ELIXIR

DEPLOYMENT

WHAT WE WILL COVER

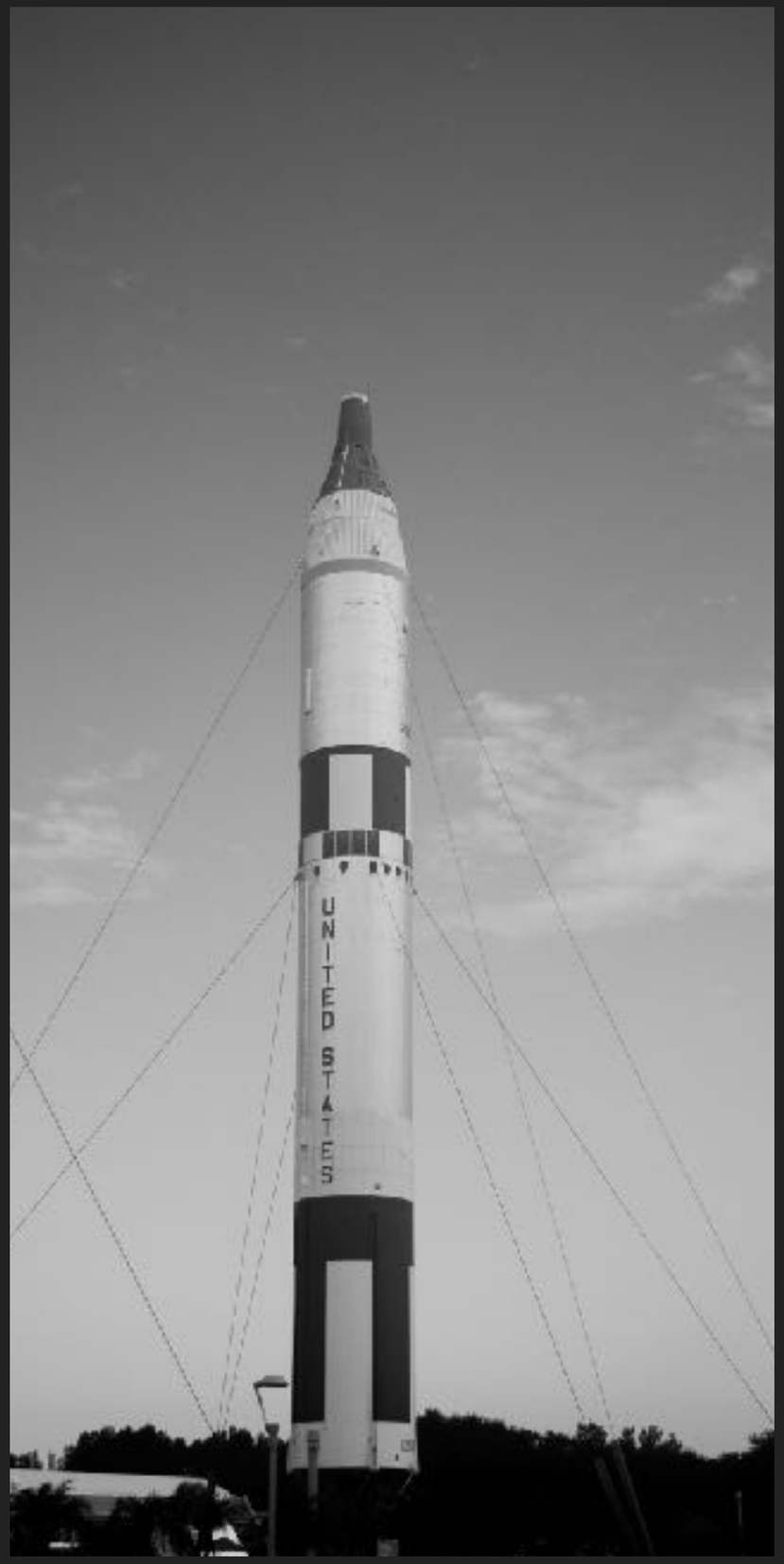
- Compilation
- Building releases with exrm (or now) distillery
- Packaging with Docker
- ENV and secrets

WON'T BE COVERING

- EDeliver or other methods
- Hot-reloading
- Erlang Clustering
- Umbrella projects

COMPILATION

- Elixir is a *compiled* language
- Gotcha #1: Must compile your release on the same *OS/architecture* as the system you intend to run it on
- Make your build-box the same thing as production
- or build inside e.g. Docker



EXRM AND DISTILLERY

- exrm: https://github.com/bitwalker/exrm
- distillery: https://github.com/bitwalker/distillery

- Both tools for generating Elixir “releases”
- These are self-contained distributions of “erts” and your application
- ERTS: Erlang Run Time System

```elixir
use Mix.Releases.Config,
    default_release: :default,
    default_environment: :prod


environment :prod do
  set include_erts: true
  set include_src: false
  set cookie: :"Lp!UZ:|ko<jC6hBzv6?J?{NV0F^)t_^vF4kkTv^O<T,],i_AAX@+[qUI>_jTa6Mc"
end

release :deploy_talk do
  set version: current_version(:deploy_talk)
end
```

rel/config.exs

```
NAME = deploy_talk
VERSION = 1

all: build_compiler compile build_release

build_compiler:
	docker build -f Dockerfile.build -t $(NAME)-build:latest .

#  The CMD in the Dockerfile actually defaults to this, but is repeated for clarity
compile:
	docker run --rm -v `pwd`:/app -e MIX_ENV=prod $(NAME)-build:latest \
		mix do compile, phoenix.digest, release --verbose

build_release:
	docker build -f Dockerfile.release -t $(NAME):$(VERSION) .

run:
	docker run --rm -p 4000:4000 $(NAME):$(VERSION)

clean:
	docker rmi $(NAME):$(VERSION)
	docker rmi $(NAME)-build:latest
	rm -rf rel/deploy_talk
```

Makefile

Dockerfile.build

```
FROM elixir:1.3

RUN mkdir -p /app/deps

RUN mix local.hex --force
# RUN mix local.rebar --force
COPY tini-static /tini-static

VOLUME /app
WORKDIR /app

ENTRYPOINT ["/tini-static", "--"]
CMD mix do compile, phoenix.digest, release --verbose
```

Dockerfile.release

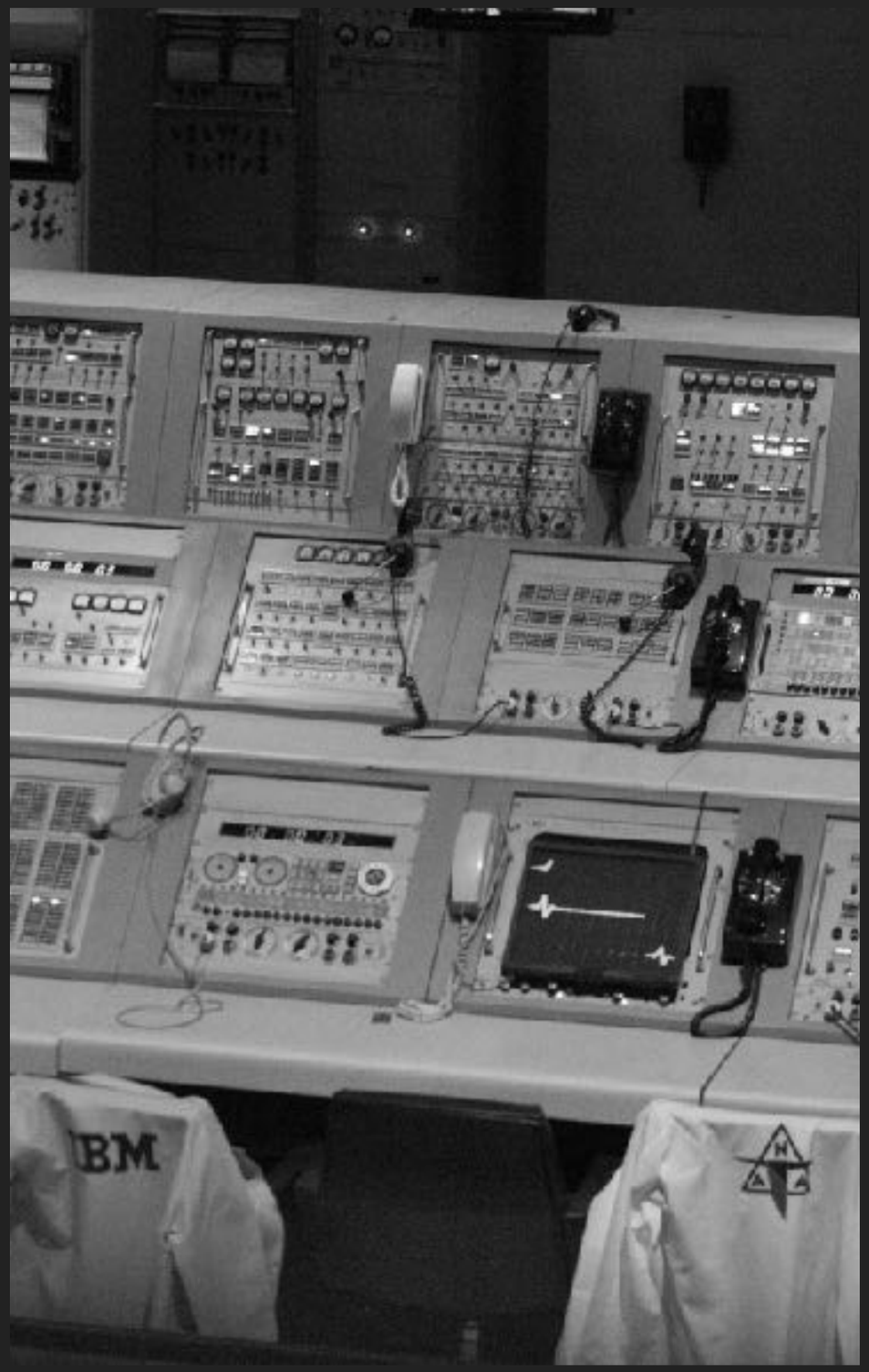
```
FROM debian:jessie

RUN apt-get update -y
RUN apt-get install -y libssl1.0.0 libsctp1

ENV LANG=C.UTF-8

COPY tini-static /sbin/tini-static

COPY rel /rel

ENV PORT=4000
EXPOSE 4000

ENTRYPOINT ["/sbin/tini-static", "--"]
CMD ["/rel/deploy_talk/bin/deploy_talk", "foreground"]
```

NATIVE LIBRARIES

- Gotcha #2: Releases don't automatically include all native dependencies!
- It's a good idea to check your release on staging before going live!

CONFIGURATION

- 12 Factor Apps
- Configuration through ENV
- Normally done with System.get_env/3
- Exrm & Distillery have a small problem though...



COMPILE-TIME VS RUNTIME ENVIRONMENT

- Gotcha #3 ?
- *System.get_env("VARIABLE")*
- ...is set at *Compile-time*
- But this can be useful...
- You can set secrets during CI build, and not have to set them in production!

COMPILE-TIME VS RUNTIME ENVIRONMENT

- How can I do runtime config then?
- https://github.com/renderedtext/ex-config
- It lets you use *{:system, "MYVAR"}* in your Mix.Config
- Use via *Config.get(:my_app, :key)*

config/prod.exs

```elixir
use Mix.Config


config :deploy_talk, DeployTalk.Endpoint,
  http: [port: {:system, "PORT"}],
  url: [host: "localhost", port: {:system, "PORT"}], # This is cr
  cache_static_manifest: "priv/static/manifest.json",
  server: true,
  root: "."

config :deploy_talk,
  my_secret: System.get_env("SECRET"),
  runtime: {:system, "RUNTIME"}

# Do not print debug messages in production
config :logger, level: :info
```

index.html.eex

```erb
<div class="jumbotron">
  <h1>Hello, Elixir Melbourne!</h1>
</div>

<div>
  <p> Secret: <%= secret %> </p>
  <p> Runtime: <%= at_runtime %> </p>
</div>
```

views/page_view.ex

```elixir
defmodule DeployTalk.PageView do
  use DeployTalk.Web, :view

  require Config

  def secret do
    {:ok, value} = Config.get(:deploy_talk, :my_secret, default: "unset")
    {:safe, value}
  end

  def at_runtime do
    {:ok, value} = Config.get(:deploy_talk, :runtime, default: "unset")
    {:safe, value}
  end

end
```

THANKS!

@JOFFOTRON