

Containers Are Always Great!

In Development

Isolated, standalone environment

Reproducible environment, easy to
share and use

In Production

Isolated, standalone environment

Reproducible environment, easy to
share and use

No surprises!

What works on your machine (in a container) will also work after deployment

Development to Production: Things To Watch Out For

Bind Mounts shouldn't be used in Production!

Containerized apps **might need a build step** (e.g. React apps)

Multi-Container projects might need to be **split** (or should be split) across multiple hosts / remote machines

Trade-offs between **control** and **responsibility** might be worth it!

Bind Mounts, Volumes & COPY

In Development

Containers should encapsulate the runtime environment but not necessarily the code

Use “Bind Mounts” to provide your local host project files to the running container

Allows for instant updates without restarting the container

In Production

Image / Container is the “single source of truth”

A container should really work standalone, you should NOT have source code on your remote machine

Use COPY to copy a code snapshot into the image

Ensures that every image runs without any extra, surrounding configuration or code

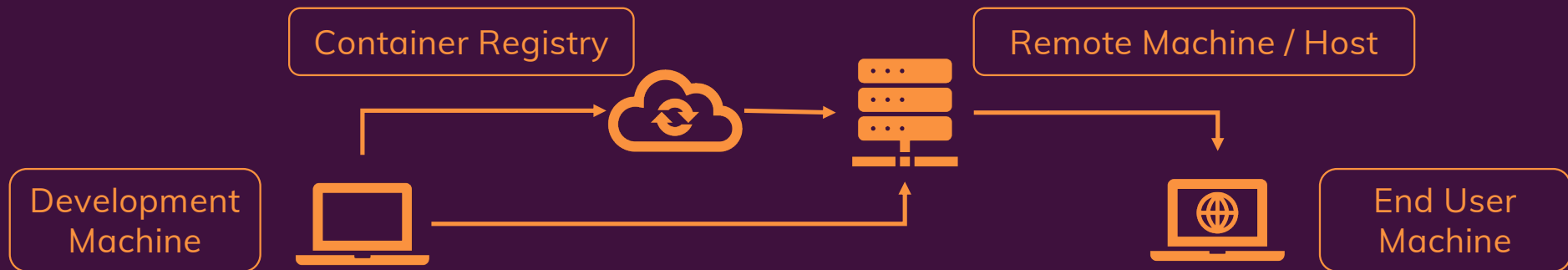
A Basic First Example: Standalone NodeJS App

Just NodeJS, no database, nothing else

1 Image & Container

Possible Deployment
Approach

Install Docker on a remote host (e.g. via SSH), push and
pull image, run container based on image on remote host



Hosting Providers

There are hundreds and thousands of Docker-supporting hosting providers out there!



Amazon Web Services



Microsoft Azure



Google Cloud

Example: Deploy to AWS EC2

AWS EC2 is a service that allows you to spin up and manage your own remote machines

1

Create and launch EC2 instance, VPC and security group

2

Configure security group to expose all required ports to WWW

3

Connect to instance (SSH), install Docker and run container

Deploy Source Code vs Image

Option 1: Deploy Source

Build image on remote machine

Push source code to remote machine,
run `docker build` and then `docker run`

Unnecessary complexity

Option 2: Deploy Built Image

Build image before deployment (e.g. on local machine)

Just execute `docker run`

Avoid unnecessary remote server work

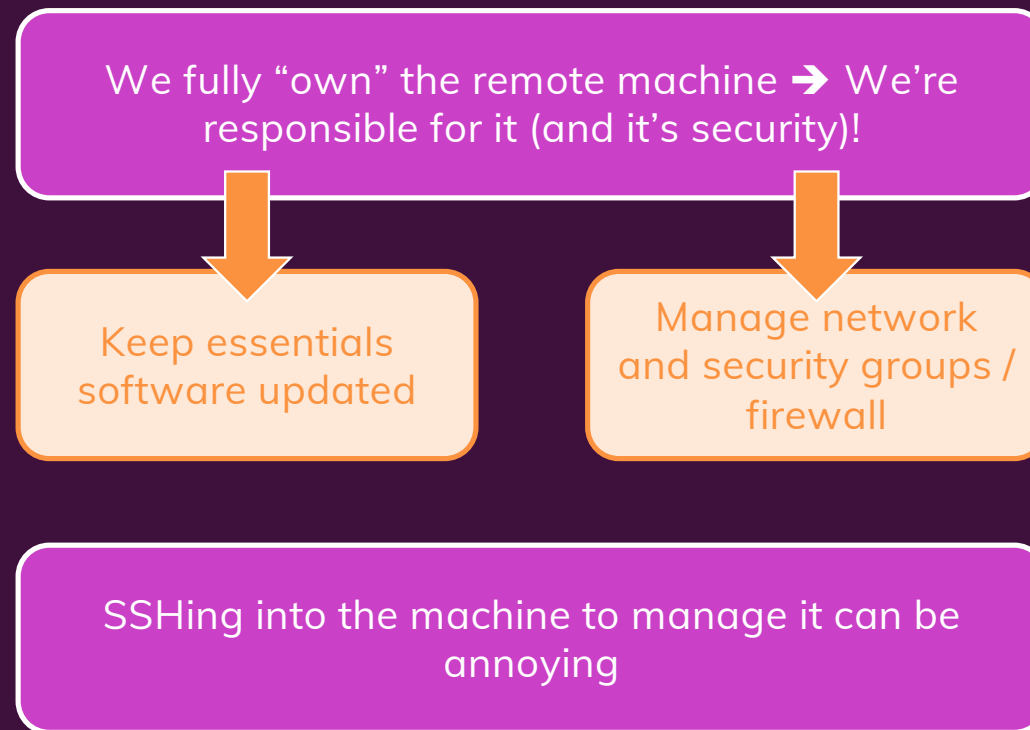
Docker Is Awesome!

Only Docker needs to be installed (no other runtimes or tools!)

Uploading our “code” is very easy

It's the exact same app and environment as on our machine

“Do-it-yourself” Approach – Disadvantages



A Managed / Automated Approach



Your Own Remote Machines
e.g. AWS EC2

You need to create them, manage them, keep them updated, monitor them, scale them etc.

Great if you're an experienced admin / cloud expert



Managed Remote Machines
e.g. AWS ECS

Creation, management, updating is handled automatically, monitoring and scaling is simplified

Great if you simply want to deploy your app / containers

A Note about Databases

You can absolutely manage your own Database containers

but ...



Scaling & managing **availability** can be challenging



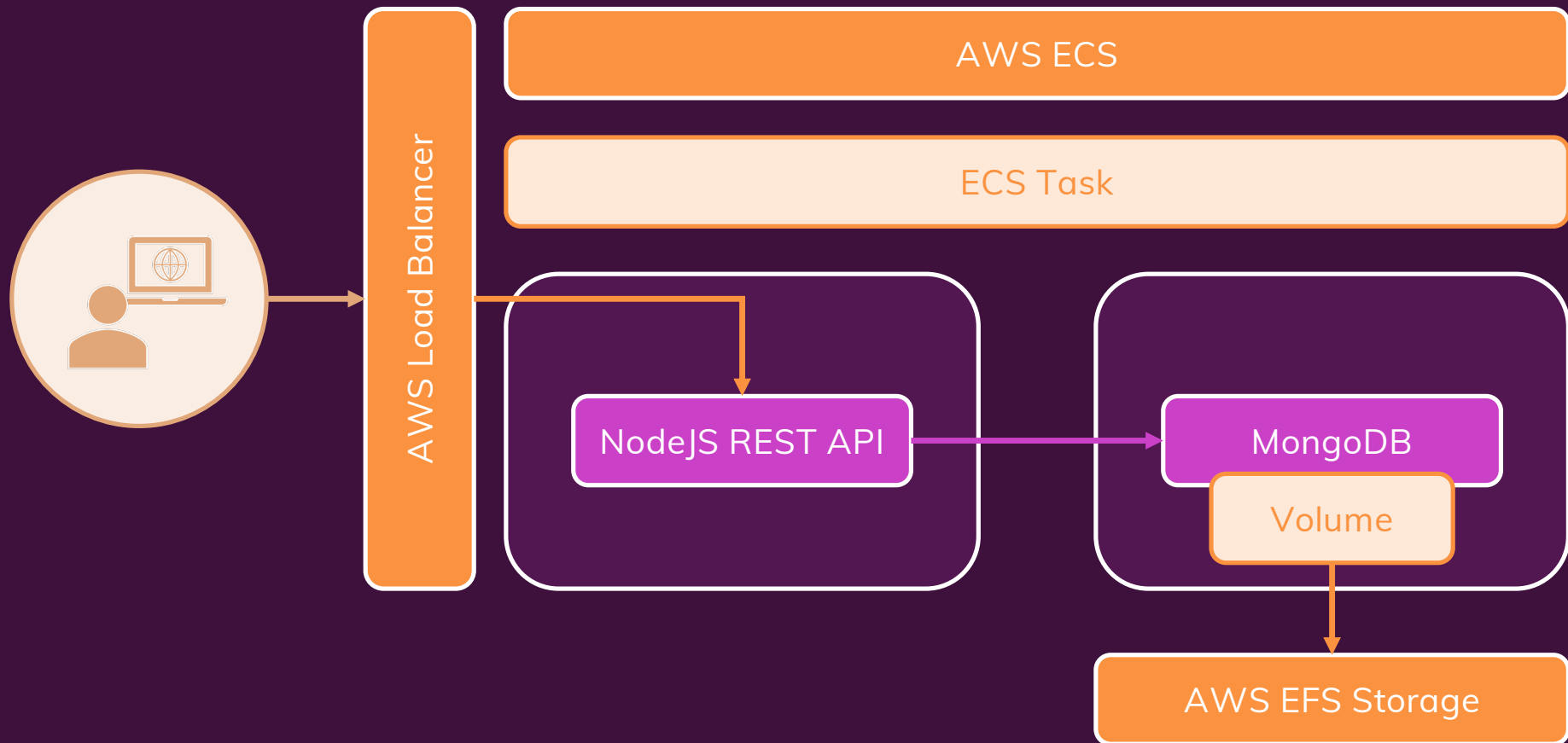
Performance (also during traffic spikes) could be bad



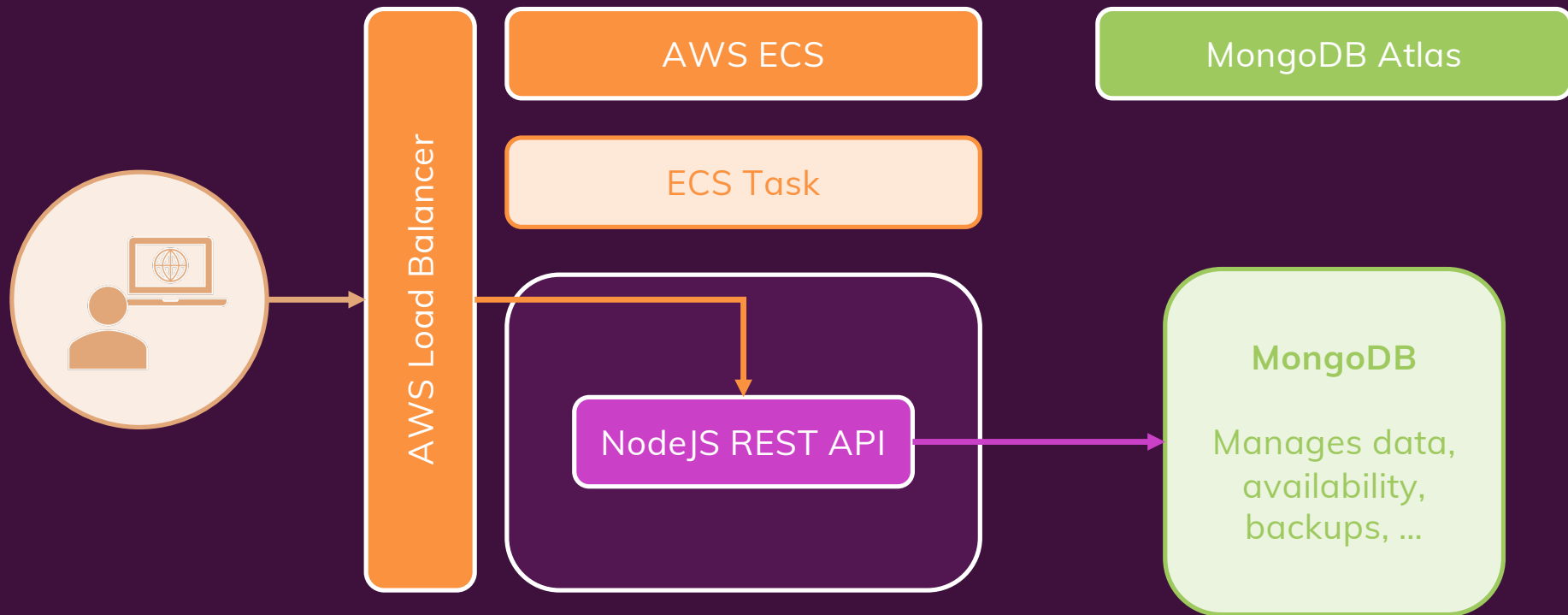
Taking care about **backups** and **security** can be challenging

Consider using a **managed Database service** (e.g. AWS RDS, MongoDB Atlas, ...)

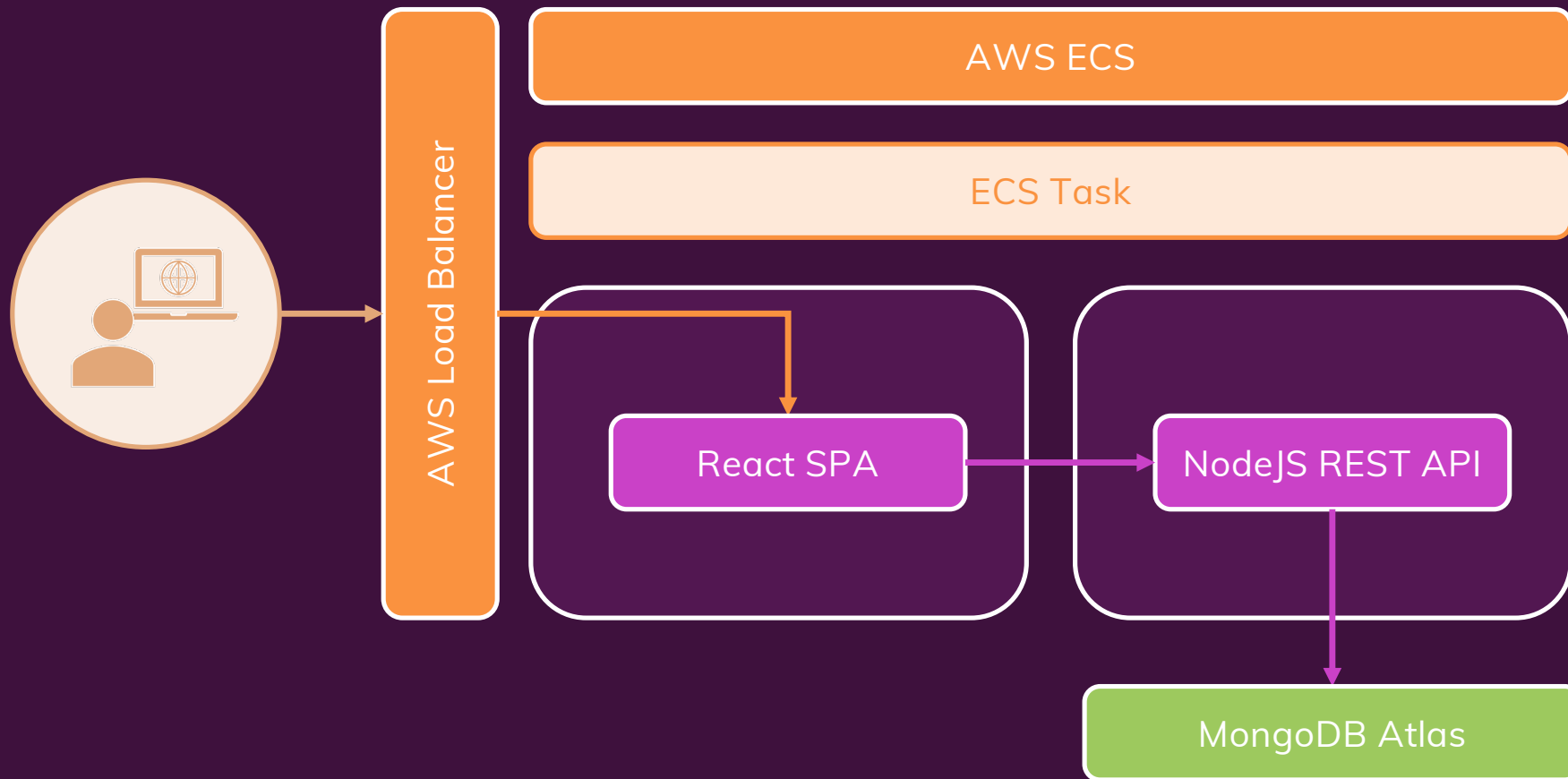
Our Current App Architecture



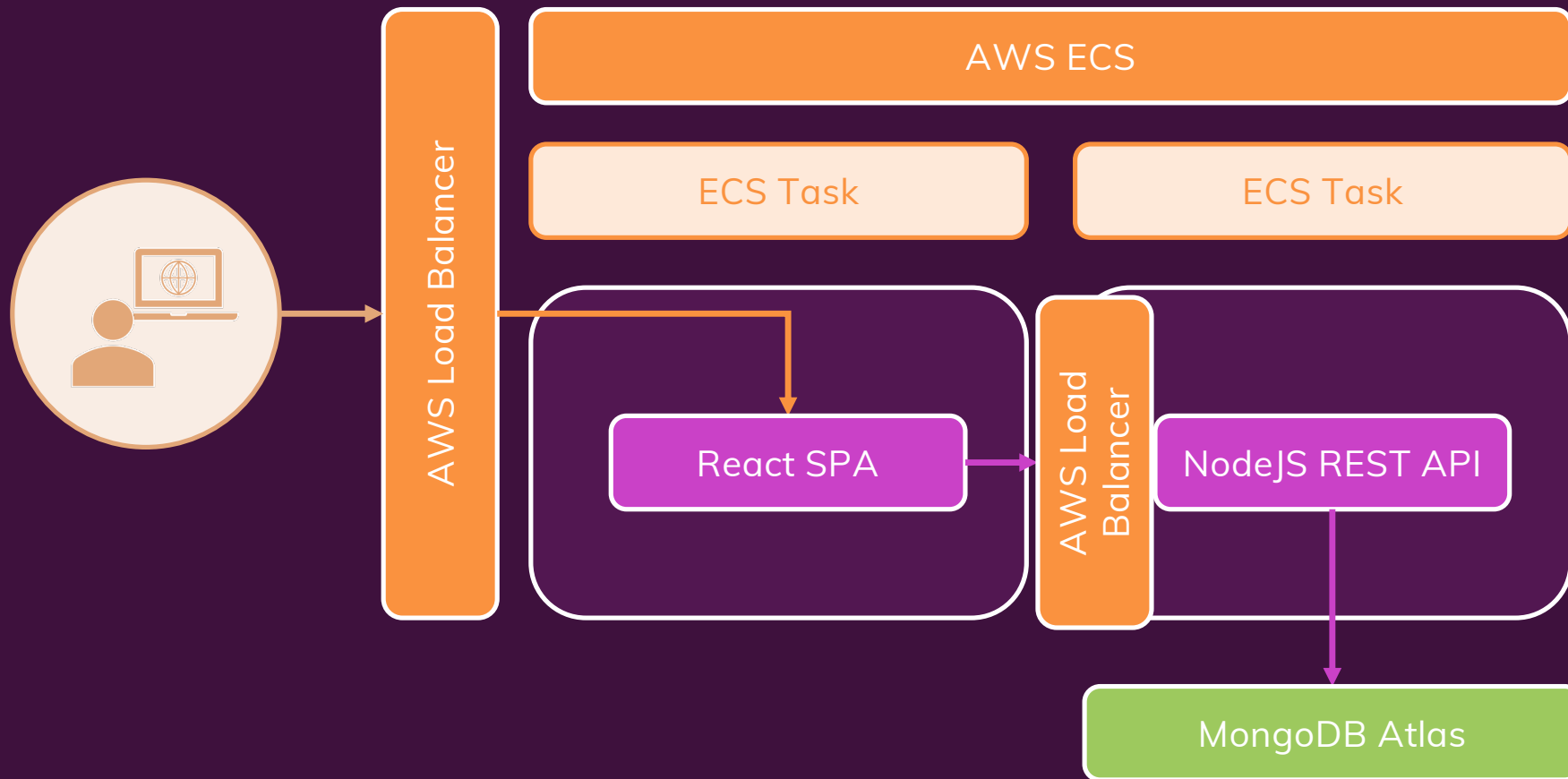
Our New App Architecture



Our Final App Architecture

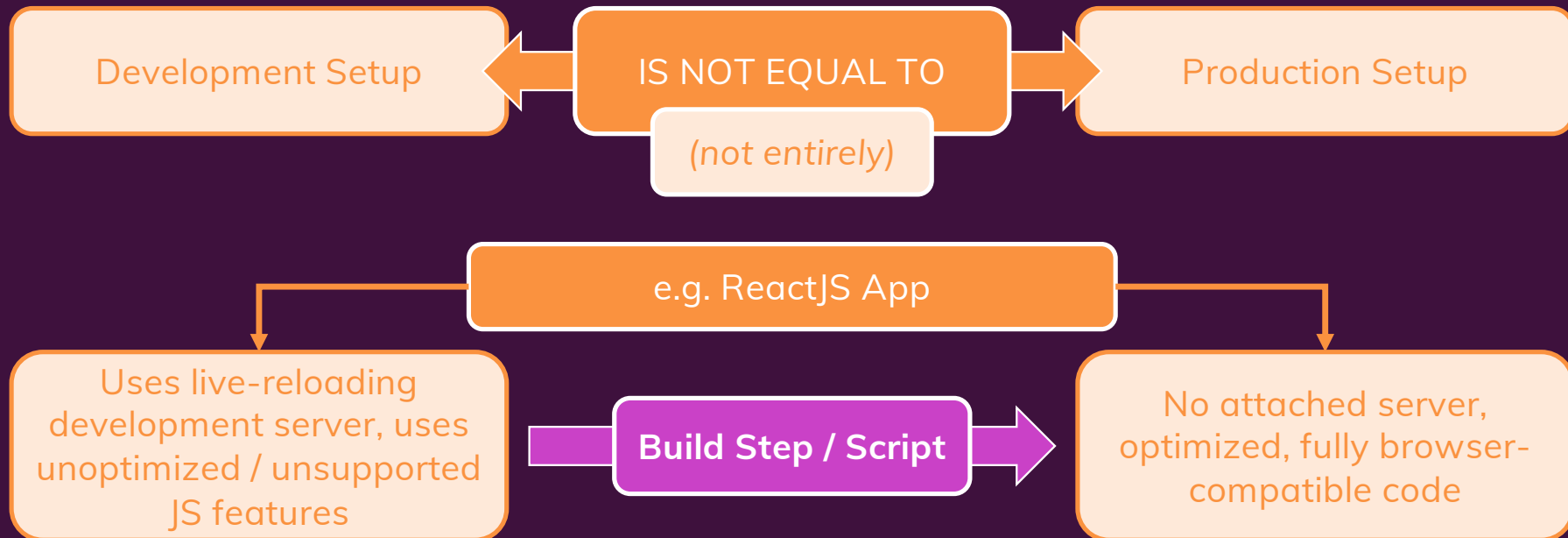


Our Final App Architecture



Apps with Development Servers & Build Steps

Some apps / projects require a build step
e.g. optimization script that needs to be executed **AFTER**
development but **BEFORE** deployment



Introducing Multi-Stage Builds

One Dockerfile, Multiple Build / Setup Steps (“Stages”)

Stages can **copy results** (created files and folders) **from each other**



You can **either build the complete image or select individual stages**

From AWS To Other Providers

AWS was just the example provider in this section!



**Manual installation of
Docker + management of the
underlying server**

Possible with any provider
that gives you fully controlled
remote hosts / instances



**Managed Container/ Docker
services like ECS**

Many cloud providers offer
managed Docker / Container
services

Can We Do Better?

Containers allow us to **encapsulate app code and environment** for both **development and production**

If we DON'T manage Docker and remote machines manually, we must work with the **tools and rules imposed by the managed service**

Thinking about production forces us to build containers / app code with **more scenarios** in mind (e.g. multi-stage builds)

Different cloud providers == Different rules

Depending on provider, **features** like load balancing might be **challenging** to implement



Kubernetes