Continuous Delivery mit Docker, Gitlab CI und Ansible

4. Januar 2017 / 0 Kommentare / in shoptimax, Technisches / von Stefan Moises

Im letzten Jahr haben wir bei shoptimax den Entwicklungs- und Deployment-Workflow stark optimiert, um Projekte möglichst effektiv umsetzen, automatisiert testen und Fehler vermeiden zu können, die z.B. bei manuellen Uploads oder Änderungen direkt auf dem Server auftreten können. Mit einem passenden Continuous Delivery Workflow kann man erreichen, dass Änderungen zeitnah und zuverlässig ausgespielt werden können und dass auf einem Server immer ein genau definierter Zustand herrscht. Nicht zuletzt werden dadurch auch die Entwickler entlastet, da sie sich nicht mehr um das Hochladen von Änderungen, Abgleichen unterschiedlicher Datei-Versionen usw. kümmern müssen und sie durch automatisierte Tests und Validierungen frühzeitig Feedback bekommen, sollte es Probleme im Zusammenspiel der Software geben.

Alle neuen Projekte werden seit einigen Monaten bereits mit automatisiertem Deployment und nach dem Grundsatz **„Continuous Delivery“** umgesetzt, ältere bzw. größere laufende Projekte stellen wir nach und nach um.

Entwicklungs-Workflow

Grundsätzlich gilt: jeder Entwickler sollte primär in seiner lokalen Umgebung entwickeln können, wozu Docker [https://www.docker.com/] eingesetzt wird. Der Entwickler arbeitet, testet und debugged lokal auf seinem eigenen Rechner, „committed“ und „pusht“ seine Arbeit anschliessend in unsere **GIT-Versionsverwaltung**, für welche wir einen internen Gitlab [https://about.gitlab.com/] -Server nutzen. Im Docker-Container der Entwickler sind zudem Tools wie Blackfire.io [https://blackfire.io/] oder XHProf [http://php.net/manual/de/book.xhprof.php] sowie Xdebug [https://xdebug.org/] verfübar.

Sobald mit GIT in den sogenannten *„develop“-Branch* gepusht wird, übernimmt unser Gitlab-Server das automatische Deployment auf ein internes Test-System, auf welchem dann Projektleiter oder andere Beteiligte (Grafiker usw.) den aktuellen Stand begutachten können.

Pusht ein Entwickler in den sog. *„stage“-Branch*, wird der Stand, nach optionalem Durchlaufen von Tests usw., auf einen externen Entwicklungsserver gespielt, z.B. auf

einen Kunden-Staging-Server.

Ein Release-Manager kann schliesslich noch in den sog. *„master"-Branch* pushen, dieser wird dann automatisch auf den Live-Server ausgerollt, allerdings wird hier ein Feature von Gitlab CI genutzt, welches es ermöglicht, neue Deployments über die Gitlab-Oberfläche manuell per Button zu „aktivieren" – dazu reicht ein simples

> when:
> – manual

in der Datei *„.gitlab-ci.yml"*.

So kann der genaue Zeitpunkt bestimmt werden, wann der neue Stand online geht und es können vorher z.B. noch Kontrollen durchgeführt werden. Der vorherige Stand wird auf dem Server belassen, so ist ein blitzschnelles **„Rollback"** über die Gitlab-Oberfläche oder die Shell möglich, sollte es doch einmal Probleme mit der neuen Version geben.

Build und Testing

In allen drei Fällen (develop, stage und master Branch) können optional vorab in einem automatisch von Gitlab gestarteten Docker-Container z.B. **Akzeptanz-Tests** usw. mit Codeception [https://github.com/Codeception/Codeception] durchgeführt oder **Coding-Standards** mittels PHP Codesniffer [https://github.com/squizlabs/PHP_CodeSniffer] und ähnlicher Tools geprüft werden etc.
Ausserdem können mit einem auf „ioly [https://github.com/ioly/ioly] " bzw. „OXID Modulconnector" [https://github.com/OXIDprojects/OXID-Module-Connector] basierenden Installer im Shop benötigte Module aus anderen GIT-Repositories oder auch von Drittanbietern automatisch installiert und aktiviert werden.
Schlägt einer der automatisierten Tests fehl, wird das Deployment gestoppt. Sind die Tests erfolgreich, wird der komplette, getestete Shop im Docker-Container als *.tar.gz gepackt.

Ausrollen mit Ansible

Im nächsten Step wird innerhalb des Docker-Containers unser **Ansible-Deployment Repository** aus GIT geclont.

> Ansible [https://www.ansible.com/] ist eine Server-Orchestrierungs/Konfigurations-Software, womit man von einem mit Ansible ausgestatteten Rechner beliebig viele „entfernte" Rechner/Server administrieren kann. Dazu ist auf den Remote-Rechnern keinerlei Client- oder Server-Komponente nötig, es muss lediglich ein Public Key hinterlegt werden. Die Ansible Befehle definiert man in sog. „Playbooks", die einfach im YAML-Format geschrieben werden können.

Unser Ansible-Playbook kann nun den kompletten Shop auf beliebige Server hochladen/verteilen und diverse zusätzliche Tasks ausführen, z.B. *Symlinks anlegen* (die in einer JSON-Datei definiert werden können) sowie *Shell- oder PHP-Befehle ausführen*

(ebenfalls in JSON beliebig definierbar, z.B. Tmp-Verzeichnis leeren, Datenbank-Views neu erzeugen, Module aktivieren usw.).
Abschliessend kann von Ansible automatisch oder auch manuell per Button der Symlink auf das Haupt-Shopverzeichnis geändert und so der neue Stand live geschaltet werden. Wie schon erwähnt ist auch ein schnelles Rollback kein Problem, da der vorherige Stand (sowie einige ältere Stände) noch auf dem Server vorgehalten werden.

In *weiteren Blog-Beiträgen* werden wir sowohl Gitlabl CI näher vorstellen als auch einige Ansible-Features näher beleuchten. Teil 2 zum Thema „Ansible“ jetzt hier lesen [http://www.shoptimax.de/blog/technisches/continuous-delivery-teil-2-ansible-tipps] !

Schlagworte: ansible, blackfire, ci, continuous, deployment, docker, integration, oxid, xdebug, xhprof

Teile diesen Eintrag

0

ANTWORTEN

© Copyright - shoptimax 2017