Fullstackentwicklung mit MacOS

Bastian Nolte

18. März 2019

1 Einleitung

Dieser Artikel beschreibt die Installation und Konfiguration von verbreiteten Werkzeugen zur Fullstackentwicklung unter MacOS, am Beispiel von Java und JavaScript/TypeScript.

2 Vorbedingungen

Es wird ein Computer mit MacOS als Betriebssystem, sowie ein grundlegendes Verständnis dieses Betriebssystems vorausgesetzt.

Inhaltsverzeichnis

3 Xcode

Xcode ist eine integrierte Entwicklungsumgebung für macOS mit einer Reihe von Softwareentwicklungswerkzeugen, die von Apple für die Entwicklung von Software für macOS, iOS, watchOS und tvOS entwickelt wurden.

Installiere Xcode über den AppStore von Apple.

3.1 Xcode command line tools

Xcode verfügt auch über einen Satz von Kommandozeilenwerkzeugen. Diese installierst Du am besten über das Terminal. Das Terminal kannst Du öffnen, indem Du ⌘+Leertaste drückst und dann `terminal` eingibst und ⏎ drückst.

Danach kann die Installation der `Xcode command line tools` wie folgt durchgeführt werden.

```
xcode-select --install
```

4 Paketmanager

MacOS bringt keinen eigenen Paketmanager mit, wie man ihn von anderen Unix-Derivaten kennt. Daher haben es sich verschiedene Projekte zur Aufgabe gemacht, diese Lücke zu füllen.

Die folgende Liste liefert einen Überblick aktueller Paketmanager im März 2019.

Mac Paketmanager	Pakete	Benötigt Rootrechte
Homebrew	4635	Nein
Homebrew Cask	4051	Nein
Nix package manager	15858	Nein
pkgsrc	18560	Ja
MacPorts	20572	Ja

Die Informationen wurden aus dem Artikel What are the best Mac package managers? der Seite Slant bezogen.

4.1 Homebrew installieren

Homebrew kann einfach über das Terminal installiert werden.

Die Installation erfolgt durch Eingabe des folgenden Kommandos im soeben geöffneten Terminal:

```
HOMEBREW_URL='https://raw.githubusercontent.com/Homebrew/install/master/install'
/usr/bin/ruby -e "$(curl -fsSL $HOMEBREW_URL)"
```

4.2 Die wichtigsten Homebrewkommandos

```
# Nach einem bestimmten Paket suchen
brew search <paketname>

# Das OpenSource-Paket mit dem Namen <paketname> installieren
brew install <paketname>

# Ein nicht OpenSource-Paket installieren,  z.B. kommerzielle MacOS-Apps.
brew cask install <paketname>

# Alle bereits installierten Pakete und deren mitinstallierte Abhängigkeiten
# in einer Baumansicht auflisten.
brew deps --tree --installed

# Alle installierten Pakete in einer flachen Ansicht auflisten
brew list && brew cask list
```

```
# Informationen zu einem bereits installierten Paket erhalten
brew info <paketname>
```

Weitere Informationen findest Du auf der Homepage des Homebrew-Projektes.

5 iTerm2

iTerm2 ist ein Open-Source-Ersatz für Apples Terminal. Es ist sehr anpassungsfähig und verfügt über viele nützliche Funktionen.

Du kannst es wie folgt installieren:

```
brew cask install iterm2
```

Du startest iTerm2 indem Du ⌘+Leertaste drückst und dann `iterm` eingibst und ↩ drückst. Eine Darstellung der Vorteile, die sich durch den Einsatz von `iTerm` ergeben, findest Du auf der Featureseite von iTerm.

5.1 Tastenkombinationen anpassen

Wahrscheinlich kennst Du die Tastenkombinationen mit denen Du wortweise (⌥) vor- und zurückspringen, beziehungsweise zum Anfang und Ende der Zeile (⌘) navigieren kannst.

Auch iTerm kannst Du so einstellen. Öffne dazu die iTerm2-Einstellungen (⌘+,) und navigiere dann zu `Profile` > `Schlüssel` und klicke auf das Symbol + um eine neue Tastenkombination hinzufügen.

Keyboard Shortcut	Action	Send
⌘←	Send escape sequence	OH
⌘→	Send escape sequence	OF
⌥←	Send escape sequence	b
⌥→	Send escape sequence	f

Nun kannst Du wortweise vor- und zurück, sowie an den Anfang und das Ende der Zeile navigieren.

6 Z-Shell

Die Z-Shell, die Du vielleicht unter dem Namen zsh kennst, ist eine Unix-Shell, die auf der Standard-Shell für macOS aufbaut und diese um weitere nützliche Funktionen erweitert.

Installieren kannst Du die zsh durch Eingabe des folgenden Befehls:

```
brew install zsh
```

Du solltest ein Framework mit der zsh installieren, da es die Konfiguration und den Einsatz von Plugins und Themes deutlich erleichtert. Es gibt zwei besonders populäre Frameworks, Oh My Zsh und Prezto. Bitte beachte, dass Du nur ein Framework und nicht beide installieren solltest, da Du sonst mit Problemen rechnen musst.

6.1 Oh My Zsh

Ich habe mich dafür entschieden `Oh My Zsh` einzusetzen. Um es zu installieren, gibst Du folgenden Befehl in Dein Terminal ein.

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/robbyrussell\
/oh-my-zsh/master/tools/install.sh)"
chsh -s $(which zsh) # Setzt die zsh als Standardshell.
```

Ein Upgrade kannst Du mit dem Kommando `upgrade_oh_my_zsh` durchführen. Starte `iTerm2` neu, um zsh zu aktivieren.

6.2 Oh My Zsh anpassen

Die Einstellungen von `Oh My Zsh` nach der Installation sind schon recht brauchbar. Es bietet Dir allerdings diverse Möglichkeiten sein Aussehen und Verhalten an Deine Anforderungen anzupassen. Wie das funktioniert kannst Du im Wiki nachlesen.

Wir wollen für den Anfang einige Plugins installieren. Öffne hierzu die Datei `~/.zshrc`, finde die Zeile die mit `plugins=` beginnt und trage Plugins wie folgt ein:

```
plugins=(
  colored-man
  colorize
  docker
  git
  github
  history
  mvn
  ng
  node
  npm
  osx
  zsh-syntax-highlighting
)
```

Ob Du alle Plugins in eine Zeile schreibst oder jedes in eine eigene Zeile, ist unerheblich. Wichtig ist nur, dass Du die Plugins durch mindestens ein Leerzeichnen voneinander trennst.

Zudem könnte es sich als praktisch erweisen, in der Eingabehistorie weiter zurückblicken zu können. Dies kannst Du erreichen, indem Du Deiner Konfigurationsdatei folgende Zeilen hinzufügst.

```
export HISTSIZE=10000
export HISTFILESIZE=10000
```

Die Änderungen werden wirksam, sobald Du iTerm neu gestartet hast.

7 Secure Shell

Secure Shell oder SSH bezeichnet sowohl ein Netzwerkprotokoll, als auch entsprechende Programme, mit deren Hilfe man auf eine sichere Art und Weise eine verschlüsselte Netzwerkverbindung mit einem entfernten Gerät herstellen kann. Häufig wird diese Methode verwendet, um lokal eine entfernte Kommandozeile verfügbar zu machen, das heißt, auf einer lokalen Konsole werden die Ausgaben der entfernten Konsole ausgegeben und die lokalen Tastatureingaben werden an den entfernten Rechner gesendet. Genutzt werden kann dies beispielsweise zur Fernwartung eines in einem entfernten Rechenzentrum stehenden Servers.[1]

7.1 Einen SSH-Schlüssel erzeugen

Ein SSH-Schlüssel besteht aus einem Paar von Dateien. Eine davon beinhaltet den privaten Schlüssel, der niemals an Dritte weitergegeben werden sollte. Die andere beinhaltet den öffentlichen Schlüssel. Diesen öffentlichen Schlüssel kannst Du dazu verwenden Dich an Systemen anzumelden, ohne Benutzername und Passwort eingeben zu müssen.

Ein Schlüsselpaar kann wie folgt erzeugt werden:

```
ssh-keygen -t rsa
```

7.2 Den öffentlichen Schlüssel in die Zwischenablage kopieren

```
pbcopy < ~/.ssh/id_rsa.pub
```

7.3 Den öffenlichen Schlüssel übertragen

Überträgt man den öffentlichen SSH-Schlüssel auf einen anderen Computer, z.B. einen Server und hinterlegt diesen dort, dann kann man sich künftig an diesem Server anmelden, ohne Benutzernamen und Passwort eingeben zu müssen.

Der Schlüssel kann mit folgendem Befehl übertragen und hinterlegt werden:

```
# ssh-copy-id -i ~/.ssh/key_rsa.pub <benutzername>@<servername oder ip>, z.B.
ssh-copy-id -i ~/.ssh/key_rsa.pub musterfrau@webserver.example.com
```

7.4 SSH-Schlüssel, Phassphrasen und der MacOS-Schlüsselbund

Ich empfehle Dir Deine SSH-Schlüssel mit einer Passphrase zu versehen. Wenn Du darauf verzichtest, kann sich jede Person die in den Besitz Deines privaten Schlüssels gelangt, mit Deinem Benutzeraccount anmelden und das an allen Systemen auf denen Du Deinen Schlüssel hinterlegt hast.

Normalerweise müsstest Du dann aber bei jeder Verwendung des Schlüssels Deine Passphrase eingeben. Dies kann schnell anstrengend werden oder - zum Beispiel bei der automatisierten Kommunikation zwischen zwei Computern - sogar unmöglich sein.

Dies kannst Du vermeiden, indem Du Deine privaten Schlüsselidentitäten dem SSH-Authentifizierungsagenten hinzufügst und die Passphrasen im Schlüsselbund Deines MacOS-Benutzers hinterlegst.

```
# ssd-add fügt Deine SSH-Schlüsselidentitäten
# dem SSH authentication agent hinzu
# -K sorgt dafür, dass Deine Passphrasen in Deinem Schlüsselbund hinterlegt werden.
ssh-add -K
```

SSH-Schlüssel kannst Du auch nachträglich mit einer Passphrase versehen.

```
ssh-keygen -p
```

7.5 Individuelle Schlüssel pro Server oder Domäne

Ich persönlich benutze gerne individuelle Schlüssel für Dienste unterschiedlicher Anbieter, zum Beispiel einen eigenen Schlüssel für `github`.

Du kannst dies realisieren, indem Du im Verzeichnis `~/.ssh` eine Datei mit dem Namen `config` erstellt. In dieser Datei kannst Du die Domänen- bzw. Hostnamen den jeweiligen Schüsseln zuordnen.

```
UseKeychain yes

Host github.*
IdentityFile ~/.ssh/github_id_rsa

Host *.github.*
IdentityFile ~/.ssh/github_id_rsa

Host *.bergsee.consulting
IdentityFile ~/.ssh/bergsee_id_rsa
```

```
Host *
IdentityFile ~/.ssh/id_rsa
```

8 Git

Git ist eine freie Software zur verteilten Versionsverwaltung von Dateien.

Installiere Git durch Eingabe des folgenden Kommandozeilenbefehls.

```
brew install git
```

Weitere Informationen zu Git, findest Du im umfassenden und kostenlosen Buch Pro Git auf der Projekt Homepage, das auch in deutscher Sprache angeboten wird.

8.1 Eine globale .gitignore-Datei anlegen

Erzeuge eine `.gitignore`-Datei in Deinem Nutzerverzeichnis `~` die Du global nutzen wirst. Diese Datei könnte zum Beispiel, wie folgt aussehen:

```
# Dokumentation: http://help.github.com/ignore-files/
# Beispiele für diverse Tools und Sprachen: https://github.com/github/gitignore

# MacOS: Konfigurationsdateien für die Verzeichnisansicht
.DS_Store
Desktop.ini
.AppleDouble
.LSOverride

# MacOS: Cache-Dateien für Miniaturansichten
._*
Thumbs.db

# Dateien, die in Wurzelvereichnissen auftauchen können
.DocumentRevisions-V100
.fseventsd
.Spotlight-V100
.TemporaryItems
.Trashes
.VolumeIcon.icns
.com.apple.timemachine.donotpresent

# Caches,Backups und Temporärdateien
*.tmp
*.bak
*.swp
tmp/

# IntelliJ-IDEA
.idea/
/**/*.iml
```

```
atlassian-ide-plugin.xml

# Eclipse-IDE
/**/.classpath
/**/.project
/**/.settings/
*.launch
.factorypath
*~.nib

 # Visual Studio Code IDE
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
!.vscode/launch.json
!.vscode/extensions.json

# node_modules - Über den Node Package Manager oder yarn verwaltete Abhängigkeiten
/node_modules

# Cache-Dateien des SASS-Compiler
.sass-cache/

# Maven-Kompilate
target/

# Code coverage Informationen
coverage/

 # Latex / Tex
*.aux
*.bbl
```

Registriere die Datei in Deiner globalen Git-Konfiguration, um die Einstellungen wirksam werden zu lassen.

```
git config --global core.excludesfile ~/.gitignore
```

9 Maven

Apache Maven ist ein in Java geschriebenes Build-Management-Werkzeug. Maven versucht das Pattern "Konvention vor Konfiguration"konsequent für den gesamten Lebenszyklus einer Software abzubilden. Neben der Verwaltung von Abhängigkeiten, wird sowohl das Kompilieren und Packen der Software, als auch die Verteilung unterstützt.

9.1 Alternativen

Neben Maven existieren noch diverse weitere Build-Management-Werkzeuge. Einen guten Überblick gibt der Artikel Build-Tools im Vergleich der Zeitschrift jaxenter.

Ich persönlich denke, dass (Stand 2019) Gradle besonders gut für die Arbeit in Java-Projekten geeignet ist. Gradle versucht die guten Seiten von Ant und Maven zu vereinen und ist dabei so

erfolgreich, dass Google es als primäres Build-Tool bei der Entwicklung von Android verwendet. Statt XML nutzt Gradle eine domänenspezifische Sprache (DSL), die auf Groovy basiert. Daher gibt es die Tendenz, dass Gradle-Skripte häufig wesentlich kürzer und klarer formuliert sind, als jene für Ant oder Maven.

9.2 Maven installieren

Maven kann mit Homebrew wie folgt installiert werden.

```
brew install maven
```

10 npm

npm ist eines der führenden Build-Management-Werkzeuge in der JavaScript-Welt. Ähnlich wie Maven in der Javawelt, ist es hier sehr verbreitet.

10.1 Alternativen

Aufgrund verschiedener Probleme, unter anderem mit der Verwaltung von indirekten Abhängigkeiten und wegen teils schlechter Performance, erlangte ein neues Build-Management-Werkzeug namens Yarn das Interesse der Entwicklergemeinde. Viele Gründe, die für den Wechsel auf Yarn sprachen, sind in der aktuellen Version von npm nun aus der Welt geschaffen. Es findet eine Rege Diskussion in der Entwicklergemeinde darüber statt, welches der beiden Tool nun das bessere sei.

10.2 npm installieren

npm kann mit Homebrew wie folgt installiert werden.

```
brew install npm
```

11 Entwicklungsumgebung

Ich benutze die kommerzielle Entwicklungsumgebung IntelliJ IDEA Ultimate für die Fullstackentwicklung mit Java, ECMAScript (auch JavaScript) und TypeScript.

Der Superuser Deines Teams kann für Dich anfragen, ob es möglich ist, Dir eine Lizenz auszustellen.

11.1 Alternativen zu IntelliJ IDEA

Es gibt diverse alternative IDEs, die Du einsetzen kannst. Da Build-Prozess, sowie Deployment und Delivery unabhängig von der Entwicklungsumgebung funktionieren, kannst Du eigentlich jede IDE einsetzen die Dir gefällt.

Populäre Alternativen für die Java-Entwicklung sind zum Beispiel die Eclipse IDE und die Netbeans IDE.

Visual Studio Code erfreut sich in den letzten Jahren wachsender Beliebtheit.

Ich bin überzeugter Anwender der Jetbrains-Toolchain. Das muss aber nicht bedeuten, dass es nicht für Dich eine andere IDE geben kann, die besser zu Deiner Arbeitsweise passt.

Eine gute Übersicht an IntelliJ-Alternativen bietet die Seite Slant - What is the best alternative to IntelliJ IDEA?

11.2 IntelliJ IDEA installieren

```
brew cask install intellij-idea
```

11.3 Formattierung und Zeichenkodierung

Egal für welche IDE Du Dich entschieden hast, Dein Team wird es Dir danken, wenn Du jederzeit sicherstellst, dass Du die Formatierungs-Regeln Deiner Firma bzw. Deines Teams importierst, aktivierst und die Zeichencodierung (encoding) auf UTF-8 stellst. Diese Massnahmen werden es Deinen Kollegen erleichtern Deine Codeänderungen zu reviewen oder im allgemeinen festzustellen, was Du mit einem git-commit geändert hast.

Aktuelle Projekte beinhalten oft eine .editorconfig-Datei aus, die von den meisten aktuellen IDEs berücksichtigt wird. Für ältere Projekte musst Du wahrscheinlich eine Formatierungsdatei im XML-Format (meist Eclipse) importieren.

12 Google Chrome

Google Chrome ist zurzeit der am weitesten verbreitete Browser. Es bringt zudem gleich Tools für die Webentwicklung mit, die Du mit der <F12>-Taste öffnen kannst.

Installiere Google Chrome wie folgt:

```
brew cask install google-chrome
```

13 REST-Services testen mit Postman

Die Postman-App macht es einfach REST-API zu testen.

Du kannst Postman wie folgt installieren:

```
brew cask install postman
```

14 Entwicklungs-Toolchain für SPA

...

Die populärsten Frameworks für die Erstellung von SPA sind aktuell Angular, React und Vue.

14.1 angular-cli installieren

Das Angular Command Line Interface (angular-cli) unterstützt Dich dabei neue Angular-Anwendungen zu erstellen, die den Best Practices des Angular-Entwickler-Teams genügen. Zudem kann angular-cli Komponenten, Routen, Dienste und Pipes für Dich erstellen.

```
brew install node
brew cask install angular-cli
```

15 vim

Vim ist ein hochgradig konfigurierbarer und erweiterbarer kommandozeilenbasierter Texteditor, der auf nahezu jedem Unixderivat vorgefunden werden kann. Die Bedienung wird von Nutzern, die es gewohnt sind auf grafischen Systemen zu arbeiten, häufig als gewöhnungsbedürftig oder gar kompliziert wahrgenommen.

Hat man die anfänglichen Hürden überwunden, so ermöglicht er komplexe Textmanipulationen.

Auf MacOS ist vim bereits installiert. Ein einfacher Weg, eine sinnvolle Grundkonfiguration vorzunehmen, ist die Verwendung von The ultimate Vim configuration: vimrc.

Die Konfiguration kann wie folgt angestossen werden:

```
git clone https://github.com/amix/vimrc.git \~{}/.vim\_runtime
sh \~{}/.vim\_runtime/install\_awesome\_vimrc.sh
```

16 Kommandozeilenbefehle

Die Kommandozeile auf unixähnlichen Betriebssystemen ist sehr mächtig. In Folge eine Auswahl von Kommandozeilenbefehlen, die ich besonders nützlich finde:

16.1 Dokumentation von Kommandozeilenbefehlen

Schaue die Dokumentation zu einem Kommandozeilenbefehl an

```
man <kommandozeilenbefehl>
```

16.2 Aktuellen Nutzer ausgeben

```
whoami
```

16.3 Aktuellen Pfad (Verzeichnis) ausgeben

```
pwd
```

16.4 In ein Verzeichnis wechseln

Wechsle in das Verzeichnis mit dem Namen `tmp`.

```
cd /tmp
```

Wechsle in das Nutzerverzeichnis (Heimatverzeichnis)

```
cd ~
```

Wechsle zurück in das vorherige Verzeichnis

```
cd -
```

Tip: Nutze <tab> und <tab><tab> für automatische Ergänzungen.

16.5 Verzeichnis erstellen

```
mkdir <verzeichnisname>
```

16.6 Datei oder Verzeichnis verschieben

```
mv <quelle> <ziel>
```

16.7 Datei oder Verzeichnis kopieren

```
cp <quelle> <ziel>
```

16.8 Zeige den Inhalt einer Datei an

```
# cat <dateiname>, z.B.
cat /var/log/meinserver.log
```

16.9 Datei überwachen und auf der Konsole ausgeben

Die neuesten Inhalte einer Datei kannst Du wie folgt ausgeben:

```
tail -f /var/log/messages

# Mit Unterstützung von Logrotate
tail -F /var/log/messages

# Nur Zeilen ausgeben, die das Wort "warning" enthalten
tail -F /var/log/messages | grep 'warning'

# Nur Zeilen ausgeben, die das Wort "warning" enthalten und zusätzlich in Datei schreiben
tail -F /var/log/messages | grep 'warning' | tee /log/warnings-only.log
```

16.10 Texte ersetzen (substituieren)

Alle Vorkommnisse eines regulären Ausdruckes in einer Datei ersetzen (substituieren), wobei die ursprüngliche Zustand in einer Kopie erhalten bleibt.

```
# sed -i.bu 's/<vorher>/<nachher>/g' <dateiname>, z.B.
sed -i.bu 's/Vorher/Nachher/g' ./test.txt
```

Eine kompakte Einführung in die Welt der regulären Ausdrücke findest Du im Tutorial: Reguläre Ausdrücke von Daniel Fett.

16.11 Finde alle Dateien mit bestimmten Merkmalen

Beispiel: Dateien die in den letzten 10 Minuten geändert wurden.

```
find . -mmin -10
```

`find` ist ein sehr mächtiges Werkzeug. Es lohnt sich auf jeden Fall ein Blick in die zugehörige Manpage mit `man find`.

16.12 Die Befehlshistorie anzeigen

```
history
```

Eine bestimmte Zeile aus der Historie nochmals ausführen

```
# !<Nummer>, z.B.
!42
```

Den letzen Befehl nochmals als root ausführen

```
sudo !!
```

Literatur

[1] *Secure Shell*. Wikipedia, Die freie Enzyklopädie. 2004. URL: https://de.wikipedia.org/wiki/Secure_Shell (besucht am 26. 02. 2019).