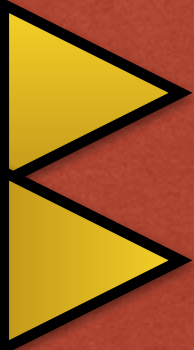


TALES OF THE CLEVER

COYOTE





JOHNNY WINN

@johnny_rugger

<https://github.com/nurugger07>

ELIXIR FOUNTAIN

@elixirfountain

<https://soundcloud.com/elixirfountain>



enbala

POWER NETWORKS



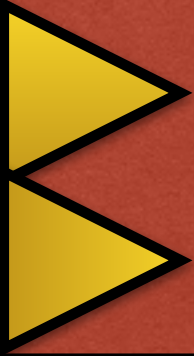
enbala

POWER NETWORKS



CLEVER COYOTE






ONCE UPON A TIME,

long ago, a horrible monster stole all the buffalo from the plains and put them in his mountain hideout...



WHAT I SET OUT TO DO VS

WHAT I DISCOVERED I WANTED TO DO...



COYOTE AND THE COLUMBIA

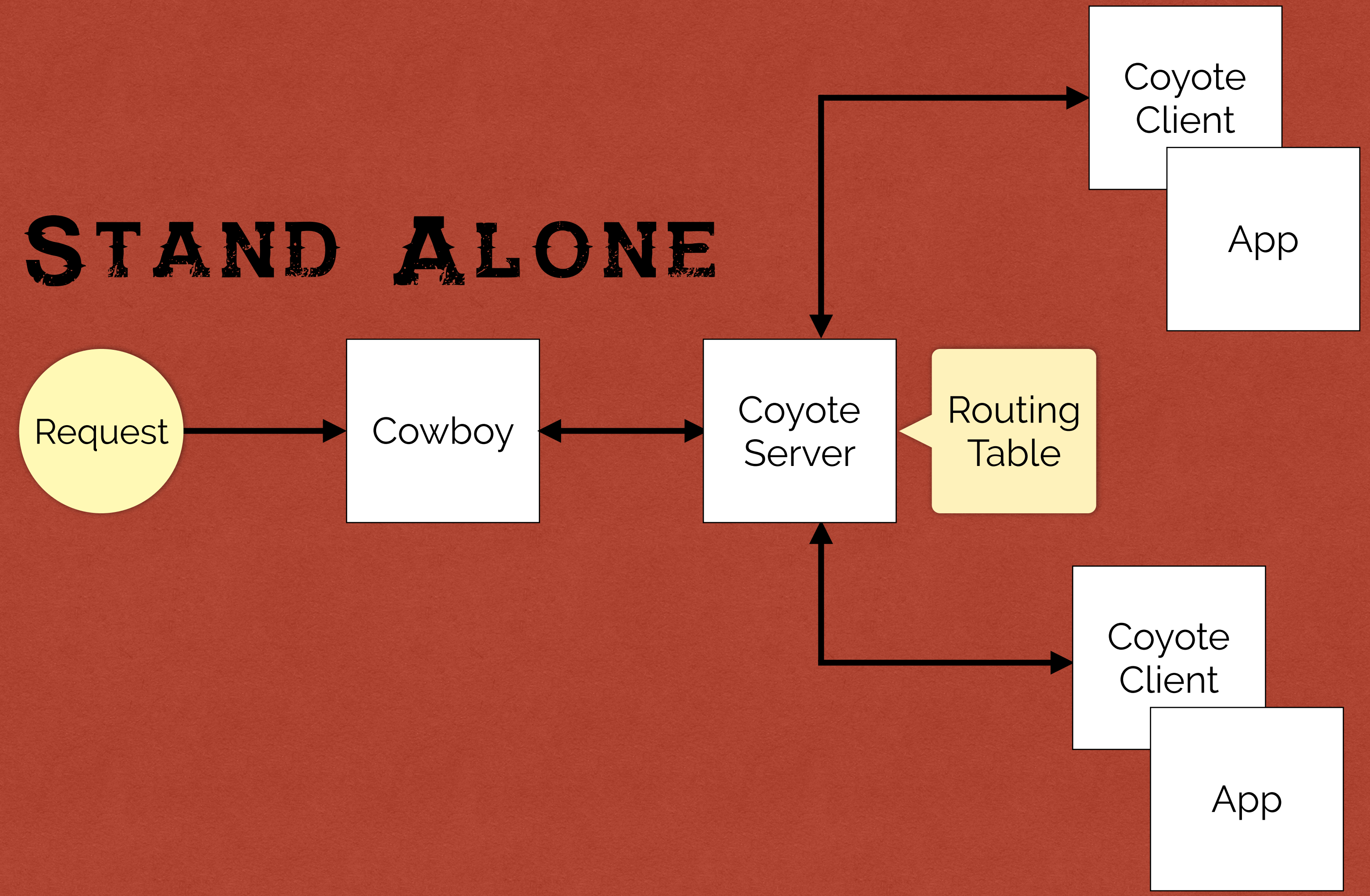


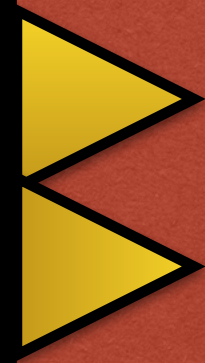


ONE DAY, COYOTE

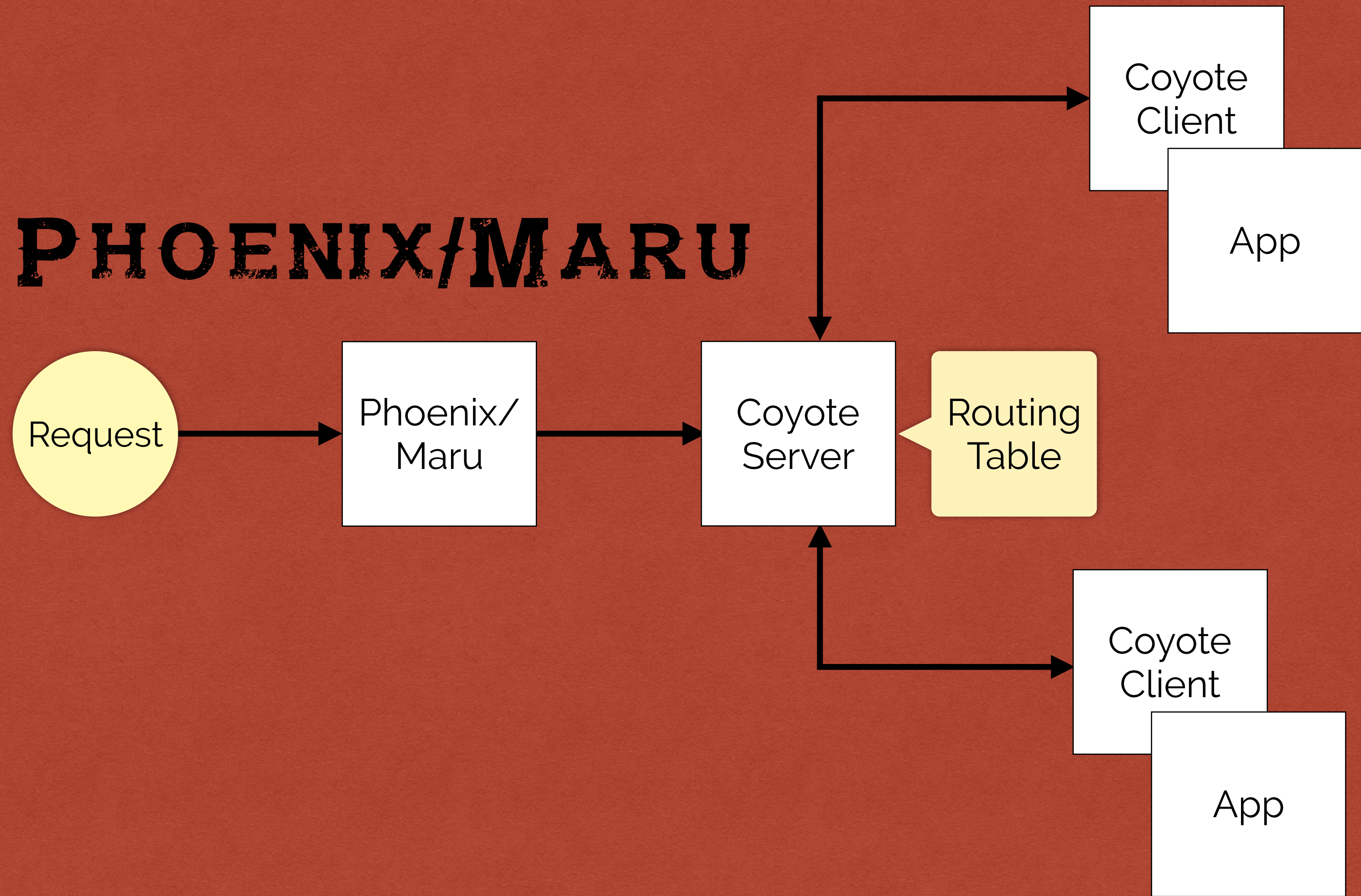
was walking along. The sun was shining brightly, and Coyote felt very hot. "I would like a cloud," Coyote said...

STAND ALONE





PHOENIX/MARU



COYOTE ROUTER

```
1 defmodule Coyote.Router do
2
3   defmacro routes(do: routes) do
4     quote do
5       def __routes__() do
6         unquote(routes)
7       end
8     end
9   end
10
11 end
```

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

COYOTE CLIENT

```
10 defmacro __using__(opts) do
11   quote do
12     use GenServer
13
14     import Coyote.Router, [only: [routes: 1]]
15
16     @server unquote(opts[:leader]) || Coyote
17
18     @name __MODULE__
19     @nodes unquote(opts[:leader_nodes]) ||
20       Application.get_env(:coyote, :leader_nodes, [:nonode@nohost])
21     @topology unquote(opts[:topology]) ||
22       Application.get_env(:coyote, :topology, :default)
23
24     def start_link,
25       do: GenServer.start_link(__MODULE__, [], name: @name)
26
27     def init(_args) do
28       send(self(), :register_routes)
29       {:ok, []}
30     end
31
32     def handle_info(:register_routes, state) do
33       Enum.each(@nodes, &register_routes/1)
34
35       {:noreply, state}
36     end
37
38     def handle_info({:watch_leader, node}, state) do
39       case Node.ping(node) do
40         :pong ->
41           register_routes(node)
42         :pang ->
43           send(self(), {:watch_leader, node})
```


COYOTE CLIENT REGISTER

```
69     end
70
71     defp register_routes(node) do
72       {mod, binary, file} = get_code()
73
74       route_func = Application.get_env(:coyote, :routes, &mod.__routes__/0)
75
76       case Node.ping(node) do
77         :pong ->
78           send(@server, node), {:register, {mod, binary, file, route_func, @topology, {self(), Node.self()}}}
79
80         Process.monitor(@server, node)
81         :pang ->
82         :ok
83       end
84     end
85
86     defp get_code,
87       do: :code.get_object_code(__MODULE__)
88
89     def call(req, from, state),
90       do: {:error, "No override provided for call/3"}
91
92     def cast(req, state),
93       do: {:error, "No override provided for cast/2"}
94
95     def info(req, state),
96       do: {:error, "No override provided for info/2"}
97
98     defoverridable [call: 3, cast: 2, info: 2]
99   end
100 end
101 end
```



COYOTE SERVER WATCH

```
38 def handle_info({:watch_leader, node}, state) do
39   case Node.ping(node) do
40     :pong ->
41       register_routes(node)
42     :pang ->
43       send(self(), {:watch_leader, node})
44   end
45   {:noreply, state}
46 end
47
48 def handle_info({:DOWN, _ref, :process, {Coyote, node}, reason}, state) do
49   send(self(), {:watch_leader, node})
50   {:noreply, state}
51 end
52
53 def handle_call(req, from, state) when is_tuple(req) do
54   try do
55     call(req, from, state)
56   rescue
57     FunctionClauseError ->
58       {:reply, {:error, "Undefined FunctionClauseError"}, state}
59   end
60 end
61
62 def handle_cast(req, state) when is_tuple(req) do
63   try do
64     cast(req, state)
65   rescue
66     FunctionClauseError ->
67       {:noreply, state}
68   end
69 end
70
```


COYOTE SERVER

```
29 def handle_info({:register, {mod, binary, file, func, topology, node}}, _state) do
30   {:module, mod} = :code.load_binary(mod, file, binary)
31
32   @route_bridge.update_routing_table(func.(), mod, node, topology)
33   {:noreply, []}
34 end
35
36 def handle_cast({method, path, args, topology}, _state) do
37   cast_route({method, path}, args, topology)
38   {:noreply, []}
39 end
40
41 def handle_cast({method, path, args}, _state) do
42   cast_route({method, path}, args, :default)
43   {:noreply, []}
44 end
45
46 def handle_cast(req, _state) when is_tuple(req) do
47   cast_route(req, [], :default)
48   {:noreply, []}
49 end
50
51 def handle_call({method, path, args, topology}, from, _state),
52   do: {:reply, call_route({method, path}, args, from, topology), []}
53
54 def handle_call({method, path, args}, from, _state),
55   do: {:reply, call_route({method, path}, args, from, :default), []}
56
57 def handle_call(req, from, _state) when is_tuple(req),
58   do: {:reply, call_route(req, [], from, :default), []}
59
60 defp call_route(req, args, from, topology),
61   do: Coyote.Relay.call({from, {req, args, topology}})
```


PRODUCER RELAY

```
1 defmodule Coyote.Relay do
2   use GenStage
3
4   def start_link(),
5     do: GenStage.start_link(__MODULE__, :ok, name: __MODULE__)
6
7   def call({from, request}, timeout \\ 5000),
8     do: GenStage.call(__MODULE__, {:call, {from, request}}, timeout)
9
10  def cast(request, _timeout \\ 5000),
11    do: GenStage.cast(__MODULE__, {:cast, request})
12
13  def init(:ok),
14    do: {:producer, {:queue.new, 0}, dispatcher: GenStage.BroadcastDispatcher}
15
16  def handle_demand(incoming_demand, {queue, demand}),
17    do: dispatch_requests(queue, incoming_demand + demand, [])
18
19  def handle_call({_call, {origin, request}}, from, {queue, demand}),
20    do: dispatch_requests(:queue.in({from, {origin, request}}, queue), demand, [])
21
22  defp dispatch_requests(queue, demand, requests) do
23    with d when d > 0 <- demand, {value, {from, request}}, queue <- :queue.out(queue) do
24      GenStage.reply(from, :ok)
25      dispatch_requests(queue, demand - 1, [request | requests])
26    else
27      _ ->
28        {:noreply, Enum.reverse(requests), {queue, demand}}
29    end
30  end
31
32 end
```

CONSUMER SUPERVISOR

```
1 defmodule Coyote.RequestSupervisor do
2   use ConsumerSupervisor
3
4   @max_demand 50
5
6   def start_link(),
7     do: ConsumerSupervisor.start_link(__MODULE__, :ok)
8
9   def init(:ok),
10    do: {:ok,
11         children(),
12         strategy: :one_for_one,
13         subscribe_to: [
14           {Coyote.Relay, max_demand: @max_demand}
15         ]}
16
17   defp children,
18    do: [worker(Coyote.RequestHandler, [], restart: :temporary)]
19
20 end
```

```
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```


REQUEST HANDLER

```
4 require Logger
5
6 @route_bridge Application.get_env(:coyote, :route_bridge, Coyote.RouteBridge)
7
8 def start_link({from, ref}, request),
9   do: GenServer.start_link(__MODULE__, {{from, ref}, request})
10
11 def init({from, _ref}, request) do
12   send(self(), :handle_request)
13   {:ok, {from, request}}
14 end
15
16 def handle_info(:handle_request, {from, {req, args, topology}}) do
17   response = case @route_bridge.find_route(req, topology) do
18     {:ok, %Coyote.Topology.Route{pid: pid, route: {method, path}}} ->
19     case GenServer.call(pid, {method, path, args}) do
20       {:error, message} ->
21         Logger.error(message)
22         {:error, "Routing error"}
23       {:ok, _response} = response ->
24         response
25     response ->
26       {:ok, response}
27   end
28   {:error, "No matching routes"} = error ->
29   error
30   nil ->
31     {:error, "No matching route"}
32 end
33
34 send(from, response)
35
36 {:stop, :normal, {}}
37 end
```


CLIENT APPLICATION

```
1 defmodule Deadwood.Saloon do
2   use Coyote.Client
3
4   @text """
5     <h1>The Clever Coyote</h1>
6     <p>
7       The monster hunted and hunted, but the buffalo had scattered without a trace. Late
8       that night, when the monster returned to his lair, young warriors were waiting.
9       They killed the monster, much to the relief of one small boy and all of the people
10      and all of the animals.
11     </p>
12     <p>
13       That is why the elders say it is Coyote to whom we owe the buffalo. Even today, the
14       people still give thanks to clever Coyote. If it had not been for the smart head and
15       warm heart of one little dog, that horrible monster would have kept all the buffalo
16       for himself forever.
17     </p>
18     """
19
20   routes do
21     [
22       {:GET, "/"}
23     ]
24   end
25
26   def call({:GET, "/", _params}, _from, state),
27     do: {:reply, @text, state}
28
29 end
~
~
~
~
~
```

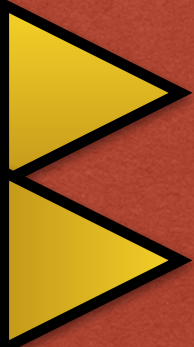
CLIENT APPLICATION

```
1 defmodule Deadwood.Menu do
2   use Coyote.Client, leader_nodes: [:"api@brooke-air"]
3
4   routes do
5     [
6       {:get_items, "/food"},
7       {:put_item, "/food"}
8     ]
9   end
10
11   def call({:get_items, "/food", _params}, _from, state),
12     do: {:reply, build_response(), state}
13
14   def call({:put_item, "/food", params}, _from, state) do
15     Deadwood.Food.put(params)
16
17     response = build_response()
18     |> Map.merge(%{params: params})
19
20     {:reply, response, state}
21   end
22
23   def build_response do
24     items = Deadwood.Food.get()
25     count = Enum.count(items)
26     average_price = items
27     |> Enum.map(&(&1.cost))
28     |> Enum.sum()
29     |> Kernel./(count)
30     |> Float.round(2)
31
32     %{
33       average_price: average_price,
```



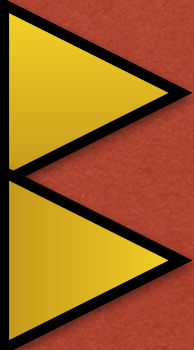

COYOTE AND THE SKUNK





COYOTE WAS TRAVELING

one day and began to feel very hungry, when he met up with skunk. "Hello, brother," Coyote said to him...



JOHNNY WINN

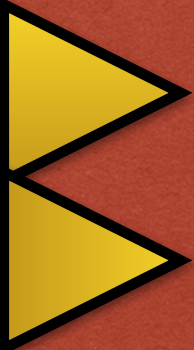
@johnny_rugger

<https://github.com/nurugger07>

ELIXIR FOUNTAIN

@elixirfountain

<https://soundcloud.com/elixirfountain>



COYOTE

<https://github.com/nurugger07./coyote>

GENSTAGE

https://github.com/elixir-lang/gen_stage