

# SHORTER FEEDBACK LOOPS WITH LIVEBOOK

Linus De Meyere

[github.com/linusdm/livebook\\_fosdem](https://github.com/linusdm/livebook_fosdem)

# WHO KNOWS LIVEBOOK?



# GOALS

- Introduction to Livebook
- How to get Livebook
- Livebook stories from the trenches
- Starting in the middle with Livebook

# Home

Import

New notebook

/Users/linus/dev/ + Fork Open

- ..
- amazing\_elixir
- blog
- data\_schema
- dovecot
- flatpickr\_phoenix\_exa...
- flow
- livebook
- livebook\_fosdem
- one-to-many-form
- single\_file\_phx\_bumbl...

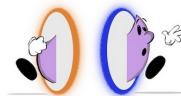
## LEARN

[See all →](#)



### Welcome to Livebook

Get to know Livebook, see how it works, and learn its features.



### Distributed portals with Elixir

A fast-paced introduction to Elixir by building distributed data-transfer portals.



### Elixir and Livebook

Learn how to use some of their unique features together.

## RUNNING SESSIONS (1)

27.2 GB / 34.4 GB

Edit

Date

### My first Livebook

/Users/linus/dev/livebook\_fosdem/start.livemd

0 MB Created 3 minutes ago



# My first Livebook



```
1 Mix.install([
2   {:kino, "~> 0.8.1"}
3 ])
```

Evaluated

nil

## Some title

```
1 IO.puts("Hello from Livebook!")
```

Evaluated

Hello from Livebook!

:ok

Any markdown goes here



# LEARNING


Livebook v0.8.1

- Home
- Learn
- Settings

?

## Learn


Check out a number of examples showcasing various parts of the Elixir ecosystem. Click on any notebook you like and start playing around with it!



### Welcome to Livebook


Get to know Livebook, see how it works, and learn its features.

[Open notebook](#)



### Distributed portals with Elixir

A fast-paced introduction to Elixir by building distributed data-transfer portals.




### Elixir and Livebook

Learn how to use some of their unique features together.



### Introduction to Kino

Make your notebooks interactive with inputs, controls, and more.



### Plotting with VegaLite

Learn how to quickly create numerous plots for your data.



### Maps with MapLibre

Seamlessly plot maps using geospatial and tabular data.



5 notebooks

Deep dive into Kino

**IT'S JUST MARKDOWN**

paulo-valim Rename the Explore section to Learn (#1424) ... ✓

Latest commit 285bc92 on Sep 21, 2022  History👤 2 contributors  

☰ 158 lines (112 sloc) | 5.12 KB

Raw

Blame



# Welcome to Livebook

---

## Basic usage

---

Livebook is a tool for crafting **interactive** and **collaborative** code notebooks.

Each notebook consists of a number of cells, which serve as primary building blocks. There are **Markdown** cells (such as this one) that allow you to describe your work and **Code** cells to run your Elixir code!

To insert a new cell move your cursor between cells and click one of the revealed buttons. 📌

```
# This is a Code cell - as the name suggests that's where the code goes.  
# To evaluate this cell, you can either press the "Evaluate" button above  
# or use `Ctrl + Enter` (or Cmd + Enter on a Mac)!
```

```
message = "hey, grab yourself a cup of 🍷"
```

Subsequent cells have access to the bindings you've defined:

```
String.replace(message, "🍷", "☕")
```

Note however that bindings are not global, so each cell sees only stuff that goes above itself. This approach helps to keep the notebook clean and predictable as you keep working on it!

## Sections

---



# IT'S JUST MARKDOWN

livebook / lib / livebook / notebook / learn / intro\_to\_livebook.livemd in livebook-dev:main

Cancel changes

<> Edit file  Preview

Spaces 2 Soft wrap

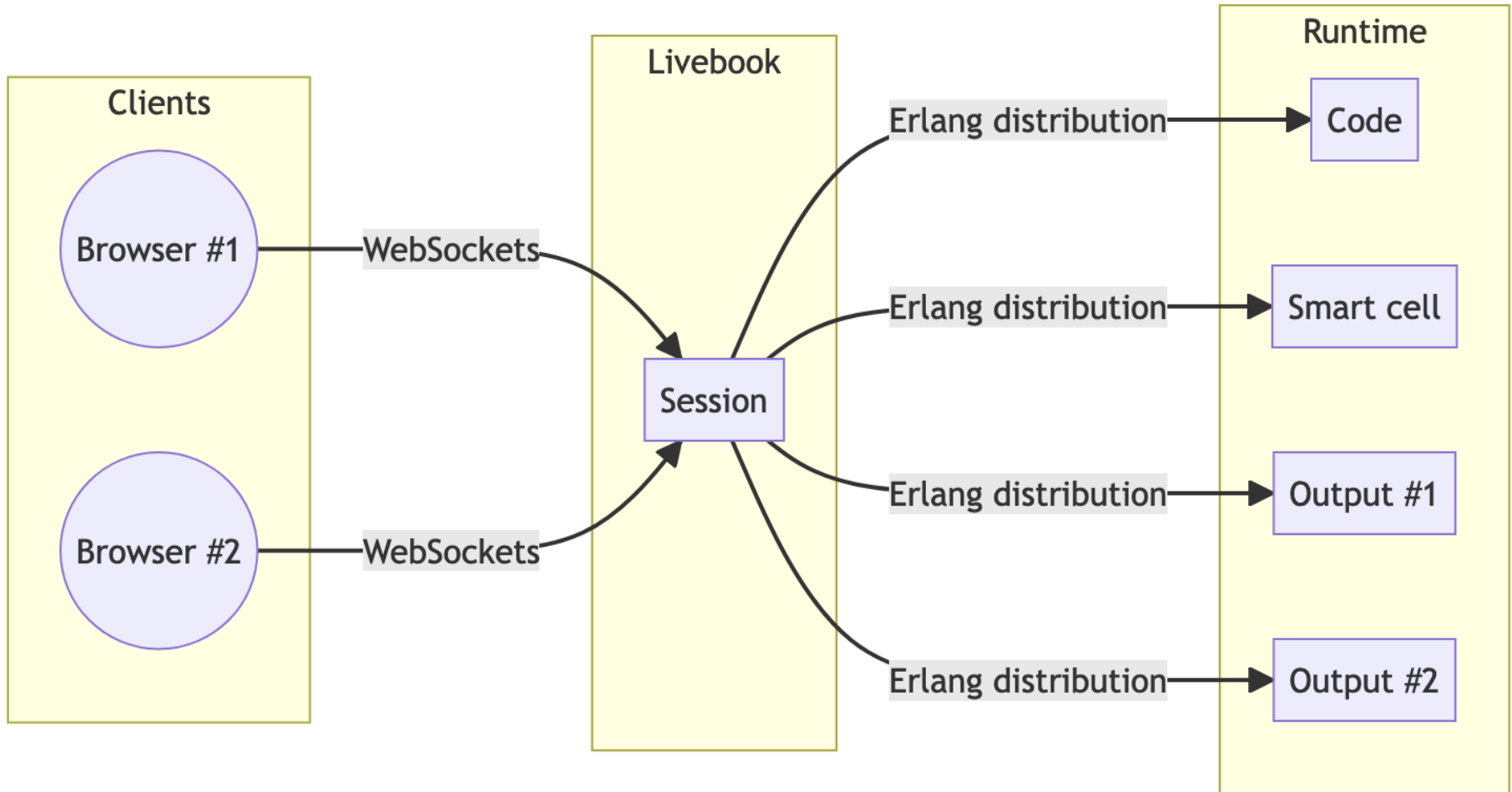
```
1 # Welcome to Livebook
2
3 ## Basic usage
4
5 Livebook is a tool for crafting interactive and collaborative code notebooks.
6
7 Each notebook consists of a number of cells, which serve as primary building blocks.
8 There are Markdown cells (such as this one) that allow you to describe your work
9 and Code cells to run your Elixir code!
10
11 To insert a new cell move your cursor between cells and click one of the revealed buttons. 📌
12
13 ```elixir
14 # This is a Code cell - as the name suggests that's where the code goes.
15 # To evaluate this cell, you can either press the "Evaluate" button above
16 # or use `Ctrl + Enter` (or Cmd + Enter on a Mac)!
17
18 message = "hey, grab yourself a cup of ☹️"
19 ```
20
21 Subsequent cells have access to the bindings you've defined:
22
23 ```elixir
24 String.replace(message, "☹️", "😁")
25 ```
26
27 Note however that bindings are not global, so each cell sees only stuff that goes
28 above itself. This approach helps to keep the notebook clean and predictable
29 as you keep working on it!
30
31 ## Sections
32
33 You can leverage so called sections to nicely group related cells together.
34 Click on the "Book" icon (<i class="ri-livebook-sections"></i>) in the sidebar
```

- Automatically save to filesystem
- Autocompletion
- Inline documentation
- Code formatting

# REPRODUCIBILITY

- No global mutable state
- Sequential model for running code cells
- Efficient change tracking - *stale* cells
- Branching sections
- Package management in the notebook itself

# ERLANG VM PROCESSES AND DISTRIBUTION EVERYWHERE



# INSTALLATION

- Desktop application (Windows and Mac)
- Escript
- Docker image
- In the cloud - somewhere remote

```
mix escript.install hex livebook
```

```
docker run -p 8080:8080 -p 8081:8081 --pull always livebook/liveb
```

# STARTING

run the desktop app, or

```
livebook server
```

```
livebook server new
```

```
livebook server path/to/directory/
```

```
livebook server path/to/some.livemd
```

```
livebook server https://example.com/some_public.livemd
```

```
livebook server --help
```

^ lots of startup/deployment options here

*“Start with the riskiest parts of your development”*

Every project manager

# BENEFITS

- Start in the middle
- Increase transparency
- Document the process
- Livebooks as shareable deliverables
- Lower the barriers to entry (also for non-coders)



# CONTEXT

- Small software shop doing custom development
- Many projects at the same time
- Small teams (teams of two)
- Important to have good DX
- Good documentation really helps
- Communication with clients is key



## CASE #1

# EXPLORING AN UNDOCUMENTED LEGACY API

- Low level TCP protocol
- Use `:gen_tcp` to send and receive messages
- Stub out the server for end-to-end scenarios
- Great for documentation purposes (no meta info available)
- Collaborate and create a shared understanding of the system

# Documenting the various messages

## Login

Log the user in and get a reference that is used to authenticate subsequent requests.

input [client code, username, password]

## Successful login

▶ Reevaluate ▾



```
1  {:ok, ["10", _message | _]} = Client.send_request_and_receive_response(54, credentials, endpoint)
```

Evaluated ●

```
{:ok,  
 [  
  "10",  
  "Login OK",
```

^ pattern matching is awesome 

## CASE #1








# EXPLORING AN UNDOCUMENTED LEGACY API


- Reproduce bugs (can be referenced in github issues)
- Facilitates discussions on the right level of abstraction
- Verify bugfixes, without having to integrate in a real application
- idea: record test fixtures (thanks [Adam Lancaster](#))
- idea: generate template for documentation

# Mix project integration

leans on [Mix.install/2](#)

my\_app/ ..

- >  assets
- ▼  config
  - 💧 config.exs
  - 💧 dev.exs
  - 💧 prod.exs
  - 💧 runtime.exs
  - 💧 test.exs
- >  lib
- ▼  livebooks
  - ≡ start.livemd
- >  priv
- >  test
  - 💧 mix.exs
  - 💧 mix.lock
-  [README.md](#)

 start.livemd


```
1  # My app
2  ```elixir
3  root_path = Path.join(__DIR__, "..")
4
5  Mix.install(
6    [
7      {my_app, path: root_path},
8      {kino, "~> 0.8"},
9    ],
10   config_path: Path.join(root_path, "config/config.exs"),
11   lockfile: Path.join(root_path, "mix.lock")
12 )
13 ```
```

# TYPICAL LIFECYCLE WITH LIVEBOOK


1. Experiment with code in Livebook
2. Maybe add tests
3. Add `:path` dependency on local mix project
4. Promote reusable code to local mix project

# Manipulating stub server responses

```
1 frame = Kino.Frame.new() |> Kino.render()
2
3 fun = fn input ->
4   # inspect the input
5   Kino.Frame.render(frame, input)
6
7   # return mock result
8   {:ok, ["one", "two"]}
9 end
10
11 Kino.nothing()
```

Evaluated 

```
%{command: 99, reference: "0000000123", request: ["input 1", "input 2"]}
```

```
1 Supervisor.start_link([{:MyApp.Server, ref: :mock_server, fun: fun}], strategy: :one_for_one) Evaluated* 
```

```
{:ok, #PID<0.762.0>}
```

```
1 :ranch.get_port(:mock_server)
```

Evaluated 

```
51509
```



# Managing secrets

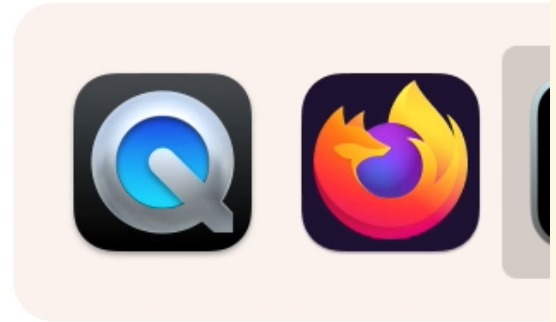
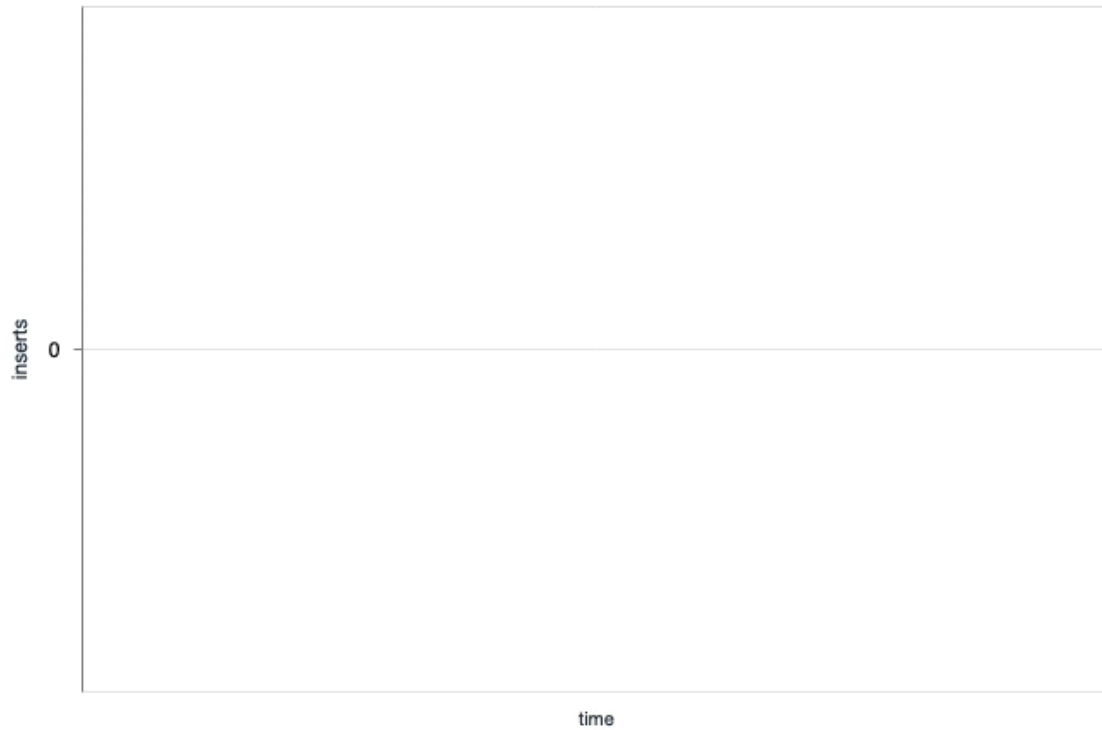
The screenshot shows a user interface for managing secrets. On the left, under the heading "APP SECRETS", there is a toggle switch for "ENDPOINT\_CREDENTIALS" which is currently turned on. The text "Toggle to share with this session" is visible below the heading. On the right, a code editor window titled "Reevaluate" contains two lines of code: `1 credentials = System.fetch_env!("LB_ENDPOINT_CREDENTIALS") |> String.split(":")` and `2 :ok`. The code is highlighted in a dark theme. Below the code, the output of the execution is shown as `:ok`. The code editor also features a toolbar with icons for zooming, settings, sharing, and other actions.

^ don't keep sensitive data in your notebooks

## CASE #2

# CONCURRENT ETL PIPELINE

- CSV → Postgres
- Concurrent data processing in Livebook
- The power of processes is available
- Fun with **Flow**
- Using Ecto from Livebook



```
1 ImportFlow.import_all_flow(model, update_fn: update_fn)
2 |> Flow.stream(link: false)
3 |> Stream.run()
```

Aborted ●

## CASE #3

# CONNECTING TO AN ONLINE ENVIRONMENT

- Remember, it's all erlang distribution behind the scenes
- You need your node's name/sname and cookie
- Great for one-off tasks
- Setup your first admin user
- Poke around your live system
- Implement features without a UI yet
- Remember, it's all live!

# Change your runtime settings to "attached node"

## Runtime settings

(esc) X

Elixir standalone

Attached node

Connect the session to an already running node and evaluate code in the context of that node. Thanks to this approach you can work with an arbitrary Elixir runtime. Make sure to give the node a name and a cookie, for example:

```
iex --sname test --cookie mycookie
```

Then enter the connection information below:

Name

my-app@45a0:b366:8af7:9dd1:3241:3e00:0e0d:539b

Cookie

mz5z5R2h1U6pDhMwkj4azrbwCf64FXpwHmSXSC1ov2ZOZfhEI+V6gChSRnr5nBX4

Connect

# TESTING IN LIVEBOOK

- Doctests are executed automatically
- You can write regular ExUnit test cases

# DBG() IN LIVEBOOK

- `dbg ( )` was recently added (elixir v1.14)
- Manipulate your pipeline live

## OTHER RESOURCES

- [The DockYard Academy](#): open source curriculum to help students learn Elixir
- [Project Bumblebee](#): Neural Networks in Livebook (GPT2, Stable Diffusion, ... on your computer!)





**NO NUMBATS HERE!**

