;Handlers

```clojure
(defn echo-handler [request]
(ok (with-out-str (pp/pprint request))))
```

```
(defn weapons-query-handler [{{:strs [name]} :params :keys [conn] :as request}]
```

(**ok**

```
(if name
(w/everybodys-weapons @conn))))
```

```
(w/weapons @conn [name])
```

```clojure
(defn add-weapons-handler [{:keys [params conn] :as request}]
(d/transact! conn data)
```

(do

```
(ok (w/weapons @conn (keys params)))))
```

```
(let [data (map (fn [[k v]] {:name k :weapon v}) params)]
```

```clojure
(defn files-handler [{:keys [sql-conn]}]
;Routes - All data
```

```
(ok (f/all-processed-files sql-conn)))
```

```
(def basic-routes
[["/echo" {:get echo-handler}]
["/files" {:get files-handler}]])
```

```
(def weapons-routes
[["/weapons" weapons-query-handler]
["/add_weapon" add-weapons-handler]])
```

;Router

```
(def router
```

(**ring/router**

```
;We were able to compose the routes here
```

```
[basic-routes weapons-routes]
```

```
:middleware [params/wrap-params
```

```
{:data {:coercion  reitit.coercion.spec/coercion
middleware/wrap-format]}}))
```

```
;Global handler
```

router

```
(ring/ring-handler
(def handler
(constantly (not-found "Not found"))))
```

```
(f/setup conn))
```

```
(defmethod ig/init-key ::jdbc/init [_ {:keys [conn]}]
(defn line->record [line]
(let [[name & weapons] (map cs/trim (cs/split line #","))]
(seq weapons)
```

```
(assoc :weapon weapons))))
```

```
(cond-> {:name name}
```

```clojure
(defn file-handler [{:keys [queue conn] :as ctx} {:keys [^File file kind] :as event}]
(when (and (#{:modify :create} kind)
(.exists file)
```

```
(.isFile file)
```

```
(cs/ends-with? (.getName file) ".csv"))
```

```clojure
(timbre/debug (str "Detected change to file: " (.getName file)))
```

```
(with-open [r (io/reader file)]
```

```clojure
(timbre/debug "Adding data to queue.")
```

```
(doseq [line (line-seq r)]
(dq/put! queue :my-queue line)))
```

```
(f/insert-file conn {:name (.getName file) :processed (Date.)}))))
```

```
ctx)
```

```clojure
(defn queue->dsdb [{:keys [queue-name queue dsdb]}]
;(timbre/debug "Checking for new items in queue...")
```

```clojure
(when-some [task (dq/take! queue queue-name 10 nil)]
(timbre/debug "Putting data into datascript")
```

```
(d/transact! dsdb [(line->record @task)])
```

```
(dq/complete! task))))
```

```
(def config
{::jdbc/connection        {:connection-uri "jdbc:h2:mem:mem_only"}
```

```
::jdbc/init                          {:conn (ig/ref ::jdbc/connection)}
```

::datascript/connection w/schema

```
::hawk/watch                    {:groups [{:paths   ["example"]
:handler #'file-handler}]
```

```
:queue  (ig/ref ::durable/queues)
```

```
:conn   (ig/ref ::jdbc/connection)}
```

```
::durable/queues          {:delete-on-halt? true
:directory          "/tmp"}
```

```
::scheduling/job                {:job                #'queue->dsdb
```

```
:queue-name :my-queue
```

```
:schedule  {:in [5 :seconds] :every :second}
```

```
:queue          (ig/ref ::durable/queues)
```

```
:dsdb          (ig/ref ::datascript/connection)}
```

```
::web/server             {:host    "0.0.0.0"
```

```
:sql-conn (ig/ref :jdbc/connection)
```

```
:port    3000
```

```
:conn     (ig/ref ::datascript/connection)
```

```
:handler #'handler}})
```

```
(defonce sys (create config))
```

```
(def valid-options
region :static-dir
```

```
#{:path :dispatch? :trust-managers :key-managers :keystore :buffer-size :auto-start :buffers-per-
```

```
:worker-threads :port :host :ssl-context :io-threads :client-auth :ajp-port :direct-buffers? :trust-password
```

```
:virtual-host :key-password :truststore :configuration :contexts :http2? :servlet-name :filter-map :ssl-
```

```
port})
```

```
(defmethod ig/init-key ::server [_ {:keys [handler] :as m}]
(timbre/debug "Launching Immutant web server.")
```

```
(immutant/run (wrap-component handler m) (select-keys m valid-options)))
```

```
(defmethod ig/halt-key! ::server [_ server]
(timbre/debug "Stopping Immutant web server.")
```

```
(immutant/stop server))
```