SOFTWARE ENGINEERING PATTERNS

# 12 FACTOR APP

# ABOUT ME

- Docker, 12 Factor Apps, Massive Cloud Scaling is a bit new to me.

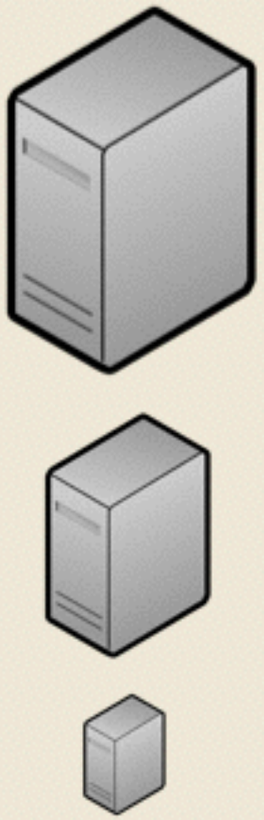
- Discovered this over the last 3 months through researching new workflows with Docker.

- Know a thing or two but am no Linux guru.

# 12 FACTOR APP?

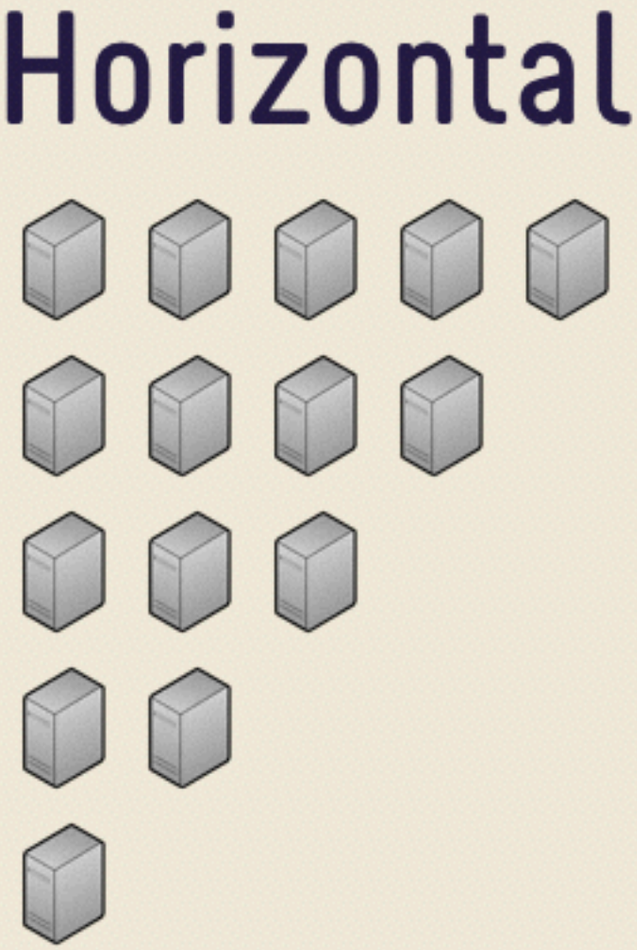
# CONCEPTS

# CLOUD SCALABILITY



Vertical

vs.



Horizontal

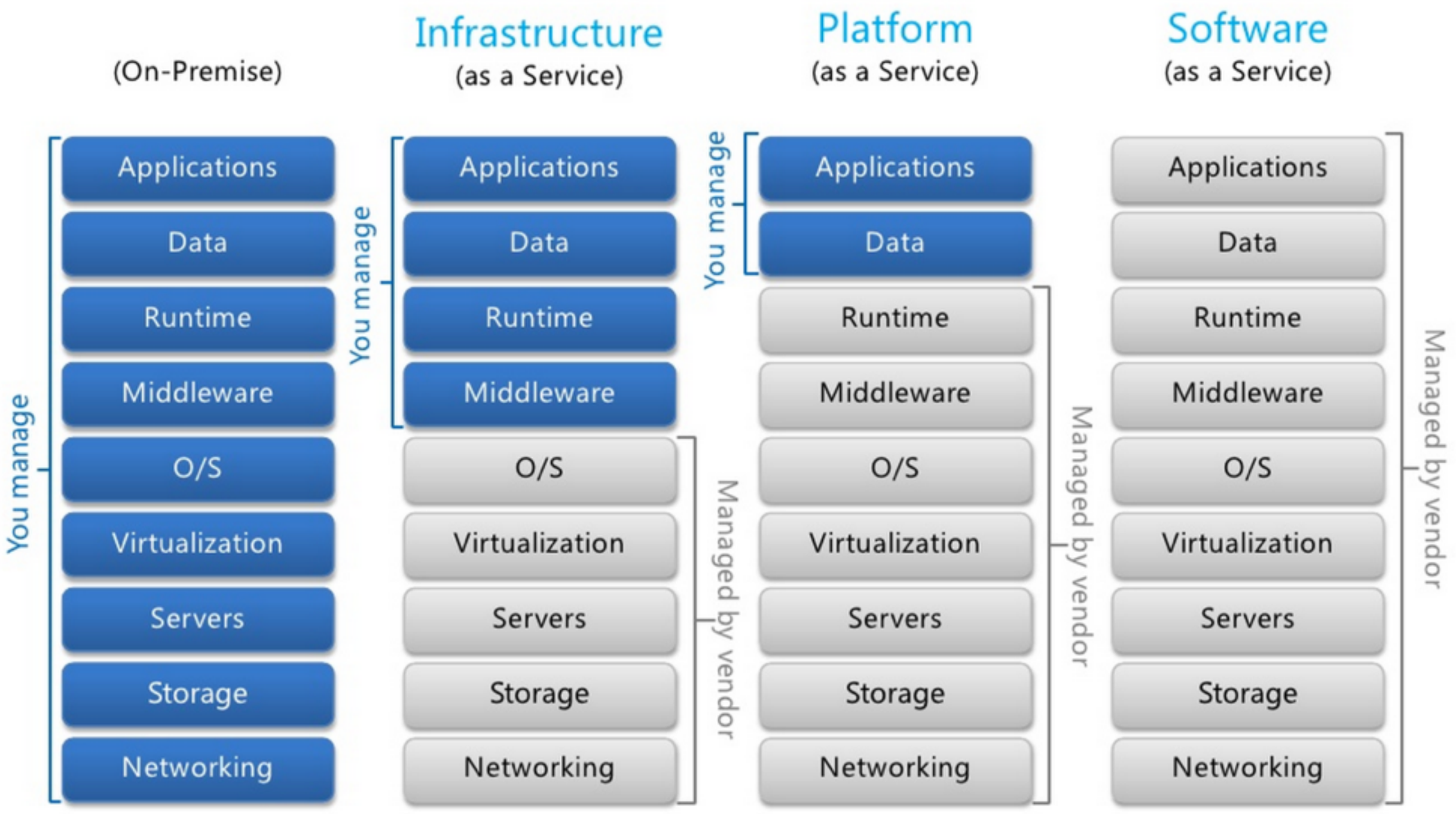
CLOUD SCALABILITY =

HORIZONTAL SCALABILITY

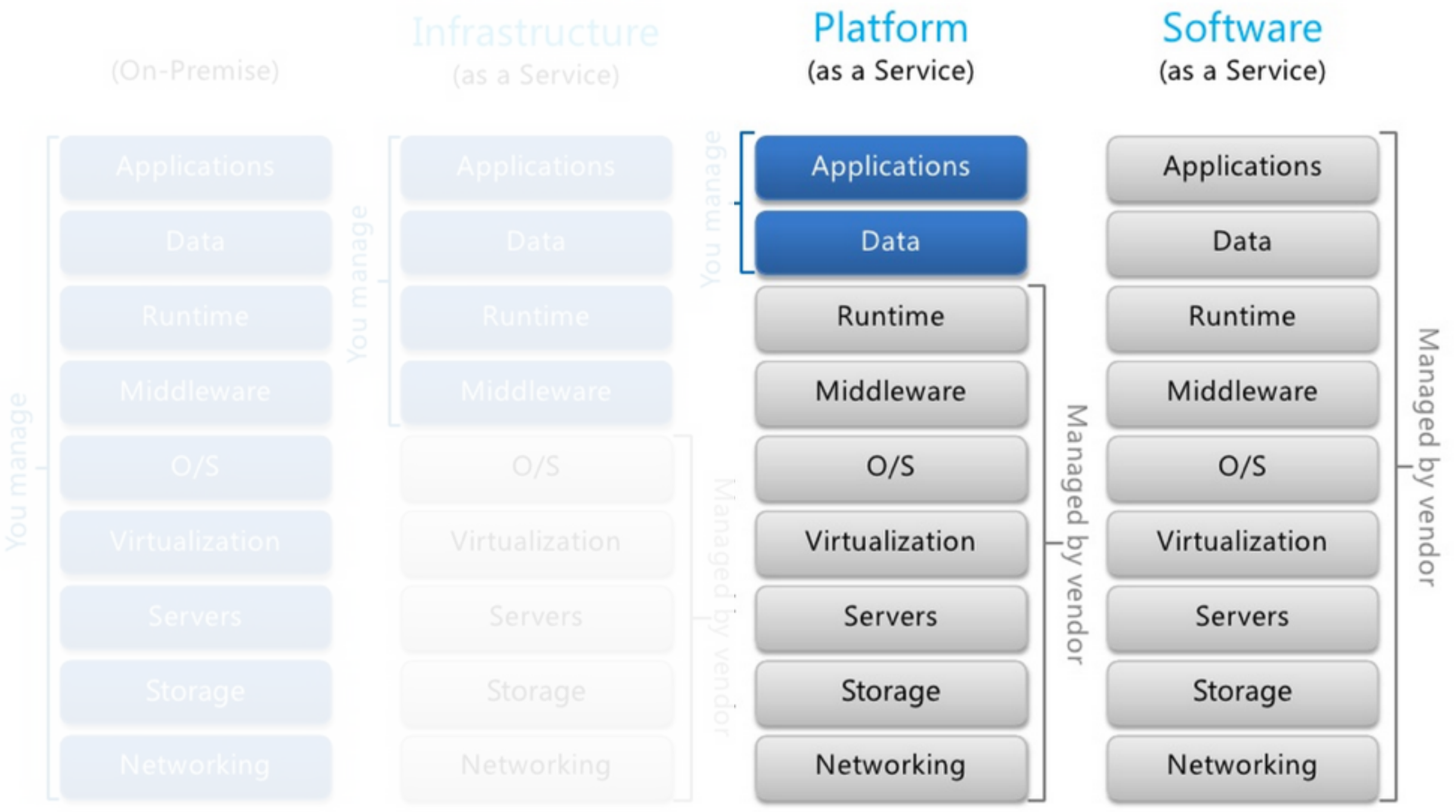
# ...AAS MODELS

## ... AS A SERVICE MODELS

# AS A SERVICE MODELS



# AS A SERVICE MODELS



http://12factor.net

–12 FACTOR APP MANIFESTO

# WHAT ARE THE 12 FACTORS?

- They are guidelines for writing scalable SaaS apps

# WHAT IS A 12 FACTOR APP?

http://12factor.net

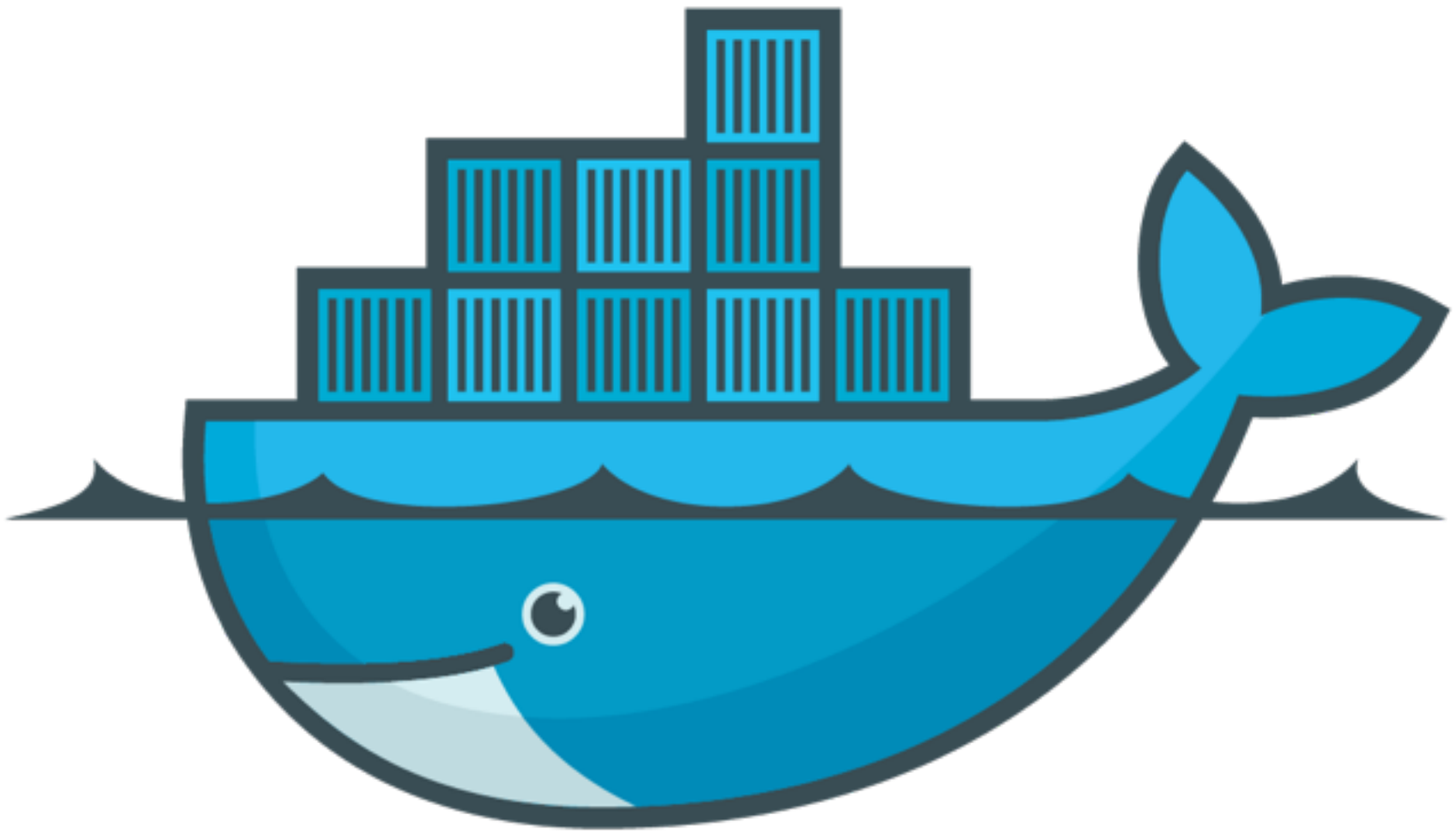
- Manifesto written around 2012
- by **Adam Wiggins**
  - **Heroku** ( Early cloud PaaS )

# WHY SHOULD I CARE?

# WHY SHOULD I CARE?

- Many start up apps get written quickly
- Limited scalability
- Tangled dependencies
- Need a rewrite to scale easily

OH AND...



docker

*If you use docker in production you probably*

***already*** *have a 12 factor app!*

# I WISH I KNEW THIS BEFORE I STARTED WITH DOCKER!

LESSONS LEARNT

# THE FACTORS

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# THE FACTORS

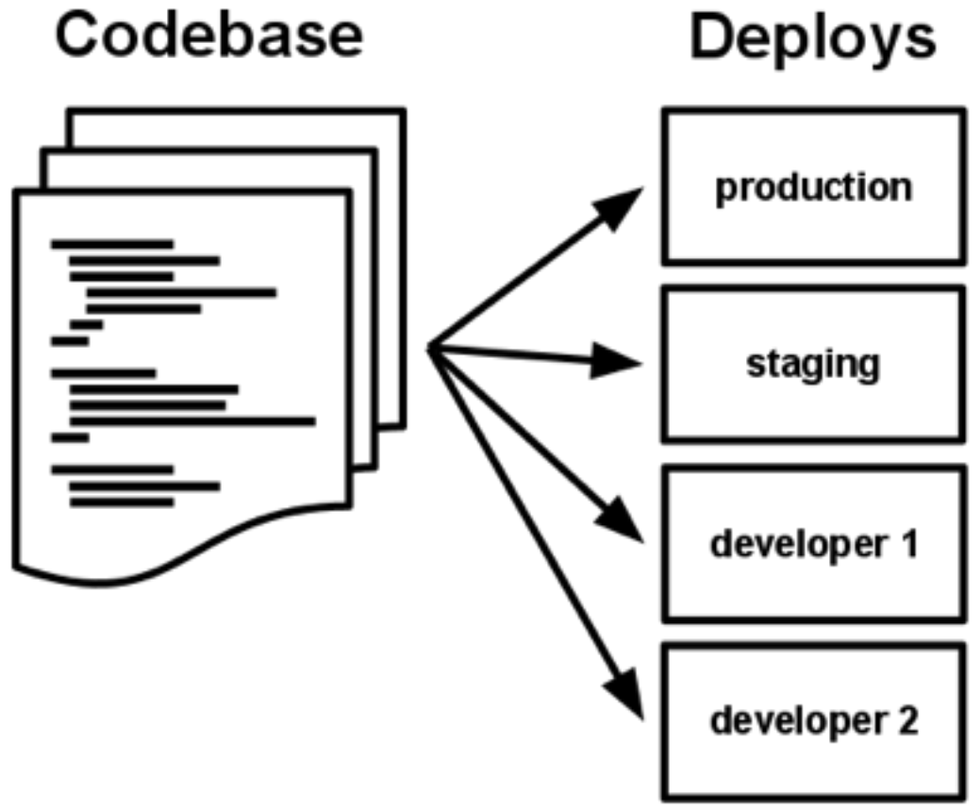
1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

# 1. CODEBASE

ONE CODEBASE MANY DEPLOYS

# CODEBASE

- codebase = repo
- **one** repo - **many** deploys
- app **!=** many repos
- many repos = distributed system



# 2. DEPENDENCIES

## EXPLICITLY DECLARE AND ISOLATE

# DEPENDENCIES

Explicitly declare and isolate dependencies

1. Declare **dependencies** in a **manifest**
2. Use **isolation** tools
3. **Specific** versions are important
4. Avoid **shelling** to unbundled system tools.

# DEPENDENCIES

Examples:

- Dependency Manifest = **Gemfile**
- Isolation tools = **bundle exec**

```
gem "redis-rails", "~> 3.2.3"
```

# 3. CONFIG

## STORE IN THE ENVIRONMENT

# CONFIG

- Config is the **specific** information required to host a deployment of your codebase
- Does not include things like routes etc.
- Mainly database **credentials**, **paths**, **resource urls** etc.

# CONFIG

Store config in the environment.

- Keep your config **outside** the app
- No config in git
- Open source test

# CONFIG

Use **environment vars**

Can you make your repo open source today?

```
ENV['DATABASE_URL']
```

```
production:
  adapter: mysql2
  database: <%= ENV['DB_ENV_MYSQL_DATABASE'] %>
  username: <%= ENV['DB_ENV_MYSQL_USER'] %>
  password: <%= ENV['DB_ENV_MYSQL_PASS'] %>
  port: <%= URI(ENV['DB_PORT']).port %>
  host: <%= URI(ENV['DB_PORT']).host %>
```

# 4. BACKING SERVICES

## AS ATTACHED RESOURCES

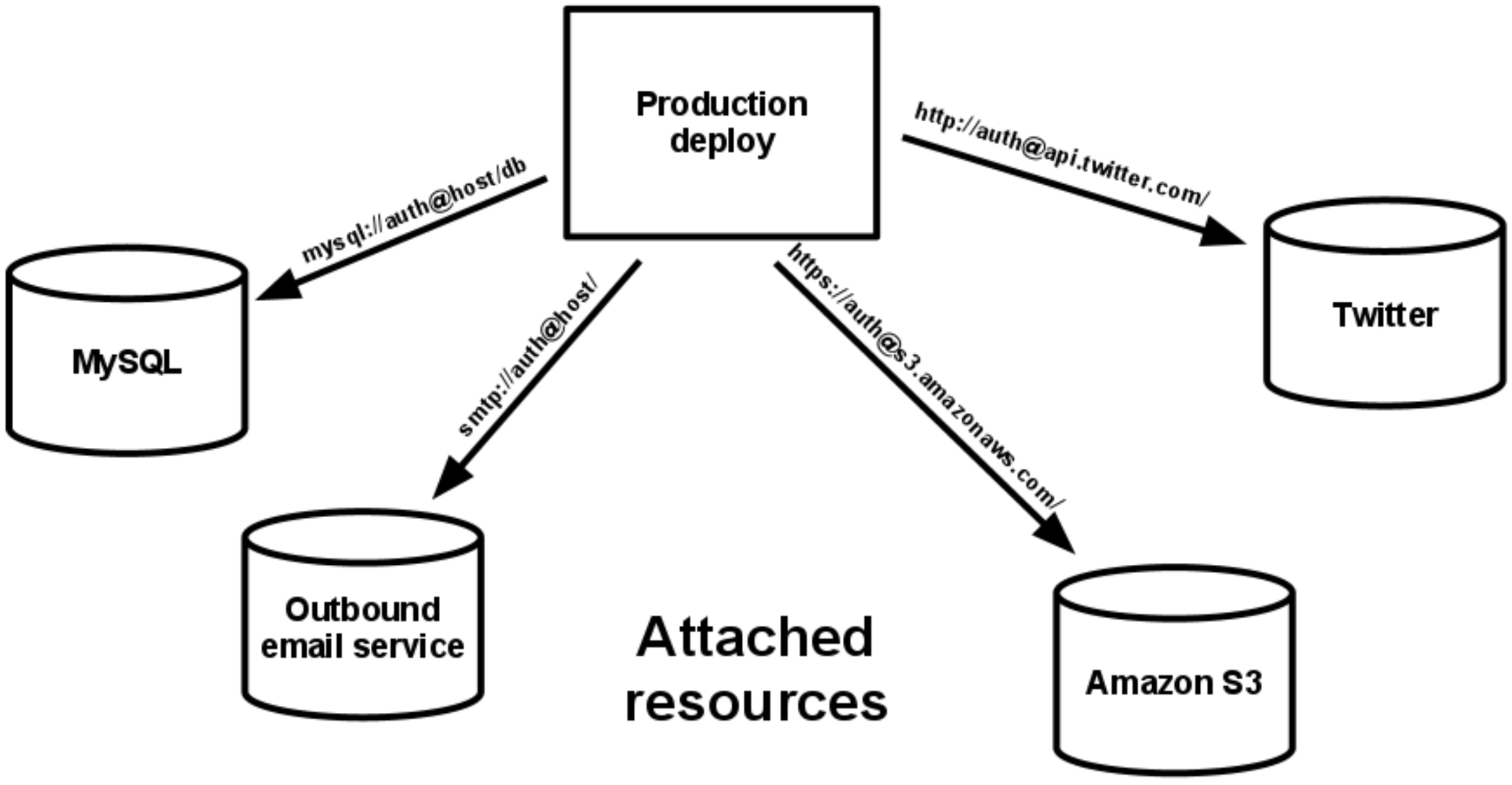
# 4. BACKING SERVICES

Treat backing services as attached resources

What's a backing service?

- **Datastore**
- **SMTP**
- **Caching systems**
- **Amazon S3**

Make no distinction between local and third party services


Production deploy
mysql://auth@host/db
http://auth@api.twitter.com/
MySQL
Twitter
smtp://auth@host/
https://auth@s3.amazonaws.com/
Outbound email service
Amazon S3
Attached resources

# 5. BUILD RELEASE RUN

STRICTLY SEPARATE BUILD & RUN STAGES

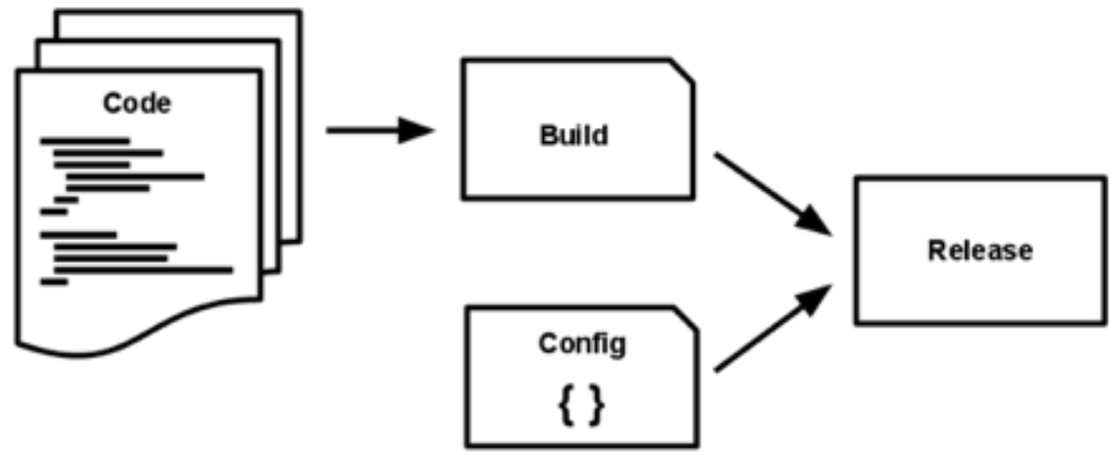
# BUILD, RELEASE, RUN

**BUILD** = codebase + dependencies + assets

**RELEASE** = **BUILD** + config

**RUN** = run process against **RELEASE**

# BUILD, RELEASE, RUN



# BUILD RELEASE RUN

- Strict **separation** between stages
- **Cannot change** code at **runtime**
- **Rollback** = just use the last release instead.
- Release has unique release **ID**

# 6. PROCESSES

EXECUTE THE APP AS ONE OR MORE STATELESS PROCESSES

# PROCESSES

- Does the **running** of the **release**
- Is **stateless**
- **Shares nothing** with other processes
- Uses **single transaction** only caches
- Session **db storage** …over sticky sessions
- Asset **pre-compilation** …over runtime calculation

# 7. PORT BINDING

## EXPORT SERVICES VIA PORT BINDING

# PORT BINDING

**http://192.168.123.45 : 5555**

The **contract** with the **executive environment** is binding to a **port** to serve **requests**.

# PORT BINDING

E-Commerce System:

**http:// 192.168.123.45 : 5555**

Booking System:

**http:// 23.123.65.48 : 3306**

# 8. CONCURRENCY

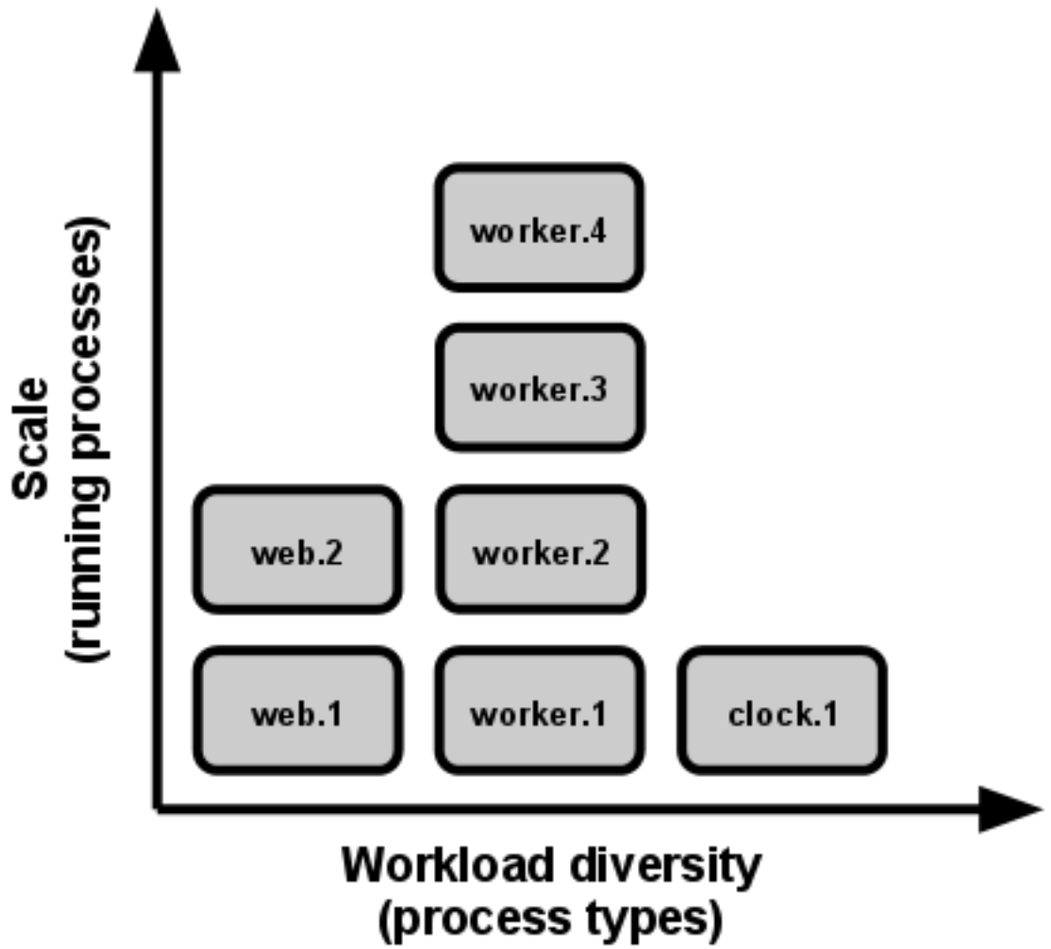
## SCALE OUT VIA THE PROCESS MODEL

# CONCURRENCY

- **Scale out** via the **process model**
- Processes are **first class** citizens.
- Assign work to a process **type** (web, db, worker etc.)
- Can then handle **own multiplexing**
- Processes **don't daemonize** or write PID files.
- Instead, use system **process manager**

# CONCURRENCY

Scale out by running multiple processes of different types



# 9. DISPOSABILITY

## MAXIMISE ROBUSTNESS WITH FAST STARTUP AND GRACEFUL SHUTDOWN

# DISPOSABILITY

Processes are **disposable**

- Start **quickly**
- Shut down **gracefully**
- Be robust against **sudden death**

# DISPOSABILITY

Example: Worker Process

- Return the current job to the job queue
- All jobs are **reentrant**
- Or at least **idempotent**

# 10. DEV PROD PARITY

KEEP DEVELOPMENT, STAGING, AND PRODUCTION AS SIMILAR AS POSSIBLE

# DEV PROD PARITY

**Gaps** exist between development and production:

- **Time** (days to push)
- **Personnel** (dev vs ops)
- **Tools**

# DEV PROD PARITY

Need to **shorten the gaps!**

- CI & deploy **ASAP** after writing code
- Get Developers **involved** in Operations
- Environments should be as **similar** as possible:
  - Eg. **Resist the urge** to use **different** backing services between development and production

# 11. LOGS

## TREAT LOGS AS EVENT STREAMS

# LOGS

Within your app treat logs as **event streams**

- **Don’t** route or store **logs in files**
- **Stream** to stout instead and be captured and handled by the **environment**

# 12. ADMIN PROCESSES

RUN ADMIN/MANAGEMENT TASKS AS ONE-OFF PROCESSES

# ADMIN PROCESSES

One-off admin tasks include:

- Database **migrations**
- **Console**
- One-time **scripts**

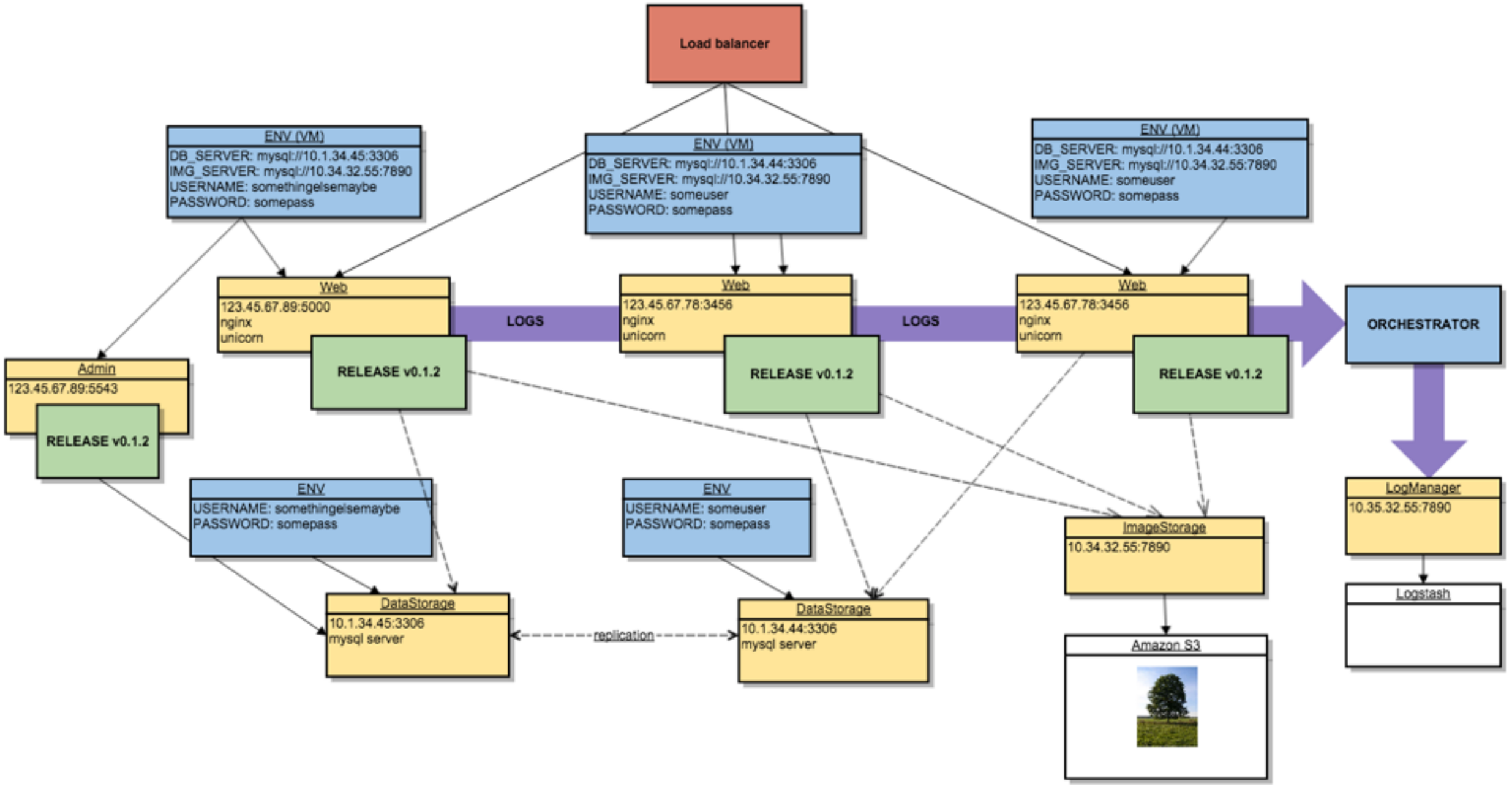
# ADMIN PROCESSES

- Run as **separate** process
- Run against the **same release**
- **Admin code** ships with **app code**

# THE FACTORS

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

WHAT MIGHT THIS ALL LOOK LIKE?


Load balancer
ENV (VM)
DB_SERVER: mysql://10.1.34.45:3306
IMG_SERVER: mysql://10.34.32.55:7890
USERNAME: somethingelsemaybe
PASSWORD: somepass
ENV (VM)
DB_SERVER: mysql://10.1.34.44:3306
IMG_SERVER: mysql://10.34.32.55:7890
USERNAME: someuser
PASSWORD: somepass
ENV (VM)
DB_SERVER: mysql://10.1.34.44:3306
IMG_SERVER: mysql://10.34.32.55:7890
USERNAME: someuser
PASSWORD: somepass
Web
123.45.67.89:5000
nginx
unicorn
RELEASE v0.1.2
LOGS
Web
123.45.67.78:3456
nginx
unicorn
RELEASE v0.1.2
LOGS
Web
123.45.67.78:3456
nginx
unicorn
RELEASE v0.1.2
ORCHESTRATOR
Admin
123.45.67.89:5543
RELEASE v0.1.2
ENV
USERNAME: somethingelsemaybe
PASSWORD: somepass
ENV
USERNAME: someuser
PASSWORD: somepass
ImageStorage
10.34.32.55:7890
LogManager
10.35.32.55:7890
DataStorage
10.1.34.45:3306
mysql server
replication
DataStorage
10.1.34.44:3306
mysql server
Amazon S3
Logstash

# THANK YOU

QUESTIONS?