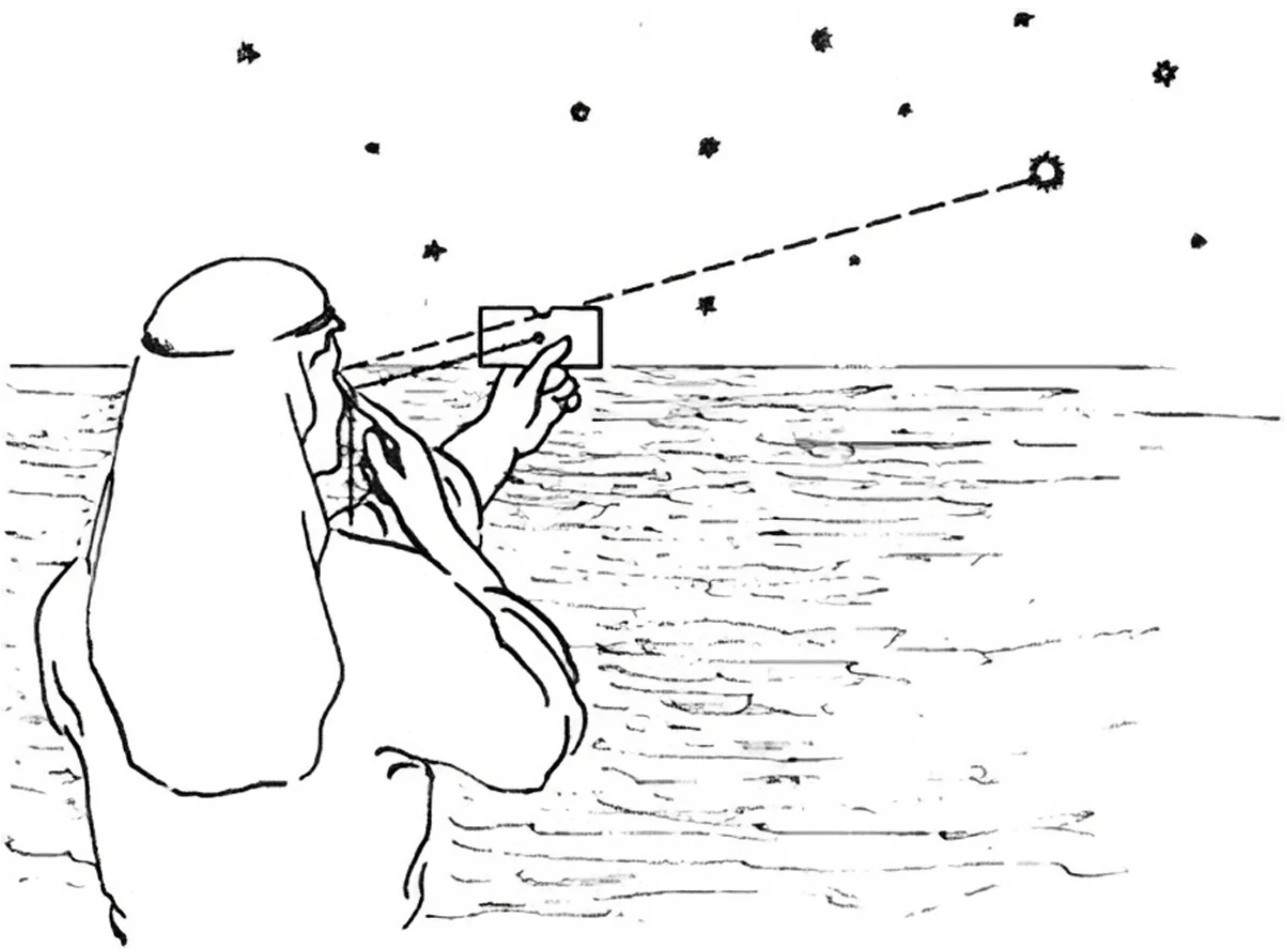
Deploying Phoenix application with Kamal 2

Kamal is a deployment tool designed for deploying containerized applications. It's specifically optimized for use with docker based applications, simplifying the process of building, deploying and managing applications across multiple servers.



(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/kamal.webp?ssl=1)

Recently (26 Sept 2024), Kamal 2 was released. This new version, which we are going to use in this tutorial, brings some important improvements over its predecessor, simplifying deployments and adding new features, like the ability to deploy multiple application to a single server.

While Kamal originated the Ruby on Rails ecosystem, it is versatile enough to be used with application that can be containerized with Docker. Its design focus on container orchestration and deployment, meaning that, as long as the application runs in a Docker container, Kamal can deploy it. To demonstrate that, in this tutorial we will be deploying a Phoenix (Elixir) application.

Requirements

To follow along, we will need to meet some requirements: a server where we'll be deploying our application; the application (phoenix application in our case) to be deployed; and Kamal 2.

For the application, let's create a new brand new phoenix application. The only requirement for this application, is that it needs to have a route for heath checking. By default this path is `/up`.

Preparing the application

```
$ elixir --version
Erlang/OTP 27 [erts-15.1] [source] [64-bit] [smp:12:12] [ds:12:12:10]
[async-threads:1] [jit]

Elixir 1.17.3 (compiled with Erlang/OTP 27)

$ mix phx.new --version
Phoenix installer v1.7.14

$ mix phx.new blogx
* creating blogx/lib/blogx/application.ex
* creating blogx/lib/blogx.ex
...
Fetch and install dependencies? [Yn] Y
* running mix deps.get
...

$ cd blogx
$ mix ecto.create

$ mix phx.gen.html Blog Post posts title:string content:text
$ mix ecto.migrate
```

We just created a new phoenix application, we setup the database (PostgreSQL) and we generated our blog context and post controller for the Post resource. Before we move on, there is few changes we need to make.

First I added the routes we need for this tutorial. I added the root path, the `/up` path and the post resources paths:

```
scope "/", BlogxWeb do
  pipe_through :browser

  get "/", PostController, :index
  get "/up", PageController, :health_check

  resources "/posts", PostController
end
```

I also changed the `PageController`, replacing the `home/2` action by our new `health_check/2` action:

```
defmodule BlogxWeb.PageController do
  use BlogxWeb, :controller

  def health_check(conn, _params) do
    send_resp(conn, 200, "OK")
  end
end
```

We this, we can fire up our local server and be able to visit the following paths. We can even interact with our new app creating, edit and deleting posts. So, we just created a basic CRUD for posts.

```
http://localhost:4000/
http://localhost:4000/up
http://localhost:4000/posts (CRUD)
```

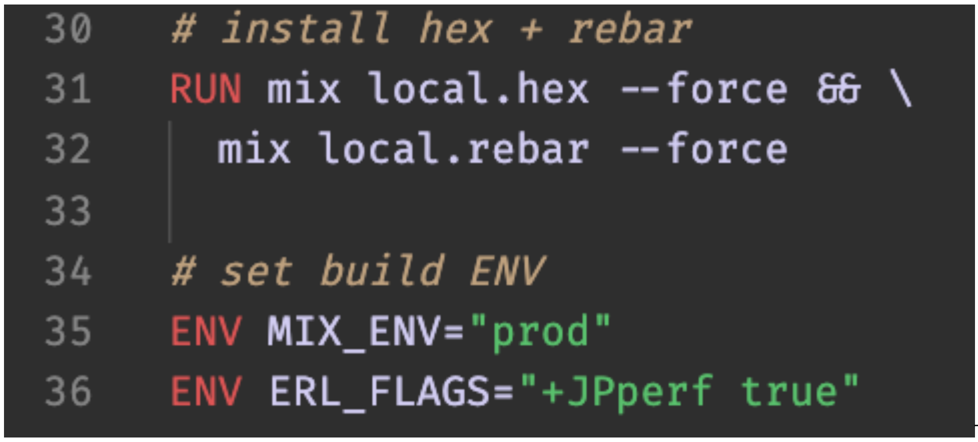
At this point, we have our application running on localhost. Now we need to prepare it, to be released as a docker image, by assembling the release inside de Docker container. To learn more about releases, please check: https://hexdocs.pm/phoenix/releases.html (https://hexdocs.pm/phoenix/releases.html).

```
$ mix phx.gen.release --docker
```

Check the previous command output to se which files were generated. For the purposes of this tutorial, we just need to make two small changes in the **Dockerfile**. first, where we will be exposing the PORT. So, open the file "/path_to_app/Dockerfile" and at the end, before the last line (CMD ["/app/bin/server"]), add the instruction to expose the port 4000. The end of your Dockerfile will look like this:

```
# If using an environment that doesn't automatically reap zombie processes, it is
# advised to add an init process such as tini via `apt-get install`
# above and adding an entrypoint. See https://github.com/krallin/tini for details
# ENTRYPOINT ["/tini", "--"]

EXPOSE 4000

CMD ["/app/bin/server"]
```

The other change is also in Dockerfile. In the begging of the file, approx. on line 35, we will find an instruction to set the MIX environment to "prod". Under this line, put another variable (ENV ERL_FLAGS="+JPperf true"), like the following image:



```
# install hex + rebar
RUN mix local.hex --force && \
  mix local.rebar --force

# set build ENV
ENV MIX_ENV="prod"
ENV ERL_FLAGS="+JPperf true"
```

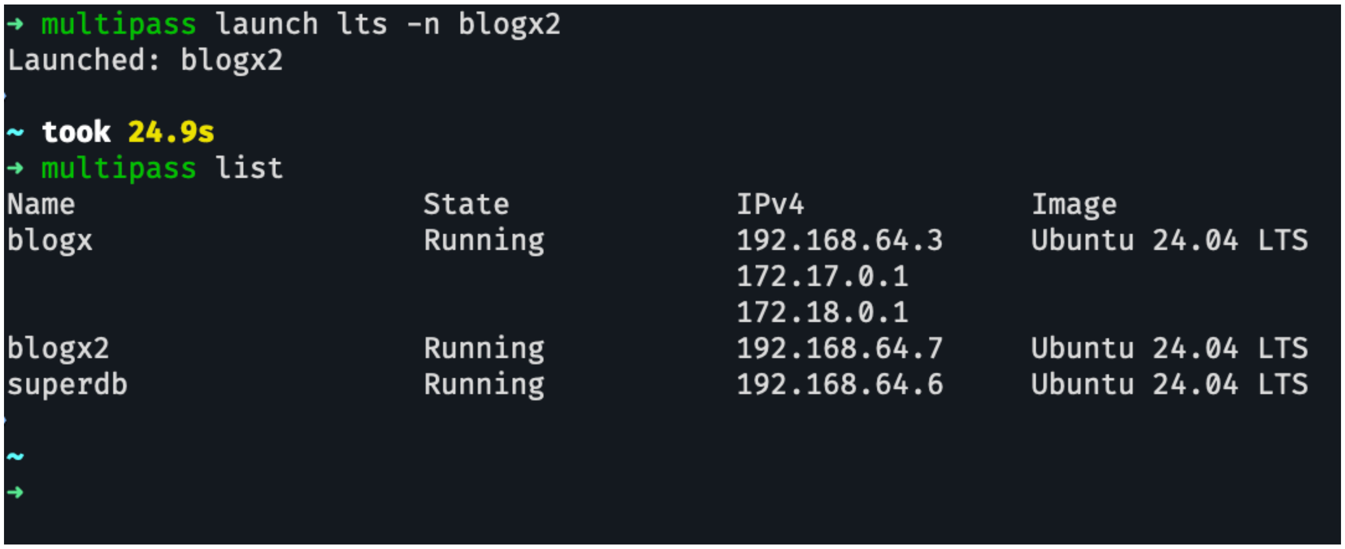
(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-18.55.08.png?ssl=1)

Our release is prepared. Now we need to setup Kamal.

Setup the server

Before we continue, we need to setup a server. for that you can use a VPS from some cloud providers like, Hetzner (https://hetzner.cloud/?ref=ZTyMH2nGOEfN), Linode (https://www.linode.com/lp/refer/?r=6deba21e3f06572f94f491b81165403e795c7f37), DigitalOcean, OVH, etc. but for this tutorial, I'm going to use Multipass to create servers locally.

Multipass is a lightweight virtualization tool created by Canonical that allows users to easily crate, manage and run Ubuntu virtual machines on local system. Multipass works on macOS, Windows and Linux. To learn more, visit: https://multipass.run/docs (https://multipass.run/docs).



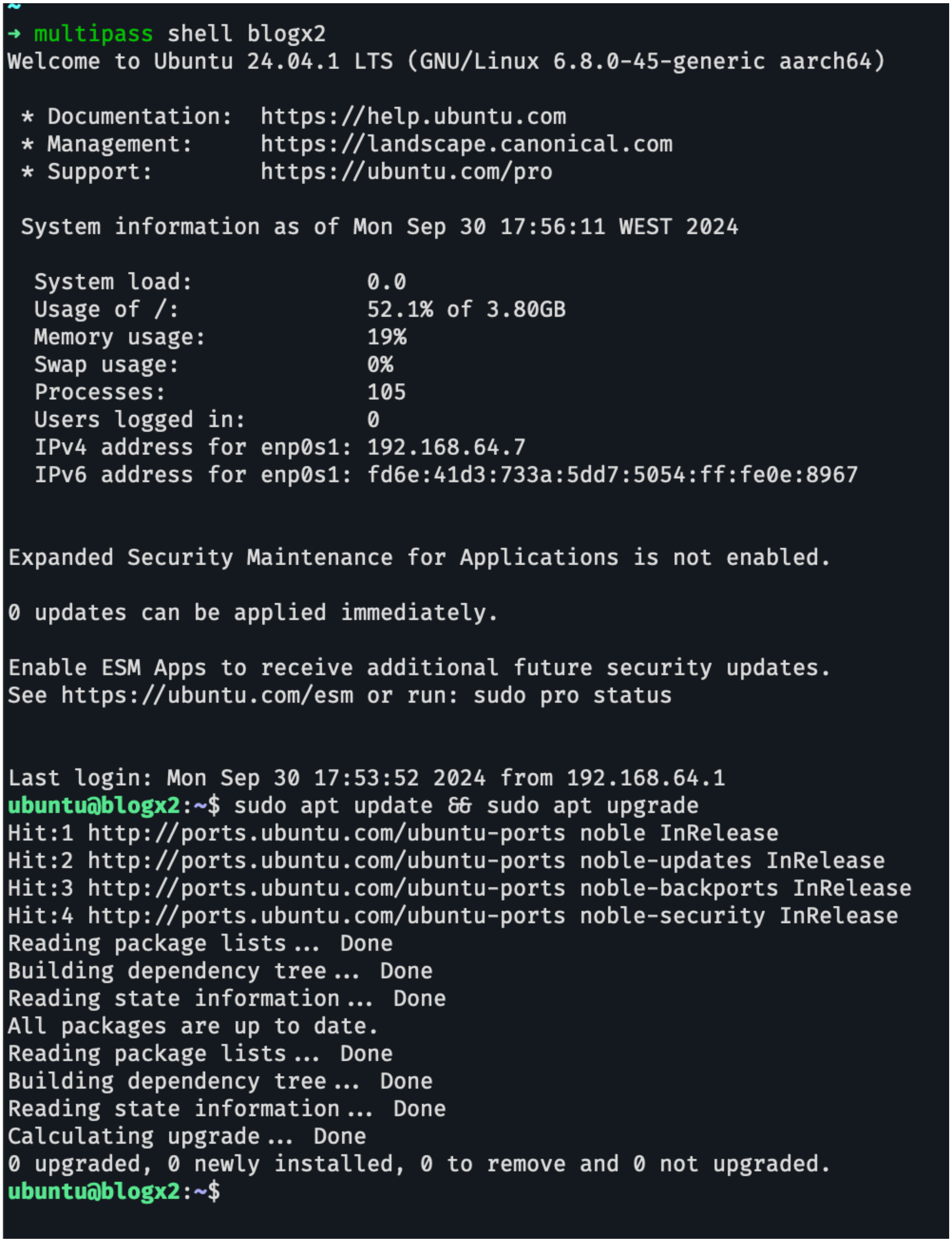
```
→ multipass launch lts -n blogx2
Launched: blogx2

~ took 24.9s
→ multipass list
Name                    State             IPv4             Image
blogx                   Running           192.168.64.3     Ubuntu 24.04 LTS
                                          172.17.0.1
                                          172.18.0.1
blogx2                  Running           192.168.64.7     Ubuntu 24.04 LTS
superdb                 Running           192.168.64.6     Ubuntu 24.04 LTS

~
→
```

(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-17.50.58.png?ssl=1)

As you can see, we just created and launched a brand new virtual machine, running Ubuntu 24.04. By the way, this VM is using the IP address 192.168.64.7, which we will need to setup Kamal. And for the database we will be using a separate server, where we already have PostgreSQL installed. ITs IP is

192.168.64.6 and the credentials are de defaults one: “postgres” as both username and password. Note that, if you follow this approach, you need to make sure that you setup the Postgresql to accept connection from outside.

Now we just need to login via SSH into our server and make sure its up-to-date.



```
~
➜ multipass shell blogx2
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-45-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Mon Sep 30 17:56:11 WEST 2024

  System load:             0.0
  Usage of /:              52.1% of 3.80GB
  Memory usage:            19%
  Swap usage:              0%
  Processes:               105
  Users logged in:         0
  IPv4 address for enp0s1: 192.168.64.7
  IPv6 address for enp0s1: fd6e:41d3:733a:5dd7:5054:ff:fe0e:8967


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Mon Sep 30 17:53:52 2024 from 192.168.64.1
ubuntu@blogx2:~$ sudo apt update && sudo apt upgrade
Hit:1 http://ports.ubuntu.com/ubuntu-ports noble InRelease
Hit:2 http://ports.ubuntu.com/ubuntu-ports noble-updates InRelease
Hit:3 http://ports.ubuntu.com/ubuntu-ports noble-backports InRelease
Hit:4 http://ports.ubuntu.com/ubuntu-ports noble-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ubuntu@blogx2:~$
```

(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-17.57.37.png?ssl=1)

There one other think you need to do, unless you are using "root" user. In my case, I am using "ubuntu" user. In order to be able to manage docker inside this server, this user must be part of the "docker" group. At this stage this group does not exist, so I will create it and add the user to it.

```
$ sudo groupadd docker
$ sudo usermod -aG docker ubuntu
```

Our servers are ready. We have both IPs (192.168.64.7 for the application and 192.168.64.6 for the database). You could run both on the same server, though. I just prefer to separate them.

Also, if you are using Multipass, don't forget to setup you SSH public key on the server or the password to be able to connect to the server as "ubuntu" user.

Setting up Kamal 2

First we need to install Kamal. There is two ways to install Kamal. If you have a Ruby environment available, you can install kamal like this:

```
$ gem install kamal
```

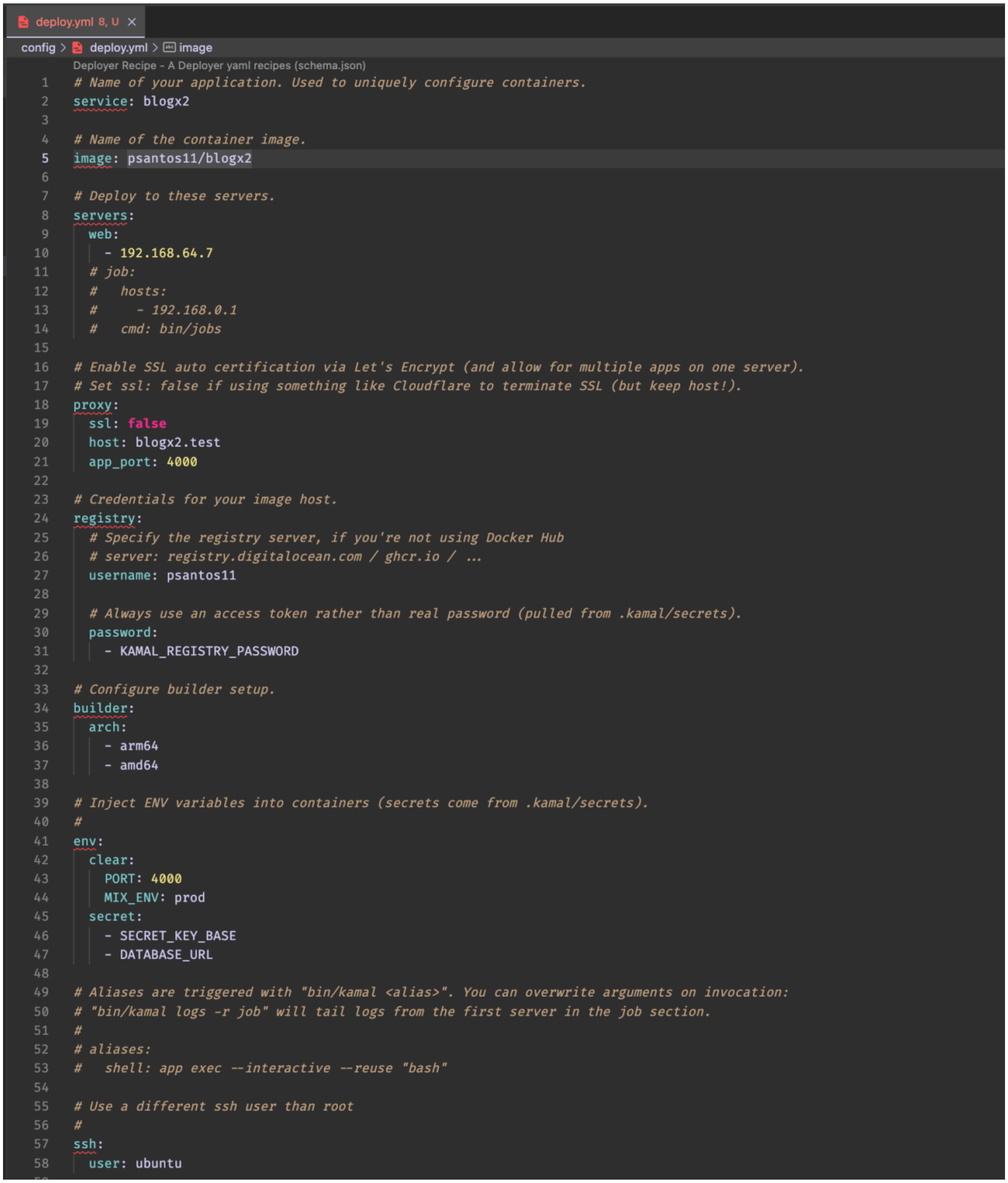
Since we are trying to deploy an Elixir (Phoenix) application, I will assume that we don't have Ruby environment available. For such situation, the option is to use the dockerized version of Kamal, by creating an alias as follows:

```
# For macOS users:
alias kamal='docker run -it --rm -v "${PWD}:/workdir" -v "/run/host-services/ssh-auth.sock:/run/host-services/ssh-auth.sock" -e SSH_AUTH_SOCK="/run/host-services/ssh-auth.sock" -v /var/run/docker.sock:/var/run/docker.sock ghcr.io/basecamp/kamal:latest'

# For Linux users:
alias kamal='docker run -it --rm -v "${PWD}:/workdir" -v "${SSH_AUTH_SOCK}:/ssh-agent" -v /var/run/docker.sock:/var/run/docker.sock -e "SSH_AUTH_SOCK=/ssh-agent" ghcr.io/basecamp/kamal:latest'
```

With this, we will have "kamal" command available and we can proceed initializing the configuration by running the following command. Note that this command must be run inside the app directory.

```
$ kamal init
```

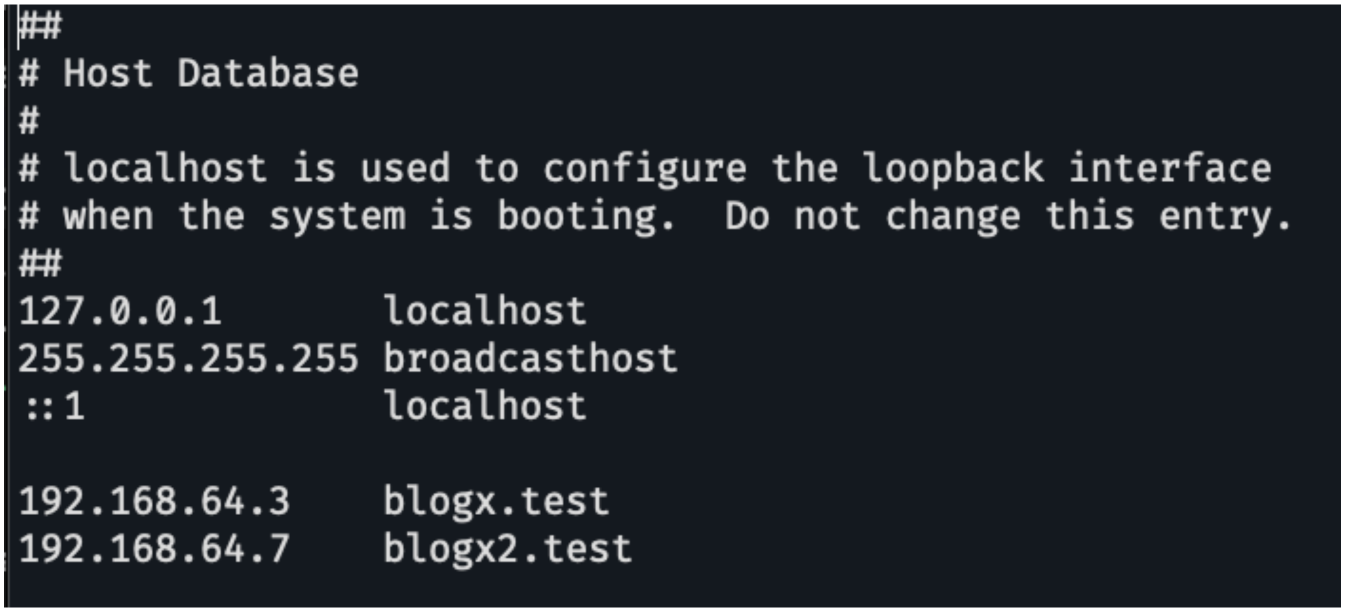
The previous command will create ".kamal" directory, within a "secrets" file we will use to manage our secrets, and a "config/deploy.yml" file. Lets start modifying this file:


deploy.yml 8, U
config > deploy.yml > image
Deployer Recipe - A Deployer yaml recipes (schema.json)

```yaml
# Name of your application. Used to uniquely configure containers.
service: blogx2

# Name of the container image.
image: psantos11/blogx2

# Deploy to these servers.
servers:
  web:
    - 192.168.64.7
  # job:
  #   hosts:
  #     - 192.168.0.1
  #   cmd: bin/jobs

# Enable SSL auto certification via Let's Encrypt (and allow for multiple apps on one server).
# Set ssl: false if using something like Cloudflare to terminate SSL (but keep host!).
proxy:
  ssl: false
  host: blogx2.test
  app_port: 4000

# Credentials for your image host.
registry:
  # Specify the registry server, if you're not using Docker Hub
  # server: registry.digitalocean.com / ghcr.io / ...
  username: psantos11

  # Always use an access token rather than real password (pulled from .kamal/secrets).
  password:
    - KAMAL_REGISTRY_PASSWORD

# Configure builder setup.
builder:
  arch:
    - arm64
    - amd64

# Inject ENV variables into containers (secrets come from .kamal/secrets).
#
env:
  clear:
    PORT: 4000
    MIX_ENV: prod
  secret:
    - SECRET_KEY_BASE
    - DATABASE_URL

# Aliases are triggered with "bin/kamal <alias>". You can overwrite arguments on invocation:
# "bin/kamal logs -r job" will tail logs from the first server in the job section.
#
# aliases:
#   shell: app exec --interactive --reuse "bash"

# Use a different ssh user than root
#
ssh:
  user: ubuntu
```

(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-18.11.38.png?ssl=1)

The visible part from the previous image is the only part of the file I had to change. The comments on the file does a good job explains each key in the file. But I would still like to comment about few of them. Under proxy key, we first disabled "ssl", because I still on local development and its for testing purposes, so I don't want to deal with SSL certificates at this time; "host" is the domain I will be using to access this app. I am also using a test domain. For that I had to change my "/etc/hosts" file. You could use something else like "dnsmasq (https://thekelleys.org.uk/dnsmasq/doc.html)" is you prefer. And I also added "app_port". But default, Kamal 2 expects the container to be exposing port 80. But in our case, phoenix is running on port 4000. So we need to specify this custom port.



```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost

192.168.64.3    blogx.test
192.168.64.7    blogx2.test
```

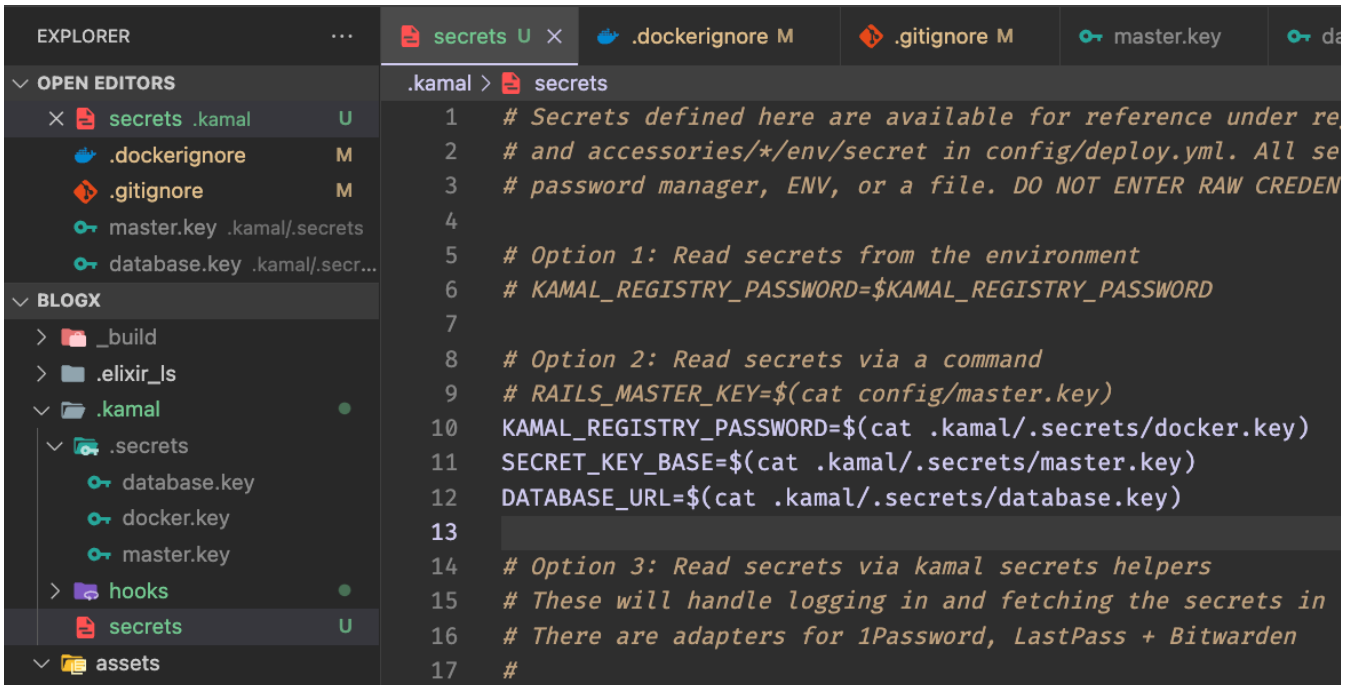
(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-18.17.44.png?ssl=1)

The other change I had to do is on "builder" key. As you can see from the previous image, I added "arm64" architecture, since I am using Mac with apple chip, and consequently my VMs are using this architecture.

Next we defined the environment variables that will be available for the container. Starting again with PORT and ENV where our app will be running. The other two environment variables are secret, meaning their values comes from .kamal/secrets. Is there where we will define how Kamal will found this values for KAMAL_REGISTRY_PASSWORD (the token from you your container image registry), SECRET_KEY_BASE and for DATABASE_URL.

Last not least, Kamal expect by default to be deploying using the root user. Normally I prefer to have a separate user for this tasks. In my case, ubuntu user. Remember to setup the SSH key so that you can access the server from your computer without tying the password.

The last thing we need to check and adjust is the ".kamal/secrets" file. From where will will pick the values to the environment variables we just specified in our "config/deploy.yml" file.



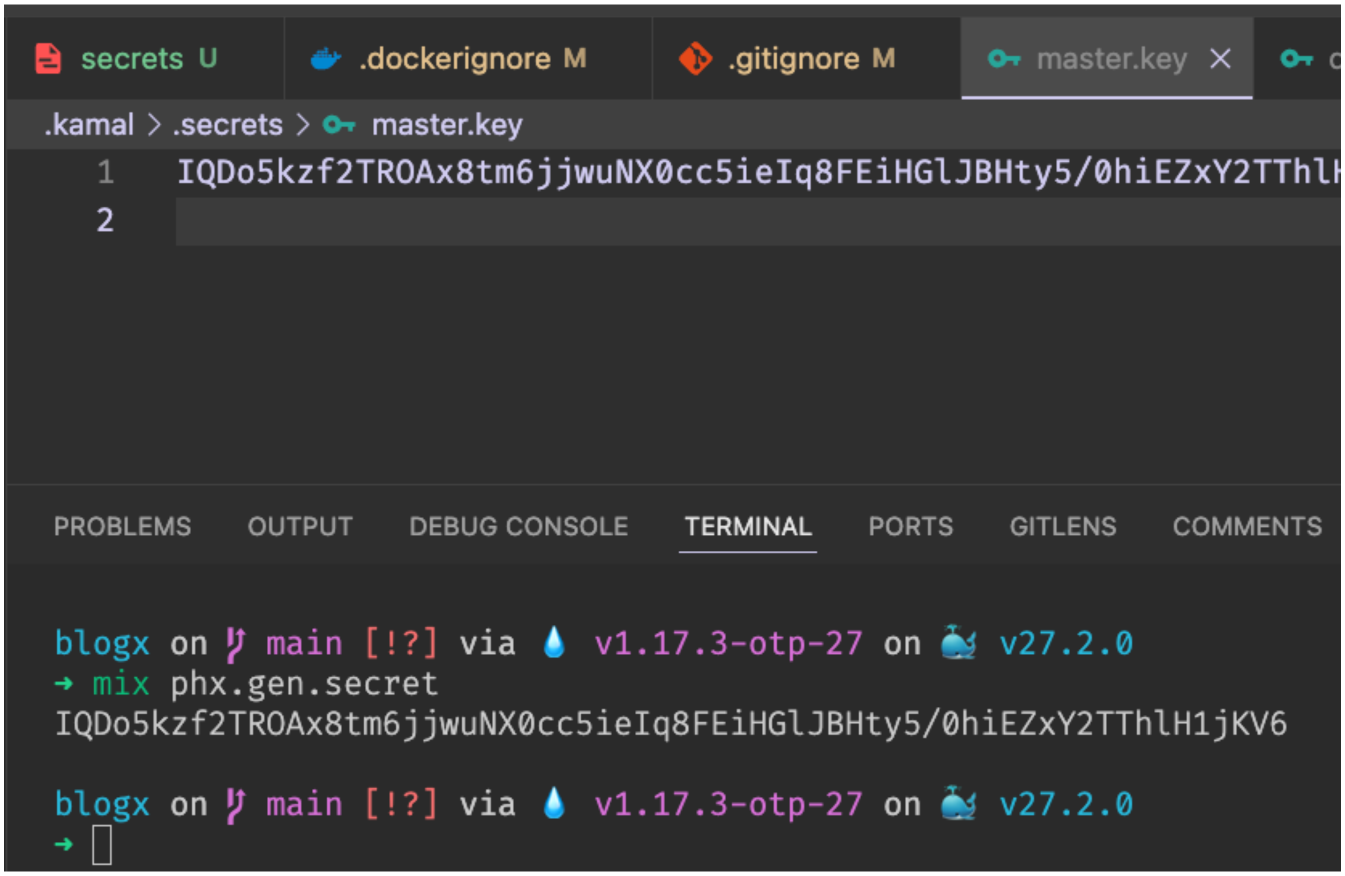
```
# Secrets defined here are available for reference under re
# and accessories/*/env/secret in config/deploy.yml. All se
# password manager, ENV, or a file. DO NOT ENTER RAW CREDEN

# Option 1: Read secrets from the environment
# KAMAL_REGISTRY_PASSWORD=$KAMAL_REGISTRY_PASSWORD

# Option 2: Read secrets via a command
# RAILS_MASTER_KEY=$(cat config/master.key)
KAMAL_REGISTRY_PASSWORD=$(cat .kamal/.secrets/docker.key)
SECRET_KEY_BASE=$(cat .kamal/.secrets/master.key)
DATABASE_URL=$(cat .kamal/.secrets/database.key)

# Option 3: Read secrets via kamal secrets helpers
# These will handle logging in and fetching the secrets in
# There are adapters for 1Password, LastPass + Bitwarden
#
```

(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-18.34.46.png?ssl=1)

As you can see from the comments, there are different ways to read the values. I prefer using the cat command. But for this option, pay double attention to never commit those files. They need always to be ignored both on .gitignore and .dockerignore files.

To generate the SECRET_KEY_BASE, you can run the following command:

```
$ mix phx.gen.secret
```



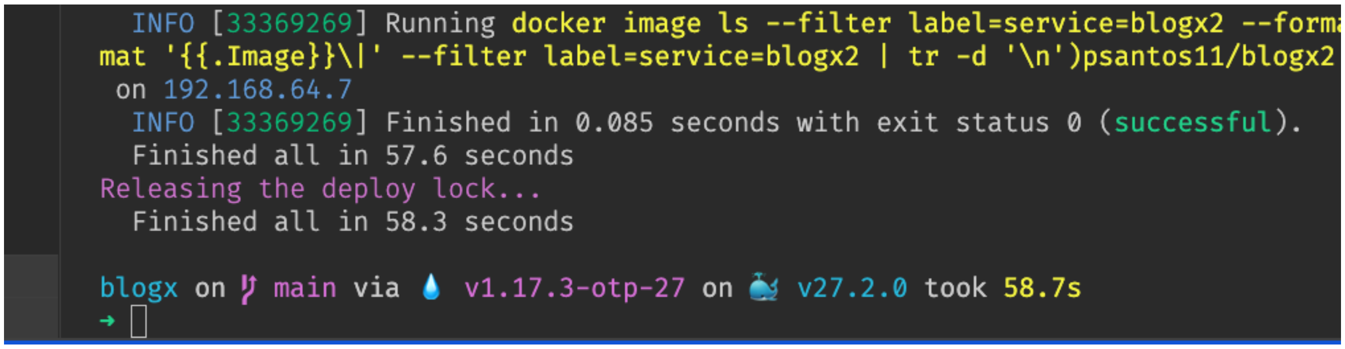
```
IQDo5kzf2TROAx8tm6jjwuNX0cc5ieIq8FEiHGlJBHty5/0hiEZxY2TThlH
```

```
blogx on  main [!?] via 💧 v1.17.3-otp-27 on 🐳 v27.2.0
➜ mix phx.gen.secret
IQDo5kzf2TROAx8tm6jjwuNX0cc5ieIq8FEiHGlJBHty5/0hiEZxY2TThlH1jKV6

blogx on  main [!?] via 💧 v1.17.3-otp-27 on 🐳 v27.2.0
➜
```

[(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-18.40.17.png?ssl=1)](https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-18.40.17.png?ssl=1)

With all this in place. We are ready to run the "kamal setup". Which will setup our server and deploy our application.

After running "kamal setup" we will get the confirmation that our app was successfully deployed.



```
  INFO [33369269] Running docker image ls --filter label=service=blogx2 --forma
mat '{{.Image}}\|' --filter label=service=blogx2 | tr -d '\n')psantos11/blogx2
 on 192.168.64.7
  INFO [33369269] Finished in 0.085 seconds with exit status 0 (successful).
  Finished all in 57.6 seconds
Releasing the deploy lock...
  Finished all in 58.3 seconds

blogx on  main via 💧 v1.17.3-otp-27 on 🐳 v27.2.0 took 58.7s
➜
```

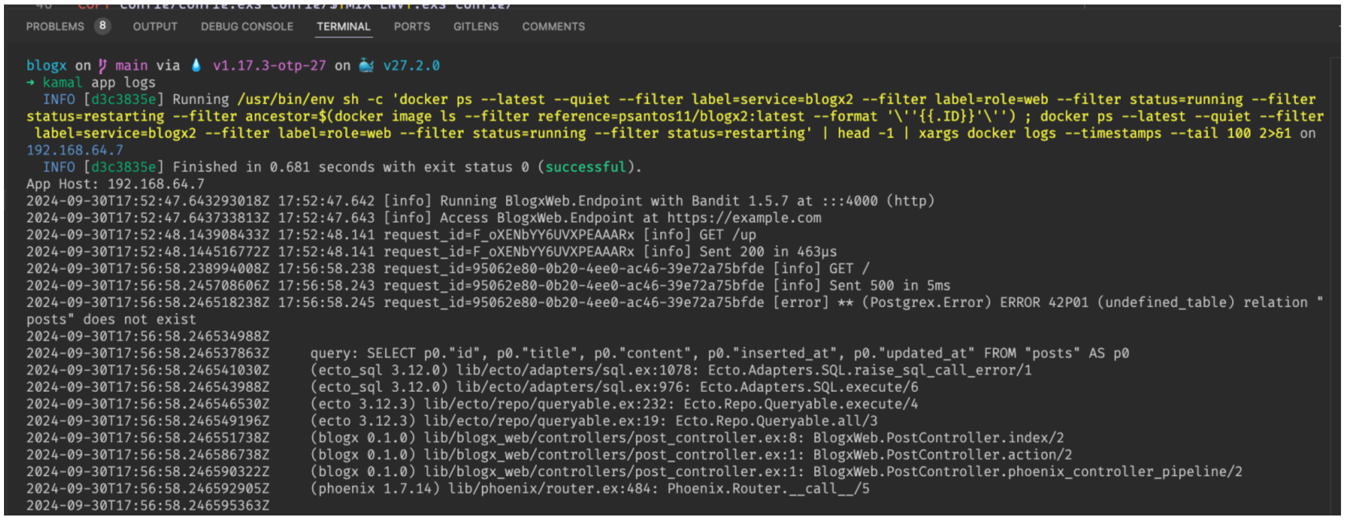
[(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-18.55.41.png?ssl=1)](https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-18.55.41.png?ssl=1)

Now, we can try to visit our application, using the domain we chose previously. In my case: [http://blogx2.test/ (http://blogx2.test/)](http://blogx2.test/) . But doing so, I got an error. 😢

To understand what is going on, we can start checking the logs, by running:

```
$ kamal app logs
```

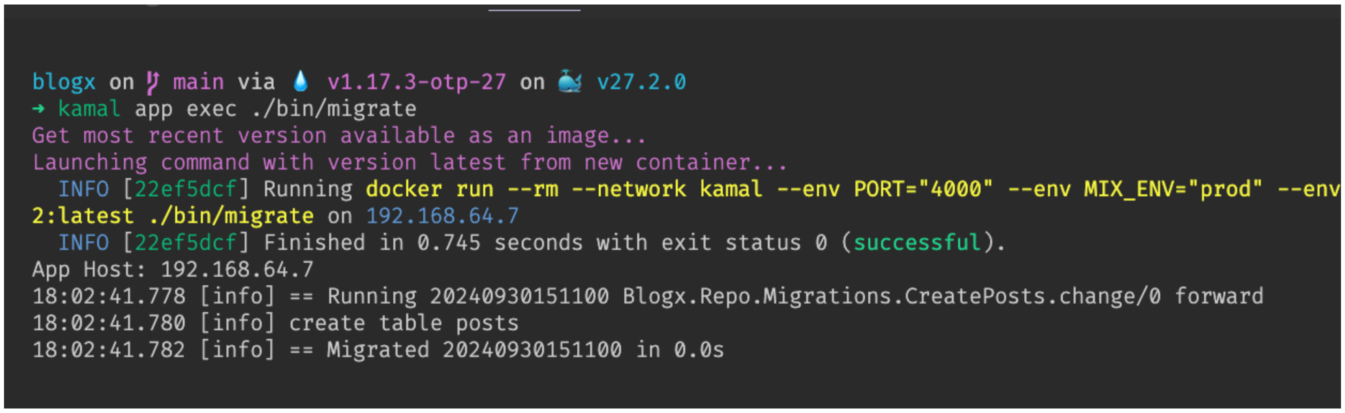
From the output, it will be easy to figure out what is going on. In our case, we forgot to run the migrations. So, how could we do that?



```
blogx on ᚴ main via 💧 v1.17.3-otp-27 on 🐳 v27.2.0
➜ kamal app logs
  INFO [d3c3835e] Running /usr/bin/env sh -c 'docker ps --latest --quiet --filter label=service=blogx2 --filter label=role=web --filter status=running --filter
status=restarting --filter ancestor=$(docker image ls --filter reference=psantos11/blogx2:latest --format '\''{{.ID}}'\'') ; docker ps --latest --quiet --filter
 label=service=blogx2 --filter label=role=web --filter status=running --filter status=restarting' | head -1 | xargs docker logs --timestamps --tail 100 2>&1 on
192.168.64.7
  INFO [d3c3835e] Finished in 0.681 seconds with exit status 0 (successful).
App Host: 192.168.64.7
2024-09-30T17:52:47.643293018Z 17:52:47.642 [info] Running BlogxWeb.Endpoint with Bandit 1.5.7 at :::4000 (http)
2024-09-30T17:52:47.643733813Z 17:52:47.643 [info] Access BlogxWeb.Endpoint at https://example.com
2024-09-30T17:52:48.143908433Z 17:52:48.141 request_id=F_oXENbYY6UVXPEAAARx [info] GET /up
2024-09-30T17:52:48.144516772Z 17:52:48.141 request_id=F_oXENbYY6UVXPEAAARx [info] Sent 200 in 463µs
2024-09-30T17:56:58.238994008Z 17:56:58.238 request_id=95062e80-0b20-4ee0-ac46-39e72a75bfde [info] GET /
2024-09-30T17:56:58.245708606Z 17:56:58.243 request_id=95062e80-0b20-4ee0-ac46-39e72a75bfde [info] Sent 500 in 5ms
2024-09-30T17:56:58.246518238Z 17:56:58.245 request_id=95062e80-0b20-4ee0-ac46-39e72a75bfde [error] ** (Postgrex.Error) ERROR 42P01 (undefined_table) relation "
posts" does not exist
2024-09-30T17:56:58.246534988Z
2024-09-30T17:56:58.246537863Z     query: SELECT p0."id", p0."title", p0."content", p0."inserted_at", p0."updated_at" FROM "posts" AS p0
2024-09-30T17:56:58.246541030Z     (ecto_sql 3.12.0) lib/ecto/adapters/sql.ex:1078: Ecto.Adapters.SQL.raise_sql_call_error/1
2024-09-30T17:56:58.246543988Z     (ecto_sql 3.12.0) lib/ecto/adapters/sql.ex:976: Ecto.Adapters.SQL.execute/6
2024-09-30T17:56:58.246546530Z     (ecto 3.12.3) lib/ecto/repo/queryable.ex:232: Ecto.Repo.Queryable.execute/4
2024-09-30T17:56:58.246549196Z     (ecto 3.12.3) lib/ecto/repo/queryable.ex:19: Ecto.Repo.Queryable.all/3
2024-09-30T17:56:58.246551738Z     (blogx 0.1.0) lib/blogx_web/controllers/post_controller.ex:8: BlogxWeb.PostController.index/2
2024-09-30T17:56:58.246586738Z     (blogx 0.1.0) lib/blogx_web/controllers/post_controller.ex:1: BlogxWeb.PostController.action/2
2024-09-30T17:56:58.246590322Z     (blogx 0.1.0) lib/blogx_web/controllers/post_controller.ex:1: BlogxWeb.PostController.phoenix_controller_pipeline/2
2024-09-30T17:56:58.246592905Z     (phoenix 1.7.14) lib/phoenix/router.ex:484: Phoenix.Router.__call__/5
2024-09-30T17:56:58.246595363Z
```

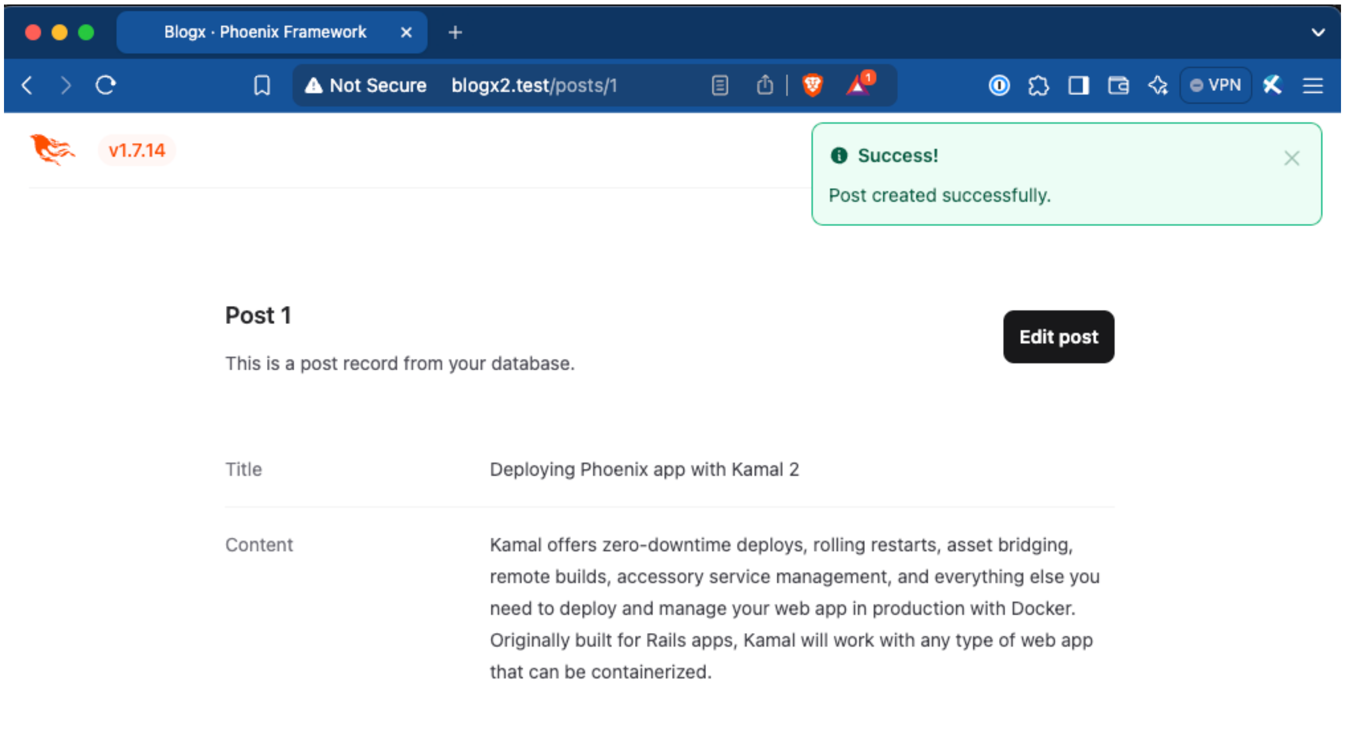
[(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-19.01.02.png?ssl=1)](https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-19.01.02.png?ssl=1)

Turns out, with kamal, we can run commands on the server using “kamal app exec CMD”. and the command we need to run is “./bin/migrate”.



```
blogx on ᚴ main via 💧 v1.17.3-otp-27 on 🐳 v27.2.0
➜ kamal app exec ./bin/migrate
Get most recent version available as an image...
Launching command with version latest from new container...
  INFO [22ef5dcf] Running docker run --rm --network kamal --env PORT="4000" --env MIX_ENV="prod" --env
2:latest ./bin/migrate on 192.168.64.7
  INFO [22ef5dcf] Finished in 0.745 seconds with exit status 0 (successful).
App Host: 192.168.64.7
18:02:41.778 [info] == Running 20240930151100 Blogx.Repo.Migrations.CreatePosts.change/0 forward
18:02:41.780 [info] create table posts
18:02:41.782 [info] == Migrated 20240930151100 in 0.0s
```

[(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-19.02.53.png?ssl=1)](https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-19.02.53.png?ssl=1)



Success!
Post created successfully.

Post 1

This is a post record from your database.

Title	Deploying Phoenix app with Kamal 2
Content	Kamal offers zero-downtime deploys, rolling restarts, asset bridging, remote builds, accessory service management, and everything else you need to deploy and manage your web app in production with Docker. Originally built for Rails apps, Kamal will work with any type of web app that can be containerized.

(https://i0.wp.com/blog.psantos.dev/wp-content/uploads/2024/09/Screenshot-2024-09-30-at-19.04.34.png?ssl=1)

And that is it for today. I hope you liked it. If you find an issue, please let me know so we can figure out together.