Zierfisch oder Wal – lokale Software-Entwicklung mit Docker

25. März 2017 / 0 Kommentare / in Allgemein, Technisches / von Stefan Moises

Pünktlich zum 4. Geburtstag von Docker [https://www.docker.com/] möchte ich ein kleines Fazit ziehen, ob und wie sich Docker-Container für die lokale Entwicklung im Agentur-Alltag mit sehr heterogenen Entwicklungs-Systemen (Windows Home / Pro, von 7 bis 10, Mac OS, Linux) der einzelnen Mitarbeiter bewähren können, welche Vorteile, aber auch Schwachstellen und Probleme es ggf. bei der täglichen Arbeit gibt und ob es sich lohnt, auf den blauen Wal zu setzen.

[http://www.shoptimax.de/wp-content/uploads/2017/03/docker_logo.png]

Die Anfänge

Vor knapp zwei Jahren fing ich an, mich erstmals mit Docker auseinanderzusetzen. Damals wie heute ging es dabei zentral darum, mit möglichst wenig Aufwand eine **einheitliche, einfach aufzusetzende lokale Entwicklungsumgebung für verschiedene Software-Projekte** zu haben, die idealerweise systemunabhängig zur Verfügung stehen sollte.

Lösungen mit MAMP [https://www.mamp.info/de/], XAMP, LAMP usw. waren kaum beherrschbar, die Versionen der einzelnen Komponenten (Webserver, Datenbank, PHP-Umgebung, etc.) auf den Systemen der Mitarbeiter wucherten vor sich hin, unterschiedlichste Projekte stellten auch unterschiedlichste Anforderungen (ZendGuard Loader, ioncube Loader, Opcode-Caches, diverse PHP-Versionen, NodeJS, etc. etc.). Oder es wurde gar gemeinsam gleichzeitig auf nur einem Test-System entwickelt, die Mitarbeiter überschrieben sich „fröhlich“ gegenseitig ihre Änderungen. Gab es irgendwo einen Fehler im Code mussten alle warten, bis der Schuldige das Problem behoben hatte und das System wieder lief… kurzum: **totale Anarchie** :)

Für jedes Projekt aber eigene **VMs** zu „basteln“ war ebenso zeitaufwändig wie hardware-hungrig… was also tun? Docker schien eine gute Lösung zu sein, mit Docker Toolbox [https://www.docker.com/products/docker-toolbox] war es auch für Windows und Mac OS verfügbar, es gab vorgefertigte Docker-Images für Apache, PHP, Mysql auf Docker Hub [https://hub.docker.com/] usw.

Allerdings war Docker Compose [https://docs.docker.com/compose/] noch im Entstehen, grafische Tools zur Container-Administration waren Mangelware, einzelne Container wurden also auf der Kommandozeile über *„docker run …“* gestartet. Für „richtige“ Entwickler natürlich ok, für Template- und Frontend-Kollegen eher „so naja“ … ;)
Der Einstieg war also eher holprig, erste Container waren der einfacheren Administration halber „all-in-one“, also entgegen der Docker-Prinzipien liefen Apache, mehrere PHP-Versionen über „phpbrew“, Mysql, NodeJs usw. in einem gemeinsamen Container und waren damit eigentlich schon fast ein „VM-Ersatz“.

Docker ist cool – aber nicht für alle

Damit hatten jetzt aber alle Entwickler eine einheitliche Umgebung – ***build once, run anywhere***!? Mitnichten … damit z.B. ein hochkomplexer OXID [https://www.oxid-esales.com/] EE Shop mit dutzenden Modulen und hunderten Megabyte an Datenbank-Daten auf einem lokalen Entwicklungsrechner „out-of-the-box“ in einem Docker-Container lief, waren erst einmal je nach Projekt teilweise mehrere Tage Setup und Tuning nötig… irgendwie mussten die Module automatisch installiert und aktiviert werden, die Datenbank sollte möglichst automatisch eingespielt werden, diverse Konfigurations-Einstellungen waren anzupassen usw. Hierfür ist z.B. unser „ioly Installer“ [https://github.com/shoptimax/ioly_installer] entstanden, der genau solche Dinge erledigt.
Das wäre natürlich auch für eine „normale“ VM oder ein lokales MAMP o.ä. nötig gewesen, von daher ist Docker hier aussen vor, dennoch muss man sich der Aufwände bewusst werden, will man auf „lokale Entwicklung“ setzen.

War es schliesslich so weit, dass ein solcher Shop nahezu automatisch in einem Docker-Container „zusammengebaut“ war, hiess das allerdings noch lange nicht, dass er auch überall „lief“ … leider gab und gibt es bis heute immer wieder Probleme, die nur auf diesem einen Windows PC (oder Macbook, oder….) mit dieser oder jener Docker Version auftreten… mal machte Docker Toolbox bei einem Mitarbeiter Probleme, dann wieder Docker for Windows bei einem anderen.
Projekt A lief bei Kollege X problemlos, in Projekt B wollte partout der Datenbank-Container nicht starten oder den Dump nicht vollständig einspielen (nachdem wir diesen schliesslich vom Apache/PHP-Container getrennt hatten, um die Daten auch einigermassen persistent zu halten und nicht immer halbstündige DB-Importe laufen zu haben, wenn man die Container neu startet).
Wieder andere hatten Probleme, weil der Prozessor zu schwach oder der Arbeitsspeicher zu gering waren, schliesslich stellt so ein Enterprise Shop mit mehreren Mandanten und Live-Artikeldaten teilweise auch ordentliche Hardware-Ansprüche. Oder Container X wollte einfach nicht mit Container Y kommunizieren… oder Hyper-V wollte erst gar nicht funktionieren. Oder VirtualBox streikte …
So gut das Konzept, so schwierig der tägliche Einsatz bei vielen verschiedenen Mitarbeitern mit noch mehr verschiedenen Systemen, kombiniert mit

unterschiedlichsten Anforderungen der einzelnen Projekte. Erschwerend kommt hinzu, dass Mitarbeiter oft auch an mehreren Projekten pro Tag arbeiten, wofür dann jedesmal eine lokale Umgebung für das Projekt benötigt wird.

Schliesslich stiessen wir noch auf erhebliche Performance-Probleme auf Windows [https://github.com/docker/for-win/issues/188] (und teilweise auch MAC)-Systemen, gerade mit komplexen **Symfony**– bzw. **Shopware**-Projekten. Ein Klick dauerte hier teilweise bis zu 30 Sekunden, um eine Kategorie- oder Detailseite im Shop aufzurufen, was scheinbar am Dateisystem-Handling im Zusammenspiel mit den Docker Volumes [https://docs.docker.com/engine/tutorials/dockervolumes/] liegt.
Deaktivieren von Javascript- und CSS-Minimierung in den Shopware-Theme-Einstellungen im Backend brachten etwas Linderung, dennoch war so an ein einigermassen effizientes Arbeiten nicht zu denken.

Für Mac OS gab es hier mit **„Docker Sync" [http://docker-sync.io/]** einen guten Workaround, für Windows zwar unzählige Foren-Posts und Blogeinträge zum Thema, aber keine „native", absehbare Lösung. Was nun?

Zurück zur VM?

Gut, dachten wir uns… wenn Docker also auf Nicht-Linux-System ein Problem hat, performant mit vielen Dateiänderungen in Volumes umzugehen, dann könnte man doch eine Linux-VM nehmen und in dieser VM einfach „wirklich natives Docker" nutzen.
Also ähnlich wie Docker Toolbox, aber nicht mit regulären Docker Volumes auf dem Host-System, sondern mit einem Verzeichnis in der VM, das über das **Samba**-Protokoll mit dem Host verbunden ist.
Das Docker-Volume könnte dann also komplett in der VM bleiben, das gemeinsame Verzeichnis wird „nach aussen" als Netzlaufwerk verbunden und das Performance-Problem wäre gelöst!
Extra-Bonus: jeder Entwickler hat endlich ein einheitliches Docker-System in der Linux VM – kein Docker for WIN/MAC oder Toolbox mehr, kein Support für verschiedenste System und Docker-Varianten mehr! Das klang zu gut, um wahr zu sein :)

Natürlich gilt wie immer – wo Licht, da auch Schatten … auch bei diesem Ansatz stiessen wir auf einige Schwachstellen und Probleme.
Eine **Schwachstelle** ist z.B., dass man in einem gemounteten Netzlaufwerk nicht vernünftig arbeiten kann – weder eine IDE wie PHPStorm oder Netbeans noch ein Git-Client wie Sourcetree oder SmartGit arbeiten hier performant oder zuverlässig, die Hersteller selbst raten eindeutig davon ab.
Einzige Lösung also – die Projekte *zweifach* aus GIT auschecken, einmal in einem lokalen Arbeitsverzeichnis, einmal im „geteilten" Verzeichnis der VM. Dann die IDE so konfigurieren, dass beim Speichern von Dateien die Änderungen automatisch in das verbundene Netzlaufwerk kopiert werden. Funktioniert zwar und die Performance ist damit wirklich um einiges besser als mit „nativem" Docker für Windows oder MAC, elegant ist allerdings etwas anderes :)

Schliesslich schafften es Entwickler, nach kurzer Zeit gar die VM komplett „kaputt" zu machen und mussten sie neu einrichten. Dann wieder funktionierte Samba nicht mehr richtig, Dateien wurden nicht mehr gespeichert, Netzlaufwerke mussten nach einem Neustart wieder neu verbunden werden, kurz: der Support-Aufwand verschob sich vom

Docker-Support hin zu VirtualBox-, Samba-, Linux- und IDE-Support … die Performance war zwar besser, aber das Handling und der Support-Aufwand eher suboptimal.

Der Hybrid

Das VM-Experiment war also mehr oder weniger gescheitert und das Performance-Problem drückte. Bei der Recherche bezüglich der Performance-Probleme mit Volumes speziell auf Windows stiessen wir dann auf Teracy-Dev [http://blog.teracy.com/2016/12/20/teracy-dev-the-only-truly-universal-productive-development-platform-with-docker-on-macos-linux-and-windows/] , ein Tool, das in eine ähnliche Richtung ging wie unser VM-Konzept, zusätzlich aber Vagrant inkl. „rsync"-Plugin nutzte… hatten wir den „Heiligen Gral" gefunden?

> „Teracy-dev – the Only Truly Universal Productive Development Platform With Docker on macOS, Linux and Windows"

Das klang doch schon wieder viel zu gut, um wahr zu sein … oder?

Und weiter, und das war uns wirklich aus dem Herzen gesprochen:

> „Docker works great on Linux, however, it's very challenging to make it work universal and consistent on Mac and Windows. There are lots of efforts to solve this problem, from Docker themselves and from Docker community, too. However, we haven't achieved that stage yet"

Und tatsächlich, mit kleineren Abstrichen hält das Konzept bisher, was es verspricht – es funktioniert, löst die Performance-Probleme auf Windows und MAC und dank „vagrant-gatling-rsync" [https://github.com/smerrill/vagrant-gatling-rsync] landen die Dateien ohne Umweg über Samba und gemountete Netzwerk-Verzeichnisse direkt vom Host in der VM und damit auch in den gestarteten Docker-Containern!
Man muss ggf., wenn man in Docker selbst wichtige Dateien hinzufügt, manuell das „rsync-back"-Plugin [https://github.com/smerrill/vagrant-rsync-back] für Vagrant bemühen, damit die Daten auch zurück aus dem Container auf den Host kopiert werden – dieses ist aber bereits im Teracy-Dev Toolset integriert und somit problemlos nutzbar. Das Teracy-Dev Team reagiert auch sehr schnell auf Anfragen und Pull-Requests, daher gibt es von uns eine eindeutige Empfehlung, das „Teracy-Dev" Projekt einfach einmal zu testen, v.a. natürlich wenn man unter denselben Wehwehchen mit Docker in heterogenen Teams leidet wie wir :)

Natürlich gibt es auch hier Nachteile, die Entwickler müssen nun auch Vagrant Basics beherrschen und es fällt damit sicher nun auch etwas zusätzlicher Vagrant-Support an, dennoch macht die einheitliche Docker-Umgebung in der VM und die gute Performance auf MAC und Windows vieles wett und ist eindeutig ein Fortschritt.

Fazit

Arbeiten in lokalen Entwicklungsumgebungen hat viele Vorteile, jeder Entwickler kann für sich neue Features entwickeln, in Ruhe Probleme debuggen usw. Allerdings sollte man sich bewusst sein, dass dadurch der Einrichtungs- und Supportaufwand für die

einzelnen Mitarbeiter durch System-Administratoren, das Dev-Ops Team oder wer auch immer sich um die Betreuung der Entwicklungsumgebungen kümmert, initial immens ansteigen kann.
Letztlich führt allerdings in einem Team mit mehreren Entwicklern kein Weg an unabhängiger, lokaler Entwicklung vorbei, wodurch jeder einzelne ungestört in seinem System arbeiten kann. Nicht zuletzt spart man sich mittelfristig natürlich auch viel Zeit, die durch Probleme beim gemeinsamen Arbeiten auf einem einzigen, gemeinsamen Testsystem z.B. definitiv verloren geht.

Man sollte aber idealerweise sicherstellen, dass die Entwickler **möglichst homogene Entwicklungs-Systeme** haben, die einigermassen **aktuell und hardwaremässig entsprechend gut ausgestattet** sind. Und dass man zumindest initial **genügend personelle Ressourcen** hat, um sich um das optimale Setup der Docker-Images zu kümmern und um die Entwickler bei ihren ersten Schwimmversuchen mit dem blauen Wal zu unterstützen :)

Docker selbst hat natürlich in anderen Bereichen viele, viele weitere Vorteile, auf die wir hier nicht eingehen konnten, eine lokale Entwicklungsumgebung aufzusetzen ist schliesslich nur einer von vielen Anwendungsbereichen.

Schlagworte: compose, docker, hyperv, lamp, mac, mamp, oxid, performance, php, samba, shopware, symfony, teracy, toolbox, virtualbox, vm, windows, xamp

Teile diesen Eintrag

0

ANTWORTEN