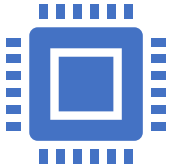
C++ in the Cloud

One NIF at a Time with Elixir

Sakshi Verma | Senior Software Engineer

Adobe

What is Elixir?



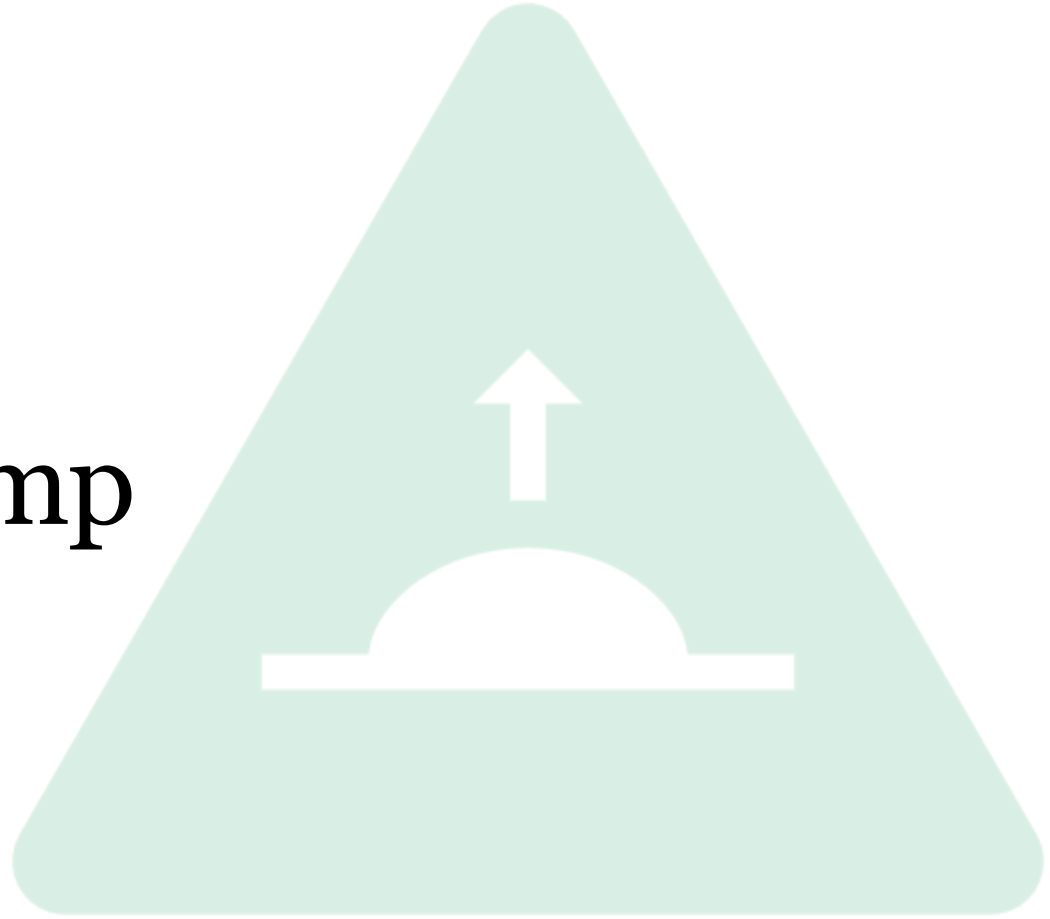
Functional, concurrent language built for scalability.

Ideal for managing large-scale cloud applications.

Built on the Erlang VM, ensuring resilience and fault tolerance.

The Smooth Sailing...

…and then The Bump



Enter Native Operations

Option 1: Rewrite all the native functionality in Elixir.

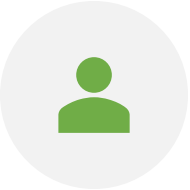
Option 2: Use already existing functionality from Native Code.


Here comes the NIFs

Native Implemented Functions (NIFs)

Call C/C++ from Elixir

Compiles to a shared library



Loaded into Erlang VM at startup

https://www.linkedin.com/in/sakverma

Why Write a NIF?

01

Erlang/Elixir ≠ CPU-heavy tasks.

02

Direct hardware interaction.

03

Interop with graphics libraries.

04

Existing Native libraries too complex to rewrite.


NIF Code: A Closer Look

```c
#include <erl_nif.h>
#include <xgboost/c_api.h>

ERL_NIF_TERM xgboost_build_info(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
        char out[1024];
        int result = XGBuildInfo(out);

        if (result == 0) {
          return enif_make_string(env, out, ERL_NIF_LATIN1);
        }
        else {
          return enif_make_badarg(env);
    }
}

// NIF initialization (maps Elixir function to C++ function)
static ErlNifFunc nif_funcs[] =
{
   {"xgboost_build_info", 0 , xgboost_build_info}
};

// Initialize the NIF module
ERL_NIF_INIT(Elixir.XGBoost.NIF, nif_funcs, NULL, NULL, NULL, NULL)
```

```elixir
defmodule XGBoost.NIF do
    @on_load :on_load

    # Load the NIF when the module starts
    def on_load do
     :erlang.load_nif("./libxgboost", 0)
    end

    # Elixir function that interfaces with the C++ NIF
    def xgboost_build_info do
      raise "NIF not loaded"
    end
end
```

Content Credentials: Generated with Microsoft Copilot

Crashes affect the whole VM.

Lacks fault tolerance and isolation.

Not interrupted by the Erlang scheduler.


Bon Voyage