That Eureka! Moment:

An Elixir
For Understanding

BEAM, Erlang, and OTP

Ray McCord

05/2017

The Notion of Machine

Given energy and material *input*

|> perform work *processing*

|> produce a result *output*

Classical Machines and Engines

Hardware

Physically implements input, processing, and output mechanisms

Classical Machines and Engines

Software

Human ingenuity to physically configure mechanisms at work time

The Computational Engine

Hardware

Physically implements input, processing, and output mechanisms

- Physically implements computational algorithms and formulas
- Physically conveys and processes data/information

The Computational Engine

Software

Human ingenuity to physically configure mechanisms at work time

- Recorded configurations in physically encoded form, for later use at work time

The Modern Digital Computer

Hardware

Physically implements input, processing, and output mechanisms

- Physically implements computational algorithms and formulas
- Physically implements digitally-encoded (binary) processing instructions
- Physically stores, conveys and processes instructions and data/information

The Modern Digital Computer

Software

Collections of higher-level instructions for interacting with hardware

- **Kernel**
  - Manages and provides software interfaces to interact with hardware
  - Integrates software to interact with specific hardware (drivers)
- **Operating System**
  - Provide essential services to application software
  - Provides higher-level vocabularies for lower-level kernel interactions
    - System programming languages, libraries, compilers, and interpreters
- **Application**
  - Recorded instructions for performing specific user work

The Virtual Machine (VM)

Physical Machine Host

A modern digital computer

The Virtual Machine (VM)

Virtual Machine Environment

An application running on a physical machine host that provides a complete virtual representation of a modern digital computer

- **Virtual Hardware (Virtualization Layer)**
  - Translates interactions between virtual and physical machine kernels
  - Software emulation of hardware not available via the physical machine
- **Virtual Software (Runtime Environment)**
  - Provides kernel services and interfaces
  - Provides operating system services
  - Provides higher-level vocabularies
    - System programming languages, libraries, compilers, and interpreters

The Network

- Two or more physical machines communicating
- Local and remote applications working together to achieve an ultimate result
- Broadcast or routed communications

The Cloud (Network Virtual Machine)

- Dividing or combining the aspects of physical machines virtually across a network
- Local and remote hardware transparently acting as one resource
- Local and remote software and services orchestrated into a single virtual application

Our Stack

BEAM

A network virtual machine

- Technically, the system programming language is BEAM intermediate language (bytecode)
- Technically, the system interpreter is the BEAM intermediate language interpreter

Our Stack

Erlang

The first of a number of programming languages that compile to the BEAM intermediate language

- The application level programming language BEAM was built specifically to implement and execute
- Provides a REPL, standard libraries, basic language tools, and HiPE for compiling to machine language
- Provides constructs for interchangeable code (modules) and hot module swapping (via BEAM)
- Provides virtual processes and per-process resource management (more on this later!)

Our Stack

Open Telecom Platform (OTP)

Robust, free telecommunications platform for Erlang

- Runs approximately half of the global telecommunications infrastructure
- Provides industrial-grade communications protocols, libraries, and tools
- Provides orchestration of Erlang virtual processes (important!)
- Provides location-agnostic referential transparency
  - Any named resource is accessed the same, whether local or remote
  - Uses message-passing instead of memory address reference pointers
  - Allows factorial (!) horizontal scaling (yes, really)
- Engineered to the goal of being fault-tolerant and self-healing
  - Let individual virtual processes crash, but never the overall system
  - Expect bad input and unreliable network conditions

Erlang Virtual Processes

- Are managed collections of virtual resources given a unique name
- Have no correlation with "machine," "kernel," or "OS" threads or processes
- Exist only within the BEAM virtual environment
- Have their own memory, heap, and garbage collection
- Can be used to model virtually anything in your application:
  - Program instructions
  - State information
  - Input, output, and stream buffers
  - Program data structures
  - Raw binary data
  - Local and remote services
  - Diagnostics, analytics, and logging

Cloud In A Box

- **Virtual processes**
  - Every logical resource is its own process
- **Virtual process orchestration and management**
  - Spawn, kill, clone, monitor, persist, and pass messages
- **Treat every process as a networked node**
  - Broadcast or route messages between virtual processes
    - This is where the "cast" and "call" come from
- **Manage the life cycle policy of your virtual processes**
  - Whether and when restarted, replaced, persisted, restored, etc.
- **Embrace the cloud architecture**
  - Local and remote resources composed into a single virtual application
  - Expand into choreography amongst multiple virtual applications
    - That means you could build a bona fide platform

Infernal Games Platform

- Account services
- Lobby servers
- Game servers
- Command and control (admin)
- Asset servers (CDN)
- In-game purchases and DLC management
- Presence
- Multi-channel in-game text chat
- Guild/team/squad services
- In-game hand-off to other game servers (maps, dimensions, levels, battles, events, cross-server)
- Voice chat routing (discord, etc.)
- Shard management
- Support servers
- Hosted community
- Partner hosting
- Game and DLC catalog
- Title (specific game) websites
- Public network website
- Developer network website
- Developer documentation and support
- Build services and continuous integration

What About Elixir?

Everything we've discussed applies.

Elixir is Erlang

- Erlang lets you expand the language through meta-programming
- Elixir is just Erlang expanded through normal meta-programming

Elixir is Magical

- Elixir can directly call into and get called from Erlang code
- Elixir can pass messages to and from Erlang virtual processes
- The entire Erlang and OTP ecosystem is available to Elixir for free
- Bridges exist to inter-operate with other programming languages
- Phoenix and Ecto ('nuff said.)

Elixir is Elegant Power

- You get the power of Erlang with a modern, elegant, and expressive syntax
- Elixir was designed to take full advantage of the cloud-in-a-box architecture
- Your Elixir applications stand on the shoulders of giants
- You can expand Elixir yourself and have a direct impact in the evolution of the core language, community, and ecosystem

Get It, Now?

(Q & A)

Eureka!

Now go build something mind-blowing.

ray.mccord@gmail.com