# Voordelen van overstappen

## Hergebruik van code

Bij het werken in het Clojure-ecosysteem kun je de kennis van de programmeertaal hergebruiken in meerdere omgevingen. Voor server side ontwikkeling maak je gebruik van Clojure op de JVM. Voor front end development kun je gebruik maken van ClojureScript, wat compileert naar JavaScript. De ClojureScript-compiler kan naast JavaScript voor de browser ook JavaScript genereren voor NodeJS. De implementaties van Clojure op diverse omgevingen zijn zoveel mogelijk hetzelfde gehouden en wijken alleen als dat afgedwongen wordt door beperkingen van de omgeving, bijvoorbeeld de afwezigheid van multithreading in een browser. Naast de JVM en de browser wordt ook het .NET-platform bediend met ClojureCLR.

Veel Clojure-libraries bieden ondersteuning voor zowel Clojure als ClojureScript.

Dus ook de kennis die je hebt van libraries kun je hergebruiken over omgevingen heen.

## Minder complexiteit

Een van de doelen van Clojure is om software-ontwikkeling eenvoudiger te maken. In Clojure worden objecten gerepresenteerd door immutable hashmaps in plaats van classes zoals in Java. Ze zijn eenvoudig te manipuleren met functies. De filosofie van Clojure sluit aan bij een uitspraak van Alan J. Perlis: "It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures."Als je steeds dezelfde datastructuren hergebruikt, hoef je immers niet steeds het wiel opnieuw uit te vinden om je data in een gewenste vorm te krijgen. Dit resulteert in aanzienlijk minder code dan wanneer je voor elk soort object een eigen class met allerhande methodes zou schrijven. Een klein voorbeeld. In de [Ring](https://github.com/ring-clojure) library, waarmee je webapplicaties kan schrijven, is een handler niks anders dan een functie die een hashmap verwacht (het request) en een hashmap teruggeeft (de response):

(defn what-is-my-ip [request]

{:status 200

:headers {"Content-Type" "text/plain"}

:body (:remote-addr request)})

De namespace waarin we deze handler definieren heeft geen afhankelijkheid van de Ring-library. Het enige wat we hoeven te weten is hoe een request er in Ring uitziet, dus, welke keys we in de hashmap kunnen verwachten en welke keys we terug moeten geven om een geldige response te kunnen vormen. Alles wat tussen request en response gebeurt is niets anders dan rekenen met normale hashmaps.

## Directe feedback

Clojure is een interactieve programmeertaal. Via een REPL (read-eval-print loop) krijg je direct feedback. Tijdens het ontwikkelen kun je functies herdefiniëren en opnieuw testen terwijl de applicatie draait, zonder een nieuwe compileercyclus te starten. Een plugin zoals JRebel voor hot code reloading is dus niet nodig. Het is zelfs mogelijk om over het netwerk in te loggen op een REPL-sessie en zo de stand van zaken te inspecteren op een productieserver. Ook ClojureScript kent het concept van een REPL waarmee je code in je browser (of NodeJS) kan testen en herdefiniëren. Een tool als [Figwheel](https://github.com/bhauman/lein-figwheel) maakt het tevens mogelijk om gewijzigde ClojureScript meteen in je browser te zien, zonder dat je de applicatie in je browser moet verversen. Dit alles maakt Clojure uitermate geschikt voor live coding. Je kan met Clojure zelfs live muziek maken. Bekijk hiervoor de library [Overtone](http://overtone.github.io/).

# Full Stack Clojure

Nu Clojure na acht jaar een volwassen ecosysteem heeft, is full stack web development in Clojure heel goed mogelijk. Zeker nu de tools en libraries rondom het ruim vier jaar oude ClojureScript sterk van kwaliteit zijn verbeterd.

## React

Met de komst van JavaScript library React is het bouwen van een SPA (single page app) in ClojureScript een fluitje van een cent. Reagent is een ClojureScript library die het schrijven van React-componenten vergemakkelijkt. Door middel van Hiccup-notatie, waarmee je uit een geneste Clojure-datastructuur HTML kan beschrijven en reactieve atoms, kun je met relatief weinig code een React-component schrijven. Dit is een voorbeeld van een component wat het aantal clicks telt op een knop en dit toont in een div: 

(def count-state (atom 10))

(defn counter []

[:div

@count-state

[:button {:on-click #(swap! count-state inc)}

"x"]])

(reagent/render-component [counter]

(js/document.getElementById "app"))

## Callback hell

Een van de problemen waar JavaScript-developers mee worstelen is het fenomeen callback hell. Een browser heeft maar één thread. Daarom moeten we werken met geneste callbacks. Als we callbacks diep nesten, wordt de code steeds slechter leesbaar. Clojure geeft hiervoor een oplossing in de vorm core.async library welke het werken met asynchrone code vereenvoudigd. Uiteraard werkt ook deze library op server en client. De bouwblokken van core.async zijn channels, buffers en go-blocks. Go-blocks zijn source-transformaties die de illusie geven van synchrone code, maar uiteindelijk geneste callbacks opleveren. Een voorbeeld. De volgende code haalt eerst het e-mailadres op van een gebruiker via een REST-API call. Hiervoor gebruiken we de library [http-cljs](https://github.com/r0man/cljs-http) welke het maken van ajax calls combineert met core.async. Als resultaat van een http-call krijgen een een channel terug, waar we het resultaat uit kunnen lezen met de <! (take) operatie. Vervolgens gebruiken we het e-mailadres in een tweede call om de bestellingen van deze gebruiker op te halen. Als uiteindelijk resultaat geven we het aantal bestellingen terug.

(go (let [email (:body

(<! (http/get

(str "/api/users/"

"123"

"/email"))))

orders (:body

(<! (http/get

(str

"/api/orders-by-email/"

email))))]

(count orders)))

## Reader conditionals

In de laatste stabiele versie van Clojure (1.7, uitgebracht op 30 juni 2015), werden reader conditionals toegevoegd. Dit geeft de mogelijkheid om binnen één bestand meerdere platformen te kunnen bedienen. Bij een Full Stack Clojure applicatie kun je hierdoor een gedeelte van je code hergebruiken op client en server. Een voorbeeld. De volgende code definieert een variabele met de waarde NaN. Op het Clojure/JVM-platform is dat Double/NaN en in JavaScript NaN uit de globale namespace.

(def not-a-number

#?(:clj Double/NaN

:cljs js/NaN

:default nil))

## Validatie

Op diverse punten in een full stack Clojure applicatie is het wenselijk om data te kunnen valideren, bijvoorbeeld voordat je data opslaat in een database. Een veelgebruikte library hiervoor is [Prismatic/Schema](https://github.com/Prismatic/schema). Met Schema beschrijf je zogenaamde data shapes. Het volgende fragment toont een data shape voor een persoon die een (geheel) aantal computers bezit:

(def Person

"A schema for a person"

{:name s/Str

:computers s/Int})

Een validatie hiervan ziet er als volgt uit. Als de validatie slaagt, dan wordt de gevalideerde data zelf opgeleverd. Indien de validatie niet slaagt, wordt er een Exceptie gegooid welke data bevat over de validatie:

(s/validate

Person

{:name "Rich"

:computers 1})

;; ok

(s/validate

Person

{:name :rich

:computers 1.33})

;; niet ok

ExceptionInfo Value does not match schema:

{:name (not (instance? java.lang.String :rich)), :computers (not (integer? 1.33))}

De validaties kunnen ook genest worden. Schema ondersteunt ook coercions, zodat je tijdens het valideren data om kan zetten naar het juiste type.

## Datomic

Persistentie kun je in een Clojure-applicatie uiteraard regelen met een database naar keuze. Gebruik je JDBC, dan is clojure.java.jdbc de aangewezen library om met je database te werken.

Een voor de hand liggende keuze binnen een full stack Clojure applicatie is [Datomic](http://www.datomic.com/). Datomic is een database die geen data overschrijft maar alleen nieuwe data bijschrijft en rekening houdt met de dimensie tijd. Je kan bijvoorbeeld een query uitvoeren tegen de meest recente toestand van de database of de toestand van vorige week. Ook Datomic omarmt het werken met simpele datastructuren.

Het schema, query input en output en transactie-metadata worden beschreven in hashmaps, sequences, etc. waarmee je vanuit je Clojure-programma altijd al werkt. Dit maakt het programmatisch schrijven van queries een stuk plezieriger en eenvoudiger dan het werken met stringmanipulatie om een SQL-query te vormen.