Dune

A modern build system for OCaml/Reason

How it started

- Two years ago: patchwork of tooling
- Simplify the Jane Street OSS
- Up to 50x faster

Now

- Standard tool for writing OCaml applications
- Developped on github
- Community project
- MIT license

Tow main drivers

- Fast builds
- Simple and coherent user experience

What is Dune?

Dune is

- A modern and fast build system for OCaml/Reason
- It can build/cross-compile native applications
- It can build Javascript applications
- And much more

Dune configuration

Via `dune` files:

```
(executable
 (name hello_world)
 (libraries mylib)
 (preprocess (pps ppx_inline_test)))
```

```
(library
 (name mylib)
 (libraries re lwt))
```

- descriptive
- close to what they describe

Simple yet powerful

It is easy to customize the system:

```
(executable
 (name generator))

(rule
 (with-stdout-to file.ml (run ./generator.exe)))
```

Usage

Simple CLI:

```
$ dune exec ./hello_world.exe
Hello, world!
```

Packed with dev tools:

```
$ dune utop src
# Mylib.x;;
- : int = 42
```

All the common stuff

- system instalation: `dune build @install`
- documentation: `dune build @doc`
- testing: `dune build @runtest`

Composability

- Put two projects together and you get something that dune understands:

```
$ git clone github.com/me/foo
$ git clone github.com/me/bar
$ dune build # Build both foo and bar at once
```

- Faster builds
- Trivial vendoring
- Trivial large scale refactoring

Testing with Dune

Expectation testing

```
(test (name hello))
```

```
$ cat hello.expected
blah

$ dune runtest
--- hello.expected
+++ hello.output
@@ -1 +1 @@
-blah
+Hello, world!

$ dune promote
Promoting _build/default/hello.output to hello.expected.

$ dune runtest
```

Inline expectation testing

```
let%expect_test _ =
  print_string "Hello, world!";
  [%expect ""]
```

```
$ dune runtest
--- test.ml
+++ test.ml.corrected
@@ -1,3 +1,3 @@
 let%expect_test _ =
   print_string "Hello, world!";
-  [%expect ""]
+  [%expect "Hello, world!"]
```

Integration tests

```
(alias
 (name runtest)
 (deps (package my-package))
 (action (run my-prog ...)))
```

- Very useful for regression tests

Targetting Javascript

Javascript

- Via js_of_ocaml
- dev mode:
  - fast incremental compilation
  - big `.js` files
- release mode:
  - slow incremental compilation
  - small `.js` files

Compiling to Javascript

Simply request `.bc.js`, Dune knows how to build it:

```
$ dune build myapp.bc.js
$ ls _build/default/*.js
myapp.bc.js
```

Cross-compilation

Cross-compilation

```
$ dune build -x windows,android,ios main.exe
```

- windows binary at `_build/default.windows/main.exe`
- android binary at `_build/default.android/main.exe`
- ios binary at `_build/default.ios/main.exe`
- handles staging without issues

Using Dune with esy

Esy integration

- add `"@opam/dune"` to your `package.json` file
- drop a couple of `dune` files in your project
- `esy dune build`

Design choices

Proper build system

- Not a frontend or a backend
- Gives us a lot of flexibility
- Faster builds

Backward compatibility

- Upgrading should be a no-brainer
- The user states the version of Dune it expects: `(lang dune 1.4)` in `dune-project` file
    - allows breaking changes without breaking existing projects
    - no superfluous deprecation messages
    - helpful error messages

No choices if not needed

- Does not give choices when not relevant
- Prefer to correctly implement one way of doing things

Future of Dune

Future of Dune

- Automate a few more things (less configuration files)
- Scaling incremental builds
- Integrate new ideas/workflows

The end

Questions