Deploying applications

Deployment options

We have two deployment options:

- deploying to a single-host
- deploying to a cluster

Single-host deployment is simple and easy, but they don't scale up. Cluster deployment requires orchestration tools. Currently, the most popular orchestration tool is Google's Kubernetes.

Virtual Private Server

To deploy our application we need a VPS. We can procure it from platforms like Digital Ocean, GCP, Azure, AWS, etc.

Production configuration

The `docker-compose.yml` file that we wrote in the last section was great for development, but not for production.

For example, volume mappings for sharing source code with the application are not needed in production since the code will not change (until we ship a new version of course). We also don't want to be running the tests in production constantly. Testing should happen before we ship. We will create a separate compose file for our production environment. The convention is for this file to be called `docker-compose.prod.yml`. We can also change the mapping for the development server.

We also want to add a `restart` policy for each service (in case they need to be restarted). The default value is `no`. If the application crushes it will stay down until we manually bring it back up. Possible other values are:

- `always` will always restart.
- `on-failure` will restart after a failure.
- `unless-stopped` will only restart the container if we manually stopped it.

```
version: "3.8"

services:
  web:
    build: ./frontend
    ports:
      - 80:3000
    restart: unless-stopped
```

```
  api:
    build: ./backend
    ports:
      - 3001:3001
    environment:
      DB_URL: mongodb://db/vidly
    command: ./docker-entrypoint.sh
    restart: unless-stopped
  db:
    image: mongo:4.0-xenial
    ports:
      - 27017:27017
    volumes:
      - vidly:/data/db
    restart: unless-stopped

volumes:
  vidly:
```

We could create other compose files for other environments: `docker-compose.testing.yml`, `docker-compose.stg.yml`, etc. When calling `docker-compose build` we need to pass the correct file using the `-f` flag:

```
docker-compose -f docker-compose.prod.yml build
```

Likewise, when using the `docker-compose up` command, we need to supply the file name:

```
docker-compose -f docker-compose.prod.yml up
```

If we also want to do changes to the `Dockerfile` themselves and have different ones for different environments (dev, stg, prod, testing, etc) those files will also be called `Dockerfile.prod`, `Dockerfile.stg`, and so on. So now, we need to expand the declaration of the `build` in our `docker-compose.prod.yml` so that the correct `Dockerfile` is used.

```
version: "3.8"

services:
  web:
    build:
      context: ./frontend
      dockerfile: Dockerfile.prod
    ports:
      - 80:80
    restart: unless-stopped
  api:
```

```
    build:
      context: ./backend
      dockerfile: Dockerfile.prod
    ports:
      - 3001:3001
    environment:
      DB_URL: mongodb://db/vidly
    command: ./docker-entrypoint.sh
    restart: unless-stopped
  db:
    image: mongo:4.0-xenial
    ports:
      - 27017:27017
    volumes:
      - vidly:/data/db
    restart: unless-stopped

volumes:
  vidly:
```