builder image

- builder可能用rust toolchain，然后下载wasm32-xx-xx，先做runtime的镜像；

  - Rust `wasm32-unknown-unknown` target - Rust compiler backend for WebAssembly (without the need for Emscripten).
  - Wargo - a simple npm package that makes compiling Rust to WebAssembly easy on macOS or Linux.
  - RustWasmLoader - A simple Webpack loader that shells out to cargo to build a Rust project targeting WebAssembly.
  - CargoWeb - This cargo subcommand aims to make it easy and convenient to build, develop and deploy client-side Web applications written in Rust.
  - Wasm-Bindgen - A library and a CLI for Rust that facilitate high-level interactions between wasm modules and JavaScript.
  - Woz - Woz is a WebAssembly progressive web app (PWA) toolchain for building and deploying performant mobile apps with Rust. Distributing your app is as simple as sharing a hyperlink.
-

fisison + wasm

builder

- builder: rust toolchain
  - rustup toolchain install nightly
  - rustup target add wasm32-unknown-unknown --toolchain nightly
- build command
  - rustc +nightly --target wasm32-unknown-unknown -O **hello.rs**

docker file 创建builder镜像的思路

- fission/builder作为base image
- 添加编译器image
- 设置编译环境
- 添加build脚本

```
ENV LANG C.UTF-8
RUN { \
    echo '#!/bin/sh'; \
    echo 'set -e'; \
    echo; \
    echo 'dirname "$(dirname "$(readlink -f "$(which javac || which java)")")"'; \
    } > /usr/local/bin/docker-java-home \
    && chmod +x /usr/local/bin/docker-java-home
ENV JAVA_HOME /usr/lib/jvm/java-1.8-openjdk
ENV PATH $PATH:/usr/lib/jvm/java-1.8-openjdk/jre/bin:/usr/lib/jvm/java-1.8-openjdk/bin
```

```
## Fission builder specific section
ADD build.sh /usr/local/bin/build
EXPOSE 8001
```

```
1  #!/bin/sh
2  set -eou pipefail
3  mvn clean package
4  cp ${SRC_PKG}/target/*with-dependencies.jar ${DEPLOY_PKG}
```

fisison + wasm

env

- wasmedge
  - fission env create --name wasm --image **wasmedge/wasmedge**
- create package with deploy-archive
  - fission package create --name hello-wasm --env wasm --**deployarchive** hello.wasm
- create function
  - fission function create --name wasm-func --env wasm --pkg hello-wasm --entrypoint main
- run function
  - fission function test --name wasm-func

docker file 创建env镜像的思路

- 提供一个os镜像 as Base alpine
- 开辟新的VOLUME
- 添加env镜像
- (optional) 容器启动时的命令ENTRYPOINT
  - 启动server / ...

docker file 创建env镜像的思路

- (optional) 容器启动后的命令CMD
  - 启动server（java）
    - 接收请求(/v2/specilize)
    - 报告函数实例是否创建成功；
    - 容器特殊化 -- 创建函数实例；
    -
```
// Load all dependent classes from libraries etc.
while (e.hasMoreElements()) {
    JarEntry je = e.nextElement();
    if (je.isDirectory() || !je.getName().endsWith(".class")) {
        continue;
    }
    String className = je.getName().substring(0, je.getName().length() - CLASS_LENGTH);
    className = className.replace('/', '.');
    cl.loadClass(className);
}

// Instantiate the function class
fn = (Function) cl.loadClass(entryPoint).newInstance();
```