



RUSTLAB

Dockerising and deploying a full stack Rust + WASM web app

Angelo Rendina



- **Toolset**
- **Local development**
- **Deployment**



Toolset



The app

New Memo

Submit

Memos

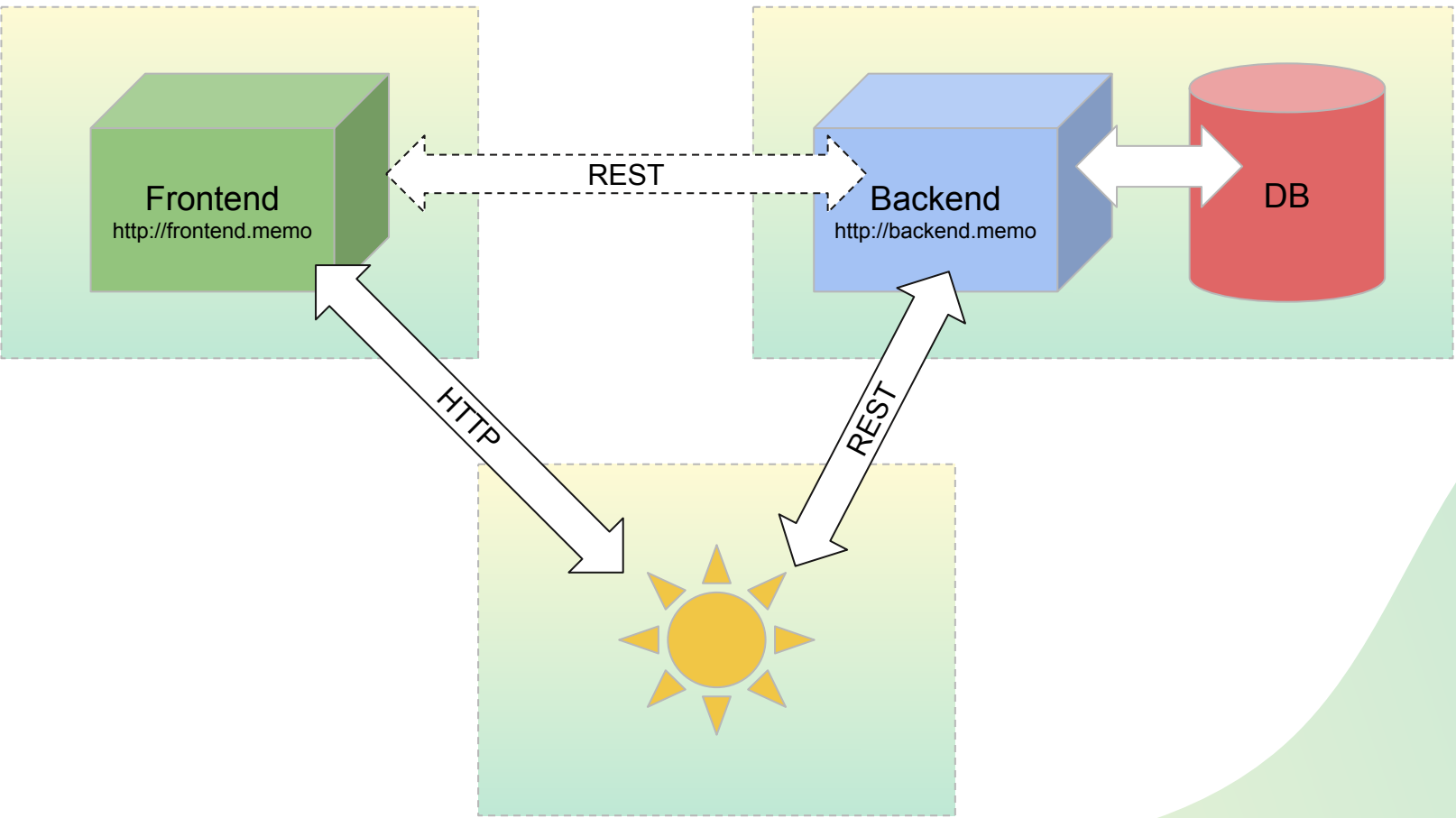
X

☒ Something to do

X

☐ Other task

Architecture



Rust monorepo

- Cargo-make
 - Task runner and build tool
 - All flows in Makefile.toml
- SQLx
 - Rust SQL toolkit
 - Command line utility to manage the database with sqlx-cli
- Actix
 - Rust web framework
 - HTTP server in a native executable
- Yew
 - Rust to Webassembly multi-threaded front-end framework
 - Single Page Application

Infrastructure

- Docker
 - Postgres official image for the database
 - Custom image from Rust for the backend
 - Custom image from Rust (wasm32-unknown-unknown) for the frontend
 - Dev stack with compose
 - Build optimised release images
- Kubernetes
 - Orchestration of the microservices
 - Management of deployments and internal routing
 - Load balancing and external routing
 - Simulate prod environment locally with minikube

Local development



Database



Using sqlx-cli

In Makefile.toml:

Generate the migration for db schema:

- `cargo make --profile local db-migration memo`

global [env] overrides already set vars!

`"X=123 cargo make some-task-with-x"`

Default profile is "development" so beware of [env.development] too

```
[env.local]
DATABASE_URL="postgresql://user:password@localhost:5432/db"
```

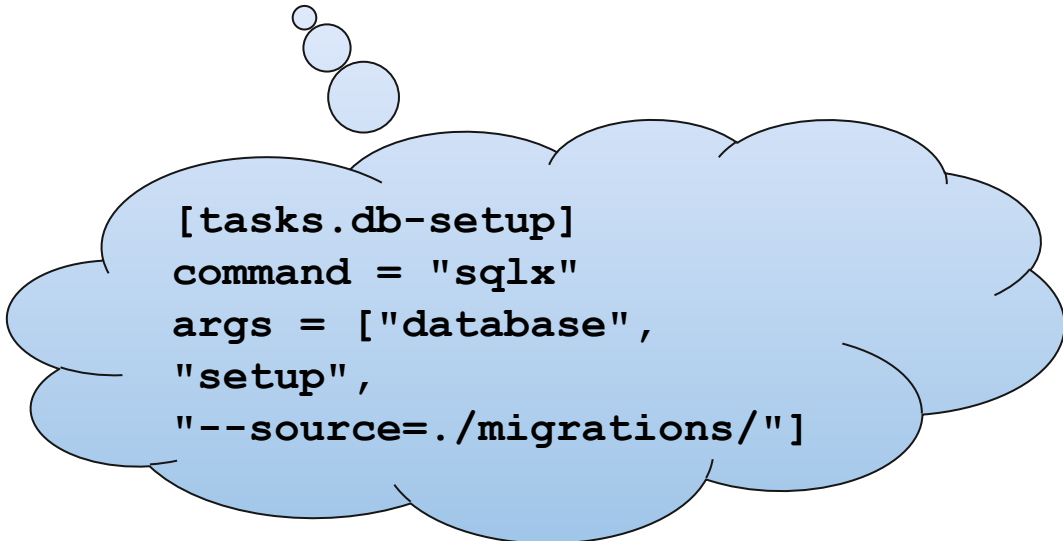
```
[tasks.db-migration]
command = "cargo"
args = ["sqlx", "migrate", "add",
"${@}"]
```

Managing migrations

In migrations/<timestamp>_memo.sql:

Run migrations:

- `cargo make --profile local db-setup`



```
[tasks.db-setup]
command = "sqlx"
args = ["database",
"setup",
"--source=./migrations/"]
```

```
CREATE TABLE memos (
    "id" UUID PRIMARY KEY NOT NULL,
    "timestamp" TIMESTAMP WITH TIME
ZONE NOT NULL,
    "done" BOOLEAN NOT NULL,
    "text" TEXT NOT NULL
);
```

Monorepo



Cargo workspace

In Cargo.toml:

Generate crates:

- `cargo new --lib common`
- `cargo new backend`
- `cargo new frontend`

Compile, build and run each package through cargo!

```
[workspace]
members = [
    "backend",
    "common",
    "frontend",
]
```

Shared types

In common/src/lib.rs:

Define DTOs to communicate between backend and frontend.

```
#[derive(Deserialize, Serialize)]
pub struct Memo {
    pub id: Uuid,
    pub timestamp: DateTime<Utc>,
    pub done: bool,
    pub text: String,
}

#[derive(Deserialize, Serialize)]
pub struct UpdateMemo {
    pub id: Uuid,
    pub done: bool,
}
```

Backend



Communicating with the database with sqlx

```

async fn index(
    executor: impl PgExecutor<'_>
) -> Result<Vec<common::Memo>, Error>{
    sqlx::query_as!(
        common::Memo,
        "SELECT id, timestamp, done, text FROM memos ORDER BY timestamp",
    )
    .fetch_all(executor)
    .await
}

```

compile-time cross-reference against
running db

(or use sqlx offline mode)

An endpoint

```
async fn get_memos(  
    app_state: Data  
) -> HttpResponse {  
    match index(&app_state.pool).await {  
        Ok(memos) => HttpResponse::Ok().json(memos),  
        Err(_) => HttpResponse::InternalServerError().finish(),  
    }  
}
```

Actix HttpServer

```
#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new()
            .app_data(app_state.clone())
            .route("/", get().to(get_memos))
            ...
    })
    .bind(("0.0.0.0", 3000))?
    .run()
    .await
}
```

shared data
(behind Arc)

Cargo make to run the backend

In Makefile.toml:

envvars for running
the stack locally

start the backend

```
[env.docker]
DATABASE_URL="postgresql://user:password@postgres:5432/db"

[tasks.backend-run]
command = "cargo"
args = ["run", "-p", "backend"]
```

URL of the database inside
docker network

Backend in Docker

In docker-compose.yml:

source code

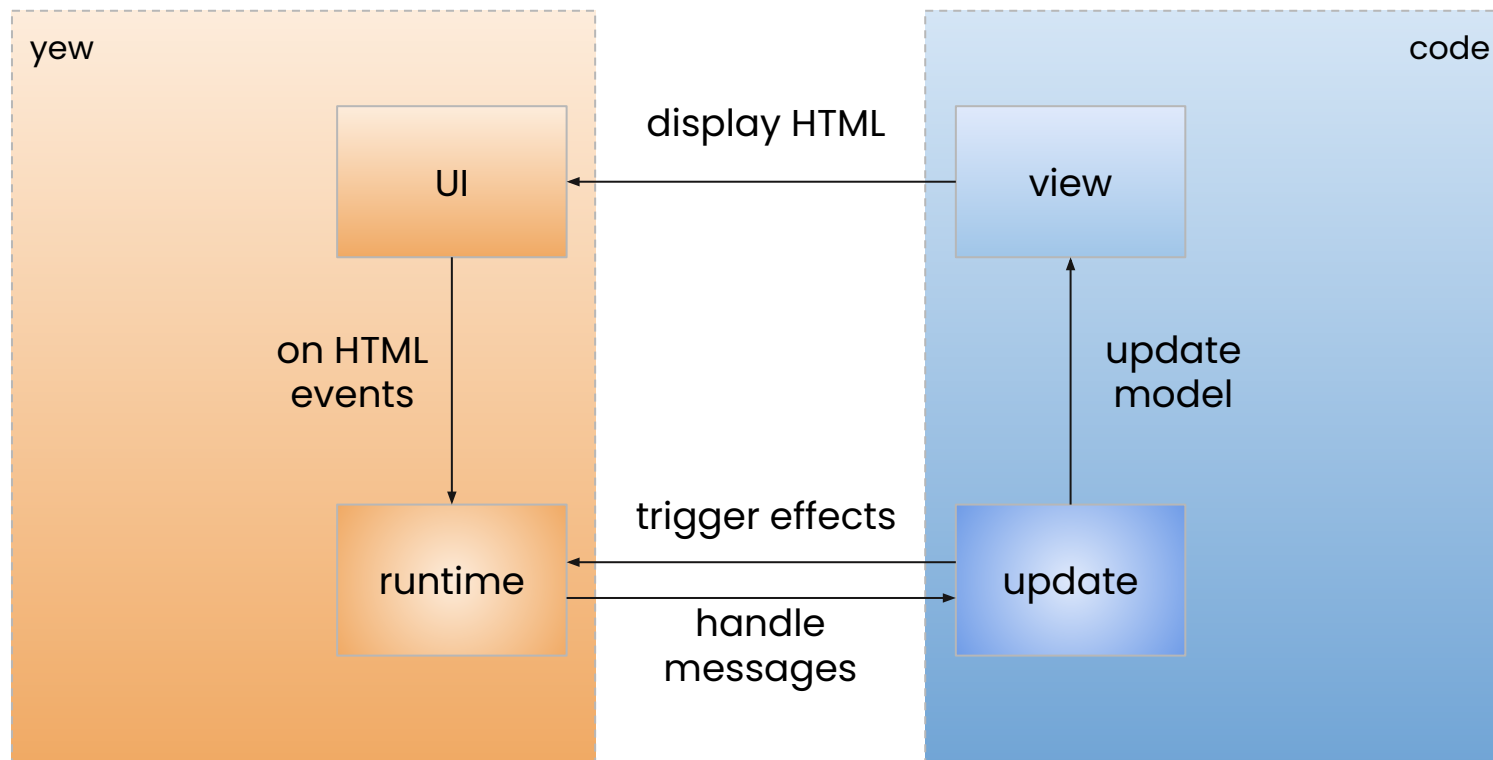
caching artifacts

```
backend:
  image: rust
  working_dir: /app
  volumes:
    - ".:/app"
    - "cargo:/app/.cargo"
    - "target:/app/target"
  environment:
    CARGO_HOME: /app/.cargo
    CARGO_TARGET_DIR: /app/target
  command: cargo make --profile docker backend-run
```

Frontend



Model-View-Update



Root component

```

struct App { memos: Vec<common::Memo> }

impl Component for App {
    fn view(&self, ctx: &Context) -> Html {
        self.memos.map(|memo| html! {
            <Viewer
                value={memo.text}
                on_delete={ctx.callback(Msg::DeleteMemo(id))} />
        })
    }

    fn update(&mut self, ctx: &Context, msg: Self::Message) {
        match msg {
            Msg::DeleteMemo(id) => { delete_memo(ctx, common::DeleteMemo { id }); }
        }
    }
}

```

Memos

☐ ☒ Something to do

☐ ☐ Other task

Making HTTP calls in yew

Code is compiled into WASM, and the envvar hardcoded!

```
const BACKEND_URL: &'static str = std::env!("BACKEND_URL");

fn delete_memo(ctx: &Context, payload: common::DeleteMemo) {
    wasm_bindgen_futures::spawn_local(async {
        let response = Request::delete(BACKEND_URL)
            .body(payload.as_string())
            .send()
            .await;

        if let Ok(_) = response {
            ctx.send_message(Msg::OnMemoDeleted(payload.id));
        }
    })
}
```


Cargo make to run the frontend

In Makefile.toml:

envvars for running the
stack locally

serve the frontend

```
[env.docker]
BACKEND_URL="http://localhost:3000"

[tasks.frontend-run]
command = "trunk"
args = ["serve", "--address", "0.0.0.0",
"./frontend/index.html"]
```

Frontend in Docker

In docker-compose.yml:

source code

caching artifacts

```
frontend:
  image: rust
  working_dir: /app
  volumes:
    - ".:/app"
    - "cargo:/app/.cargo"
    - "target:/app/target"
    - "rustup:/app/.rustup"
  environment:
    CARGO_HOME: /app/.cargo
    CARGO_TARGET_DIR: /app/target
    RUSTUP_HOME: /app/.rustup
  command: cargo make --profile docker frontend-run
```

Deployment



Backend image



Sqlx offline mode

In Makefile.toml:

Sqlx offline mode so code is compiled without live access to db.

Prepare and store locally a schema for offline compilation:

- `cargo make --profile local db-prepare`

```
[env.local]
DATABASE_URL="postgresql://user:password@localhost:5432/db"
```

```
[tasks.db-prepare]
command = "cargo"
args = ["sqlx", "prepare", "--", "-p", "backend"]
```

Cargo make build task

In Makefile.toml:

Propagate external envvars into cargo make tasks.

Enable sqlx offline mode (using prepared schema.json)

```
[env.release]
SQLX_OFFLINE=true
DATABASE_URL="${DATABASE_URL}"

[tasks.backend-build]
command = "cargo"
args = ["build", "-p", "backend",
"--release"]
```

Release image

In backend/Dockerfile:
cargo-make installed only once

accept envvars as build arguments

compile source code

small unix image with minimal runtime
install necessary dependencies

copy built binary into minimal runtime

run the backend!

```
docker build -f backend/Dockerfile
--build-arg DATABASE_URL=<?>
```

```
FROM rust AS base
RUN cargo install cargo-make
```

```
FROM base as builder
ARG DATABASE_URL
ENV DATABASE_URL ${DATABASE_URL}
COPY . /memo
WORKDIR /memo
RUN cargo make --profile release
backend-build
```

```
FROM debian:buster-slim as runner
RUN apt update
RUN apt install -y libssl1.1
```

```
FROM runner
COPY --from=builder
/memo/target/release/backend
/memo/backend
CMD ["/memo/backend"]
```

Frontend image



Cargo make build task

In Makefile.toml:

Propagate external envvars into cargo make tasks.

Build static HTML and JS bundle.

```
[env.release]
BACKEND_URL="${BACKEND_URL}"

[tasks.frontend-build]
command = "trunk"
args = ["build", "--release",
"./frontend/index.html"]
```

Release image

In frontend/Dockerfile:
cargo-make and compile dependencies
installed once

accept envvars as build arguments

compile source code

copy built bundle into minimal runtime

run the frontend!

```
docker build -f frontend/Dockerfile
--build-arg BACKEND_URL=<?>
```

```
FROM rust AS base
RUN cargo install cargo-make
RUN rustup toolchain install stable
RUN rustup target add
wasm32-unknown-unknown
RUN cargo install trunk
RUN cargo install wasm-bindgen-cli
```

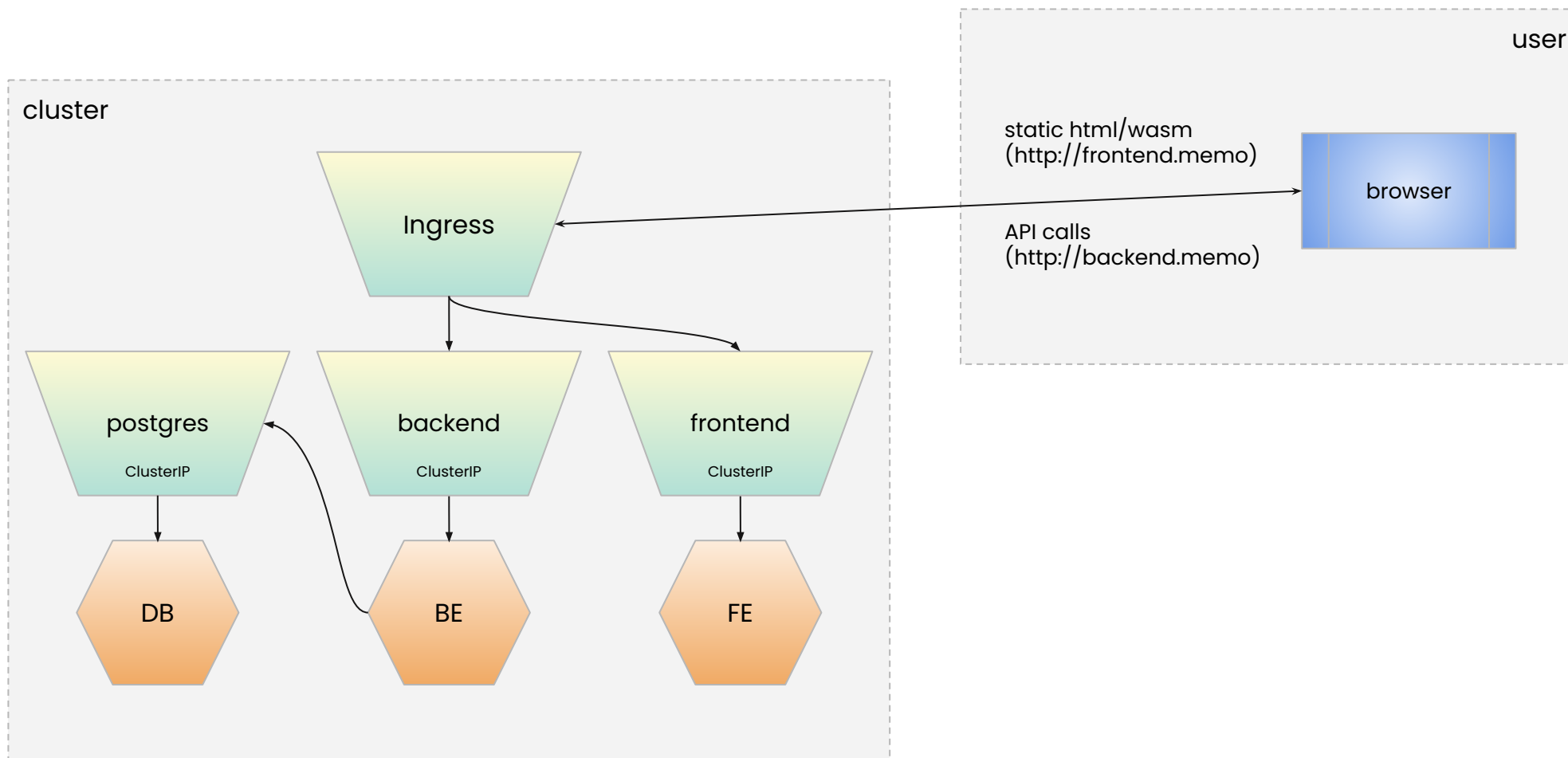
```
FROM base AS builder
ARG BACKEND_URL
ENV BACKEND_URL ${BACKEND_URL}
COPY . /memo
WORKDIR /memo
RUN cargo make --profile release
frontend-build
```

```
FROM httpd:alpine
COPY --from=builder /memo/dist/.
/usr/local/apache2/htdocs/.
```

Kubernetes



App infrastructure



K8s deployment

k8s deployment

applies to pods with "app: backend" label

generates pods with "app: backend" label

use "memo-backend" docker image

```
kind: Deployment
spec:
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - image: memo-backend
```

K8s service

k8s service
by default ClusterIP

visible within the cluster as “backend-service”

forwards traffic to any pod with “app: backend” label

```
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
```

K8s ingress

k8s ingress

if request has "backend.memo" host...

...forward to "backend-service" ClusterIP

if request has "frontend.memo" host...

...forward to "frontend-service" ClusterIP

```
kind: Ingress
spec:
  rules:
  - host: backend.memo
    http:
      paths:
      - path: /
        backend:
          service:
            name: backend-service
  - host: frontend.memo
    http:
      paths:
      - path: /
        backend:
          service:
            name: frontend-service
```

Minikube

backend and DB are in the same cluster
reference by service name

Build release images:

- `docker build -f ./backend/Dockerfile -t memo-backend --build-arg DATABASE_URL=postgresql://user:password@postgres-service:5432/db .`
- `docker build -f ./frontend/Dockerfile -t memo-frontend --build-arg BACKEND_URL=http://backend.memo .`

For each k8s item, write configuration in a YAML file and setup cluster:

- `kubectl apply -f <yml>`

Expose the cluster ingress on localhost:

- `minikube tunnel`

In `/etc/hosts`, map "backend.memo" and "frontend.memo" to "127.0.0.1".



RUSTLAB

Angelo Rendina

angelo.rendina91@gmail.com

