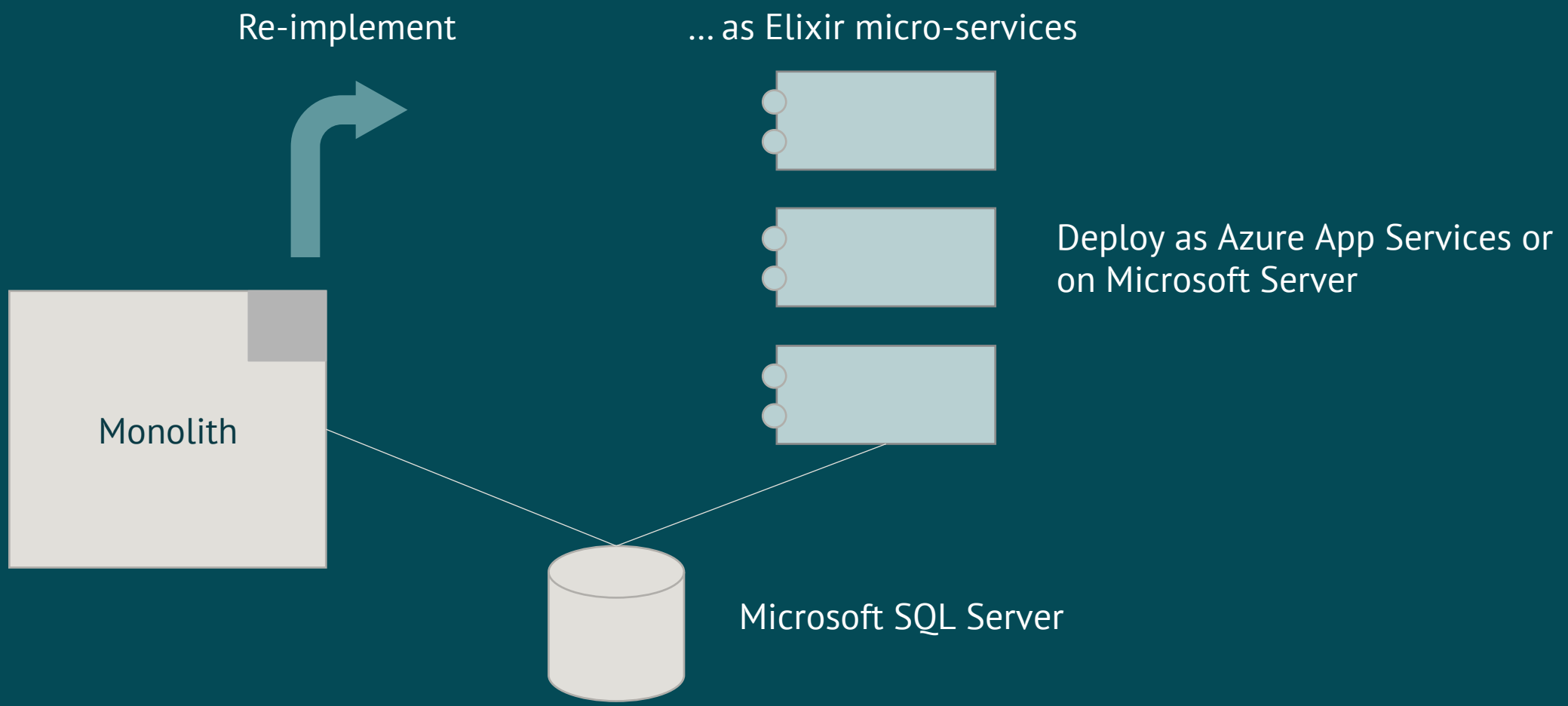
BUILDING, TESTING AND DEPLOYING ELIXIR APPLICATIONS WITH DOCKER IN AZURE

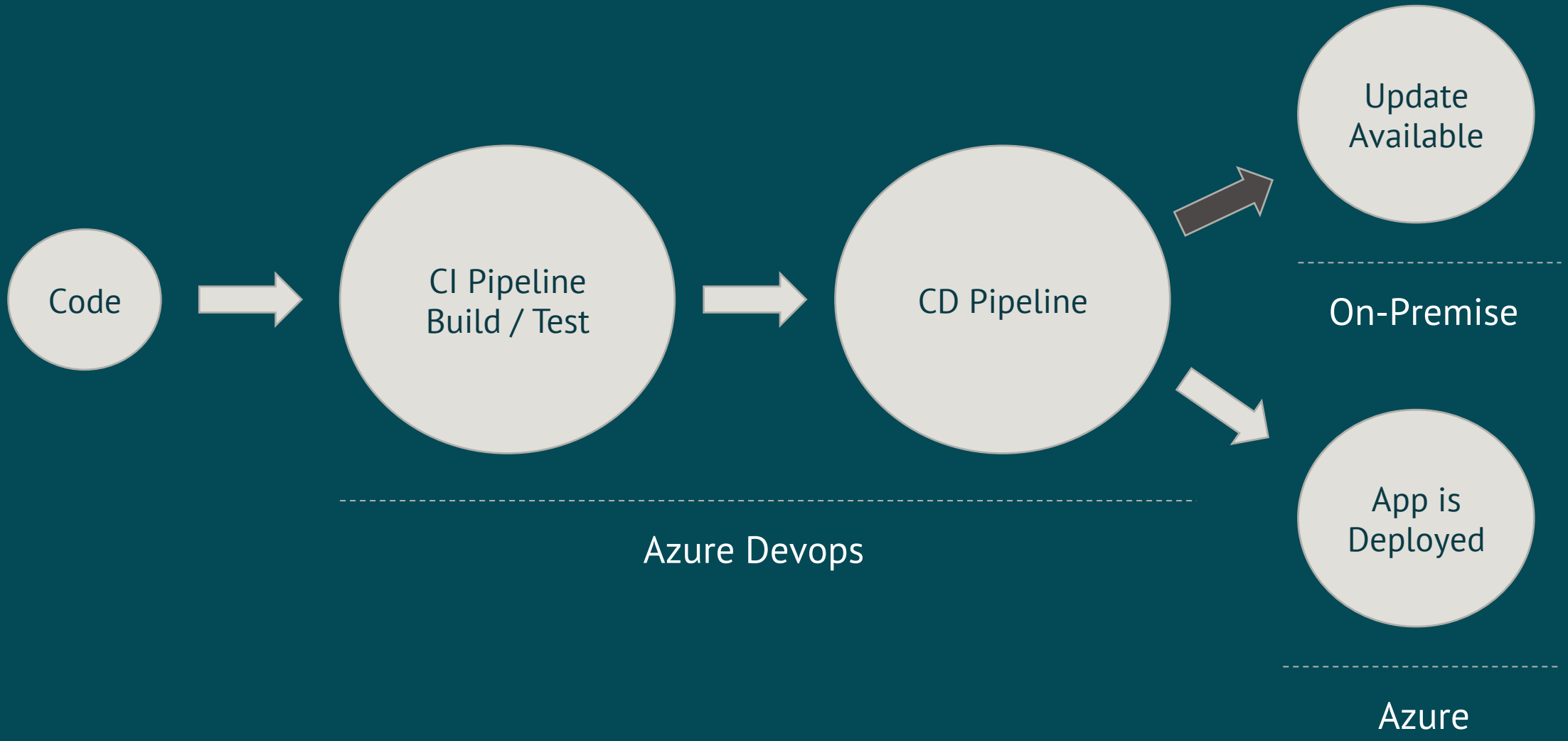
Erik Svensson - @svan-jansson on GitHub
Ibrahim Abdelkareem - @IbrahimCIS on Twitter

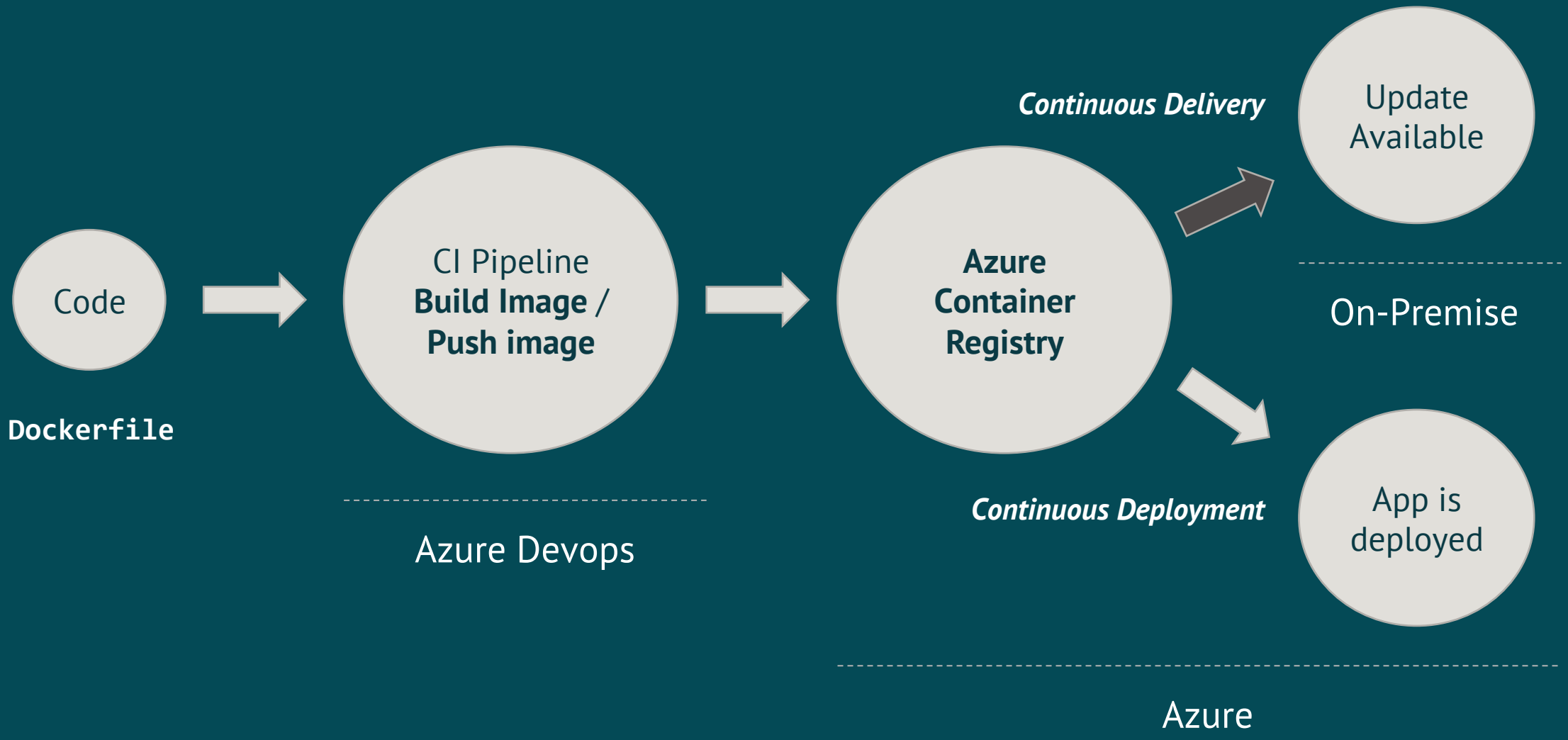
#CodeBEAMSTO
Stockholm - 16-17 May 2019

CAN YOU USE ELIXIR AT A "MICROSOFT SHOP"?

#CodeBEAMSTO
Stockholm - 16-17 May 2019


Re-implement
... as Elixir micro-services
Monolith
Deploy as Azure App Services or on Microsoft Server
Microsoft SQL Server


Code
CI Pipeline
Build / Test
CD Pipeline
Update
Available
On-Premise
App is
Deployed
Azure Devops
Azure


Code
Dockerfile
CI Pipeline
Build Image /
Push image
Azure Devops
Azure
Container
Registry
Continuous Delivery
Update
Available
On-Premise
Continuous Deployment
App is
deployed
Azure

```
# Build stage
FROM elixir:1.8.1-alpine AS build
ARG APP_NAME=our_elixir_app
ENV MIX_ENV=prod REPLACE_OS_VARS=true TERM=xterm
WORKDIR /opt/app
RUN apk update && \
mix local.rebar --force && \
mix local.hex --force

# PLT caching
COPY mix.exs mix.lock ./
COPY apps/$APP_NAME/mix.exs apps/$APP_NAME/mix.exs
COPY apps/${APP_NAME}_api/mix.exs apps/${APP_NAME}_api/mix.exs

RUN mix deps.get && \
MIX_ENV=dev mix deps.compile && \
MIX_ENV=test mix deps.compile && \
mix deps.compile && \
MIX_ENV=dev mix dialyzer --plt --halt-exit-status

COPY . .

# Check format of source code, compilation warnings, static code analysis,
run tests and release
RUN mix format --check-formatted && \
mix compile --force --warnings-as-errors && \
MIX_ENV=dev mix dialyzer --halt-exit-status && \
MIX_ENV=test mix test --exclude integration && \
mix release --env=prod --verbose && \
mkdir -p /opt/release && \
tar -xzf _build/prod/rel/${APP_NAME}/releases/*/${APP_NAME}.tar.gz -C /
opt/release && \
mv /opt/release/bin/${APP_NAME} /opt/release/bin/start_server

# Release stage
FROM alpine:3.9
RUN apk update && apk --no-cache --update add bash openssl
ENV PORT=8080 MIX_ENV=prod REPLACE_OS_VARS=true
WORKDIR /opt/app
EXPOSE ${PORT}
COPY --from=build /opt/release .
CMD ["/opt/app/bin/start_server", "foreground"]
```

What we wanted to achieve

- Build and deploy (Anywhere!)
- Spend less build time.
- Verify code changes should be integrated.
- Run a light-weight docker container in production.

```
# Build stage
FROM elixir:1.8.1-alpine AS build
ARG APP_NAME=our_elixir_app
ENV MIX_ENV=prod REPLACE_OS_VARS=true TERM=xterm
WORKDIR /opt/app
RUN apk update && \
mix local.rebar --force && \
mix local.hex --force

# PLT caching
COPY mix.exs mix.lock ./
COPY apps/$APP_NAME/mix.exs apps/$APP_NAME/mix.exs
COPY apps/${APP_NAME}_api/mix.exs apps/${APP_NAME}_api/mix.exs

RUN mix deps.get && \
MIX_ENV=dev mix deps.compile && \
MIX_ENV=test mix deps.compile && \
mix deps.compile && \
MIX_ENV=dev mix dialyzer --plt --halt-exit-status

COPY . .

# Check format of source code, compilation warnings, static
code analysis, run tests and release
RUN mix format --check-formatted && \
mix compile --force --warnings-as-errors && \
MIX_ENV=dev mix dialyzer --halt-exit-status && \
MIX_ENV=test mix test --exclude integration && \
mix release --env=prod --verbose && \
mkdir -p /opt/release && \
tar -xzf _build/prod/rel/${APP_NAME}/releases/*/$
{APP_NAME}.tar.gz -C /opt/release && \
mv /opt/release/bin/${APP_NAME} /opt/release/bin/start_server
# Rest Goes Here …
```

Setting up build stage

- We use elixir-alpine image.
- Setting up variables and build arguments.
- Install build tools & dependencies.

```
# Build stage
FROM elixir:1.8.1-alpine AS build
ARG APP_NAME=our_elixir_app
ENV MIX_ENV=prod REPLACE_OS_VARS=true TERM=xterm
WORKDIR /opt/app
RUN apk update && \
mix local.rebar --force && \
mix local.hex --force

# PLT caching
COPY mix.exs mix.lock ./
COPY apps/$APP_NAME/mix.exs apps/$APP_NAME/mix.exs
COPY apps/${APP_NAME}_api/mix.exs apps/${APP_NAME}_api/mix.exs

RUN mix deps.get && \
MIX_ENV=dev mix deps.compile && \
MIX_ENV=test mix deps.compile && \
mix deps.compile && \
MIX_ENV=dev mix dialyzer --plt --halt-exit-status

COPY . .

# Check format of source code, compilation warnings, static
code analysis, run tests and release
RUN mix format --check-formatted && \
mix compile --force --warnings-as-errors && \
MIX_ENV=dev mix dialyzer --halt-exit-status && \
MIX_ENV=test mix test --exclude integration && \
mix release --env=prod --verbose && \
mkdir -p /opt/release && \
tar -xzf _build/prod/rel/${APP_NAME}/releases/*/$
{APP_NAME}.tar.gz -C /opt/release && \
mv /opt/release/bin/${APP_NAME} /opt/release/bin/start_server
# Rest Goes Here …
```

We don’t want to wait for hours!

- Application dependencies needs to be fetched and built.
- Static code analysis needs to build persistent lookup tables (A.K.A PLT).
- The output is unlikely to change for a sequence of builds.
- Thanks to docker caching.

A little bit about docker layers.

Building a docker file

```
FROM elixir:1.8.1-alpine AS build
ARG APP_NAME=my_app
..
COPY mix.exs mix.lock ./
..
RUN mix deps.get
..
COPY . .
..
RUN mix compile --force --warnings-as-errors
..
CMD ["/opt/app/bin/start_server", "foreground"]
```

IMAGE	CREATED	CREATED BY
7184cc184ef8	2 days ago	image layer: CMD[“opt/app/bin/start_server..”]
fdd93d9c2c60	2 days ago	image layer: RUN mix compile –force –warnings-as-errors
e9539311a23e	2 days ago	image layer: COPY . .
995a21532fce	2 days ago	image layer: RUN mix deps.get
ecf7275feff3	2 days ago	image layer: Copy mix.exs mix.lock
86c81d89b023	2 days ago	image layer: ARG APP_NAME=my_app
	2 days ago	base image: elixir:1.8.1-alpine

IMAGE	CREATED	CREATED BY
cf650ef85086	0 days ago	image layer: CMD[“opt/app/bin/start_server..”]
fdd93d9c2c60	0 days ago	image layer: RUN mix compile –force –warnings-as-errors
530c750a346e	0 days ago	image layer: COPY . .
995a21532fce	2 days ago	image layer: RUN mix deps.get
ecf7275feff3	2 days ago	image layer: Copy mix.exs mix.lock
86c81d89b023	2 days ago	image layer: ARG APP_NAME=my_app
	2 days ago	base image: elixir:1.8.1-alpine

Cache will invalidate only if mix.exs or mix.lock changed or the docker Run command changed.

```
# Build stage Goes Here …
# PLT caching
COPY mix.exs mix.lock ./
COPY apps/$APP_NAME/mix.exs apps/$APP_NAME/mix.exs
COPY apps/${APP_NAME}_api/mix.exs apps/${APP_NAME}_api/mix.exs

RUN mix deps.get && \
MIX_ENV=dev mix deps.compile && \
MIX_ENV=test mix deps.compile && \
mix deps.compile && \
MIX_ENV=dev mix dialyzer --plt --halt-exit-status

COPY . .

# Check format of source code, compilation warnings, static
code analysis, run tests and release
RUN mix format --check-formatted && \
mix compile --force --warnings-as-errors && \
MIX_ENV=dev mix dialyzer --halt-exit-status && \
MIX_ENV=test mix test --exclude integration && \
mix release --env=prod --verbose && \
mkdir -p /opt/release && \
tar -xzf _build/prod/rel/${APP_NAME}/releases/*/$
{APP_NAME}.tar.gz -C /opt/release && \
mv /opt/release/bin/${APP_NAME} /opt/release/bin/start_server

# Release stage
FROM alpine:3.9
RUN apk update && apk --no-cache --update add bash openssl
ENV PORT=8080 MIX_ENV=prod REPLACE_OS_VARS=true
WORKDIR /opt/app
EXPOSE ${PORT}
COPY --from=build /opt/release .
CMD ["/opt/app/bin/start_server", "foreground"]
```

Now we build

- First copy our files to the image.
- Check code formatted.
- Check for compilation warnings.
- Run static code analysis.
- Run unit tests.
- Build a release package.

```
# Build stage Goes Here …
# PLT caching
COPY mix.exs mix.lock ./
COPY apps/$APP_NAME/mix.exs apps/$APP_NAME/mix.exs
COPY apps/${APP_NAME}_api/mix.exs apps/${APP_NAME}_api/mix.exs

RUN mix deps.get && \
MIX_ENV=dev mix deps.compile && \
MIX_ENV=test mix deps.compile && \
mix deps.compile && \
MIX_ENV=dev mix dialyzer --plt --halt-exit-status

COPY . .

# Check format of source code, compilation warnings, static
code analysis, run tests and release
RUN mix format --check-formatted && \
mix compile --force --warnings-as-errors && \
MIX_ENV=dev mix dialyzer --halt-exit-status && \
MIX_ENV=test mix test --exclude integration && \
mix release --env=prod --verbose && \
mkdir -p /opt/release && \
tar -xzf _build/prod/rel/${APP_NAME}/releases/*/$
{APP_NAME}.tar.gz -C /opt/release && \
mv /opt/release/bin/${APP_NAME} /opt/release/bin/start_server

# Release stage
FROM alpine:3.9
RUN apk update && apk --no-cache --update add bash openssl
ENV PORT=8080 MIX_ENV=prod REPLACE_OS_VARS=true
WORKDIR /opt/app
EXPOSE ${PORT}
COPY --from=build /opt/release .
CMD ["/opt/app/bin/start_server", "foreground"]
```

A Light-Weight Deployment

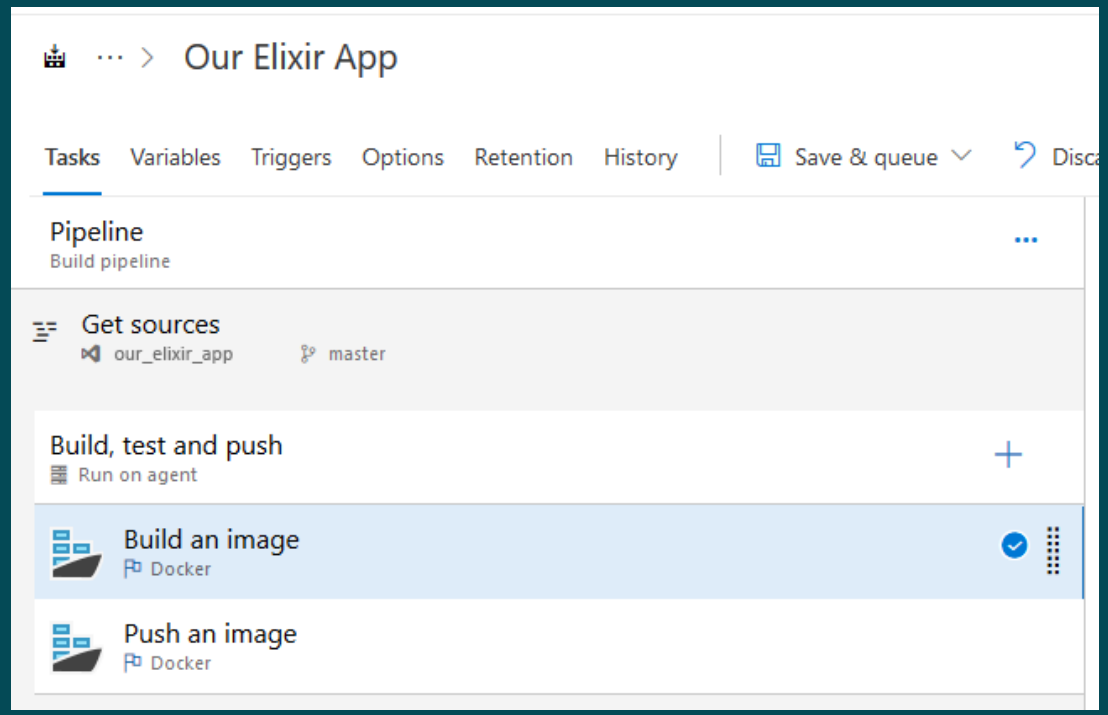
- We don’t need all the build tools, code, test results, …etc.
- Binaries and runtime is all what we really need.
- A multi-stage docker file is used.
- Only the release package is copied from the build image.

Azure build pipeline

```
pool:
  name: Hosted Ubuntu 1604

steps:
- task: Docker@0
  displayName: 'Build an image'
  inputs:
    azureSubscription: '[subscription]'
    azureContainerRegistry: ‘[CR configuration]'
    buildArguments:
     Some_ARG=$(Some_ARG)
    imageName: cr-host.io/$(Build.Repository.Name):latest’


- task: Docker@0
  displayName: 'Push an image'
  inputs:
    azureSubscription: '[subscription]'
    azureContainerRegistry: ‘[CR configuration]'
    action: 'Push an image'
    imageName: cr-host.io/$(Build.Repository.Name):latest’
```



STEP-BY-STEP GUIDE
GETTING STARTED WITH ELIXIR + DOCKER + AZURE

More articles at: medium.com/asolvi

#CodeBEAMSTO

Stockholm - 16-17 May 2019