REAL WORLD ELIXIR DEPLOYMENT

Empire City Elixir | 2016-05-21

pete gamache | pete@gamache.org | @gamache

Hi!

Summary

- Elixir and Erlang make a rock-solid platform
- Erlang excels at long-lived clusters
- "Modern" deployment practices do not take advantage of Erlang core features
- Deploying Elixir can be painful, but if you are taking advantage of platform, can be worth it

Let's talk about an ideal platform

Platform Desirables

- Horizontal scalability
- Fault tolerance
- Zero-downtime upgrades and rollbacks
- Stability and performance
- Ease of administration and maintenance

Let's talk about an average platform

Devops 2010

- Fleets of ephemeral, stateless servers
- Load-balancer or shared work queue
- All message passing happens externally
- All state held externally

Devops 2010, cont.

- Upgrade: Start new servers, add to pool, remove old servers from pool, kill old servers
- Rollback: either same as above, or hot/cold a.k.a. blue/green deployment
- Many moving parts – provisioning, load balancer, connection draining

Devops 2016

- Mostly like that
- Something something Docker
- Heavily automated
- Far-out stuff: virtualized resource pools, e.g. Mesos

Desirables Achieved?

- Horizontal scalability and fault tolerance, check
- Zero-downtime upgrades and rollbacks, mostly
- Stability and performance, yeah sure
- Ease of maintenance, not so sure

Let's talk about the Erlang platform

Things Erlang Does

- Extreme fault tolerance
- Simple scalability to dozens of server nodes
- Zero-downtime upgrades/rollbacks (and fast)
- Stability and performance

Things Elixir Helps With

- Ease of administration and maintenance

Erlang Clustering

- Designed for long-lived server clusters
- Treat OTP apps like microservices, and you get a free resource pool
- Postpone sharding
- Postpone outboard message passing

Erlang vs The World

- Erlang doesn't look like Devops 2010, but delivers much of Devops 2016
- Go with the flow of the platform
- Startups love time-to-market advantages

Real Talk

- Don't use Elixir in prod just because it's shiny
- Evaluate advantages and disadvantages
- If Elixir and Erlang have the features to get you to market faster or better, consider it
- Also consider Erlang + Devops 2016 if that works for your use case
- Nothing's free in the long run – scale hurts

Deployment

- The sum of tools and topology
- Topology is a big topic, so here I will abbreviate to "at least two of everything"
- Now let's discuss deployment tools on the Erlang/Elixir platform

In the Beginning...

1. Write OTP applications for your first software iteration
2. Compile them
3. Build a release (1.0.0) using Reltool. It must have debug info and no `.ez` archive.
4. Make sure you create the RELEASES file at some point before starting your production application. You can do it with `release_handler:create_RELEASES(RootDir, ReleasesDir, Relfile, [{AppName, Vsn, LibDir}])`.

22. Call `release_handler:unpack_release("NameOfRel-Vsn").`
23. Call `release_handler:install_release(Vsn).`
24. Call `release_handler:make_permanent(Vsn).`
25. Make sure things went fine. If not, rollback by installing an older version.

You might want to write a few scripts to automate this.

Source: http://learnyousomeerlang.com/relups

exrm

- Elixir Release Manager
- Handles most of steps 1-25 in previous slide
- Generates Erlang releases
- Then you like, put them on servers or something? lol

edeliver

- Based on *deliver*, a *bash*-based deploy tool
- Works with *rebar*, *mix*, *relx*, *exrm*
- Handles builds, deployment, versioning
- Actively developed
- https://github.com/boldpoker/edeliver

How to *edeliver*

- Add *edeliver* as a dependency/app in *mix.exs*
- Create and configure *.deliver/config* file
- Set up servers, logins, dirs, and configs
- Use git tags to mark *mix.exs* versions
- Run mix tasks, make money

```
 18    def application do
 19      [mod: {Myapp, []},
~ 20       applications: [:phoenix, :cowboy, :logger,
+ 21         :gettext, :edeliver]]
 22    end
 23
 24    # Specifies which paths to compile per environment.
 25    defp elixirc_paths(:test), do: ["lib", "web", "test"]
 26    defp elixirc_paths(_),     do: ["lib", "web"]
 27
 28    # Specifies your project dependencies.
 29    #
 30    # Type `mix help deps` for examples and options.
 31    defp deps do
 32      [{:phoenix, "~> 1.1.4"},
 33       {:gettext, "~> 0.9"},
 34       {:edeliver, "~> 1.2.6"},
mix.exs  [line=34/37(91%)col=30Bufs#1[0][0x0]
```

```
#!/usr/bin/env bash

MIX_ENV=${MIX_ENV:-"prod"}
APP=myapp
BUILD_HOST=localhost
BUILD_USER=$USER
BUILD_AT=$HOME/myapp-builds
CONFIGS_AT=$HOME/myapp-configs
LINK_VM_ARGS=$HOME/myapp-vmargs/${MIX_ENV}.vm.args

STAGING_HOSTS=localhost
STAGING_USER=$USER
TEST_AT=$HOME/myapp-staging

PRODUCTION_HOSTS=localhost
PRODUCTION_USER=$USER
DELIVER_TO=$HOME/myapp-prod
.deliver/config [+] Line:1/37[2%]Col:1Buf:#1[35][0x23]
```

```
19 pre_erlang_get_and_update_deps() {
20     status "Copying configs from $CONFIGS_AT to $BUILD_AT"
21     __sync_remote "cp $CONFIGS_AT/* $BUILD_AT/config/"
22 }
23
24 git_checkout_remote() {
25     local _revision=$1
26     status "Checking out $_revision, the rad way"
27     __sync_remote "
28         [ -f ~/.profile ] && source ~/.profile
29         set -e
30         cd $DELIVER_TO
31         git stash
32         git checkout $_revision
33     "
34 }
35
```

edeliver Directories

- Don't create $BUILD_AT, $TEST_AT, $DELIVER_TO yourself
- *config/*.secret.exs*, etc. go in $CONFIG_AT
- **.vm.args* files handle node names, clustering
- Normal Unix permissions rules apply

How to Clustering

- *name* and *setcookie* params in *vm.args*
- *sync_nodes_mandatory/sync_nodes_optional* in *:kernel* config
- http://erlang.org/doc/design_principles/distributed_applications.html

```
## Name of the node
-name myapp@10.0.0.1

## Cookie for distributed erlang
-setcookie very_secret

## Heartbeat management; auto-restarts VM if it dies or
## (Disabled by default..use with caution!)
##-heart

## Enable kernel poll and a few async threads
##+K true
##+A 5

## Increase number of concurrent ports/sockets
##-env ERL_MAX_PORTS 4096
~
```

```
# start per endpoint:
#
#     config :myapp, Myapp.Endpoint, server: true
#
# You will also need to set the application root to `.`>
# for the new static assets to be served after a hot up>
#
#     config :myapp, Myapp.Endpoint, root: "."


config :kernel,
  sync_nodes_optional: [:"myapp@10.0.0.1",
                        :"myapp@10.0.0.2"],
  sync_nodes_timeout: 10000,
  inet_dist_listen_min: 9100,
  inet_dist_listen_max: 9105

config/prod.exs [+] Line:63/73[86%]Col:1Buf:#1[0][0x0]
```

edeliver and Versions

- *edeliver* has some new features around auto-versioning
- I do not recommend using them (yet)
- Just stick to git tags named exactly like the *mix.env* version (i.e., "0.0.1" not "v0.0.1")

Let's take *edeliver* out for a spin

Build a release

```
$ mix edeliver build release --version=0.0.1

BUILDING RELEASE OF MYAPP APP ON BUILD HOST

-----> Authorizing hosts
-----> Ensuring hosts are ready to accept git pushes
-----> Pushing new commits with git to: gamache@localhost
-----> Resetting remote hosts to 2456fcea1738e26294e913918
-----> Cleaning generated files from last build
-----> Copying configs from /Users/gamache/myapp-configs to
-----> Fetching / Updating dependencies
-----> Compiling sources
-----> Detecting exrm version
-----> Generating release
-----> Copying release 0.0.1 to local release store
-----> Copying myapp.tar.gz to release store

RELEASE BUILD OF MYAPP WAS SUCCESSFUL!
```

Deploy the first release

```
-----> Generating release
-----> Copying release 0.0.1 to local release store
-----> Copying myapp.tar.gz to release store

RELEASE BUILD OF MYAPP WAS SUCCESSFUL!

$ mix edeliver deploy release to production \
>   --version=0.0.1 --start-deploy

DEPLOYING RELEASE OF MYAPP APP TO PRODUCTION HOSTS

-----> Authorizing hosts
-----> Uploading archive of release 0.0.1 from local releas
-----> Extracting archive myapp_0.0.1.tar.gz
-----> Starting deployed release

DEPLOYED RELEASE TO PRODUCTION!
```

```
$ cd ~/myapp-prod/myapp

$ ls
bin/                lib/          relup
erl_crash.dump      log/          running-config/
erts-7.2.1/         releases/     tmp/

$ bin/myapp ping
Using /Users/gamache/myapp-prod/myapp/releases/0.0.1/myapp.s
pong

$
```

Build a hot upgrade

```elixir
defmodule Myapp.Mixfile do
  use Mix.Project

  def project do
    [app: :myapp,
     version: "0.0.2",
     elixir: "~> 1.0",
     elixirc_paths: elixirc_paths(Mix.env),
     compilers: [:phoenix, :gettext] ++ Mix.compilers,
     build_embedded: Mix.env == :prod,
     start_permanent: Mix.env == :prod,
     deps: deps]
  end

  # Configuration for the OTP application.
  #
  # Type `mix help compile.app` for more information.
```

mix.exs [+] line:16/37(43%)[col:23][buf:#1][0][0x0]

```
$ git commit -am 'added Myapp.version; bump to 0.0.2'
[master 854b1d0] added Myapp.version; bump to 0.0.2
 3 files changed, 9 insertions(+), 1 deletion(-)

$ git tag 0.0.2

$ mix edeliver build upgrade --from=0.0.1 --to=0.0.2

BUILDING UPGRADE OF MYAPP APP ON BUILD HOST
-----> Authorizing hosts
-----> Ensuring hosts are ready to accept git pushes
-----> Pushing new commits with git to: gamache@localhost
-----> Resetting remote hosts to 854b1d0f7e18db0d41807e14274
-----> Cleaning generated files from last build
-----> Checking out 0.0.1, the rad way
-----> Copying configs from /Users/gamache/myapp-configs to
-----> Fetching / Updating dependencies
```

```
-----> Copying configs from /Users/gamache/myapp-configs to
-----> Fetching / Updating dependencies
-----> Compiling sources
-----> Detecting exrm version
-----> Generating release
-----> Checking out 0.0.2, the rad way
-----> Copying configs from /Users/gamache/myapp-configs to
-----> Fetching / Updating dependencies
-----> Compiling sources
-----> Checking version of new release
-----> Generating release
-----> Removing built release 0.0.1 from remote release dire
-----> Copying release 0.0.2 to local release store
-----> Copying myapp.tar.gz to release store

UPGRADE BUILD OF MYAPP WAS SUCCESSFUL!
```

Deploy a hot upgrade

```
-----> Removing built release 0.0.1 from remote release dire
-----> Copying release 0.0.2 to local release store
-----> Copying myapp.tar.gz to release store

UPGRADE BUILD OF MYAPP WAS SUCCESSFUL!


$ mix edeliver deploy upgrade to production --version=0.0.2

DEPLOYING UPGRADE OF MYAPP APP TO PRODUCTION HOSTS

-----> Authorizing hosts
-----> Uploading archive of release 0.0.2 from local release
-----> Upgrading release to 0.0.2

DEPLOYED UPGRADE TO PRODUCTION!
```

Hot upgrades are cool

```
21
22    # Tell Phoenix to update the endpoint configuration
23    # whenever the application is updated.
24    def config_change(changed, _new, removed) do
25      Myapp.Endpoint.config_change(changed, removed)
26      :ok
27    end
28
29    def version(app \\ :myapp) do
30      Application.loaded_applications
31      |> Enum.filter(&(elem(&1, 0) == app))
32      |> List.first
33      |> elem(2)
34      |> to_string
35    end
36  end
37
```

lib/myapp.ex [+] line:37/37(100%)col:1Buf#1[0][0x0]

```
$ cd ~/myapp-prod/myapp

$ bin/myapp remote_console
Using /Users/gamache/myapp-prod/myapp/releases/0.0.1/myapp.s
Erlang/OTP 18 [erts-7.2.1] [source] [64-bit] [smp:4:4] [asyn
lse] [dtrace]

Interactive Elixir (1.2.0) - press Ctrl+C to exit (type h()
iex(myapp_prod@127.0.0.1)1> Myapp.version
"0.0.2"
iex(myapp_prod@127.0.0.1)2> # upgrade occurs
nil
iex(myapp_prod@127.0.0.1)3> Myapp.version
"0.0.3"
iex(myapp_prod@127.0.0.1)4> # Erlang is magic
```

But they don't always work

- Sometimes it's a matter of hand-editing appups and relups
- Sometimes packages are not in a position to be hot-upgraded (e.g., Cowboy)
- Sometimes f^@% you

So, What Now?

- Rolling upgrades are your best fallback
- Necessitates more than one server
- Write your code so that function signatures don't change
- Above technique is good for DB migrations too!

Rolling Upgrade

- Build release
- Deploy release without *–start-deploy*
- Execute on each server:

```
cd $DELIVER_TO/your_app_name
releases/X.Y.Z/your_app_name.sh restart
```

Rollbacks

- Rollbacks are just upgrades
- If you could upgrade version X to Y, you can rollback Y to X just as quickly

```
$ # From version 0.0.3

$ bin/myapp upgrade 0.0.2
Using /Users/gamache/myapp-prod/myapp/releases/0.0.3/myapp.s
Release 0.0.2 is marked old, switching to it.
Generating vm.args/sys.config for upgrade...
sys.config ready!
vm.args ready!
Release 0.0.2 is marked old, switching to it.
Installed Release: 0.0.2
Made release permanent: "0.0.2"

$
```

Database Migrations

- *edeliver* now has a migration feature
- http://blog.plataformatec.com.br/2016/04/running-migration-in-an-exrm-release/
- I usually just log into the build server and:

```
cd $BUILD_AT/prod
MIX_ENV=prod mix ecto.migrate
```

Gotcha!

Gotcha! Ulimit

- a.k.a. EFILE, Too many open files
- Erlang rules at using file descriptors
- It rules so hard that Unix thinks something went wrong and must be contained
- Be sure and set ulimit very high/infinite
- Set *-env ERL_MAX_PORTS* very high in *vm.args*

Gotcha! Exceptions

- Too many exceptions too fast, and the Erlang VM will ragequit
- Band-aid Solution 1: *-heart* in *vm.args*
- Band-aid Solution 2: *bin/myapp start* in cron
- Real Solution: fix your shit

Gotcha! Segfaults

- The Erlang VM is not supposed to segfault
- How adorable
- Same instructions as previous slide

Gotcha! Cached Release

- Example: you build an upgrade from X to Z
- You then build an upgrade from Y to Z
- If you had copied X-to-Z to your servers, you can't perform the upgrade Y-to-Z
- Solution: Blow away *$DELIVER_TO/myapp/releases/Z* before building/deploying Y-to-Z

Gotcha! Websockets

- Unexplained client 400s can be due to load balancer issues
- Amazon ELB: use "TCP"/"SSL" settings, not "HTTP"/"HTTPS"
- Nginx:

```
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "upgrade";
```

Gotcha! Git Problems

```
-----> Copying configs from /Users/gamache/myapp-configs
 to /Users/gamache/myapp-builds/config/
-----> Fetching / Updating dependencies
-----> Compiling sources
-----> Detecting exrm version
-----> Generating release
-----> Checking out 0.0.2
error: Your local changes to the following files would b
e overwritten by checkout:
        mix.lock
Please, commit your changes or stash them before you can
 switch branches.
Aborting

FAILED 1:

    [ -f ~/.profile ] && source ~/.profile
    set -e
    cd /Users/gamache/myapp-builds
    git checkout 0.0.2
```

Finding a Git Solution

```
816
817  # checks the given commit out on the remote build host
818  git_checkout_remote() {
819    local _revision=$1
820    status "Checking out $_revision"
821    __sync_remote "
822      [ -f ~/.profile ] && source ~/.profile
823      set -e
824      cd $DELIVER_TO
825      git checkout $_revision
826    "
827  }
828
```

in *edeliver/libexec/erlang*

```
19 pre_erlang_get_and_update_deps() {
20     status "Copying configs from $CONFIGS_AT to $BUILD_AT"
21     __sync_remote "cp $CONFIGS_AT/* $BUILD_AT/config/"
22 }
23
24 git_checkout_remote() {
25     local _revision=$1
26     status "Checking out $_revision, the rad way"
27     __sync_remote "
28         [ -f ~/.profile ] && source ~/.profile
29         set -e
30         cd $DELIVER_TO
31         git stash
32         git checkout $_revision
33     "
34 }
35
```

```
-----> Copying configs from /Users/gamache/myapp-configs to
-----> Fetching / Updating dependencies
-----> Compiling sources
-----> Detecting exrm version
-----> Generating release
-----> Checking out 0.0.2, the rad way
-----> Copying configs from /Users/gamache/myapp-configs to
-----> Fetching / Updating dependencies
-----> Compiling sources
-----> Checking version of new release
-----> Generating release
-----> Removing built release 0.0.1 from remote release dire
-----> Copying release 0.0.2 to local release store
-----> Copying myapp.tar.gz to release store

UPGRADE BUILD OF MYAPP WAS SUCCESSFUL!
```

Another Summary

- Erlang and Elixir deployment shouldn't look like less-powerful platforms'
- *edeliver* automates most of the deployment process, mostly. Let's say 90% of the time
- Taken next to the platform's advantages, the gotchas aren't so bad
- 2016 is early in the Elixir Deployment timeline

Questions?

Thanks!

https://twitter.com/gamache

https://engineering.appcues.com

http://www.slideshare.net/petegamache/real-world-elixir-deployment