



FULLSTACK/STL
dev meetup

Clojure at Climate

Christopher Mark Gore

`cgore.com`

Thursday, October 27, AD 2016

**We write Clojure at The Climate Corporation,
and we're hiring! Come work with us!**



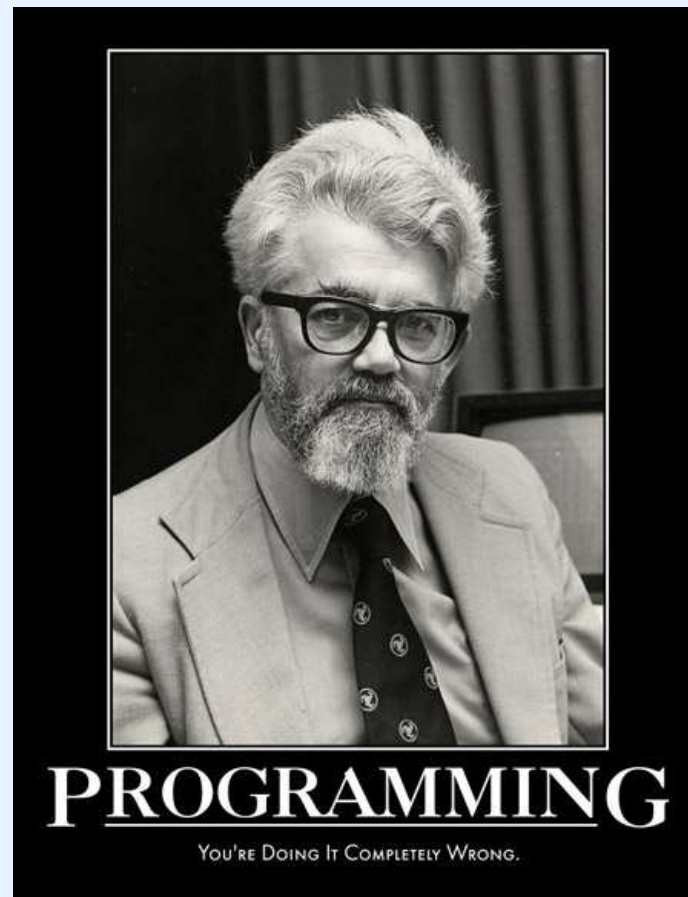
Especially now that Bayer is buying us!



It's a pretty cool place to work, we've even got a giant globe to play with.



Clojure is Lisp



Clojure is Lisp on the JVM.

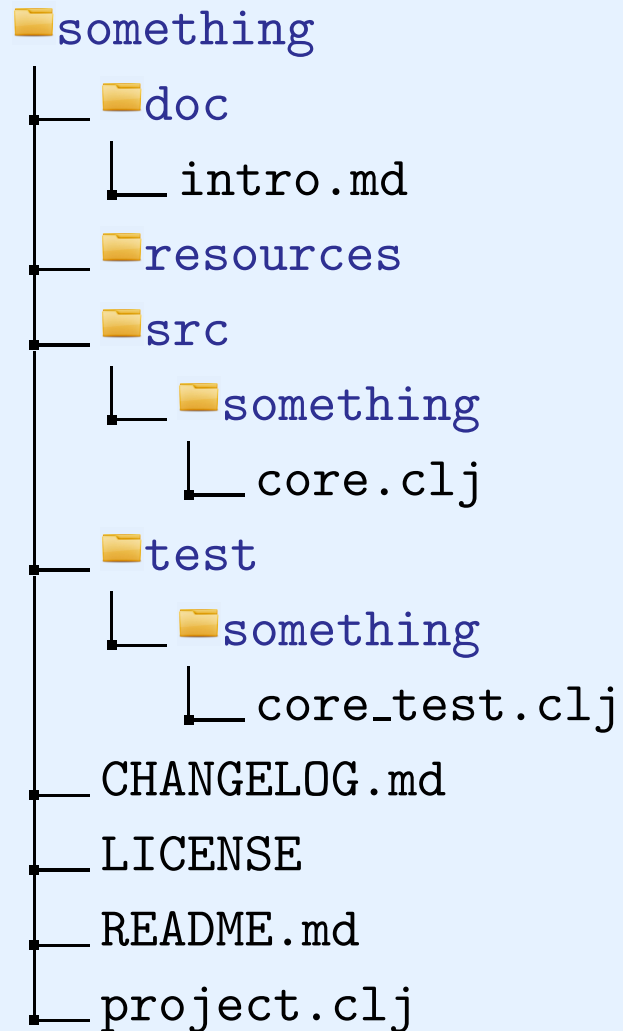


Building out a brand new Clojure app using Leiningen.



`lein new something`

What did that make for us?



src/something/core.clj

```
1 (ns something.core)
2
3 (defn foo
4   "I don't do a whole lot."
5   [x]
6   (println x "Hello, World!"))
```

test/something/core_test.clj
(You write tests, right?)

```
1 (ns something.core-test
2   (:require [clojure.test :refer :all]
3             [something.core :refer :all]))
4
5 (deftest a-test
6   (testing "FIXME, I fail."
7     (is (= 0 1)))))
```

LICENSE

This is defaulting to the Eclipse Public License, which you will maybe want to change.

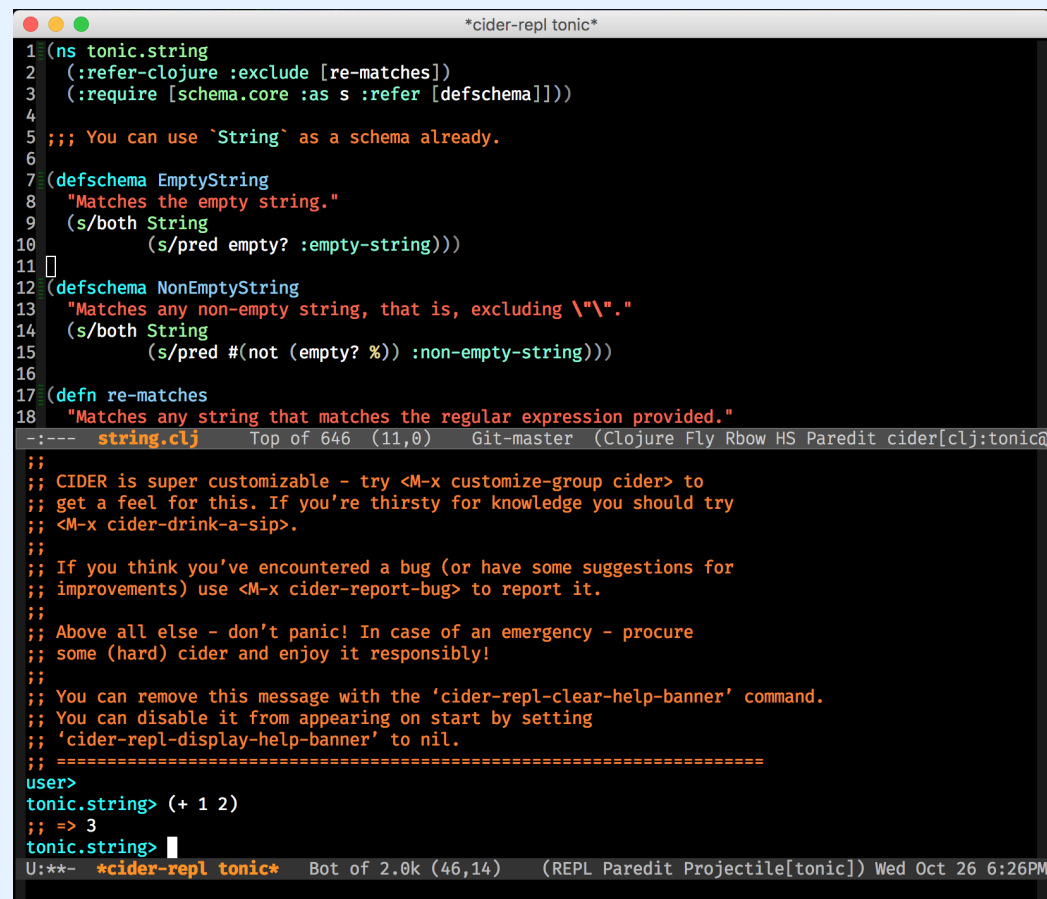
README.md

You should put some useful getting started info here, so that you can remember what to do in six months when you have to work on this component again.

project.clj

```
1 (defproject something "0.1.0-SNAPSHOT"
2   :description "FIXME: write description"
3   :url "http://example.com/FIXME"
4   :license
5   {:name "Eclipse Public License"
6    :url
7     "http://www.eclipse.org/legal/epl-v10.html"}
8   :dependencies [[org.clojure/clojure "1.8.0"]])
```

Since Clojure is a Lisp, you probably want to use Emacs. Because, nepotism.



```
*cider-repl tonic*
1 (ns tonic.string
2   (:refer-clojure :exclude [re-matches])
3   (:require [schema.core :as s :refer [defschema]]))
4
5 ;; You can use `String` as a schema already.
6
7 (defschema EmptyString
8   "Matches the empty string."
9   (s/both String
10    (s/pred empty? :empty-string)))
11
12 (defschema NonEmptyString
13   "Matches any non-empty string, that is, excluding \"\"."
14   (s/both String
15    (s/pred #(not (empty? %)) :non-empty-string)))
16
17 (defn re-matches
18   "Matches any string that matches the regular expression provided."
19   [re]
20   (fn [s] (re-matches re s)))
21
22 --- string.clj Top of 646 (11,0) Git-master (Clojure Fly Rbow HS Paredit cider[clj:tonic])
23
24 ;;
25 ;; CIDER is super customizable - try <M-x customize-group cider> to
26 ;; get a feel for this. If you're thirsty for knowledge you should try
27 ;; <M-x cider-drink-a-sip>.
28 ;;
29 ;; If you think you've encountered a bug (or have some suggestions for
30 ;; improvements) use <M-x cider-report-bug> to report it.
31 ;;
32 ;; Above all else - don't panic! In case of an emergency - procure
33 ;; some (hard) cider and enjoy it responsibly!
34 ;;
35 ;; You can remove this message with the 'cider-repl-clear-help-banner' command.
36 ;; You can disable it from appearing on start by setting
37 ;; 'cider-repl-display-help-banner' to nil.
38 ;; =====
39 user>
40 tonic.string> (+ 1 2)
41 ;; => 3
42 tonic.string>
43 U:**~ *cider-repl tonic* Bot of 2.0k (46,14) (REPL Paredit Projectile[tonic]) Wed Oct 26 6:26PM
```

Hello, World!

```
1 (println "Hello, World!")
```

That's it.

Math is lispy of course.

```
1 (/ (+ (* 1 2 3)
2      (* 4 5 6)
3      (* 7 8 9 10)))
4 12.345)
5 ;; => 418.46901579586876
```


There's Java classes underneath it all.

```
1 (type 123)
2 ;; => java.lang.Long
3 (type "123")
4 ;; => java.lang.String
5 (type (type "123"))
6 ;; => java.lang.Class
7 (type [1 2 3 4 5])
8 ;; => clojure.lang.PersistentVector
9 (type '(1 2 3 4 5))
10 ;; => clojure.lang.PersistentList
```

Java interop is pretty easy.

```
1 (.toUpperCase "stop_shouting")
2 ;; => "STOP SHOUTING"
3 (java.awt.Point. 1 2)
4 ;; => #object[java.awt.Point
5 ;;           0x4d59f5c5
6 ;;           "java.awt.Point [x=1, y=2] "]
7 (.-x (java.awt.Point. 1 2))
8 ;; => 1
```

**But objects suck and hashmaps are awesome,
so bean is your friend.**

```
1 (bean java.awt.Color/BLUE)
2 ;; =>
3 {:RGB -16776961,
4  :alpha 255,
5  :blue 255,
6  :class java.awt.Color,
7  :colorSpace #object[ ... ],
8  :green 0,
9  :red 0,
10 :transparency 1}
```

Ring is the most commonly used web application library in Clojure.

<https://github.com/ring-clojure/ring>

```
1 (defproject something "0.1.0-SNAPSHOT"
2   ...
3   :dependencies [...]
4       [ring "1.5.0"]
5       ...]
6   :ring {:handler hello-world.core/handler
7          :port 80}
8   ...)
```

Ring is very simple in it's design.

Requests come in.

Responses go out.

Handlers map from requests to responses.

Middleware adds functionality to the handlers. Nearly all additional functionality can be implemented as just more middleware functions, it's nothing but function composition.

A super-basic Ring handler function.

Ring handlers map from requests to responses, both represented as hashmaps.

```
1 (ns hello-world.core)
2
3 (defn handler [request]
4   {:status 200
5    :headers {"Content-Type" "text/html"}
6    :body "Hello World"})
```

And then just

lein ring server

at the command line to start it up.

Ring handler requests

:server-port The port handling the request.

:server-name server name / IP address.

:remote-addr The IP address of the client.

:uri The request URI (the full path after the domain name).

:query-string The query string, if present.

:scheme The transport protocol, either `:http` or `:https`.

:request-method The HTTP request method
(*:get*, *:head*, *:options*, *:put*, *:post*, or *:delete*)

:headers An hashmap of lowercase header name strings to corresponding header value strings.

:body An `InputStream` for the request body, if present.

Ring handler responses

:status HTTP status code (*200, 404, 500, ...*)

:headers hashmap of headers

:body This can be a String, ISeq, File, or InputStream.

Ring middleware

Ring middleware are functions that add functionality to handlers.

```
1 (defn wrap-content-type [handler content-type]
2   (fn [request]
3     (let [response (handler request)]
4       (assoc-in response
5                 [:headers "Content-Type"]
6                 content-type))))
```

The middleware function is taking in the handler function (and sometimes some additional arguments) and returning a new handler function.

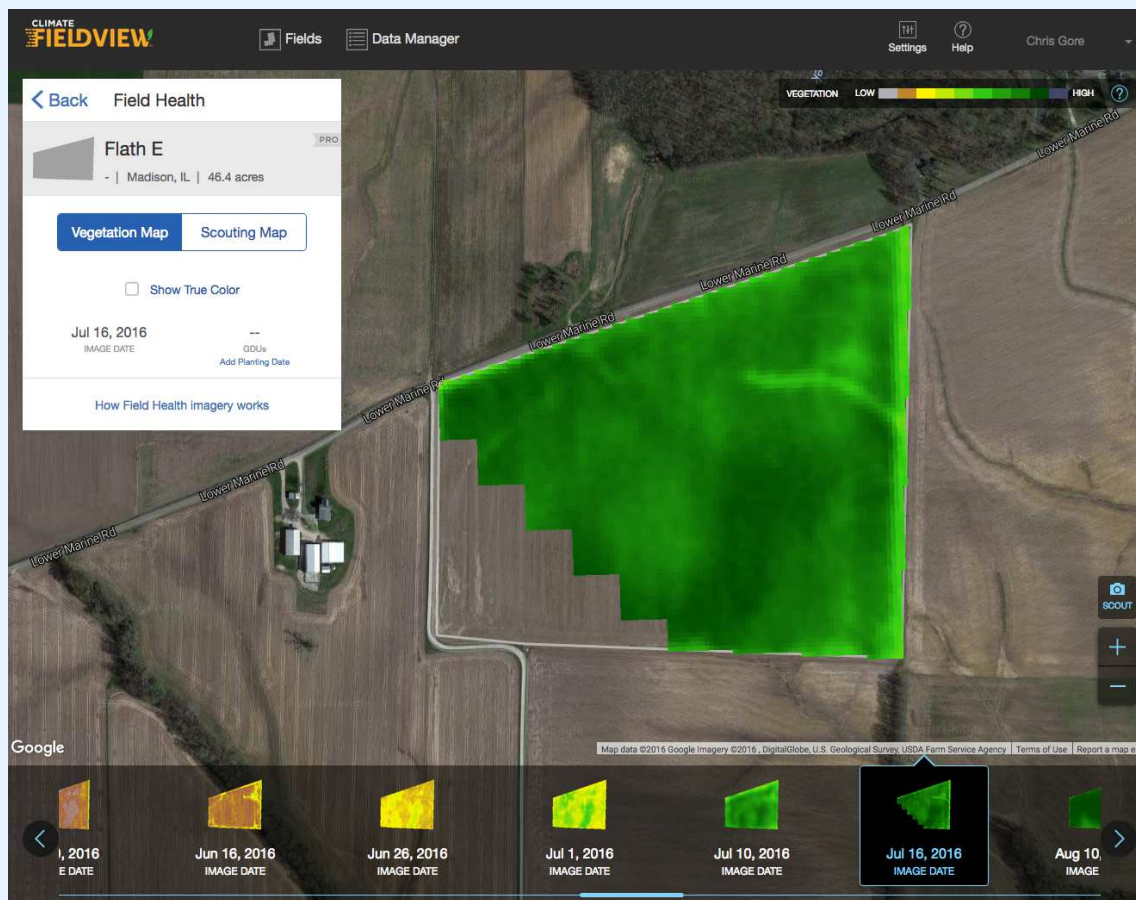
Using Ring middleware

```
1 (def app
2   (wrap-content-type handler "text/html"))
```

But you typically have more than one middleware, so the threading operator is our friend:

```
1 (def app
2   (-> handler
3       (wrap-content-type "text/html")
4       (wrap-keyword-params)
5       (wrap-params)))
```

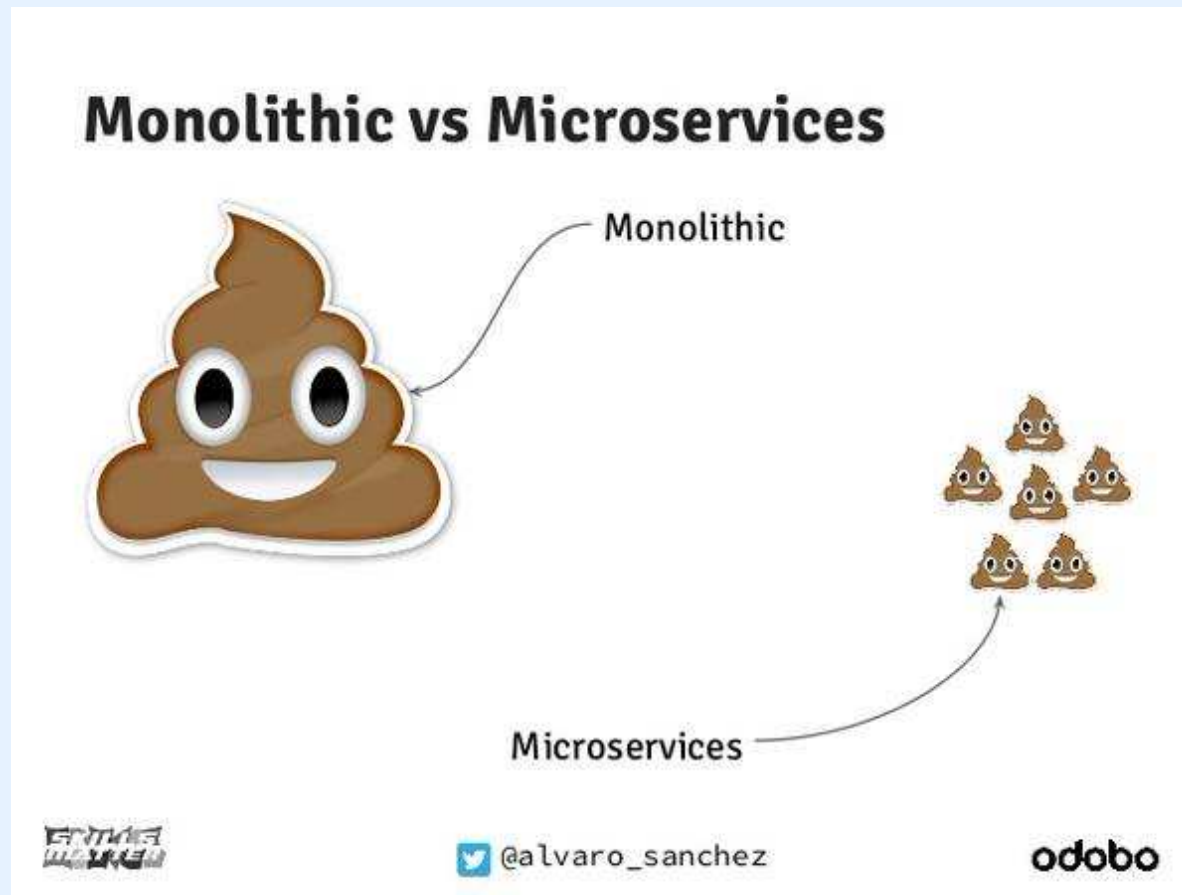
What exactly do we do at Climate?



Basically, we take a lot of photos of corn from space: we're corn paparazzi.



We have a microservice architecture at Climate.



Microservices are actually a good idea.

- If one component fails, the rest is usually still up.
- We can scale each one independently for differing demand.
- A team might only need to learn about a few microservices, instead of the whole ball of wax.
- Since they're fully independent, there's a lot of freedom in choosing tech stacks. We've got Clojure, Python, Ruby, Scala, and even Java for some reason.

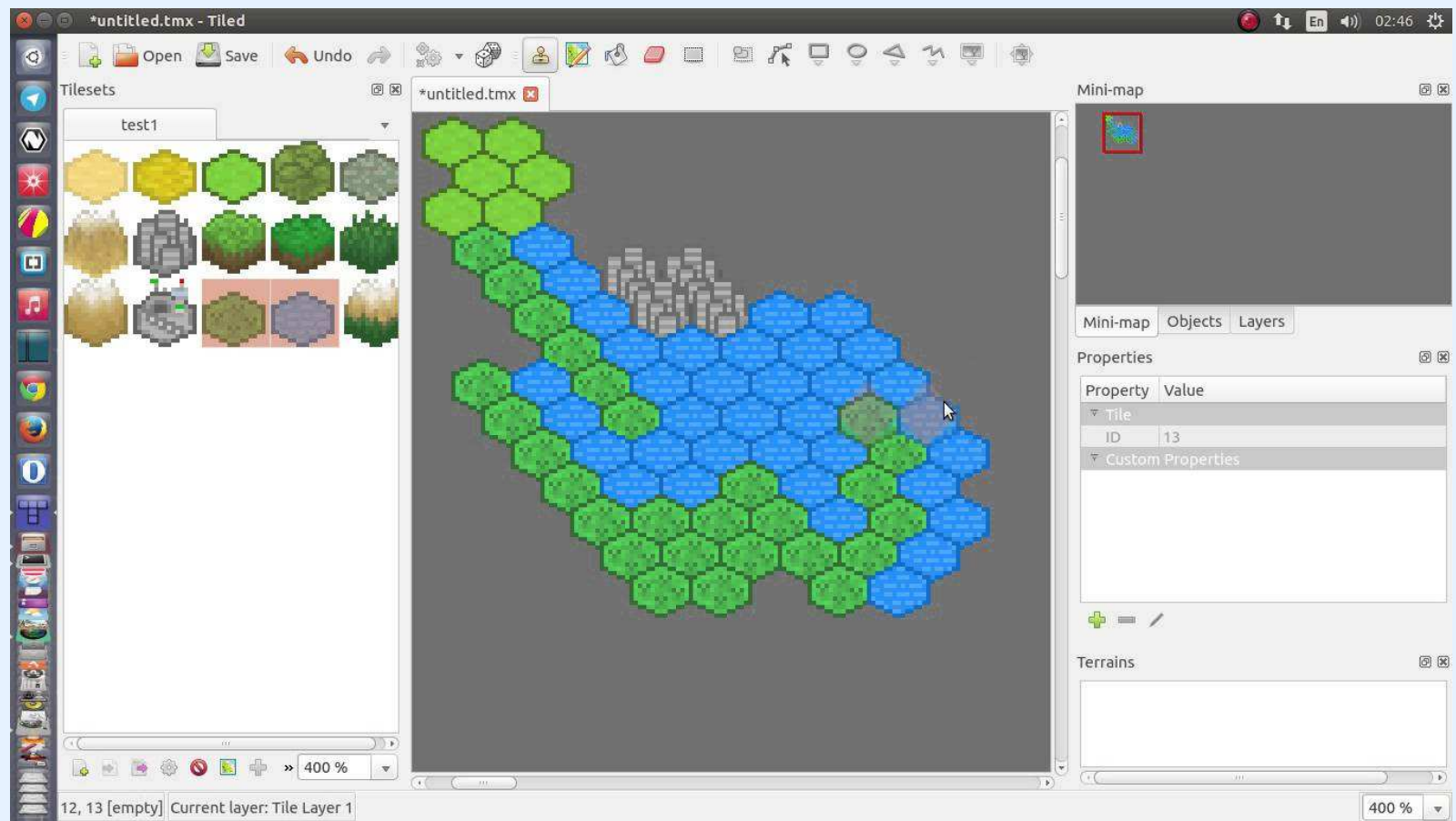
Satellite providers put up images for us to download.



A bunch of Python code downloads the imagery we care about and stages it locally.



Scrapbook (in Clojure) slices and dices the imagery into a very clever database sitting on Amazon S3 and Amazon DynamoDB.



Toomie (in Ruby) realizes that imagery needs to be generated and tells Galactus.



Galactus (in Clojure) gets the image generation request, makes the images, and puts them in a different S3/DynamoDB system.



(Most of our backend systems are named after
Marvel comics.)



Why all this stuff? It's HUGE, that's why!



- 5m satellite imagery
- for most of the USA
- about every other week
- back to 2011
- and we're expanding
- petabytes of imagery in Amazon S3
- hundreds of instances in Amazon EC2

Languages written in Clojure

jank A statically typed functional programming language

Lux A functional, statically-typed Lisp.

Sceje a tiny scheme implementation on Top of Clojure.

“If you give someone Fortran, he has Fortran. If you give someone Lisp, he has any language he pleases.”

– Guy L. Steele.

Questions?