Title: docker-compose build environment variable

Post Body:

TL;DR : How can I pass env variable when building the image with `docker-compose` and have `docker run image command` recognize them ?

I have this Dockerfile :

```
FROM mhart/alpine-node:10 ADD . /app WORKDIR /app RUN apk --no-cache add --virtual builds-deps build-base python &&\     yarn
```

and this docker-compose.yml :

```
version: '3.3' services:    api:     build:       context: .       dockerfile: Dockerfile-preprod     image: registry.git.loui
```

With this .env file (which is in the root folder, like docker-compose.yml and Dockerfile) :

```
#!/usr/bin/env bash  NODE_ENV=development PORT=9000 SECRET_SESSION=superSecr3t APP_NAME=Night Vision API_VERSION=/api/v0/ DEFA
```

And this code in the node server startup :

```
// Export the config object based on the NODE_ENV // ============================================== const config: IConfig = co
```

When I run the `docker-compose build` command, everything is fine, but for instance If I try `docker run myimage yarn run test` the Error 'Please set an environment' is thrown.

I would expect that

```
env_file:   - .env
```

makes the env variables of this file accessible in my image but that is not the case, that's why I tried to add

```
 environment:   - NODE_ENV=development
```

But still no success, I have also tried to pass my env variable as command line argument when I run the build :

```
docker-compose build --build-arg NODE_ENV=development api
```

But it gives me this message :

```
[Warning] One or more build-args [NODE_ENV] were not consumed Successfully built 9b14dd5abc3f
```

And I would really prefer to use the first or second methods

docker version : 18.06.1-ce docker-compose version : 1.19.0

Accepted Answer:

There's a note in the docs:

> **Note:** If your service specifies a build option, variables defined in environment are not automatically visible during the build. Use the args sub-option of build to define build-time environment variables.

It's also described in here. First (as mentioned above), you need to specify `ARG` in Dockerfile:

```
FROM mhart/alpine-node:10 ARG NODE_ENV ENV NODE_ENV $NODE_ENV ADD . /app WORKDIR /app RUN apk --no-cache add --virtual builds-
```

And then edit your `docker-compose` file to include the argument during build, like so:

```
build:   context: .   dockerfile: Dockerfile-preprod   args:     - NODE_ENV=development
```

Or even

```
build:   context: .   dockerfile: Dockerfile-preprod   args:     - NODE_ENV=${NODE_ENV}
```

Highest Rated Answer:

As per documentation under build args.

> You can omit the value when specifying a build argument, in which case its value at build time is the value in the environment where Compose is running.

```
args:   NODE_ENV:
```

Additionally, it will use the `.env` file as documented under variable substitution.

> You can set default values for environment variables using a .env file, which Compose automatically looks for. Values set in the shell environment override those set in the .env file.

So you can create a `.env` file like below.

```
NODE_ENV=production BASE_URL=/foo/bar/
```

And then just list them in the compose file either under `build.args` to make them available on *build*, or under `environment` to make them available on *run*.

```
version: '3.9' services:   app:     build:       context: .       # these are available on build       args:         NODE_ENV:
```

This is kind of useful. However, the problem that comes along with this approach is that if the variables are not set from shell or env file, the default values in the Dockerfile will be overwritten with an empty value.

If one wants to keep some sort of default, the following can be used.

```
version: '3.9' services:   app:     build:       context: .       args:         NODE_ENV: ${NODE_ENV:-dev}     environment:
```

This is making usage of posix parameter expansion. If the variable is not set, it will use the value after `:-`. So in the example above, it would default to `NODE_ENV=dev` and `BASE_URL=http://localhost:8080`, if those are not set.
Allowing to override them with a `.env` file or by setting a shell variable, i.e. `export NODE_ENV=prod`.

If you want to change the env file it's using, you can do that with the --env-file flag.

```
docker compose --env-file .my-env up
```