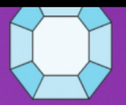
PART OF GLOBAL ELIXIR MEETUPS






INTRODUCTION TO

ASH & ASH AI

SEPTEMBER 25, 2025 IN BOCHUM, GERMANY

Who am I

Daniel Hoelzgen

Software Engineer at 9elements

Rails developer switching to Elixir

Agenda

1. Introduction to Ash
2. Modeling the domain
3. Streaming conversations
4. Calling tools

Scope: Give an overview, not a tutorial

What is Ash

Model your domain, derive the rest

What is Ash

Resource-oriented, declarative domain modeling framework

- Resource oriented
  - Domain driven, centered around resouces
- Declarative instead of Imperative
  - Data vs. Code
  - Result vs. Instructions
- Extensible

Extensions

- Core (Resources, Actions & Domains)
- Persistence (Postgres, SQLite, ...)
- Authentication & Authorization
- Background Jobs (Oban)
- Reactor
- ...

Integrate into...

- Phoenix: Forms, LiveView, ...
- GraphQL, JSON API

Is it like Rails / Django / Laravel?

No.

It is not a Web Application Framework, it is a Domain Modeling Framework.

But together with Phoenix & AshPhoenix, you can build Web Apps with it.

- Configuration over Convention
- Extensions over Customization

Modeling the Domain

Ash Core & AshPostgres Extension

A magic robot helping with events

The AshIntro Demo Project

- Chatbot providing info about events
- Events are automatically scraped from web pages
- These web pages are configured via CRUD

We work through these tasks in reverse order

Starting

- Use igniter to init project or install dependencies
- If you start from scratch, you can build the initial command on ash-hq.org
- Use generators (backed by igniter) to scaffold if you want

```
mix ash.gen.resource AshIntro.Events.EventPage --extend postgres
mix ash.gen.resource AshIntro.Events.Event --extend postgres
```

What do we get?

Event Page Resource

```
defmodule AshIntro.Events.EventPage do
  use Ash.Resource,
    otp_app: :ash_intro,
    domain: AshIntro.Events,
    data_layer: AshPostgres.DataLayer

  postgres do
    table "event_pages"
    repo AshIntro.Repo
  end
end
```

What do we get?

Event Resource

```
defmodule AshIntro.Events.Event do
  use Ash.Resource,
    otp_app: :ash_intro,
    domain: AshIntro.Events,
    data_layer: AshPostgres.DataLayer

  postgres do
    table "events"
    repo AshIntro.Repo
  end
end
```

What do we get?

Event Domain

```
defmodule AshIntro.Events do
  use Ash.Domain,
    otp_app: :ash_intro

  resources do
    resource AshIntro.Events.EventPage
    resource AshIntro.Events.Event
  end
end
```

This domain is also registered as a domain used by the app in config.exs.

Next steps?

1. Define attributes on resources
2. Define relations on resources
3. Define actions on resources
4. Expose these actions through the domain's code interface

Resource attributes

```elixir
defmodule AshIntro.Events.EventPage do
  attributes do
    timestamps()
    uuid_v7_primary_key :id

    attribute :name, :string, allow_nil?: false
    attribute :url, :string, allow_nil?: false

    attribute :crawl_state, :atom do
      constraints one_of: [:ready, :requested, :processing, :error]
      default :ready
    end
  end

  ...
```

Relations

```elixir
# In AshIntro.Events.EventPage
relationships do
  has_many :events, AshIntro.Events.Event do
    public? true
    sort event_date: :asc
  end
end


# In AshIntro.Events.Event
relationships do
  belongs_to :event_page, AshIntro.Events.EventPage do
    public? true
    allow_nil? false
  end
end
```

Actions

```
defmodule AshIntro.Events.EventPage do
  actions do
    read :read do
      primary? true
    end

    create :create do
      primary? true
      accept [:name, :url]
    end

    ...
```

Actions

Shorter for defaults

```
defmodule AshIntro.Events.EventPage do
  actions do
    default_accept [:name, :url]
    defaults [:create, :read, :update, :destroy]
  end

  ...
```

Domains / Code Interface

```
defmodule AshIntro.Events do
  resources do
    resource AshIntro.Events.EventPage do
      define :list_event_pages,
        action: :read,
        default_options: [query: [sort: [inserted_at: :asc]]]

      define :get_event_page_by_id,
        action: :read,
        get_by: [:id]

      define :create_event_page, action: :create

      ...
```

Domains / Code Interface

Also generates proper docs

```
iex(4)> h AshIntro.Events.get_event_page_by_id

            def get_event_page_by_id(id, params \\ nil, opts \\ nil)

Calls the read action on AshIntro.Events.EventPage.

## Options

  * :page - Pagination options, see Ash.read/2 for more.
  * :load (t:term/0) - A load statement to add onto the query
  * ...
```

Migrations & Snapshots

Igniter task: mix ash.codegen

- Creates snapshot of your resources
- Compares to previous snapshots
- Generates migrations based on comparison

```
mix ash.codegen create_event_tables
mix ash.migrate
```

Connecting to the Views

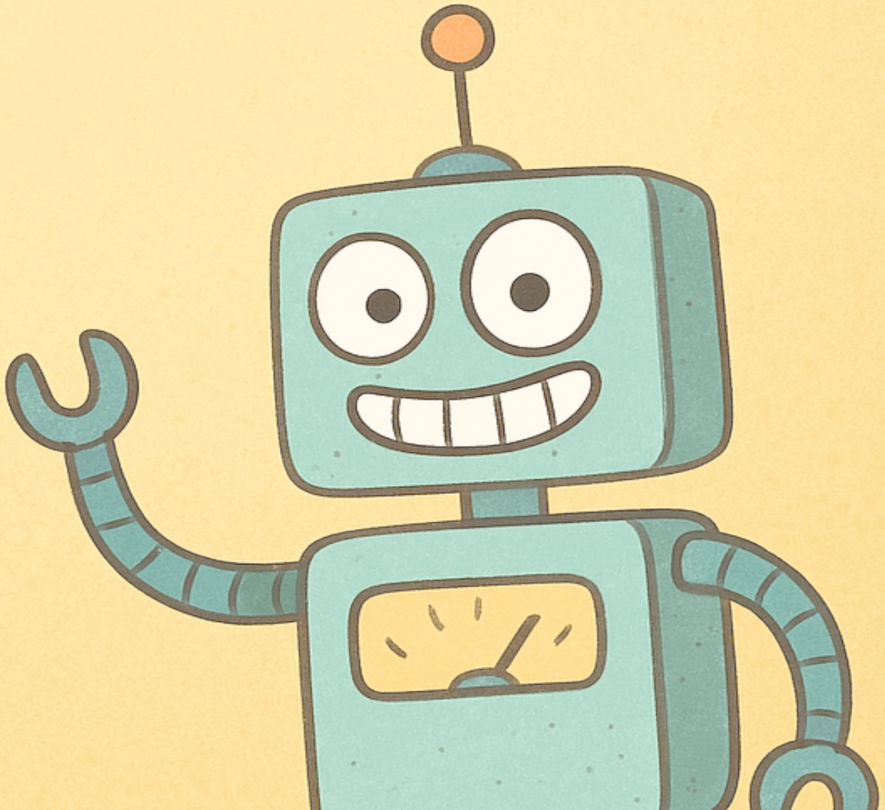
The AshPhoenix Extension

AshPhoenix

- Bridges Ash resources with Phoenix
- Translates Ash changesets into Phoenix Forms
- Includes parameter handling, error propagation & filter forms

Demo

CRUD Event Pages



Creating Event Pages

Create form

```
# remember: in domain

resource AshIntro.Events.EventPage do
  define :create_event_page, action: :create

# now: use it in live view

event_page_form =
  AshIntro.Events.form_to_create_event_page(actor: socket.assigns.current_user)
  |> to_form()

{:noreply, assign(socket, :event_page_form, event_page_form)}
```

Creating Event Pages

Validate

```
def handle_event("validate", %{"form" => params}, socket) do
  {:noreply,
   assign(
     socket,
     :event_page_form,
     AshPhoenix.Form.validate(socket.assigns.event_page_form, params)
   )}
end
```

Creating Event Pages

Submit

```elixir
def handle_event("submit", %{"form" => form_data}, socket) do
  socket =
    case AshPhoenix.Form.submit(socket.assigns.event_page_form, params: form_data) do
      {:ok, event_page} ->
        socket
        |> assign(:event_page_form, nil)
        |> stream_insert(:event_pages, event_page)

      {:error, event_page_form} ->
        socket |> assign(:event_page_form, event_page_form)
    end

  {:noreply, socket}
end
```

Works for create & update, action is baked into form.

There is more

- Validations
- Calculations & Aggregates
- Notifications
- Actors, Permissions & Policies
- Tenants
- ...

Scraping Events

AshAI / Prompt backed actions

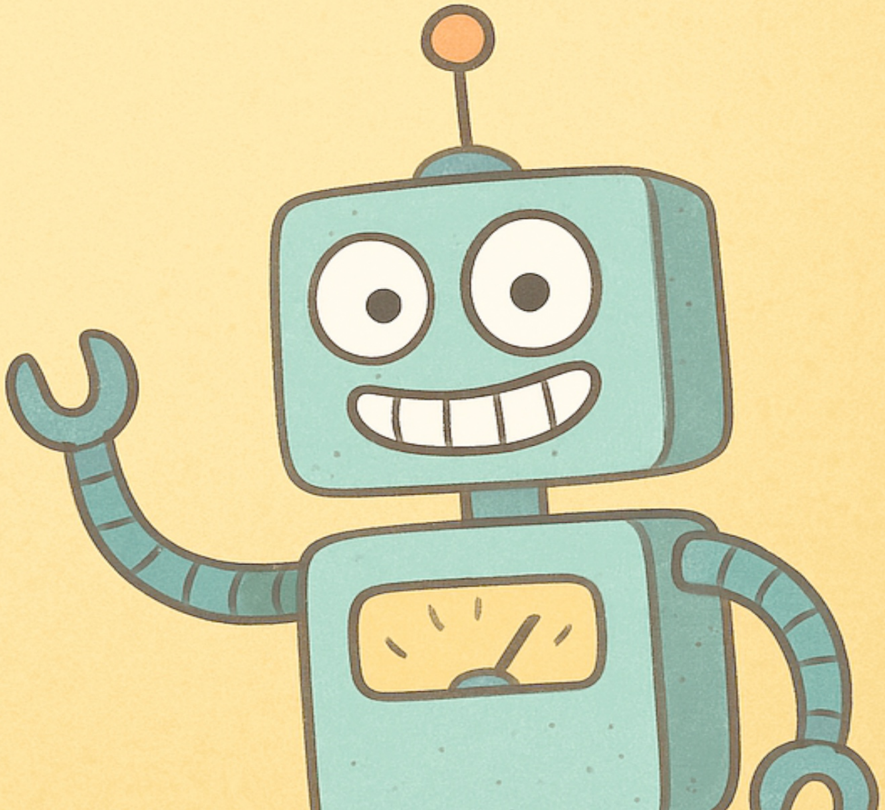
Oban, PubSub

Ash AI

- Prompt backed actions
- Actions as tools for LLMs
- Vectorization
- MCP

Demo

Scraper in action



Plan

- Run scraper in background with Oban
- Extract events with LLM backed action
- Notify (frontend) via PubSub

Preparations

Adding extensions, setting notifier

```
defmodule AshIntro.Events.EventPage do
  use Ash.Resource,
    otp_app: :ash_intro,
    domain: AshIntro.Events,
    data_layer: AshPostgres.DataLayer,
    extensions: [AshAi, AshOban],
    notifiers: [Ash.Notifier.PubSub]
  ...
```

Run scraper in background

```
update :crawl do
  accept []
  change set_attribute(:crawl_state, :requested)
  change run_oban_trigger(:process)
end


update :process do
  accept []
  require_atomic? false

  change {
    AshIntro.Scraping.Changes.Crawl,
    entity_resource: AshIntro.Events.Event, entity_foreign_key: :event_page_id
  }
end
```

Run scraper in background

```
oban do
  triggers do
    trigger :process do
      actor_persister AshIntro.ActorPersistor
      action :process
      queue :crawls
      where expr(crawl_state == :requested)
      worker_module_name AshIntro.Events.EventPage.AshOban.Worker.Process
      scheduler_module_name AshIntro.Events.EventPage.AshOban.Scheduler.Process
    end
  end
end
```

Scraper Module

We might do this later...

Right now, all we need to know is:

It scrapes the page,

then calls an extract update action on event page to extract drafts,

and then creates an entity resource based on the draft

Extract event drafts

```
action :extract, {:array, AshIntro.Events.EventDraft} do
  argument :text, :string, allow_nil?: false

  run prompt(
        LangChain.ChatModels.ChatOpenAI.new!(%{model: "gpt-4.1"}),
        tools: false,
        prompt: "Extract events from this content: <%= @input.arguments.text %>"
      )
end
```

Extract events drafts

```
defmodule AshIntro.Events.EventDraft do
  use Ash.TypedStruct

  typed_struct do
    field :name, :string, allow_nil?: false, description: "The title of the event"
    field :description, :string, description: "A brief description of the events"
    field :event_date, :string, description: "ISO-8601 UTC timestamp"

    field :event_location, :string,
      description: "The location of the event, can be Online / Zoom if applicable"
  end
end
```

Create events from drafts

```
defp create_entities(container, drafts, opts, ash_opts) when is_list(drafts) do
  event_page_id = Keyword.fetch!(opts, :event_page_id)

  drafts
  |> Enum.map(fn draft ->
    attrs =
      Map.from_struct(draft)
      |> Map.put(event_page_id, container.id)

    AshIntro.Events.Event
    |> Ash.Changeset.for_create(:create, attrs, ash_opts)
    |> Ash.create()
  end)
end
```

Notifiy the others

AshIntro.Events.EventPage

```
pub_sub do
  module AshIntroWeb.Endpoint
  prefix "event_pages"

  transform fn %{data: event_page} ->
    %{event_page: event_page}
  end

  publish_all :update, "all", except: [:process]
end
```

Notify the others

AshIntro.Events.Event

```
pub_sub do
  module AshIntroWeb.Endpoint
  prefix "events_for_event_page"

  transform fn %{data: event, action: action} ->
    %{event: event, event_page_id: event.event_page_id, action: action.name}
  end

  publish :create, "all"
  publish :update, "all"
  publish :destroy, "all"
end
```

Handle event_pages updates

```
# inside events_live.ex

def handle_info(
      %Phoenix.Socket.Broadcast{
        topic: "event_pages:all",
        payload: %{event_page: event_page}
      },
      socket
    ) do
  socket =
    socket
    |> stream_insert(:event_pages, event_page)

  {:noreply, socket}
end
```

Handle events updates

```
# inside events_live.ex

def handle_info(
      %Phoenix.Socket.Broadcast{
        topic: "events_for_event_page:all",
        payload: %{event_page_id: page_id} = payload
      },
      socket
    ) do
  send_update(
    AshIntroWeb.EventsLive.EventsForPageComponent,
    id: "events-list-container-#{page_id}",
    payload: payload
  )

  {:noreply, socket}
end
```

Handle events updates

```elixir
# inside component, piped events from parent live_view

def update(%{payload: %{action: action, event: event}}, socket) do
  {:ok, upsert_from_action(socket, action, event)}
end

defp upsert_from_action(socket, :create, ev),
  do: socket |> stream_insert(:events, ev) |> update(:event_count, &(&1 + 1))

defp upsert_from_action(socket, :update, ev),
  do: stream_insert(socket, :events, ev)

defp upsert_from_action(socket, :destroy, ev),
  do: socket |> stream_delete(:events, ev) |> update(:event_count, &max(&1 - 1, 0))
```

Streaming chats

AshAI / **ash_ai.gen.chat**

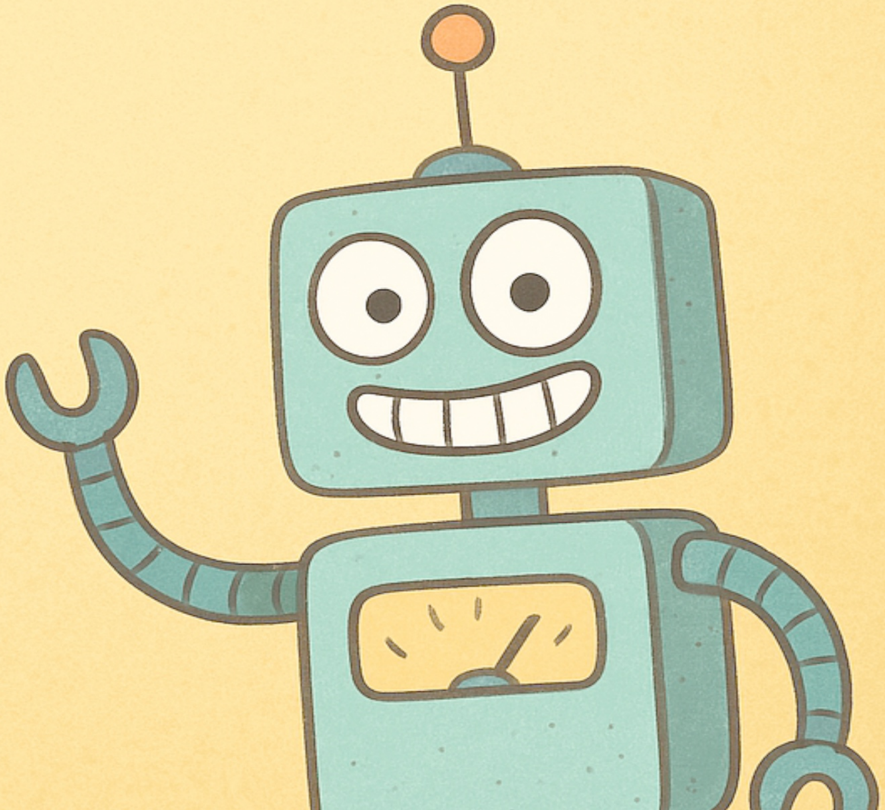
Basic idea

- Messages, organized in conversations
- User messages trigger Oban worker to respond with a system message
- System messages get streamed from LLM / LangChain

Most of the actual chat is not AshAI, but we will use it later

Demo

Chatbot



Message / Create Action

```
create :create do
  accept [:text]

  validate match(:text, ~r/\S/) do
    message "Message cannot be empty"
  end

  argument :conversation_id, :uuid do
    public? false
  end

  change AshIntro.Chat.Message.Changes.CreateConversationIfNotProvided
  change run_oban_trigger(:respond)
end
```

Message / Oban Trigger

```
oban do
  triggers do
    trigger :respond do
      actor_persister AshIntro.ActorPersistor
      action :respond
      queue :chat_responses
      lock_for_update? false
      scheduler_cron false
      worker_module_name AshIntro.Chat.Message.Workers.Respond
      scheduler_module_name AshIntro.Chat.Message.Schedulers.Respond
      where expr(needs_response)
    end
  end
end
```

Message / Needs Response Calculation

```
calculations do
  calculate :needs_response, :boolean do
    calculation expr(source == :user and not exists(response))
  end
end
```

Calculated fields, have to be loaded explicitly

Can be used in expressions (as for the trigger)

Side Note: Aggregates

A special kind of calculation

```
aggregates do
  count :event_count, :events do
    public? true
  end
end
```

Optimized for summarizing related data

Often pushed directly down the data layer (i.e. COUNT)

Message / Respond Action

```
update :respond do
  accept []
  require_atomic? false
  transaction? false
  change AshIntro.Chat.Message.Changes.Respond
end
```

```elixir
def change(changeset, _opts, context) do
  Ash.Changeset.before_transaction(changeset, fn changeset ->
    system_prompt = LangChain.Message.new_system!("... is #{Date.utc_today()}.")
    message_chain = message_chain(messages)

    %{
      llm: ChatOpenAI.new!(%{model: "gpt-4o", stream: true}),
      custom_context: Map.new(Ash.Context.to_opts(context))
    }
    |> LLMChain.new!()
    |> LLMChain.add_message(system_prompt)
    |> LLMChain.add_messages(message_chain)
    |> AshAi.setup_ash_ai(otp_app: :ash_intro, tools: [], actor: context.actor)
    |> LLMChain.add_callback(%{
      on_llm_new_delta: fn _model, data -> [...],
      on_message_processed: fn _chain, data -> [...]
    })
    |> LLMChain.run(mode: :while_needs_response)
```

How does the upsert change look like

```
change atomic_update(
         :text,
         {:atomic,
          expr(
            if ^arg(:complete) do
              ^arg(:text)
            else
              ^atomic_ref(:text) <> ^arg(:text)
            end
          )}
       )
```

Use PubSub Notifier to update frontend

```
pub_sub do
  module AshIntroWeb.Endpoint
  prefix "chat"

  publish :create, ["messages", :conversation_id] do
    transform fn %{data: message} ->
      message
    end
  end

  publish :upsert_response, ["messages", :conversation_id] do
    transform fn %{data: message} ->
      message
    end
  end
end
```

But we did that for events?

AshAI / Tool Calls

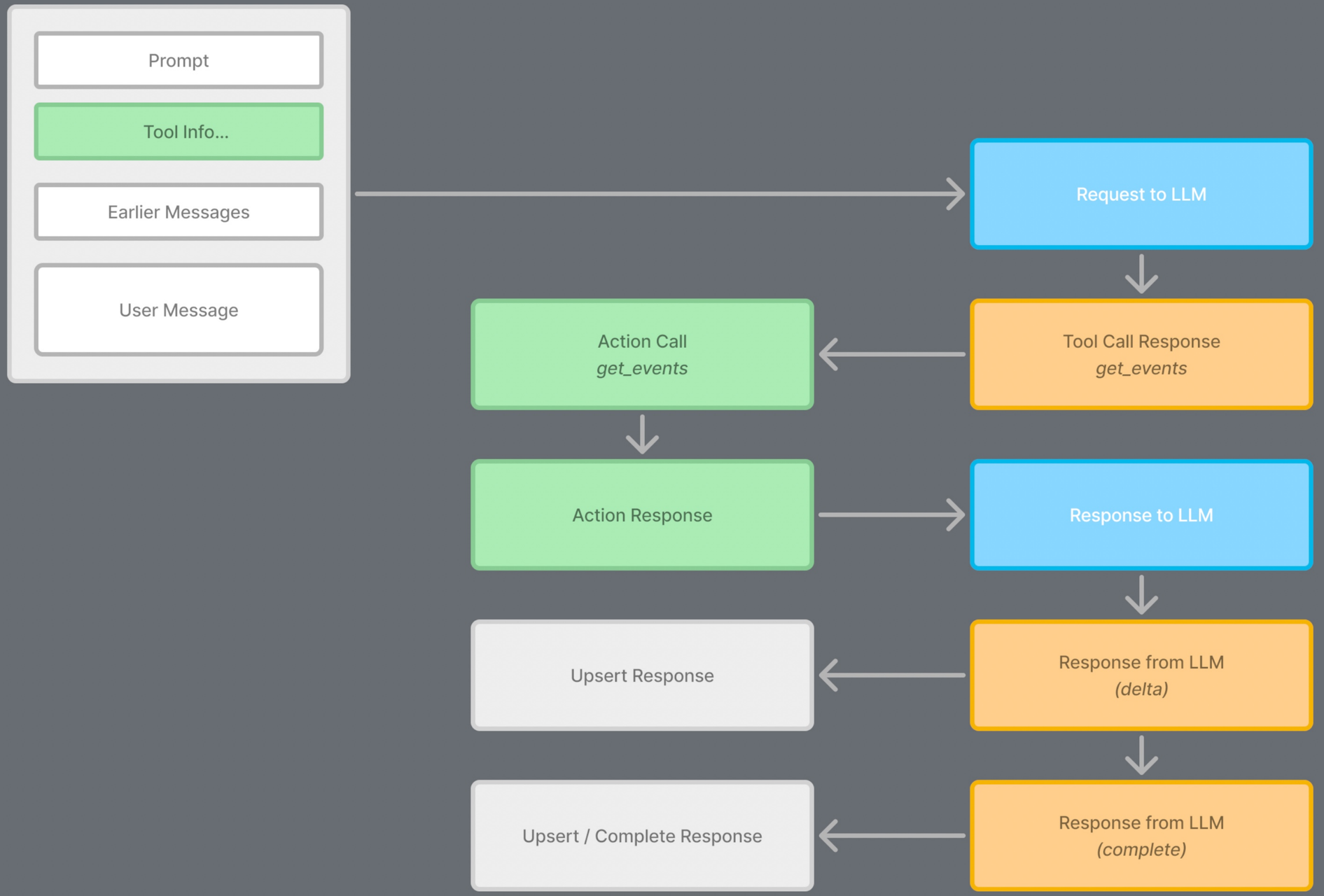
Expose tools in domain...

```
defmodule AshIntro.Events do
  use Ash.Domain,
    otp_app: :ash_intro,
    extensions: [AshAi, AshPhoenix]

  tools do
    tool :get_events, AshIntro.Events.Event, :read do
      description """
      Use this tool to get information about events.
      These events are gathered from differt sources, so
      check for duplicates before using this information.
      """
    end
  end

  ...
```

...and tell LangChain to use it

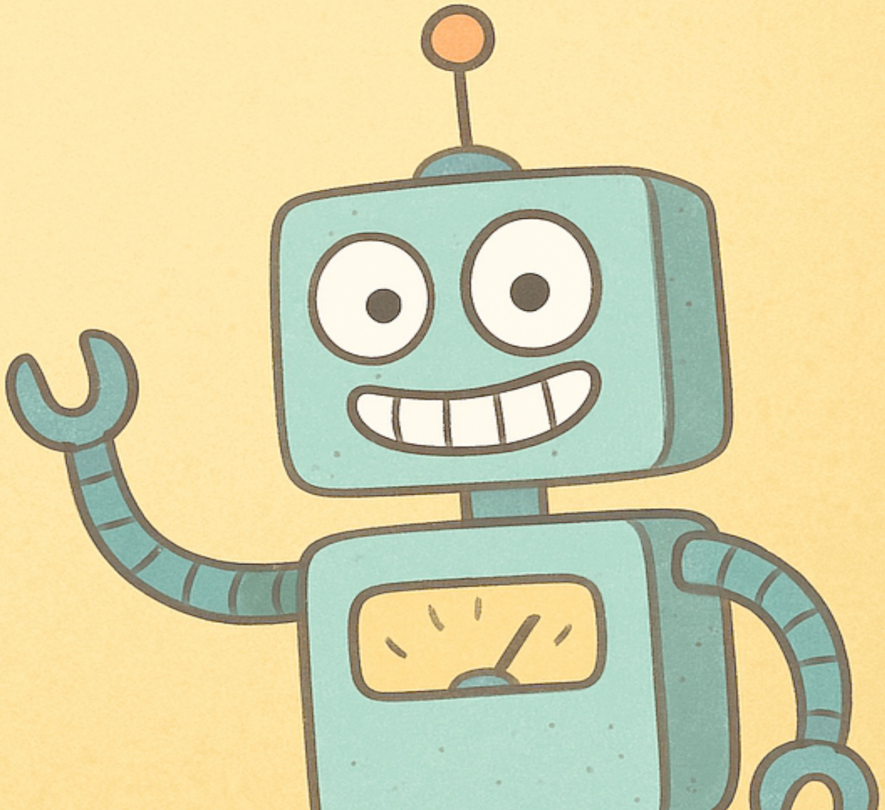
via AshAI

```
%{
  llm: ChatOpenAI.new!(%{model: "gpt-4o", stream: true}),
  custom_context: Map.new(Ash.Context.to_opts(context))
}
|> LLMChain.new!()
|> LLMChain.add_message(system_prompt)
|> LLMChain.add_messages(message_chain)
|> AshAi.setup_ash_ai(
  otp_app: :ash_intro,
  tools: [:get_events],
  actor: context.actor
)
|> LLMChain.add_callback(...
...
```


Prompt
Tool Info...
Earlier Messages
User Message
Request to LLM
Tool Call Response
get_events
Action Call
get_events
Action Response
Response to LLM
Response from LLM
(delta)
Upsert Response
Response from LLM
(complete)
Upsert / Complete Response

Demo

Chatbot



AshAI Tool Calling

- Tool exposed with just a few lines
- Complex querying based on AshQuery options out of the box
- Can use own read action for providing additional options

Exposing Attributes

As with AshJSON / GraphQL, attributes need to be marked as public

```
defmodule AshIntro.Events.Event do
  ...
  attributes do
    timestamps()
    uuid_v7_primary_key :id

    attribute :name, :string, public?: true, allow_nil?: false
    attribute :description, :string, public?: true, allow_nil?: true
    attribute :event_date, :datetime, public?: true, allow_nil?: false
    attribute :event_location, :string, public?: true
  end
```

Conclusion: Why I explored Ash

Small Projects

- No boilerplate for basic CRUD stuff
- Use Extensions for the basic building parts
- Focus on building the logic

Big Projects

- Implement and iterate on higher abstraction level
- Automatic change of 'how' without needing to touch the 'what'

Resources

- Ash HQ website as a starting point
- Ash Docs on hexdocs
- Ash Framework on PragProg by Rebecca Le & Zach Daniel
- Domain modeling with Ash Framework on devcarrots.com
- Elixir Mentor just started a course on Ash
- Usage Rules for coding with AI

```
mix igniter.install usage_rules
mix usage_rules.sync CLAUDE.md --all
```

Community

- Ash Discord - Zach & core team is very active there
- Elixir Discord
- Elixir Forum

Our own Elixir Ruhr community

- Join our Discord
- Might publish the demo code if someone's interested

THANK YOU