجاری نرم‌افزار را محک می‌زند.  
**-UI Test:** این تست تضمین می‌دهد که وقتی رابط کاربری در اختیار کاربران قرار گیرد، همان‌طور که از آن انتظار می‌رود کار خواهد کرد.

**آشنایی با مزایای Continuous Delivery**

همان‌طور که پیش از این اشاره کردیم، این فرآیند پس از CI صورت می‌گیرد بدین صورت که در هر زمانی که مشتری و یا تیم مهندسی تمایل داشته باشد، می‌توان نسخه‌ جدید نرم‌افزار را روی سرور اصلی منتشر کرد مضاف بر اینکه با سرعت بیشتری می‌توان فیچرهای جدید را در معرض دید کاربران نهایی نرم‌افزار قرار داده و بالتبع از ایشان برای اِعمال تغییرات یا بهبود بیشتر فیدبک گرفت.

همچنین نکته‌ای که در ارتباط با این فرآیند باید همواره مد نظر قرار داده شود آن است که کیفیت نرم‌افزار تحت هیچ عنوان نباید فدای سرعت انتشار آن گردد! به عبارت دیگر، گرچه CD این امکان را در اختیار تیم‌های مهندسی نرم‌افزار می‌گذارد تا همواره آماده عرضه آخرین نسخه از نرم‌افزار خود باشند، اما در عین حال داشتن **اطمینان کامل** از کارکرد صحیح بیلد یک باید است.

**آشنایی با مزایای Continuous Deployment**

این گام که آخرین مرحله از مراحل سه‌گانه است از حساسیت بسیار بالایی برخوردار می‌باشد چرا که قرار است کلیه‌ تغییرات به صورت خودکار و بدون دخالت عامل انسانی صورت گیرند و اینجا است که باید از درست تست شدن سورس‌کد کاملاً اطمینان پیدا کنیم به طوری که هرچه تست‌ها قابل‌اعتمادتر باشند، نسخه ریلیس نهایی که به صورت خودکار دیپلوی می‌شود نیز از قابلیت اعتماد بیشتری برخوردار خواهد بود.

وقتی Continuous Deployment به فرآیند توسعه نرم‌افزار ورود پیدا می‌کند، تیم توسعه با خیال راحت می‌تواند بر کدنویسی خود متمرکز باشد زیرا کدهایی که ایشان می‌نویسند قرار است ابتدا به صورت کامل توسط سیستم تست شده سپس به صورت خودکار روی سرور اصلی و بدون نیاز به دخالت ایشان دیپلوی شوند مضاف بر اینکه از دید کاربران نهایی نیز این مزیت شکل می‌گیرد که دائماً بهبودهایی را به صورت روزانه یا هفتگی در نرم‌افزار مورد استفاده خود شاهد هستند و دیگر نیازی نیست تا روزها، هفته‌ها و یا ماه‌ها برای انتشار نسخه جدید نرم‌افزار یا اپلیکیشن صبر نمایند.

نکته‌ای که در ارتباط با این فرآیند وجود دارد آن است که همواره باید به سازوکاری اندیشید تا بتوان به صورت Real-time (بلادرنگ) رفتار سیستم را مانیتور کرد چرا که در این پروسه همه چیز به صورت خودکار صورت می‌گیرد و بدیهی است که نیاز به ابزارهایی داریم تا چنانچه به هر دلیلی مشکل روی سرور ایجاد شد،‌ از طریق نوتیفیکیشن‌های مختلف از آن مطلع گردیم.

**جمع‌بندی**  
سرمایه‌گذاری روی مباحث فنی و استفاده از ابزارهای مناسب CI/CD یک روی سکه است و ایجاد فرهنگی که در آن CI/CD محترم شمرده شود روی دیگر سکه به طوری که این فرهنگ می‌باید مابین تک‌تک دولوپرها شکل گیرد که تا می‌توانند کدهای خود را سریع کامیت کنند، برای هر فیچر جدیدی که می‌نویسند یا بخشی که بهبود می‌دهند و یا باگی که فیکس می‌کنند یک یونیت تست بنویسند و در یک کلام بار را از روی دوش سایر اعضای تیم تا حد ممکن بردارند. نیاز به توضیح نیست چنانچه فردی برای مدت‌ها تغییرات لوکال خود را روی ریپازیتوری اصلی نفرستد، احتمال ایجاد نه یک بلکه چندین و چند کانفیلیکت مختلف با سایر بخش‌های نرم‌افزار بسیار زیاد خواهد بود. چنانچه در چنین فضایی به هر دلیلی باگی در بیلد نهایی یافت شود، **اولویت** اول و آخر رفع آن باگ است و نیاز به توضیح نیست که هرچه تعداد تغییرات بیشتر بوده باشد، یافتن باگ مذکور نیز دشوارتر خواهد شد و همان‌طور که قبلاً بارها اشاره کردیم، اهمیت نوشتن تست‌های اصولی در اینجا مشخص می‌شود.

**معنی CI و CD چیه؟**

در واقع CI مخفف Continuous Integration *(ادغام پیوسته)* هست که یک عمل نرم افزاری است که توی اون همه برنامه نویس ها، تغییراتی که توی کدشون میدن رو روی یک ریپازیتوری مرکزی merge میکنن *(شاید چندین بار این اتفاق در روز بیوفته)*

همچنین CD مخفف Continuous Delivery هست که در یک سطح بالای Continuous Integration قرار میگیره و عمل خودکارسازی انتشار یک پروژه نرم افزاری رو به چرخه فرایند اضافه میکنه.

در CI، هر تغیر در کد، یک زنجیره build-and-test *(ساخت و تست)* رو برای اون پروژه ای که توش کد تغییر کرده اجرا میکنه، همچنین بازخورد اون تغییر کد رو به برنامه نویس یا برنامه نویسایی که اون تغییرات رو ایجاد کردن اعلام میکنه. تمام چرخه CI (از خروجی گرفتن پروژه تا تست و بارگذاریش) همه باید کمتر از 10 دقیقه طول بکشه.

از طرفی CD شامل تهیه و تدارک زیرساخت ها هست. که میتونه گاها شامل چندین لایه/سطح باشه. چیزی که مهمه اینه که تمام این فرایند باید کاملا خودکار باشه؛ و هر نسخه جدیدی که آماده میشه و این چرخه شروع به اجرا میکنه، یک نسخه کاملی از log ها *(گزارش کامل)* ساخته بشه و در اختیار تیم فنی قرار بگیره که همه متوجه بشن اون features *(قابلیت یا قابلیت های)* جدیدی که اضافه شده، به چه صورتی و با چه بازخوردی به پروژه اضافه شدن.

**نکته:** در بعضی موارد هم CD به منظور Continuous Deployment هم در مطرح میشه و از اونجایی که خیلی روی صحبت من مرتبط باهاش نیست، اشاره ای بهش نمیکنم.

**اجزای CI/CD pipeline چی هستن؟**

شاید در یک نگاه کلی به قصه، CI/CD pipeline یک کار اضافی و سربار به نظر برسه، ولی واقعا اینطور نیست. همانطور که گفته شد، دقیقا ضروری ترین مراحل انتشار یک نسخه جدید از محصول رو به صورت خیلی حرفه ای و دقیق خودکار یا همون automated میکنه. مراحلی که باید هر سری برای انتشار یک نسخه جدید انجام و بررسی بشن.

در نبود یک pipeline خودکار، در واقع مهندسین *(یا برنامه نویس ها)* باید به صورت دستی *(و با ضریب خطای بیشتری)* این مراحل رو انجام بدن که خب همین امر باعث میشه زمان بیشتری هم بزارن و طبیعتا عملکردشون *(یا همون productively)* هم افت میکنه.

**اکثر پروژه های نرم افزاری، برای انتشار یک نسخه جدید، باید این مراحل رو طی کنن**:

همانطور که میبینی، مراحل کلی یک فرایند برای انتشار یک ورژن یا نسخه جدید از یک پروژه/محصول 4 مورد زیر هستن:

1. مرحله Source (منبع کد)
2. مرحله Build (ساخت)
3. مرحله Test (تست)
4. مرحله Deploy (استقرار، انتشار)

خطا هر کدوم از مراحل بالا به صورت یه نوتیفیکیشن *(یا از طریق ایمیل یا اسلک یا هرچی)* به برنامه نویسای پروژه ارسال میشه که برنامه نویسایی که مسئول اون بخش هستن به خطای رخ داده رسیدگی بکنن. در غیر این صورت *(اگر هیچ خطایی نباشه)* برای تمام تیم یک نوتیفیکیشن «موفقیت آمیز» از انتشار نسخه جدید ارسال میشه.

**مرحله Source (منبع کد)**

در بیشتر مواقع، شروع pipeline زمانی اتفاق میوفته که یک push به ریپازیتوری *(که کد پروژه اونجاس)* انجام میشه. به عبارت بهتر، همون تغییر کدی که گفته بودم، وقتی به سورس پروژه تزریق میشه فرایند CI/CD شروع به کار می­کنه. البته یک نوع دیگه­ای از شروع کار CI/CD هم وجود داره که بهش میگن scheduled *(یعنی زمان بندی شده)*؛ این مورد اغلب برای مواقعی هست که میخوایم در یک ساعت خاصی از روز مثلا وب سایتمون رو بروزرسانی بکنیم که میدونیم در اون ساعت تعداد کاربرهای آنلاین کمتری دارن از سایت استفاده میکنن. که در اون صورت توی تنظیمات ست میکنیم که در فلان زمان عملیات CI/CD شروع به کار کنه.

**مرحله Build (ساخت)**

در این مرحله ما کدی که به سورس پروژه اضافه شده رو ترکیب میکنیم با کد سورس اصلی و تمام وابستگی هاش (dependencies) رو هم نصب و دانلود میکنیم و سورس رو تبدیل میکنیم به همون چیزی که کاربر نهایی قراره در این نسخه ببینه. برنامه هایی که به زبان های جاوا، سی پلاس پلای یا .. نوشته شدن، نیاز دارن که کامپایل بشن (و در این مرحله میشن)، در حالی که زبان هایی مثل روبی، پایتون، جاوااسکریپت یا .. نیازی به این کار ندارن.

علاوه بر در نظر گرفتن زبان برنامه نویسی، نرم افزارهای cloud-native عموما در بستر داکر Docker قرار دارن و توسعه داده میشن، پس در این مرحله، CI/CD pipeline یک داکر کانتینر Docker containers میسازه. رخداد خطا در این مرحله ساخت، یک شاخص هست از یک مشکل اساسی در تنظیمات پروژه که باید در اسرع وقت بهش رسیدگی بشه.

**مرحله Test (تست)**

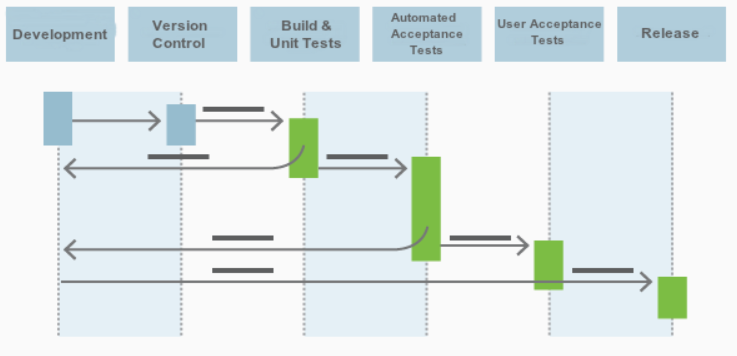
در این مرحله، به صورت خودکار، تست هایی که پروژه باید سپری بکنه تا صحتش اعتبارسنجی بشه اجرا میشن. هم توی کد و هم رفتار کلی محصول. این مهمترین مرحله هست برای حصول اطمینان از صحت عملکرد محصول در نسخه جدید؛ که باعث میشه نسخه ای که قرار انتشار پیدا بکنه و در معرض استفاده کاربر نهایی قرار میگیره، بدون باگ Bug free و بدون کرش Crash free باشه.

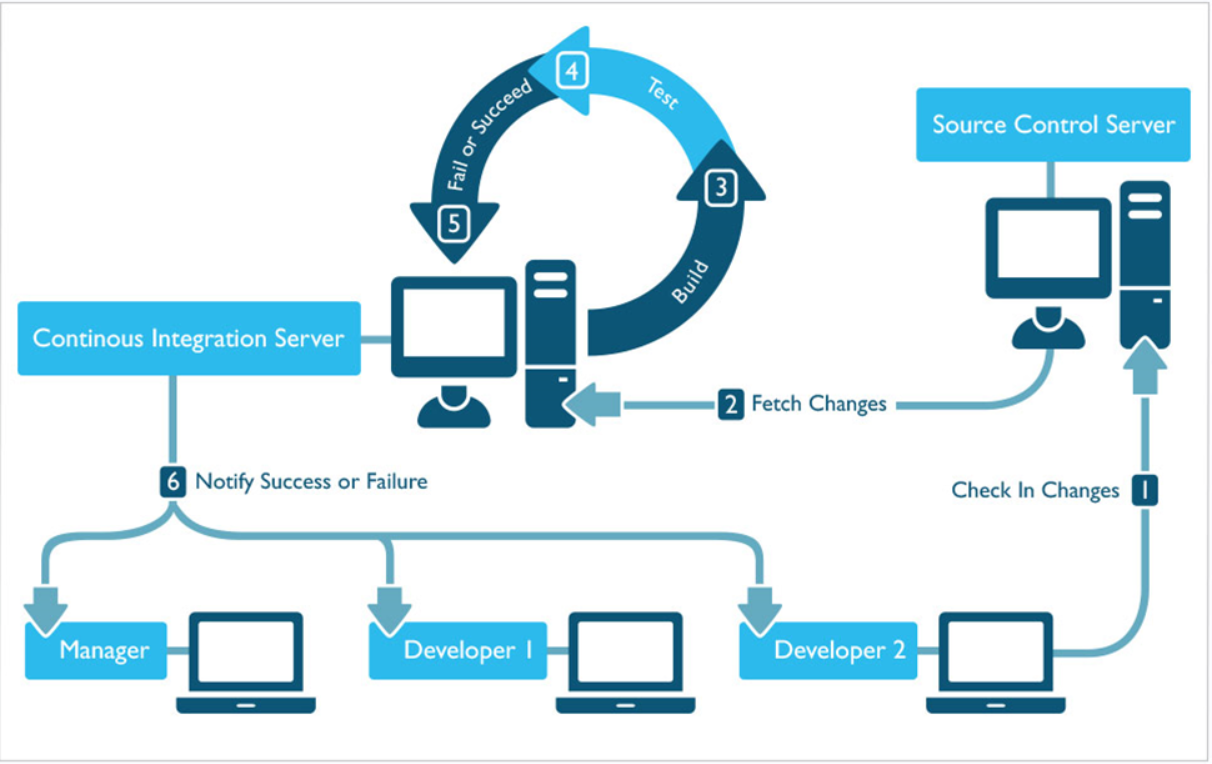
عموما دیده میشه که برنامه نویس ها خیلی تمایلی به نوشتن تست ها ندارن و معمولا پشت گوش میندازن. ولی خب همونطور که مشخص هست یک مرحله خیلی مهم هست و برای راه اندازی CI/CD pipeline جزو الزامات هستش. هر پروژه برنامه نویسی با توجه به زبان و ساختاری که داره تست های خودش رو داره که نهایتا هرگونه خطایی که تست ها نشون میدن به برنامه نویس های پروژه گزارش داده میشه.

**مرحله Deploy (استقرار، انتشار)**

خب الان ما تا قبل از این مرحله، یک ورژن جدید از پروژه رو داریم که یک سری ویژگی و قابلیت جدید بهش اضافه شده، تلفیق و ترکیب کد بیس اتفاق افتاده، تست شده و آماده به انتشار هست. در اینجا محیط های انتشار متفاوتی وجود داره، مثلا «بتا - Beta» یا «staging» که برای تیم داخلی قابل استفاده و در دسترس هست و همچنین محیط «ساخت نهایی-« production که دقیقا همونجایی هست که کاربر نهایی میبینه.

تیم هایی که متدولوژی مدیریت پروژشون مدل Agile هست، که توی مراحل توسعشون تست و مانیتور کردن وجود داره، معمولا محیط Deployشون همون staging هست که باز بتونن یه سری تست ها و ریویو ها رو روی نسخه جدید انجام بدن و بعد از تایید نهایی، به محیط production انتقال پیدا میکنه و نسخه جدید برای کاربرهای نهایی قابل روئیت و استفاده میشه.



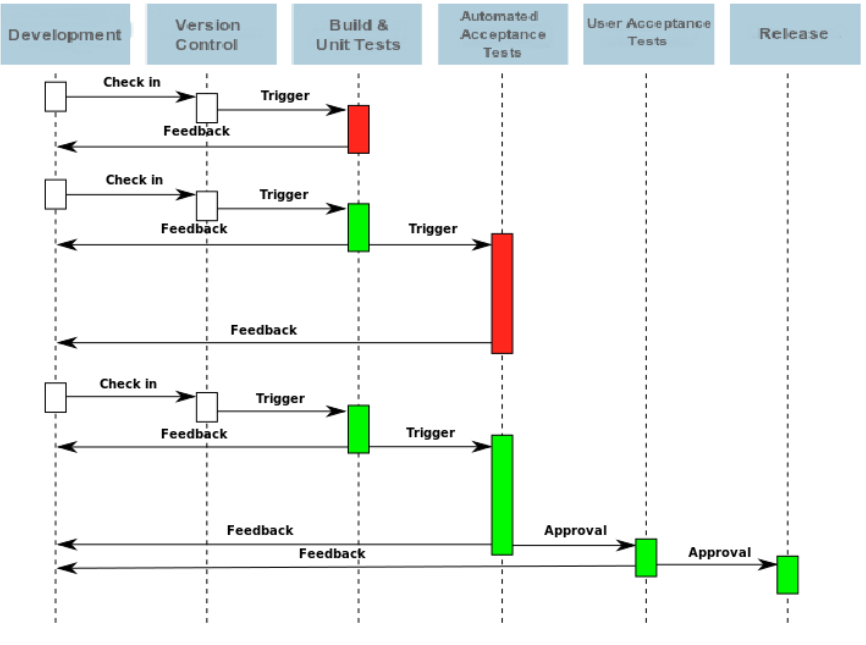


**Continuous Delivery چیست ؟**

Continuous Delivery یا (CD)، رویکردی در مهندسی نرم افزار است که تیم ها را قادر می سازد نرم افزار تولید شده را به روشی **سریع** و **مطمئن** برای **انتشار و تحویل** **آماده** کنند. این فرایند از لحظه اضافه شدن یا تغییر کد در source control شروع می شود و شامل بیلد، تست، پیکربندی و انتشار در محیط های مختلف تست و محیط عملیات می شود. این مفهوم در فارسی به “تحویل مداوم” یا “تحویل مستمر” ترجمه شده است.

به عبارت دیگر: Continuous Delivery توانایی اعمال تغییرات در محیط عملیات در هر لحظه با روشی **سریع** و **مطمئن**و به طور کاملا **پایدار** می باشد. این تغییرات شامل همه انواع آن از جمله تغییرات پیکربندی در نرم افزار، زیرساخت و پلتفرم، افرودن ویژگی های جدید، رفع باگ و خطا ها می شود.

به وسیله محیط های تست مختلف، می توان یک Release Pipeline ایجاد کرد تا بتوان یک زیرساخت جدید را به طور اتوماتیک ایجاد کرد و نرم افزار را روی آن منتشر کرد. منظور از زیرساخت، سرور، سیستم عامل، سرویس دهنده ی وب، virtualization، شبکه و پیکربندی و تنظیمات آنها می باشد. به کمک این محیط های متوالی می توان فعالیت های طولانی Integration، تست عملکرد و تست های پذیرش نهایی را به تدریج انجام داد. فرایند Continuous Delivery در Release Pipeline با [Continuous Integration](http://hidevops.com/category/continuous-integration/)  شروع میشود و با انتشار و پایان تست در هر محیط، انتشار و تست در مرحله بعدی شروع می شود. مجموع این کارها به صورت حلقه های یک زنجیر در پشت سر هم قرار گرفته و فرایند Continuous Delivery را تشکیل می دهند.



**هدف Continuous Delivery چیست ؟**

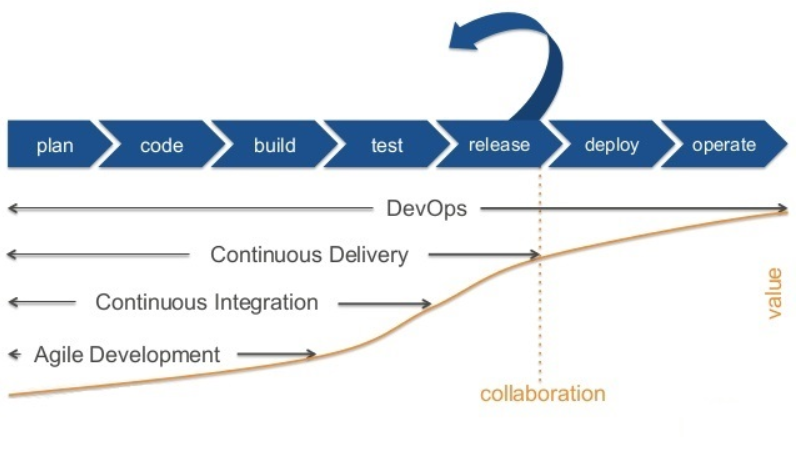
هدف CD این است که که انتشار و تحویل نرم افزار را به خصوص برای **سیستم های توزیع شده در مقیاس بسیار بزرگ** و **محیط های عملیاتی پیچیده** به یک فرایند **روتین**، **ساده** و **قابل پیش بینی** تبدیل کند. رسیدن به این هدف تنها در صورتی امکان پذیر است که **کدهای** نرم افزار **همیشه** در وضعیت **آماده برای انتشار** باشند، حتی در شرایطی که یک تیم با هزاران توسعه دهنده به طور روزانه در حال تغییر و به روزرسانی کد ها و نرم افزار هستند. به این ترتیب روش سنتی فازهای ادغام (Integration)، تست، پیکربندی و Hardening جای خود را به روش های اتوماتیک می دهد. همچنین **زمان آماده شدن برای انتشار**، **زمان مهاجرت به محیط**های دیگر (Time to Remediate) و **زمان برطرف کردن رخداد** در محیط عملیات را کاهش (Time To Mitigate or time to remediate production incidents) می دهد.

**تفاوت Continuous Delivery و Continuous Deployment چیست ؟**

بسیاری از مواقع دو مفهوم Continuous Delivery و Continuous Deployment به اشتباه به جای هم استفاده می شوند اما تقاوت این دو چیست ؟ Continuous Deployment به این معنی است که هر تغییر در نرم افزار به طور اتوماتیک (یا بر طبق یک زمان بندی مشخص) در محیط عملیات **منتشر شود**. اما Continuous Delivery به این معنی است که هر تغییر در نرم افزار، **آماده انتشار**در هر محیطی باشد اما ممکن است تصمیم این باشد که منتشر نشود (معمولا به دلایل کسب و کاری). این محیط ها شامل محیط تست تیم توسعه، محیط تست تیم کنترل کیفیت و محیط عملیات و هر محیط دیگری می شود.

**تفاوت Continuous Delivery و** [**دواپس**](http://hidevops.com/category/دوآپس/) **چیست ؟**

این دو مفهوم از جهاتی به هم نزدیک هستند و اما تفاوت های بنیادی با هم دارند. دامنه ی [DevOps](http://hidevops.com/category/devops/) بسیار گسترده تر از CD است و علاوه بر خودکارسازی فرایند تحویل نرم افزار، شامل **تغییرات فرهنگی** به خصوص **ارتباط** بین **تیم ها**ی مختلفی که در جریان آماده سازی و تحویل محصول نرم افزاری (Software Delivery) هستند هم می شود. (از جمله تیم های توسعه، عملیات، تضمین کیفیت، مدیریت و … ) اما Continuous Delivery روشی است برای **اتوماتیک سازی زنجیره تحویل محصول** و بر این موضوع تمرکز دارد که فرایند های مرتبط در کنار هم طوری اجرا شود که منجر شود محصول نرم افزاری، **سریعتر**، **امن تر** و **با تکرار بیشتر**ی به دست مشتری برسد.



**در نبود Continuous Delivery …**

در نبود CD، انتشار نرم افزار یک bottleneck برای تیم های “عملیات فناوری اطلاعات” و در نهایت برای محصول است. فرایند دستی و غیر اتوماتیک، باعث **انتشارهای غیر قابل اتکا**، **با تاخیر** و **پر از خطا** می شود. تیم های درگیر این موضوع معمولا به کارهای دستی که افراد انجام می دهند تکیه می کنند که باعث مشکلات جدی خواهد شد.

**حداقل الزامات اجرای Continuous Delivery**

برای راه اندازی و اجرای Continuous Delivery الزاماتی وجود دارد که در کمترین حالت می توان از دو مورد زیر نام برد

* زیربنایی از تست ها که تا حد قابل قبولی خیال شما را از اینکه نرم افزار به درستی کار می کند، راحت کند و به شما اعتماد به نفس لازم را برای شروع استقرار بدهد
* ابزارهای خودکارسازی (automation) و اسکریپت های لازم که به شما اطمینان دهد استقرار با موفقیت انجام خواهد شد و در صورت نیاز امکان Rollback وجود خواهد داشت

**سخن آخر**

در آخر باید این این نکته را اضافه کنم که فرایند [Continuous Delivery](http://hidevops.com/143/continuous-delivery-%da%86%db%8c%d8%b3%d8%aa-%d8%9f/) و Infrastructure as Code و Monitoring به طور قابل ملاحظه ای مکمل یکدیگر هستند. [Continuous Delivery](http://hidevops.com/143/continuous-delivery-%da%86%db%8c%d8%b3%d8%aa-%d8%9f/) برای کنترل تاثیر تغییرات در محیط عملیاتی از استراتژی های استقرار از جمله Blue-Green Deployment و همچنین تکنیک Feature Flags (یا Feature Toggles) نیز پشتیبانی می کند. (در آینده راجع به استراتژی های انتشار مطلبی خواهم نوشت)  
امروزه ارزش هایی که CD برای سازمان ها ایجاد می کند، آن را به یک نیاز ضروری تبدیل کرده است. برای رساندن ارزش به مشتریان نهایی، باید محصول را **به طور پیوسته** و **بدون خطا** منتشر کرد. Release Pipeline های پیشرفته به **توسعه دهندگان** اجازه می دهد که feature های جدید را **سریع** و **مطمئن**منتشر کنند. با کمک CD، رفع خطاها در محیط عملیات و اضافه کردن ویژگی های جدید بسیار سریع تر و با اطمینان بالا انجام خواهد شد و به دست مشتری خواهد رسید.