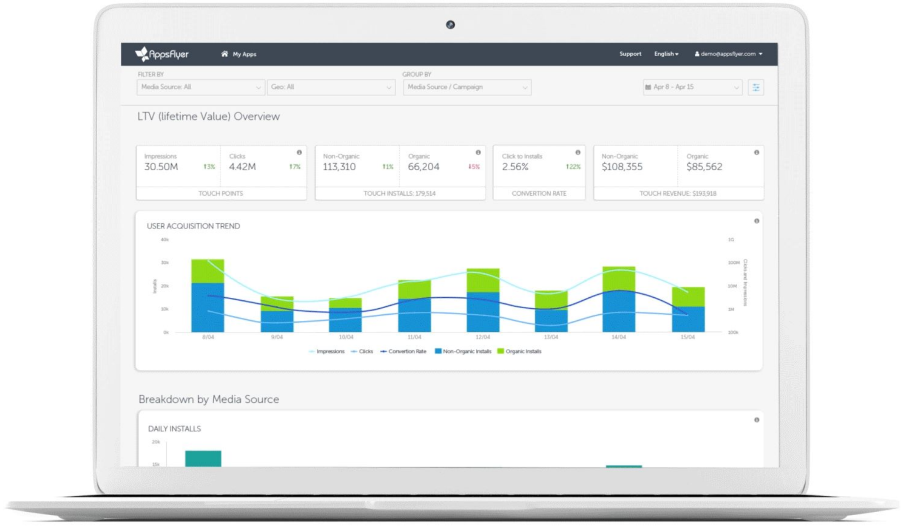
AppsFlyer

Unleash The Power Of The REPL

Dana Borinski

re:Clojure 2019

AppsFlyer in a Nutshell



Click

Install

Match

CONTENT SHARES
SOCIALS
EMAIL
SMS
Ads
OFFER
Add to cart
DESKTOP
MOBILE WEB

AppsFlyer
Engineering

AppsFlyer in Numbers

200+

Engineers Worldwide

200+

Microservices

1500

Servers

90B+

Events / Day

95%

Devices with AppsFlyer's SDK

Agenda

✓

Stack trace and prints as tools

(println “to the rescue!”)

✓

Understanding the code flow

mate-clj

✓

Debugging with the REPL

Is there a right approach?

+ REPL =

Let’s dive in

Me, every time I see a stack trace

1. Stack Trace

1. The Stack Trace

```clojure
 8 (defstate config
 9   :start {:a 0 :b 1}
10   :stop nil)
11
12 (defn my-divide [a b]
13   (println "going to divide" a "by" b)
14   (/ a b))
15
16 (defn my-calc [a b]
17   (-> (+ a b)
18       (* b)
19       (my-divide a)))
20
```

```
user=> (my-calc (:a config) (:b config))

going to divide 1 by 0
Execution error (ArithmeticException) at debugging-clojure.core/my-divide (core.clj:14).
Divide by zero
```

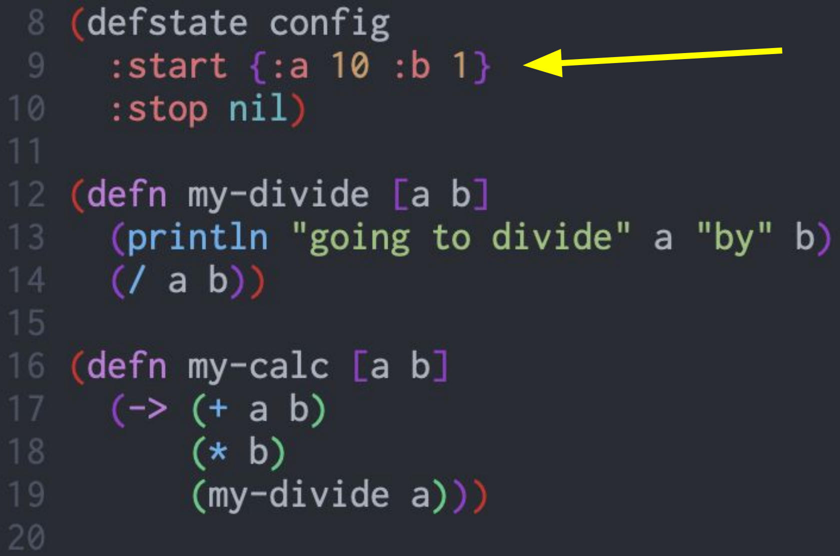
1. The Stack Trace

```clojure
 8 (defstate config
 9   :start {:a 0 :b 1}
10   :stop nil)
11
12 (defn my-divide [a b]
13   (println "going to divide" a "by" b)
14   (/ a b))
15
16 (defn my-calc [a b]
17   (-> (+ a b)
18       (* b)
19       (my-divide a)))
20
```

```
user=> (clojure.stacktrace/print-stack-trace *e)
java.lang.ArithmeticException: Divide by zero
 at clojure.lang.Numbers.divide (Numbers.java:188)
    debugging_clojure.core$my_divide.invokeStatic (core.clj:14)
    debugging_clojure.core$my_divide.invoke (core.clj:12)
    debugging_clojure.core$my_calc.invokeStatic (core.clj:19)
```

2. Synchronization

2. Code & Data Synchronization



```clojure
(defstate config
  :start {:a 10 :b 1}
  :stop nil)

(defn my-divide [a b]
  (println "going to divide" a "by" b)
  (/ a b))

(defn my-calc [a b]
  (-> (+ a b)
      (* b)
      (my-divide a)))

```

2. Code & Data Synchronization

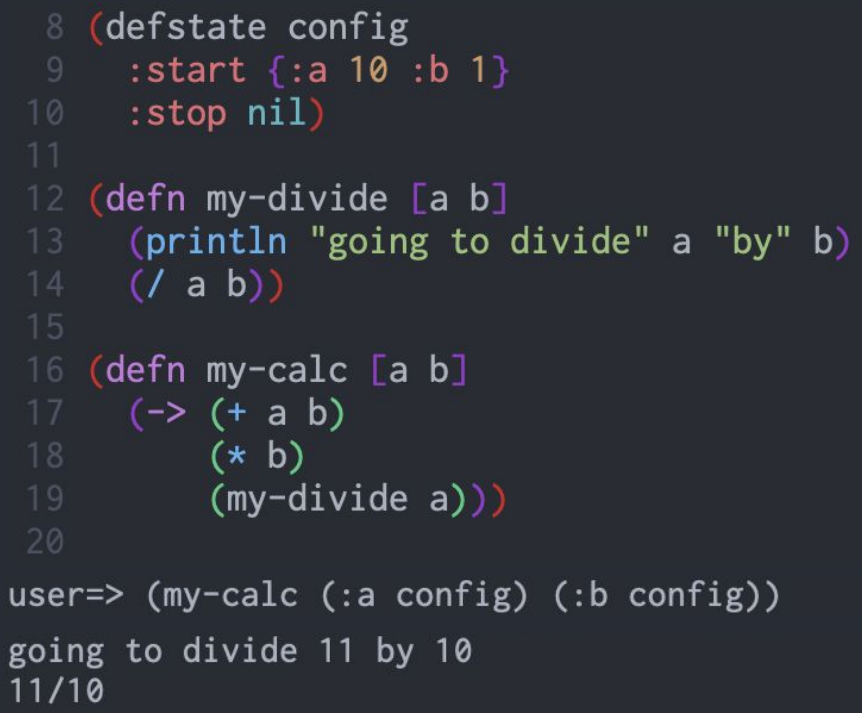
```clojure
(ns user
  (:require [clojure.tools.namespace.repl :as tn]
            [mount.core :as mount]))

(defn go
  "starts all states defined by defstate"
  []
  (mount/start)
  :ready)

(defn reset
  "stops all states defined by defstate, reloads modified source files,
  and restarts the states"
  []
  (mount/stop)
  (tn/refresh :after 'user/go))

(defn -main []
  (reset))

```

2. Code & Data Synchronization

```clojure
(defstate config
  :start {:a 10 :b 1}
  :stop nil)

(defn my-divide [a b]
  (println "going to divide" a "by" b)
  (/ a b))

(defn my-calc [a b]
  (-> (+ a b)
      (* b)
      (my-divide a)))

```

```
user=> (-main)

stoping all states...
refreshing the code...
:reloading (debugging-clojure.core debugging-clojure.core-test)
starting all states...
:ready
```

2. Code & Data Synchronization



```clojure
(defstate config
  :start {:a 10 :b 1}
  :stop nil)

(defn my-divide [a b]
  (println "going to divide" a "by" b)
  (/ a b))

(defn my-calc [a b]
  (-> (+ a b)
      (* b)
      (my-divide a)))

```

```
user=> (my-calc (:a config) (:b config))

going to divide 11 by 10
11/10
```

States Reloading

- Using the default **`user`** namespace.
- Sync the states data with the program
- Fast reloading
- My Clojure Workflow, Reloaded - By Stuart Sierra

FYSPRINGFIELD

3. Code Flow Debugging

3. Code Flow Debugging

```clojure
(def m {:name ["re:Clojure"]
        :location ["London" "Crypt on the Green"]})

(-> m
    :name
    clojure.string/upper-case
    (str " fun!"))

```

```
"[\"RE:CLOJURE\"] fun!"
```

WAIT A MINUTE.

(println “to the rescue!”)

- Simple
- Fast feedback
- Can be combined in functions and macros

3. Code Flow Debugging

```clojure
(def m {:name ["re:Clojure"]
        :location ["London" "Crypt on the Green"]})

(-> m
    (doto println)
    :name
    (doto println)
    clojure.string/upper-case
    (doto println)
    (str " fun!"))
```

```
{:name [re:Clojure], :location [London Crypt on the Green]}
[re:Clojure]
["RE:CLOJURE"]

"[\"RE:CLOJURE\"] fun!"
```

3. Code Flow Debugging

```clojure
(def m {:name ["re:Clojure"]
        :location ["London" "Crypt on the Green"]})

(-> m
    :name
    first
    clojure.string/upper-case
    (str " fun!"))

```

```
"RE:CLOJURE fun!"
```

But...

mate-clj

mate-clj

Debug your code out of the box

https://github.com/AppsFlyer/mate-clj

mate-clj

```
user=> (require '[mate-clj.core :as mate])
nil

(def m {:name ["re:Clojure"]
        :location ["London" "Crypt on the Green"]})

(mate/d-> m
          :name
          first
          clojure.string/upper-case
          (str " fun!"))

(:name m) => [re:Clojure]
(first (:name m)) => re:Clojure
(clojure.string/upper-case (first (:name m))) => RE:CLOJURE
(str (clojure.string/upper-case (first (:name m)))  fun!) => RE:CLOJURE fun!

"RE:CLOJURE fun!"
```

mate-clj

```
(mate/dcond->> 1
               true inc
               (= 3 2) (* 42)
               true (+ 100)
               (= 2 2) (* 9))
```

```
918
(+ 100 (inc 1)) => 102
(* 9 (+ 100 (inc 1))) => 918
918
```

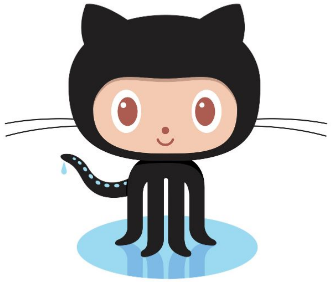
mate-clj

```clojure
(mate/dreduce + [1 3 5 7 9])
(#function[clojure.core/+] 1 3) => 4
(#function[clojure.core/+] 4 5) => 9
(#function[clojure.core/+] 9 7) => 16
(#function[clojure.core/+] 16 9) => 25

25
```

mate-clj

Pull Requests Are Welcome!

https://github.com/AppsFlyer/mate-clj



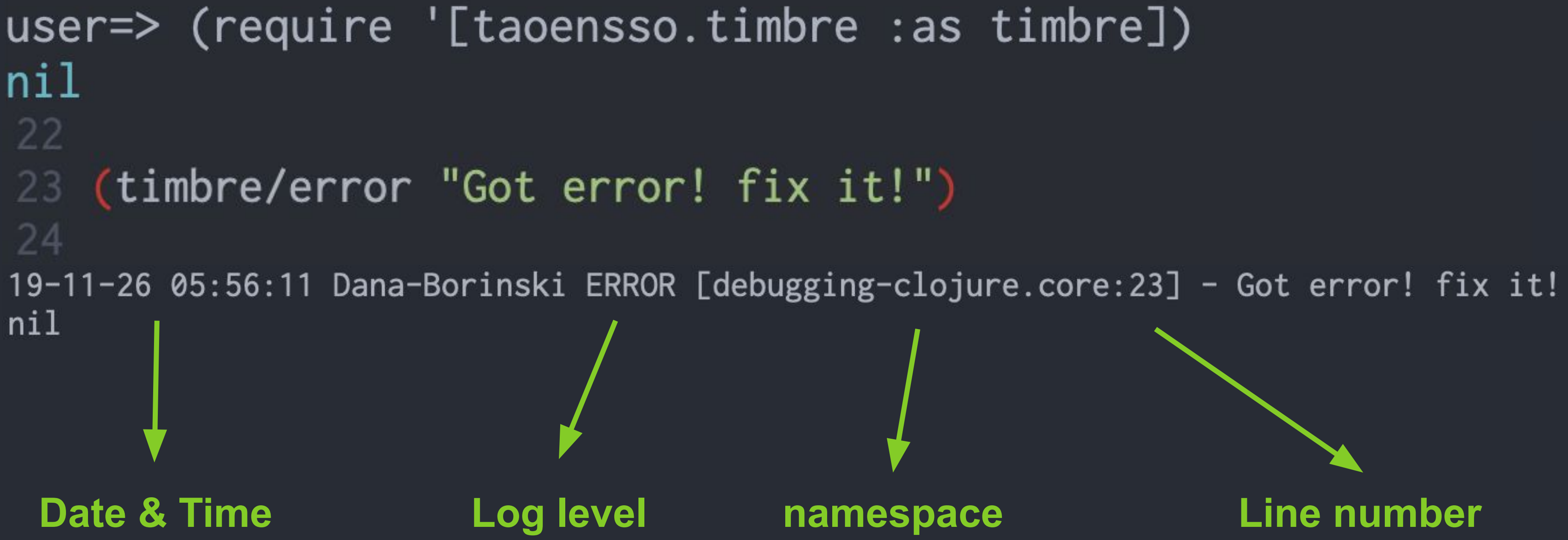
4. Logging Libraries

Logging Libraries

Advanced Prints

- Level tagging
- Line numbers
- Source namespace
- Low overhead

5. Logging Libraries



```
user=> (require '[taoensso.timbre :as timbre])
nil
 22
 23 (timbre/error "Got error! fix it!")
 24
19-11-26 05:56:11 Dana-Borinski ERROR [debugging-clojure.core:23] - Got error! fix it!
nil
```

Logging Libraries

- timbre
- tools.logging
- cambium

Recap

- The stack trace
- State data and code synchronization
- Code flow debugging using prints
- mate-clj
- Logging libraries

+ REPL = 💪

AppsFlyer

Every new line of code you willingly bring into the world is code that has to be debugged, code that has to be read and understood, code that has to be supported.

Jeff Atwood

AppsFlyer

Thank you
&
Safe Debugging!

dana.borinski@appsflyer.com