What is the 12 Factor App ?

YouTube Link: https://www.youtube.com/watch?v=l0hmRmMsGdQ

Issues faced while building applications in the traditional way:

- Time taken to obtain a server and host it on the internet
- If server goes away, our application is affected - session information, can't scale since server machine is fixed

Modern application needs to be **free of the underlying infra** - is has to be **portable**, and be **able to run on different environments** **without having the change the source code**

Modern applications need to be

- portable
- loosely couple with infra
- minimal divergence between environments
- and easily scalable
- Suitable to be deployed on modern cloud platforms

Engineers from Heroku came up with **12 principles** to follow to ensure applications have the following characteristics

I
Codebase

II
Dependencies

III
Config

IV
Backing Services

V
Build, release, run

VI
Processes

VII
Port Binding

VIII
Concurrency

IX
Disposability

X
Dev/prod parity

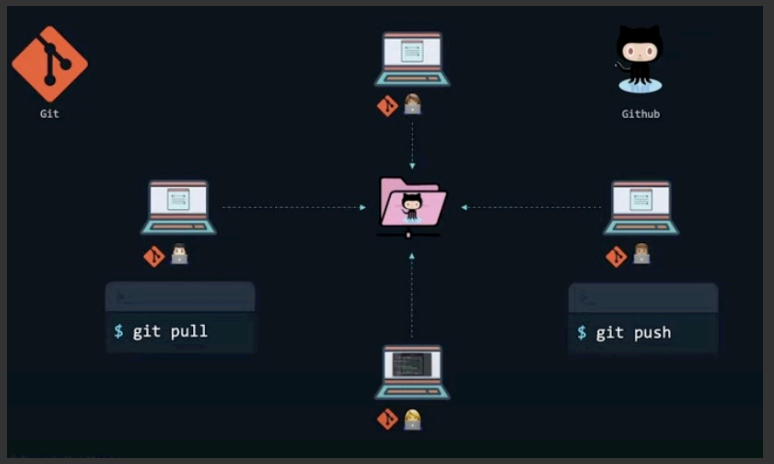
XI
Logs

XII
Admin Processes

1. Having a single code base

Initially we just build our application, but as more users come up and we hire more developers to work on the application, we need to ensure that we manage the different versions of the code base and need to have it all in a single location.

This is where Git Comes into the picture and platforms like GitHub help us with collaboration and code management.

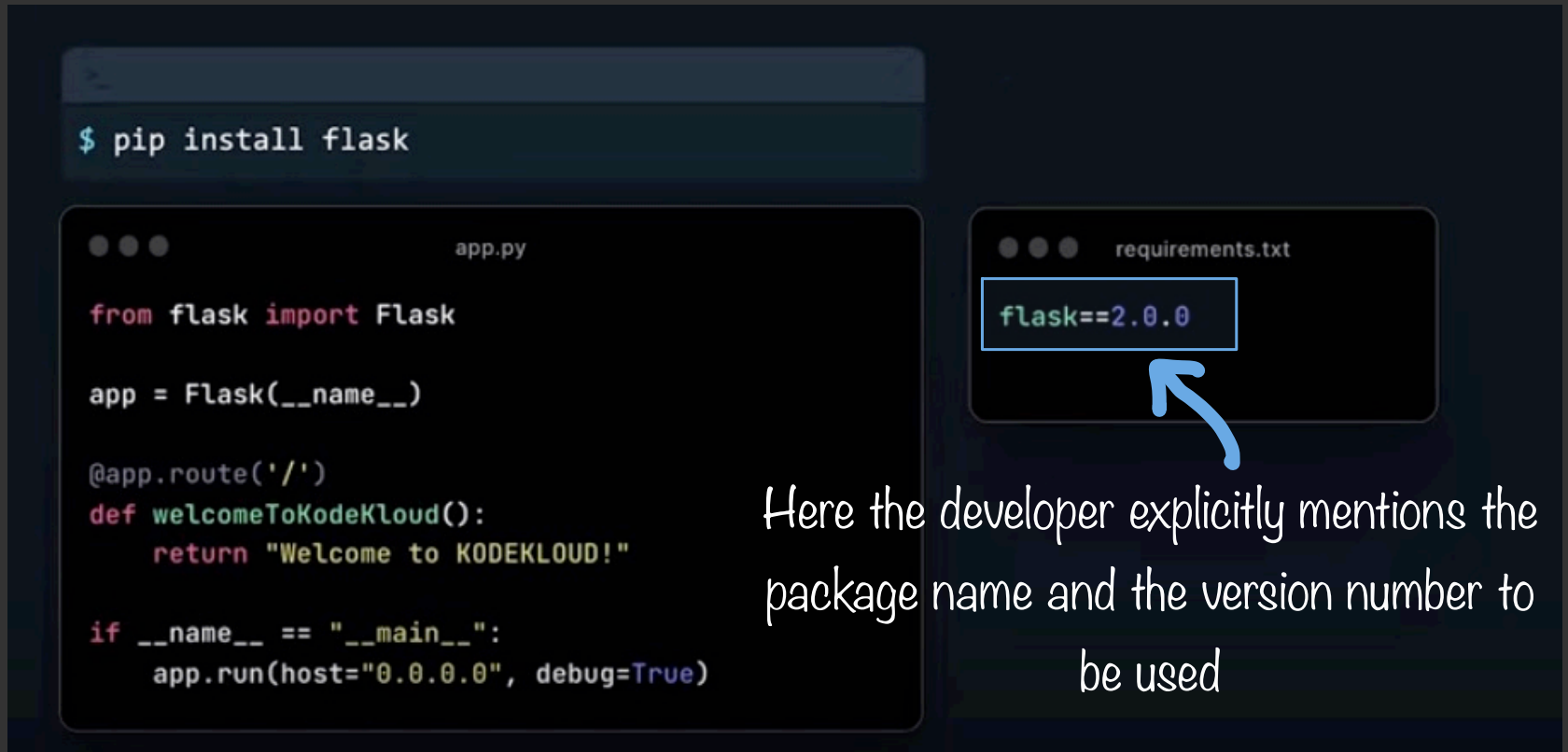


Now when our application starts growing and we start adding other features, like a payment service, a delivery service etc.. we must **not save all of those code files in the same git repo**. This is a violation of the factor 1 of the 12 factor APP. We need to maintain separate code bases.

However, we can have the multiple pipelines for dev, staging and prod for each app within the same repo.

2. Explicitly declare and isolate dependencies

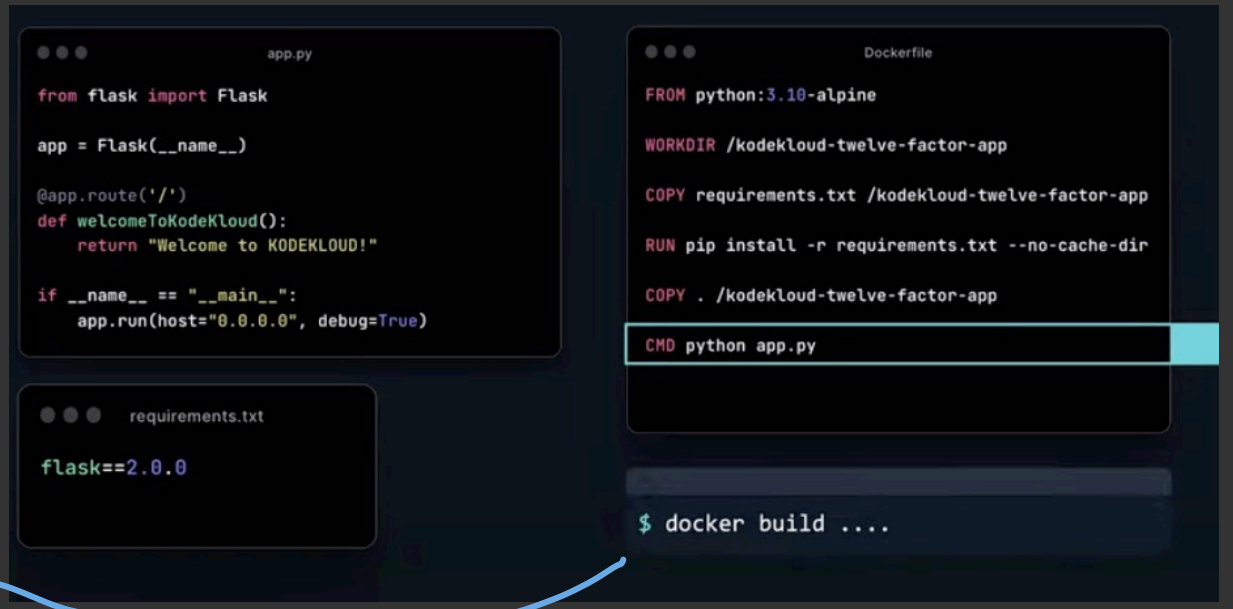
For example a simple flask application has the flask library as a dependency. But as per the 12 factor, **we must not assume that this library will be available in the environment in which we will be running the app in production**. We must explicitly ensure that there are mechanisms to set up this dependency in all environments.



2. Explicitly declare and isolate dependencies

While it might be a good practice to declare a requirements file for your main application, it is also important to do so for all the different services that your main application depends on. Similarly, to ensure uninterrupted working and prevent issues during package installation, it is key to isolate the environments and dependencies for each of the sub applications that make up the whole application.

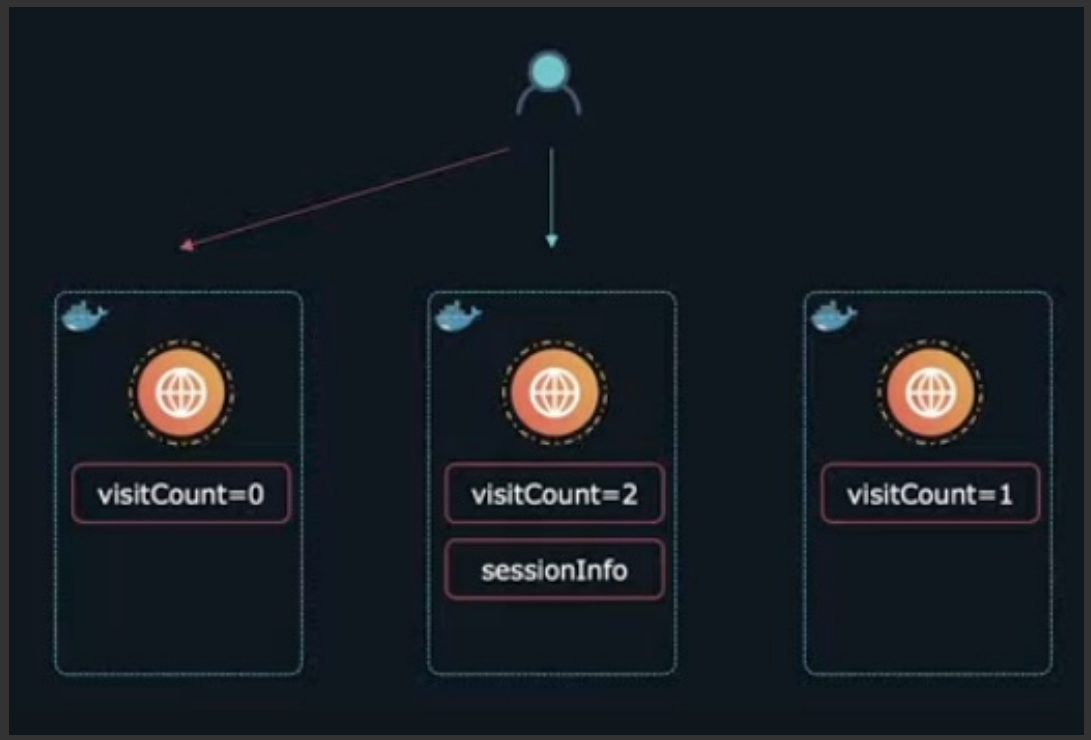
This is ' where tools like docker help us



3. Concurrency

While it is great to have one instance serving a few customers, as the customer base grows up, we will need to scale horizontally to serve all requests. This is where **building our application keeping concurrency in mind will help us with scaling.**

4. Model processes to be stateless



In this example we see that `visitCount` and `sessionInfo` is stored individually in each instances' state. This will become a problem if the instance crashes and the user's request is to be redirected to another instance.

Hence it is **crucial to keep the individual applications stateless** so any user's **request can be handled by any instance**, this helps with seamless scaling and capacity management. All state based data should be either moved to an external database or a caching service like redis

In this case, we are using redis as an attached service. But the key point to note here is that an attached service should be replaceable at any time and it should just start working with our application.

5. Backing services must be see

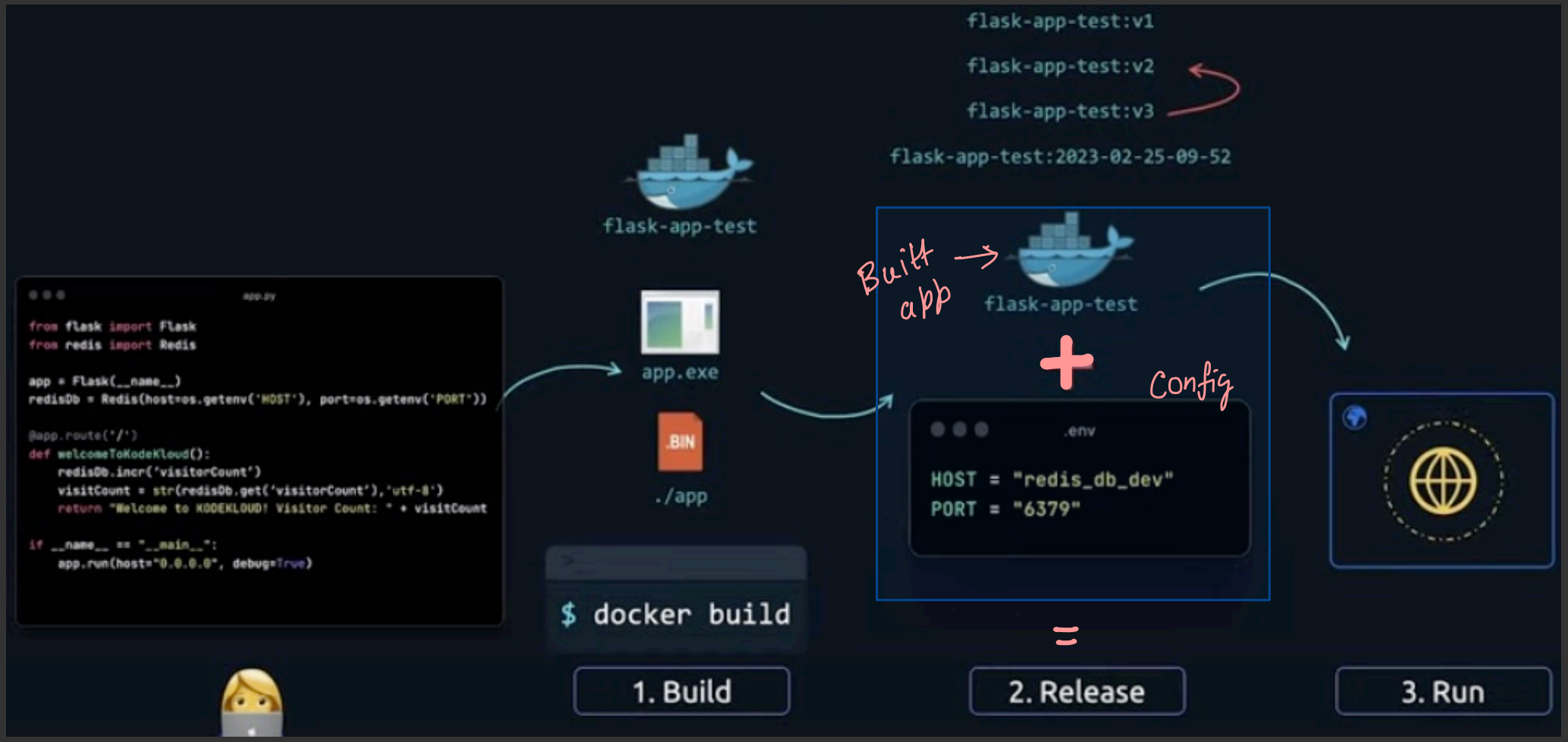
Or if one type of persistence does not work we must be able to switch to another type of persistence or another version of it without having to do any code change.

5. Backing services must be seen as attached services

6. Config needs to always be kept separate from the code base

This helps with rapid testing and deployment in different environments without any code change. Additionally this let's us open source the code at any time without worrying about sensitive data being exposed.

7. Strict, Build Release and Run Phase Separation



- Built code plus config is the release
- Having a strict separation helps us with rolling back changes when needed

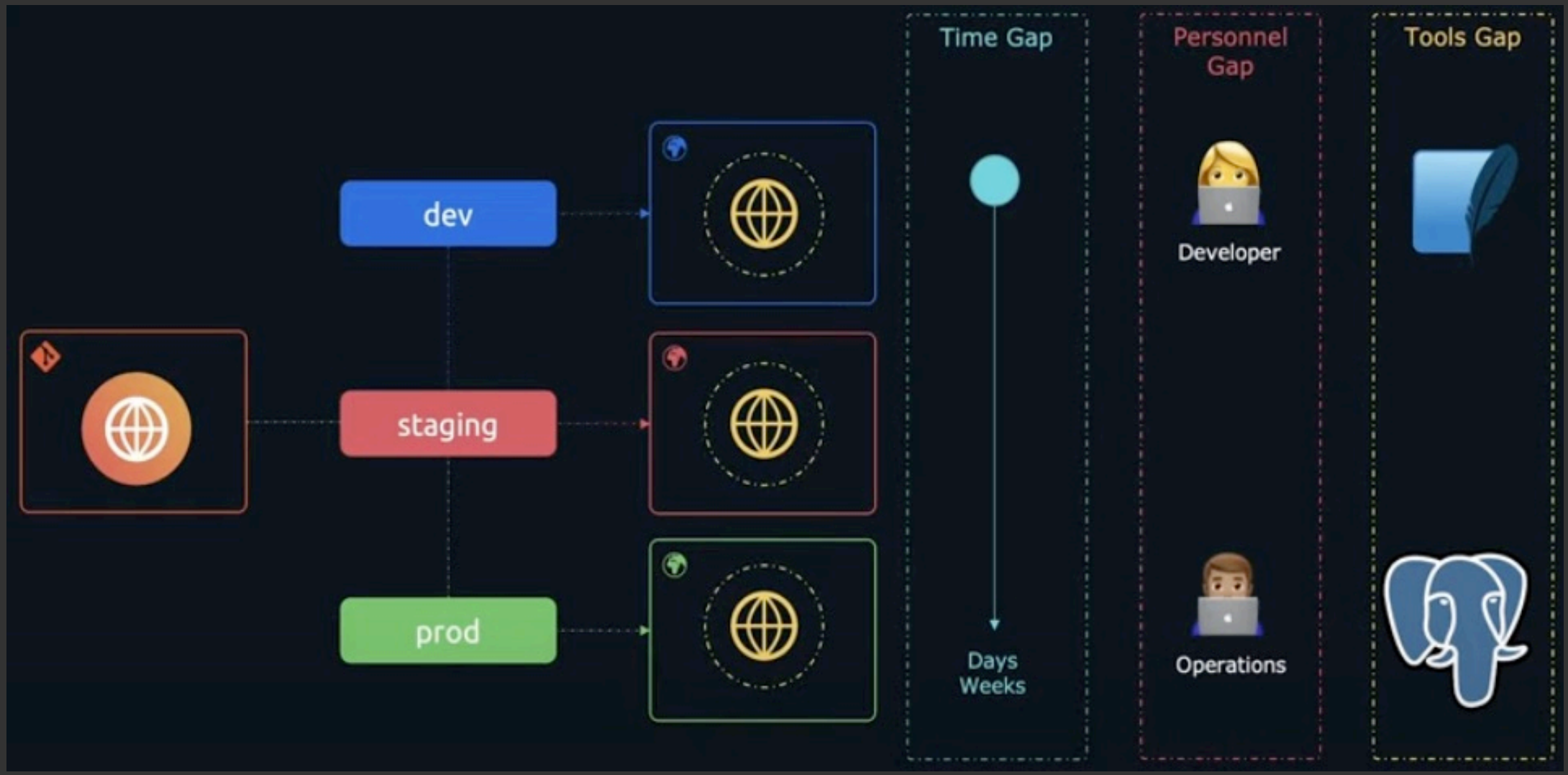
8. Bound to a port

- Unlike traditional applications the 12 factor app does not wait for specific services to become available. It picks up a port binds to it and starts listening for requests / starts serving on that port.

9. Disposability:

- The 12 factor app's processes can be started or stopped at a moment's notice.
- This helps with fast scaling and graceful shutdown whenever they receive a SIGTERM signal from the process manager
- A graceful shutdown is important to ensure users who are expecting a response, don't get impacted and there are no data leaks from processes that were running during the instance termination

10. Dev Prod Parity



- Designed for **continuous deployment** by keeping the gap between development and production small
- Developer resists the urge to use different **backing services between dev and prod**

11. Logs are treated as a continuous stream of events

- In cases where logs are written to a local log file, or just present in the terminal where the app runs, they get lost once the instance goes down
- While there are some instances of using centralized logging is encouraged, tightly coupling the logging solution to the app itself is discouraged - meaning having the logging library and the url within the application.

X

app.py

```
from fluent import sender

# for remote fluent
logger = sender.FluentSender('app', host='host', port=242

# Use current time
logger.emit('follow', {'from': 'userA', 'to': 'userB'})
```

12. Admin Process Principles

- Admin processes must be **run separately** as a one off process
- They must be run in an system that is **identical to the production environment**
- They should be automated scalable and reproducable