DEVELOPING A MULTI-TENANT SAAS USING CLOJURE

Ari-Pekka Viitanen

ME

Programmer
Architect

Working for **VINCIT**

ari-pekka.viitanen@vincit.com
@apviitanen

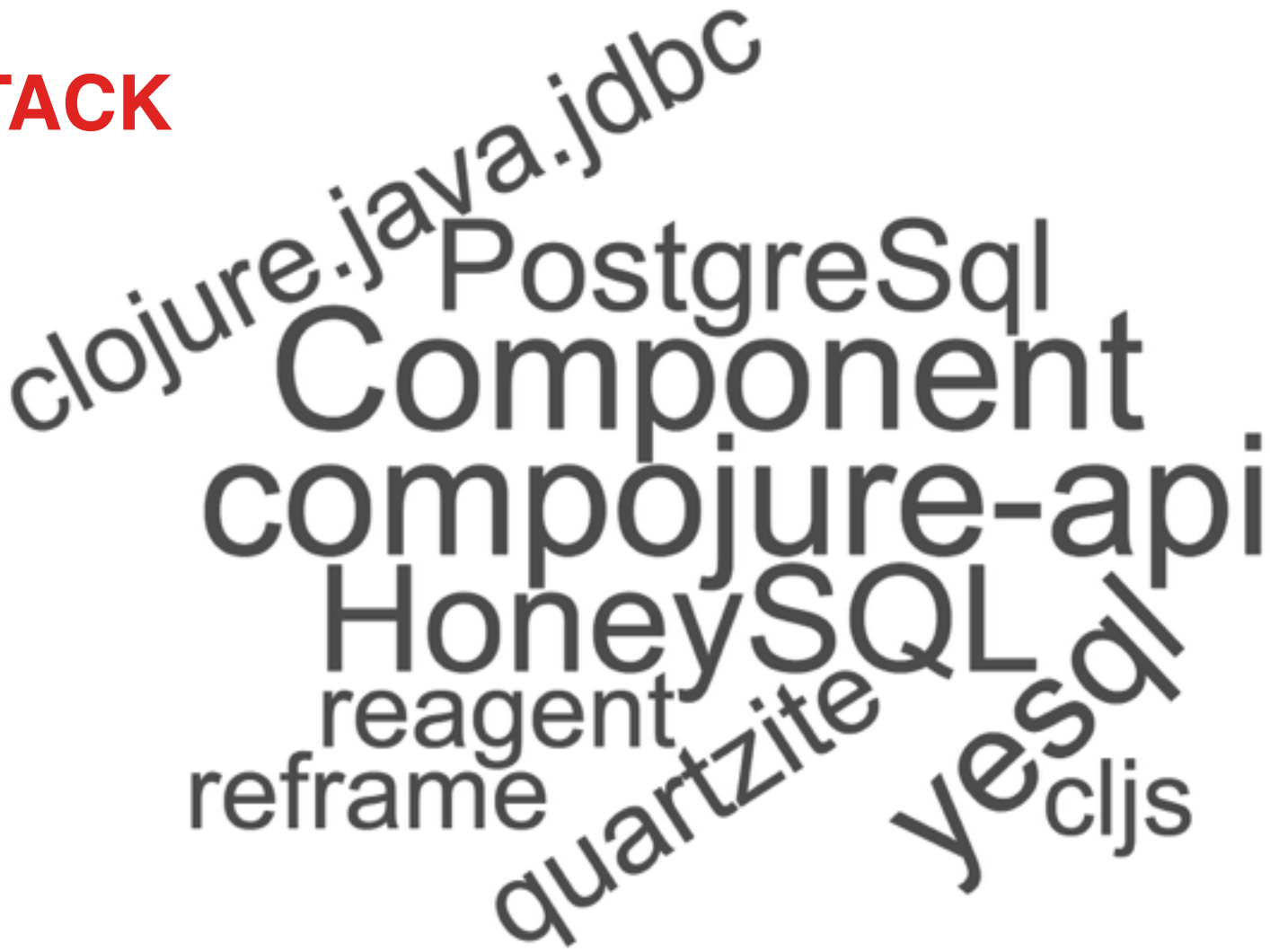
CASE BXX

WHEEL

A ground-breaking customer intelligence software

- For developing, monitoring and accelerating your brand and business

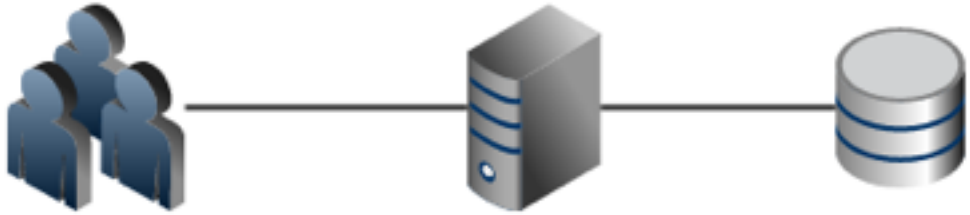
STACK


clojure.java.jdbc
PostgreSql
Component
compojure-api
HoneySQL
reagent
reframe
quartzite
yesql
cljs

DATA PERSISTENCE AND MULTI-TENANCY

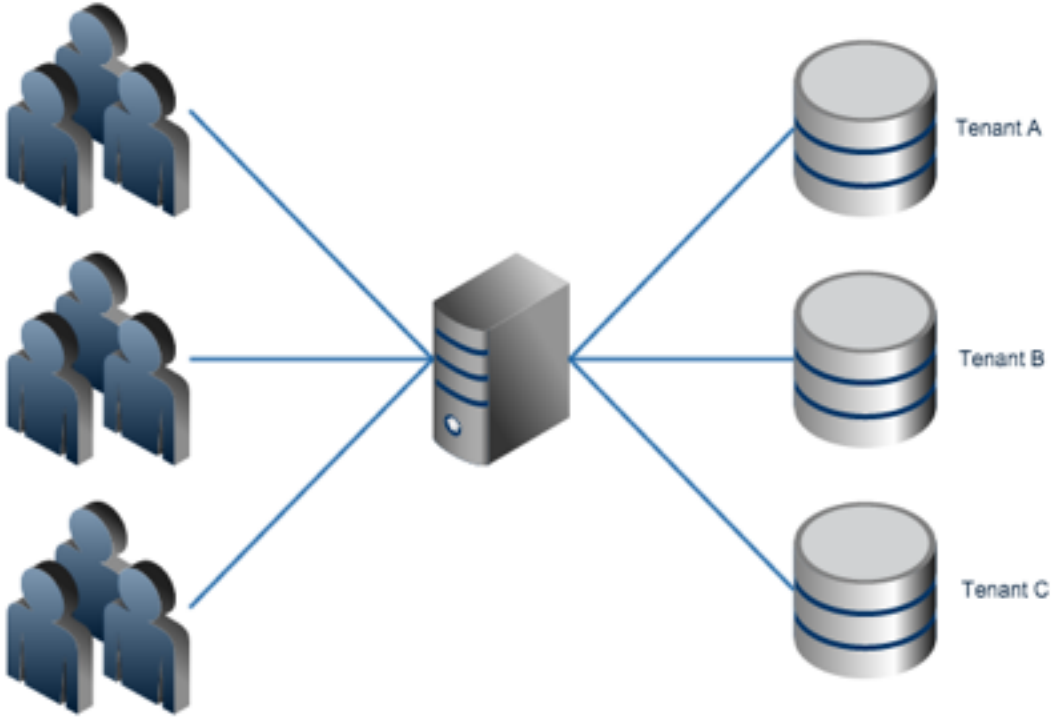
SINGLE TENANCY

TENANT A

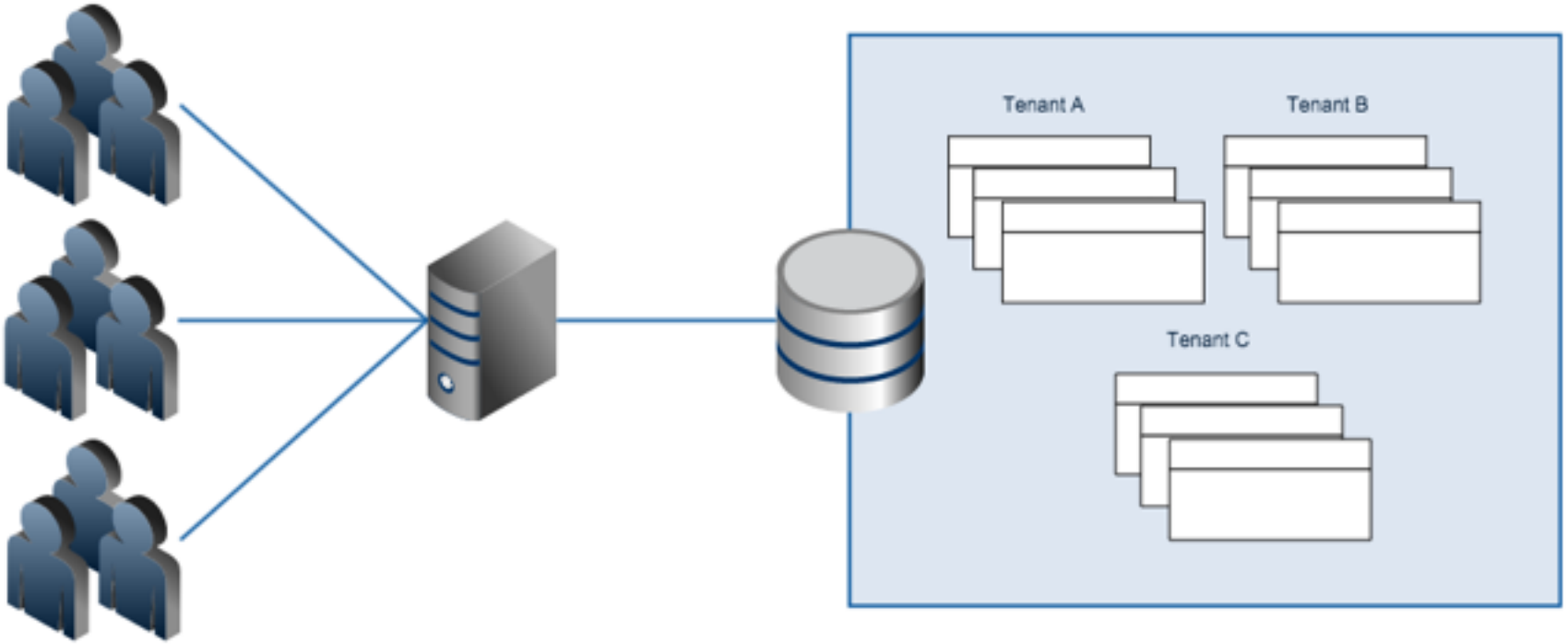


TENANT B

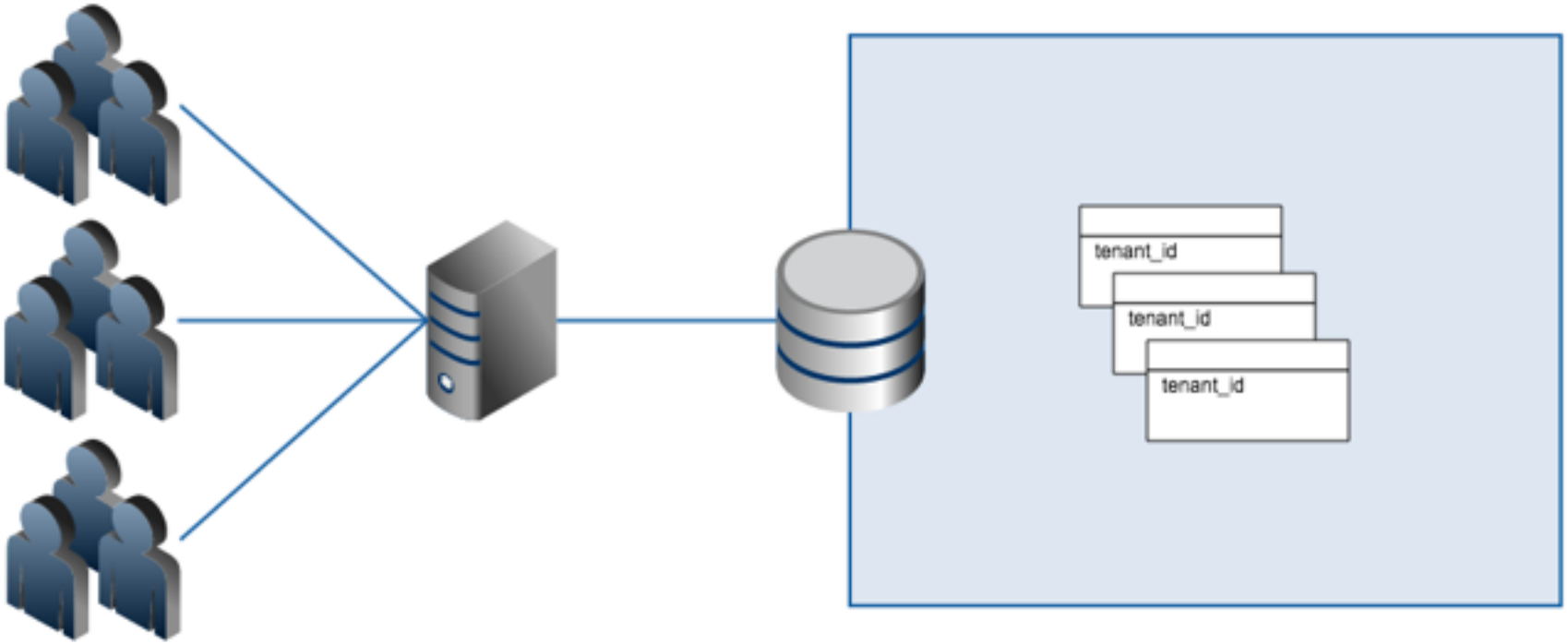
MULTI-TENANCY - SEPARATE DBs



MULTI-TENANCY - SEPARATE SCHEMAS



MULTI-TENANCY - SHARED SCHEMA



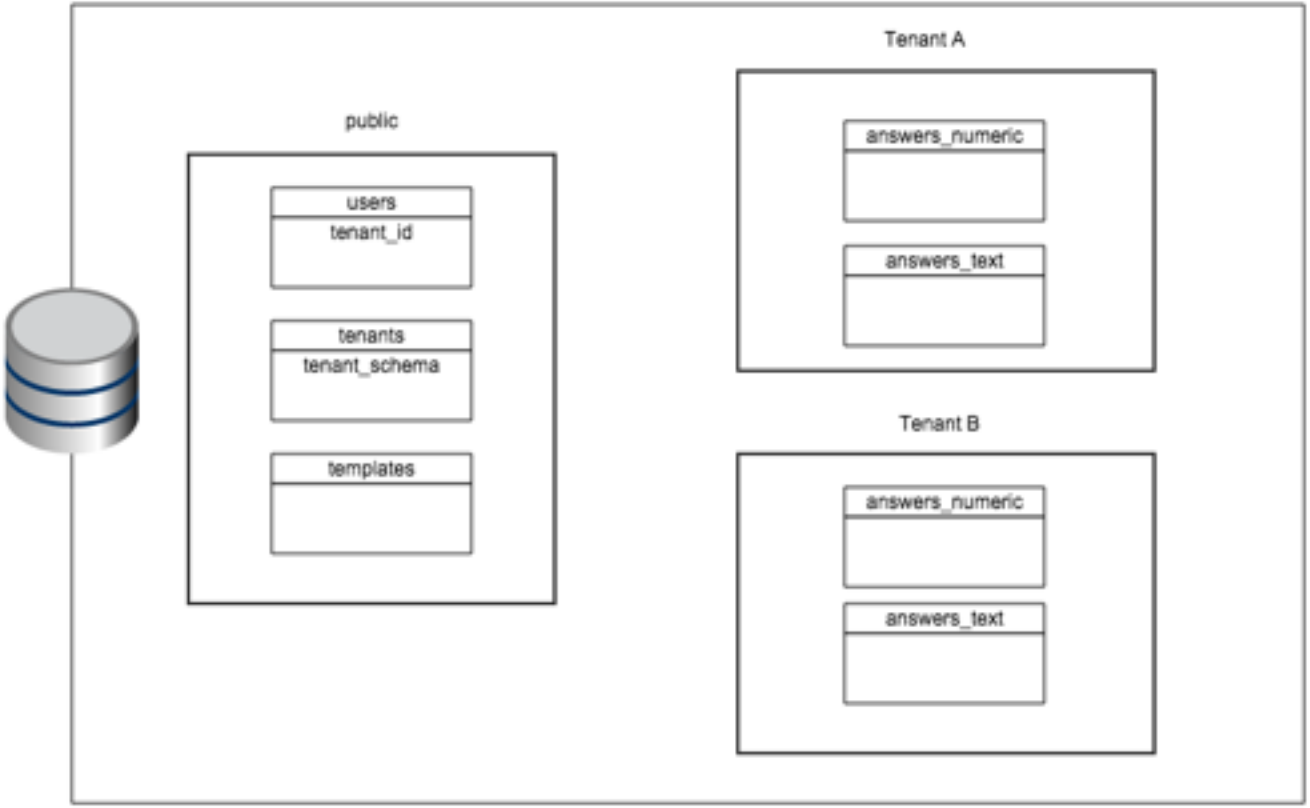
1st ATTEMPT - SHARED SCHEMA

```
 -- name: load-contact-groups
SELECT cg.id, cg.name FROM contact_group cg, tenant t
WHERE cg.tenant_id = t.id AND t.name = :tenant;

 -- name: find-contact-group-by-id
SELECT cg.id, cg.name FROM contact_group cg, tenant t
WHERE cg.tenant_id = t.id AND t.name = :tenant AND cg.id = :id;

 -- name: load-group-members
 -- loads members of group with :groupid within :tenant
 SELECT id, name, email FROM contact
 WHERE id IN
      (SELECT cgm.contact_id FROM contact_group_membership cgm, tenant  t WHERE
       cgm.tenant_id = t.id AND t.name = :tenant AND cgm.contact_group_id = :
       groupid);
```

SEPARATE SCHEMAS



SIMPLER QUERIES

```
 -- name: load-contact-groups
SELECT id, name FROM contact_group;

 -- name: find-contact-group-by-id
SELECT id, name FROM contact_group
WHERE id = :id;

 -- name: load-group-members
 -- loads members of group with :groupid within :tenant
 SELECT id, name, email FROM contact
 WHERE id IN
      (SELECT contact_id FROM contact_group_membership WHERE contact_group_id
= :groupid);
```

HOW DID WE DO THAT?

SHARING AND ISOLATING

```
set search_path to tenant_schema,public;
```

… IN CLOJURE

```
(defmacro with-tenant [t & body]
  `(binding [*tenant* ~t]
     ~@body))


(defn datasource [datasource-options]
  (HikariDataSource. (reify HikariLifecycleHooks
                       (onCheckout [_ conn]
                         (run-sql conn (change-schema-sql *tenant*)))
                       (onCheckin [_ conn]
                         (run-sql conn (change-schema-sql nil))))
    (doto (HikariConfig.)
```

THE JAVA PROGRAMMER'S SOLUTION

```
:java-source-paths ["java-src"]
```

```
public interface HikariLifecycleHooks {
    void onCheckout(final Connection connection);
    void onCheckin(final Connection connection);
}
```

```
public class HikariCallbackWrapper extends HikariDataSource implements
ConnectionCloseCallback {

    private final HikariLifecycleHooks hooks;

    public HikariCallbackWrapper(final HikariLifecycleHooks hooks, final
                                 HikariConfig config) {
        super(config);
        assert hooks != null;
        this.hooks = hooks;
    }

    @Override
    public Connection getConnection() throws SQLException {
        final Connection connection = super.getConnection();
        hooks.onCheckout(connection);
        return new LifecycleWrappedConnection(this, (IHikariConnectionProxy)
                                              connection);
    }

    @Override
    public void aboutToClose(final Connection connection) {
        hooks.onCheckin(connection);
    }
```

THIS WORKS!

```
(defn datasource [datasource-options]
  (HikariCallbackWrapper. (reify HikariLifecycleHooks
                        (onCheckout [_ conn]
                          (run-sql conn (change-schema-sql *tenant*)))
                        (onCheckin [_ conn]
                          (run-sql conn (change-schema-sql nil))))
    (doto (HikariConfig.)
```

```
(with-tenant schema-name
    (delete-contact db-spec 1))
```

WRAP EVERY ENDPOINT?

```
(defn wrap-tenant [handler]
  (fn [request]
    (with-tenant (-> request
                     :identity
                     :tenant-schema)
      (handler request))))
```

BIND IN THE MIDDLEWARE

```
(macroexpand '(-> handler
                  (wrap-tenant tenant-schema)
                  (wrap-context deps)
                  (wrap-authentication auth/auth-backend)))

=> (wrap-authentication
     (wrap-context (wrap-tenant handler tenant-schema) deps)
     auth/auth-backend)
```

AGAIN, THIS WORKS

AT LEAST FOR CUSTOMER API

BUT WE ARE USING DYNAMIC SCOPE

http://stuartsierra.com/2013/03/29/perils-of-dynamic-scope

Digital Digressions by Stuart Sierra

From programming to everything else

ABOUT ME WRITING PRESENTATIONS SOFTWARE CONTACT

On the Perils of Dynamic Scope

...

You can't dispatch to another thread. Say goodbye to Agents, Futures, thread pools, non-blocking I/O, or any other kind of asynchrony. The resource is only valid on the current thread.[3]

You can't return a lazy sequence backed by the resource because the resource will be destroyed as soon as body returns.

SURVEY RESULTS & ADMIN UI

THREADING?
LAZY-SEQ?

```
(with-tenant tenant-schema
    ...
    (map (fn [res] (... (add-completed-survey<! res ...)))))
```

BACK TO READING THE DOCS

in clojure.java.jdbc:

```
(defn get-connection
    ^java.sql.Connection
  [{:keys [connection
           factory
           datasource]
    :as db-spec}]
    (cond
        connection
        connection

        factory
        (factory (dissoc db-spec :factory))
```

BUT I LIKE THE WITH-TENANT MACRO

```
(defn factory [{:keys [db-spec tenant-schema]}]
  (let [conn (jdbc/get-connection db-spec)]
    (run-sql conn (change-schema-sql tenant-schema))
    conn))

(defmacro with-tenant-schema [[db-schema db t] & body]
  `(let [~db-schema {:factory       #'factory
                     :datasource    (:datasource ~db)
                     :tenant-schema ~t}]
     ~@body))
```

ONCE AGAIN, IT WORKS

A pragmatic solution to a real-world problem

```
(with-tenant-schema [db-schema db schema]
      (add-contact-group (assoc ctx :db db-schema) "Customers")
      (add-contact-group (assoc ctx :db db-schema) "Partners")
      (add-contact-group (assoc ctx :db db-schema) "Subcontractors")
      )
```

WHAT WE LEARNED

- Simple abstractions
- Be aware of dynamic scope
- Learn your libraries

THANK YOU