Releases: Part 4

Distillery

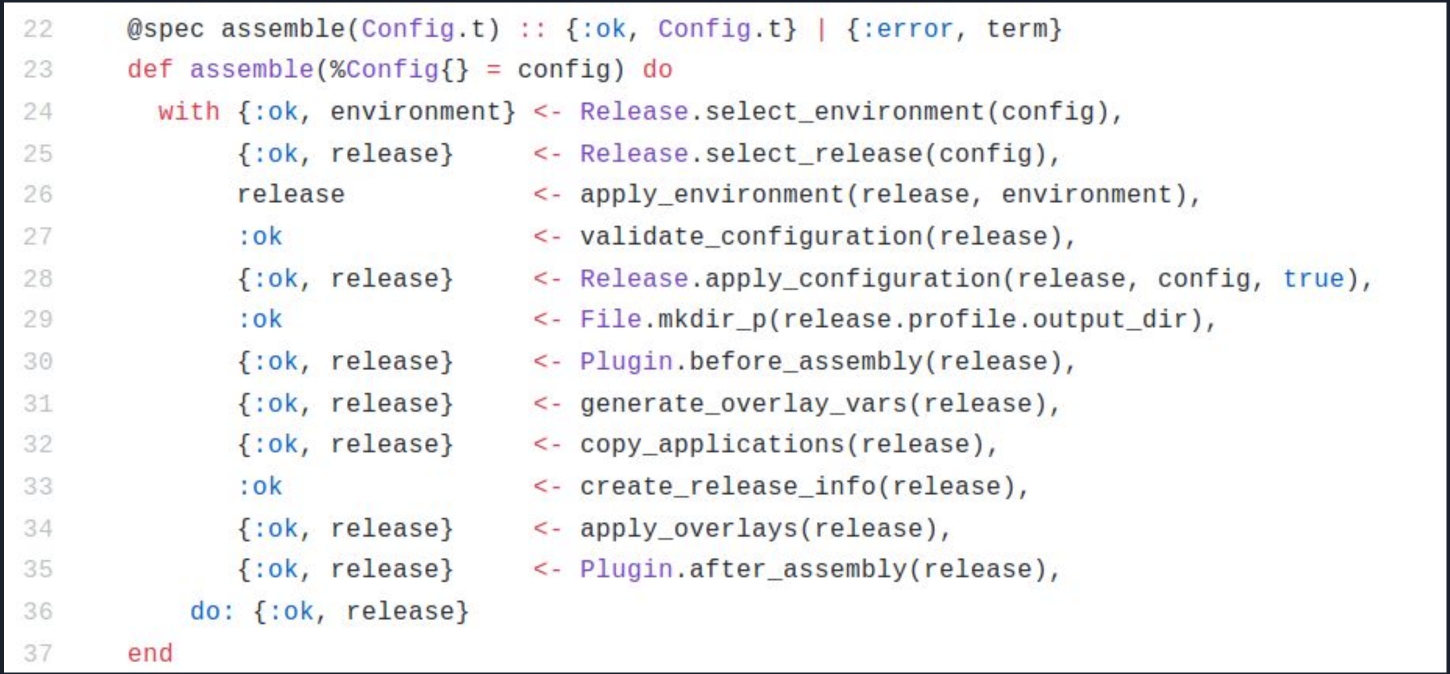
Next:

1. The Basics
2. The Application
3. The Environment

4. Distillery [you are here]
  A. Explore how Distillery builds a release
  B. Walk through the shell scripts
  C. Boot Hooks / custom commands
  D. Plugins

Distillery: assembler

lib/mix/lib/releases/assembler.ex



```elixir
  @spec assemble(Config.t) :: {:ok, Config.t} | {:error, term}
  def assemble(%Config{} = config) do
    with {:ok, environment} <- Release.select_environment(config),
         {:ok, release}     <- Release.select_release(config),
         release            <- apply_environment(release, environment),
         :ok                <- validate_configuration(release),
         {:ok, release}     <- Release.apply_configuration(release, config, true),
         :ok                <- File.mkdir_p(release.profile.output_dir),
         {:ok, release}     <- Plugin.before_assembly(release),
         {:ok, release}     <- generate_overlay_vars(release),
         {:ok, release}     <- copy_applications(release),
         :ok                <- create_release_info(release),
         {:ok, release}     <- apply_overlays(release),
         {:ok, release}     <- Plugin.after_assembly(release),
      do: {:ok, release}
  end
```

The Shell scripts
release_dir/lib-exec

- Can be found in the `priv/` in the distillery source
- Are Responsible for building up the start/stop/commands
- Templated, modular, and easy to poke around in
- https://hexdocs.pm/distillery/shell-script-api.html#content

Overlays

- Create files from templates
- Move files around
- Create new directories
- Any other basic file related task

Commands

- Migrations
- Maintenance tasks
- Anything you want to run on the server after a deploy

Plugins

Release build lifecycle hooks

- before_assembly
- after_assembly
- before_package
- after_package
- after_cleanup

Summary

- Distillery Does the hard work for us
- Erlang provides the supporting conventions
- It’s not magic - you can do it!