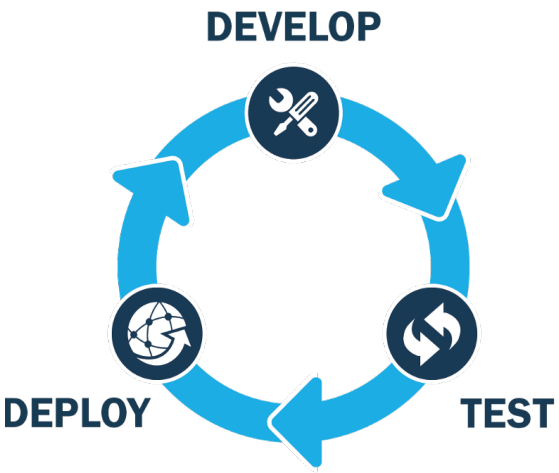
مروری بر Continuous Integration, Continuous Delivery, Continuous Deployment

مقدمه

در فرایند توسعه نرم افزار، **یکپارچه‌سازی عملیات و اجرای آنها بصورت مداوم**، یک روش کاملاً جاافتاده و توسعه یافته برای اکثر زبان‌های برنامه نویسی است... اما در اکوسیستم PHP، یک رویکرد تقریباً جدید به شمار می‌رود. یکپارچه سازی و خودکارسازی فرایندها، به ما این امکان را می‌دهد تا به سرعت و به طور مکرر، بدون دخالت انسان، نسخه های جدیدی از نرم افزارمان را ارائه دهیم. انجام این کار، روش دستی انجام task ها را کاهش می‌دهد، احتمال خطا را پایین می آورد، تست‌ها را به صورت خودکار اجرا می کند و نسخه هایی از نرم افزار را تولید می‌کند.

در حال حاضر راهکارهای یکپارچه سازی و خودکارسازی بسیار زیادی وجود دارند، اما تعداد کمی از آنها صددرصد به PHP اختصاص دارند. چراکه اکثر آنها چندزبانه و general هستند و **نیازهای خاص** توسعه‌دهندگان PHP را در نظر نمی‌گیرند. در این مقاله، هر آنچه که شما برای یکپارچه سازی و خودکارسازی فرایندها در پروژه‌ی خود باید بدانید را ارائه خواهم داد.



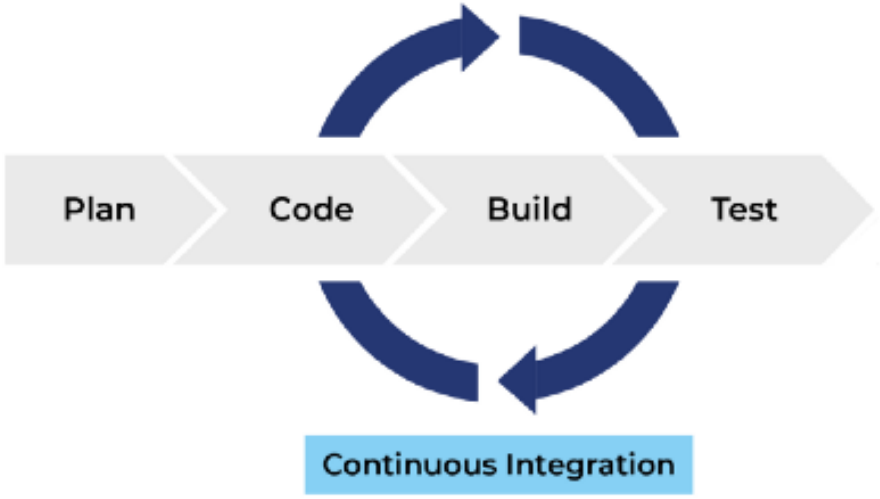
فهرست/مندرجات

روش‌ها

روش‌ها و practice های مختلفی وجود دارند که در آن، یکپارچه سازی و خودکارسازی فرایندها به شما کمک کند تا پروسه‌های تضمین کیفیت/QA را اجرایی کنید. در اینجا برخی از آنها را بررسی می کنیم:

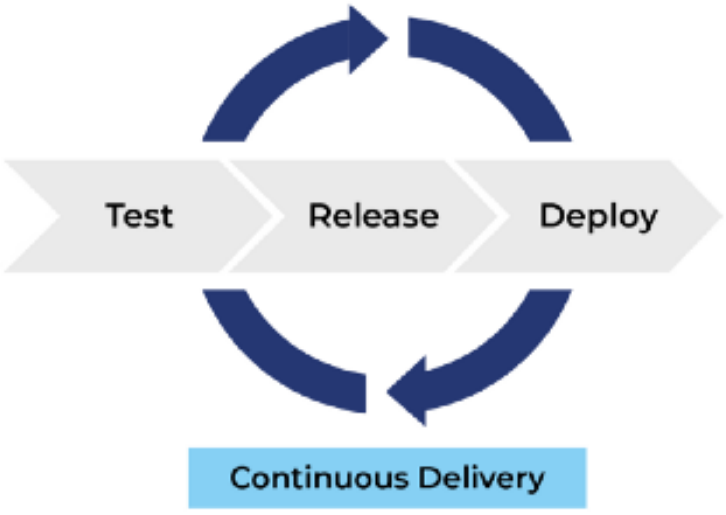
Continuous Integration / CI

Continuous Integration (یکپارچه‌سازی مداوم) مجموعه‌ای از اقداماتی است که به صورت خودکار، تغییرات کدهای پروژه را از طرف توسعه دهندگان پروژه، دریافت و ادغام/Merge میکند. درواقع روشی است که به ادغام **مداوم و خودکار** branch ها به یک branch اصلی و مشترک اشاره دارد. این روش بخشی از متدولوژی Extreme Programming / XP که یک متد Agile است، به شمار می‌رود که بر جنبه‌ی تحقق، بازخورد و توسعه اپلیکیشن تمرکز دارد. اجرا و پیاده‌سازی CI مستلزم آن است که Codebase ای برای هر commit ساخته و تست شود تا از هر پسرفت و مشکلی حین یکپارچه سازی جلوگیری کند. سپس این نسخه‌ی جدید از Codebase را می‌توان بر روی یک سرور دیگری استقرار/Deploy نمود، تا اینکه تست‌های manual را نیز بتوان انجام داد.
نکته: تست های Unit, Integration, Component/Module در این مرحله انجام می شوند.



Continuous Delivery / CD

Continuous Delivery (تحویل مداوم) یک گام فراتر از Continuous Integration عمل می‌کند، اما به Continuous Integration وابسته می باشد. Continuous Delivery تکنیکی است که در آن، تیم توسعه اطمینان می دهد که نرم افزار در هر زمان قابل انتشار/Release است. درواقع هر Codebase ای که تولید یا build شده است، آن را تست کرده و نسخه ای را بسته بندی می‌کند که آن را می‌توان تحویل end-user داد. نکته اینکه در این مرحله **بسته صرفاً ایجاد و تحویل داده میشود**، فارغ از اینکه آن بسته توسط end-user استفاده خواهد شد یا نخواهد شد! با انجام اینکار، هر commit یا feature می‌تواند سریعاً تحویل end-user داده شود.
بکمک Continuous Delivery، می توانید تصمیم بگیرید که Release بصورت روزانه، هفتگی، ماهیانه یا هر آنچه که با شرایط تجاری/مدیریتی شما مطابقت داشته باشد تحویل داده شود.
نکته: تست های (black box) System , Subsystem در این مرحله انجام می شوند.

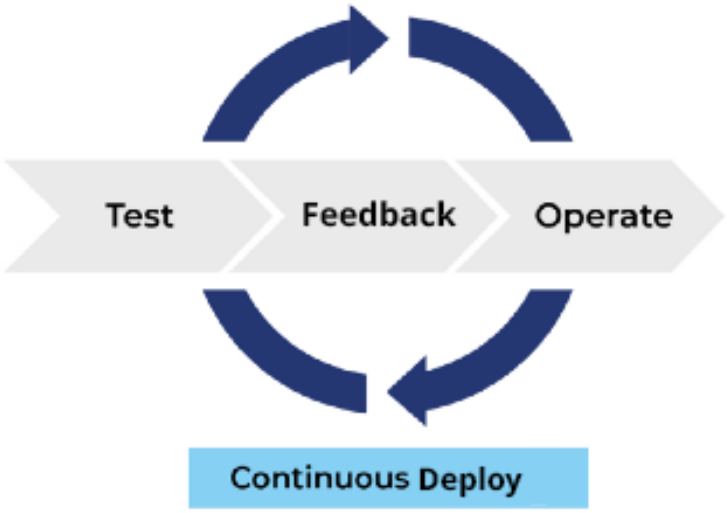


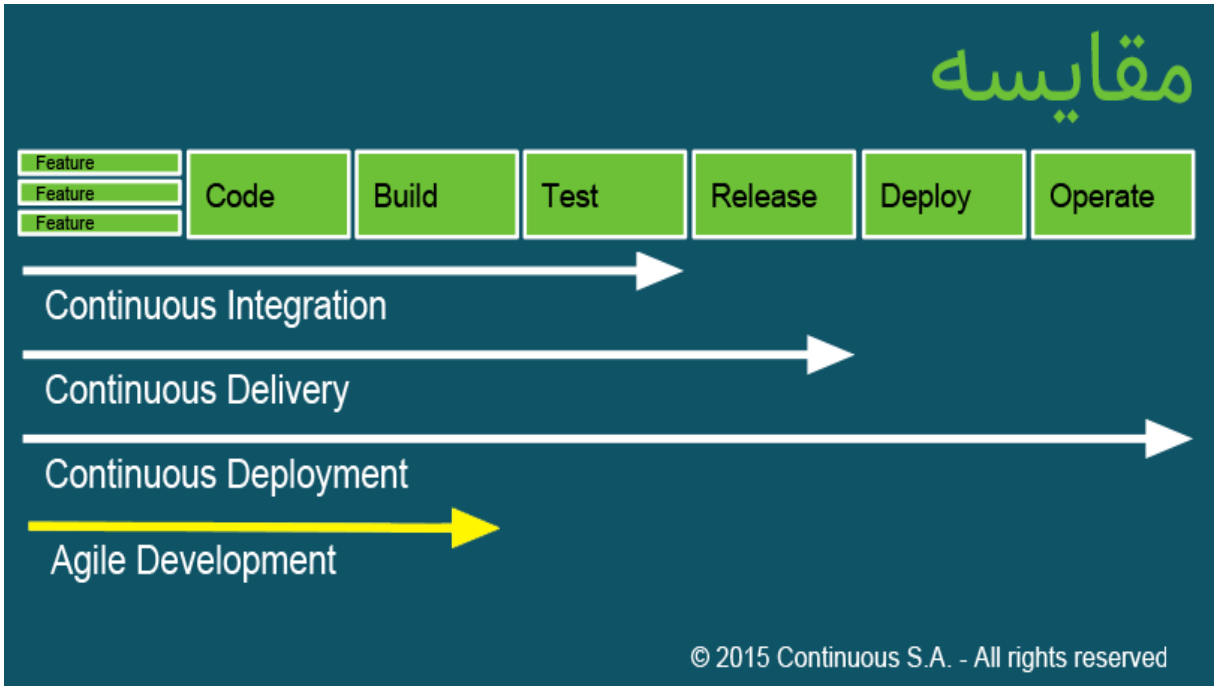
مزایای استفاده از Continuous Delivery

- خودکارسازی فرآیند انتشار نرم‌افزار
- بهبود روند توسعه نرم افزار
- سرعت در کشف، ردیابی و رفع باگهای نرم‌افزار
- سرعت در انجام بروزرسانی نرم‌افزار
- سرعت در تحویل نرم‌افزار به بازار/مصرف کننده
- کاهش میزان استرس تیم تولیدکننده

Continuous Deployment / CD

Continuous Deployment (استقرار مداوم) یک گام فراتر از Continuous Delivery عمل می‌کند، اما به Continuous Delivery وابسته می باشد. در این مرحله تمام تغییرات کد بعد از مرحله Release، به طور خودکار در یک محیط production **مستقر و اجرا** میشود. در حالیکه تمامی اصول Continuous Delivery پیاده‌سازی و اجرا می‌شوند، هر build ثابت و stable در یک محیط production استقرار می‌یابد. به این ترتیب، هر امکان و ویژگی جدید پروژه، برای تمامی end-user ها، به سریع‌ترین روش ممکن در دسترس خواهد بود. از آنجایی که تمامی فرایندها در این مرحله بصورت خودکار و بدون دخالت انسان انجام می شوند، **جمع آوری بازخورد از end-user** و تضمین کیفیت/QA بسیار مهم و کلیدی است.
نکته: تست User Acceptance در این مرحله انجام می شود.





Build

اولین چیزی که باید توجه داشت، مدیریت Dependencyهای نرم‌افزاری است. کدامیک برای اجرای تست‌ها لازم است و کدامیک برای اجرای اپلیکیشن اجباری خواهد بود؟ نیازی نیست تا با نصب و استفاده از فریم ورکهای تست بر روی یک سرور production، خود را به زحمت بیندازید. در عوض، اطمینان حاصل می‌کنیم که کتابخانه‌های third party مورد استفاده‌ی اپلیکیشن ما در دسترس باشند. برای انجام این کار، Composer بهترین همراه ما خواهد بود!
سپس به اتوماسیون/خودکارسازی می‌رسیم. باید به خاطر داشته باشیم که هم محیط production و هم محیط development، باید درون یک پروسه build پوشش داده شوند. اگر در حال حاضر از continuousphp استفاده می‌کنید (خوش شانسید!)، 2 پکیج را ایجاد خواهیم کرد؛ یک پکیج برای اجرای تست‌ها به همراه تمامی وابستگی‌های development، و یک پکیج دیگر برای deployment، بدون وابستگی‌های اضافی.
در حال حاضر، dependencyهای نرم‌افزاری براحتی تحت کنترل برنامه نویس هستند. اما هنوز کارمان تمام نشده است! در برخی موارد، باید فایل‌های configuration را ایجاد/اصلاح کرده و فایل‌های استاتیک (CSS، JavaScript،...) را کامپایل کنیم. برای انجام این کار، ما از ابزار phing، که خودکارسازی فعالیت‌ها و فرایندها را برای شما امکان‌پذیر می‌کند، استفاده خواهیم کرد. PHING با PHP و برای PHP نوشته شده است. اگر نیاز بهTask جدید user-defined وجود داشته باشد، و حتی اگر تعداد زیادی task از قبل تعبیه شده باشند، extend کردن PHING نیز آسان خواهد بود.

تست‌ها

تکنیک Continuous Deployment را نمی‌توان بدون اشاره به تست‌ها توضیح داد! ابزارهای زیادی برای تولید و انجام Unit Test در PHP وجود دارند، اما محبوبترین و پراستفاده ترین آنها PHPUnit است. از آنجایی که تست های Unit

به پایگاه‌داده و دیگر dependencyهای سیستم نیازی ندارند، پیاده‌سازی و اجرای یک Continuous Integration pipeline آسان خواهد بود.
اما تست‌های Functional بخش چالش‌برانگیزی برای پیاده‌سازی و اجرا خواهند بود. همانطور که این تست‌ها می‌توانند به برخی از dependencyهای سیستم وابسته باشند، ما باید این وابستگی‌ها را به منظور اجرای تست‌ها ارائه و فراهم کنیم. در این مورد، PHING ابزاری مفیدی خواهد بود که به ما این امکان را می‌دهد تا به راحتی این مرحله را مدیریت کنیم، و ابزاری است که برای اجرای تست‌ها لازم است. اگر بخواهیم dependency پایگاه‌داده را فراهم و ارائه کنیم، باید این مرحله را به دو بخش تقسیم کنیم. بخش اول، ایجاد/بروز رسانی شماتیک Life Cycle اپلیکیشن، از طریق ابزارهای ارتقا (DBDeploy، Phinx، Doctrine Migration،...) و بخش دوم، وارد کردن fixtureهای تست.
برای نوشتن این تست‌ها، ابزار Behat بسیار ضروری است، زیرا تنها ابزاری است که به شما این امکان را می‌دهد تا توسعه‌ی مبتنی بر رفتار (Behavior Driven Development) را در PHP انجام دهید. همچنین ابزار Gherkin نیز برای نوشتن تست‌ها مورد استفاده قرار می‌گیرد و آنها را در دسترس قرار می‌دهد.

استقرار / Deployment

این مرحله هدف هر اپلیکیشن است. استقرار به معنای کاربردی کردن، عملیاتی کردن و در دسترس قرار دادن اپلیکیشن بر روی سرور production است. ساخت یک پکیج، در صورتی که فرد بخواهد Deployment را اجرا کند و یا به نسخه‌ی خاصی rollback کند، اجباری و الزامی است. این پکیج می‌تواند برای Deployment بر روی سرورهای موجود مورد استفاده قرار گیرد، و نسخه‌ی فعلی را برای سرورهای جدید (مثلاً در یک محیط ابری) و یا نسخه‌های پیشین در صورت نیاز به rollback نیز Deployment کند. در نهایت، یک پکیج، فایل فشرده‌سازی‌شده‌ی (.tar.gz، .zip، ...) به همراه اپلیکیشن کامل و اسکریپت‌های نصب/ارتقا است.
اکنون که پکیج ما در دسترس است، باید آن را بر روی یک/چند سرور نصب کنیم. چندین ابزار برای استقرار وجود دارند که با پکیج ها کار می‌کنند. سرور Zend احتمالاً کامل‌ترین نوع در PHP است، اما ابزارهای دیگر مثل ابزار جدید Amazon CodeDeploy، واقعاً امیدوارکننده هستند. علاوه بر این، مدیریت این Deploymentها به همراه ابزارهای تامین و ارائه (Chef, Puppet, Ansible) امکان‌پذیر است.

Deployment pipeline

اکنون باید از یک اتوماسیون build مثل Jenkins، continuousphp... استفاده کنیم تا تمامی مراحل بالا را پیاده‌سازی کرده و یک Deployment pipeline را ایجاد کنیم که پس از هر commit یا هر push یک مسیر کامل build را طی کند. با ایجاد پایپ‌لاین‌های مختلف بر طبق branch/tag های موجود، می‌توانیم با روش Agile سریع کار کنیم پروژه نهایی را build کنیم. با این حال، یک پلتفرم خوب به شما این امکان را می‌دهد تا چندین build موازی را اجرا کنید. درواقع Parallelization باعث جلوگیری از کُندشدن عملیات پس از هر commit یا push خواهد شد.

توضیح مفاهیم اساسی

Task: یک Task یا یک Build task، **یک** عمل یا دستور است که باید انجام شود. مثل:
ant compile یا **gcc MyProgram.c** یا **tar -zcvf myfile.tgz myfolder**

Job: یک Job شامل چندین Task است که هر یک از آنها به ترتیب اجرا می شوند. اگر یک Task در یک Job از کار بیفتد، معمولاً کل Job ناموفق/fail شناخته می شود. مثل:

Job 1:

| ant clean
| gcc MyProgram.c
| javac MyApp.java

Job 2:

| tar -zcvf MyProgram.tgz MyProgram.exe
| tar -zcvf MyApp.tgz MyApp.jar

هر Task در Job به عنوان یک برنامه مستقل اجرا می شود بنابراین، تغییراتی که توسط یک Task در هر یک از متغیرهای آن ایجاد می شود، بر Task بعدی تأثیر نخواهد گذاشت.

Phase/Stage: یک Stage یا Phase شامل چندین Job است که هر یک از آنها می توانند مستقل از بقیه، Taskها را انجام دهند. اگر یک Job شکست بخورد، کل Stage ناموفق/fail شناخته می شود. مثل:

Stage build:

| Job 1

| Job 2

Stage deliver:

| Job 3

| Job 4

Stage deploy:

| Job 5

| Job 6

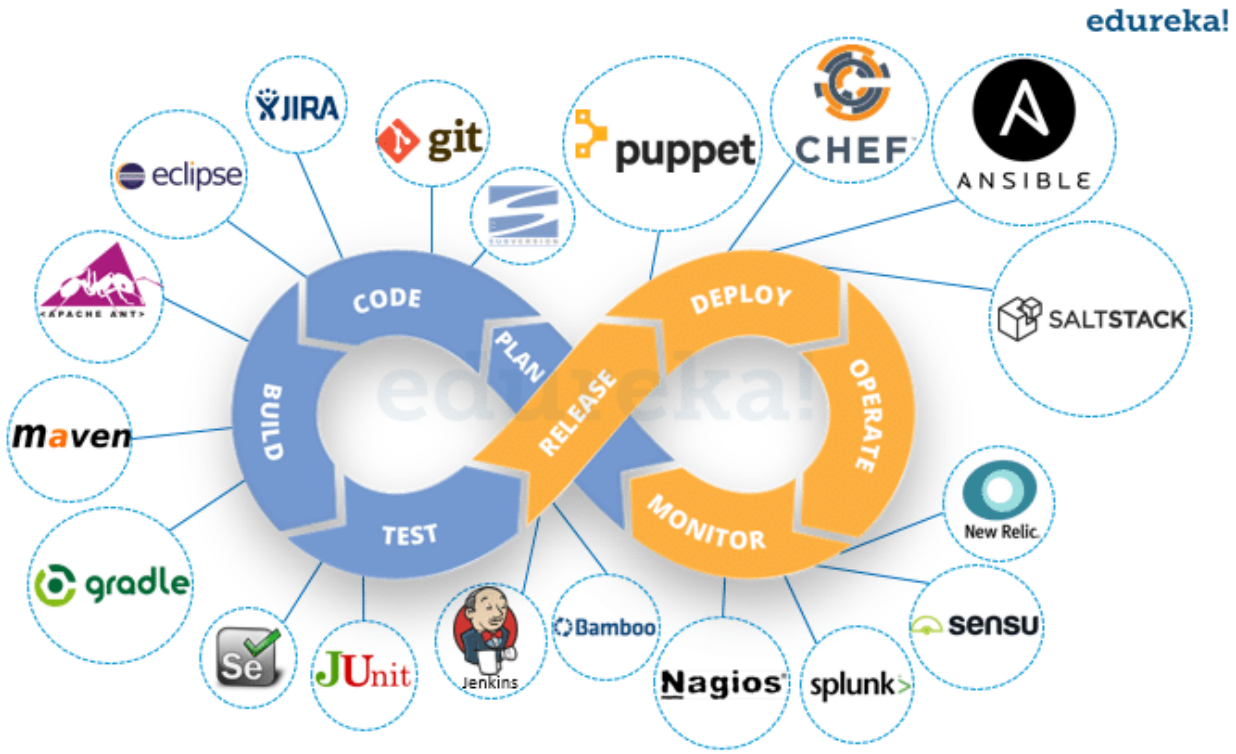
Pipeline: یک Pipeline شامل چندین Stage است که هر یک از آنها به ترتیب اجرا می شوند. اگر Stage ای خراب شود، Pipeline خراب در نظر گرفته می شود و مراحل بعدی شروع نمی شوند. مثل:

| **Stage build** → **Stage deliver** → **Stage deploy** |

Materials و Triggers: درواقع تعیین میکند که چه زمانی این Taskها، Jobها، Stageها و Pipeline اجرا شوند. یک Material دلیلی برای اجرای یک Pipeline است. معمولاً، این یک مخزن source code است (، SVN ، Git Mercurial و غیره). زیرساخت به طور مداوم materialها را بررسی می کند و هنگامی که یک تغییر جدید یافت می شود، Pipeline مربوطه اجرا یا Trigger می شود.

نتیجه‌گیری

می‌توانیم در نظر بگیریم که پیاده‌سازی یک Deployment pipeline در مقایسه با continuous integration pipeline پیچیده نیست، بلکه بسیار انعطاف‌پذیرتر است، زیرا هر مرحله از مدیریت پروژه را کنترل می‌کند. علاوه بر این، به توسعه‌دهندگان این امکان را میدهد تا در بخش Deployment مشارکت داشته و اصول و روشهای DevOps را بهتر درک کنند. اکنون می‌توانیم با سرعت بیشتر و با وقفه‌ی کمتر، اپلیکیشن را برای end-user ارائه کنیم.



نویسنده: یوشا آل ایوب

منابع:

https://continuousphp.com/tutorial/introduction-continuous-delivery/
https://ibm.com/cloud/learn/continuous-delivery
https://aws.amazon.com/devops/continuous-delivery/
https://atlassian.com/continuous-delivery