**\*\*\* 12 Factorial App \*\*\***

**Portability:**

Our application To be available all the time even in the time of server patching, adding resources, or scaling, Our application should be breakfree from underlying infrastructure where we can host this application anywhere we want ( on-prem, aws, azure, gcp )

-> to run our application on any env with out changing the code is called portability.

-> Modern applications should be portable and not tightly coupled to underlying infrastructure and should be suitable to modern cloud platforms.

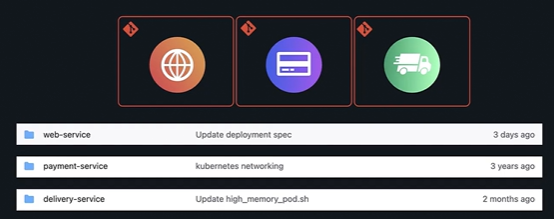
-> our application should be scalable, continuous deployment, suitable to modern cloud platforms and portable. To achive this our applications should be developed with certain principles which is

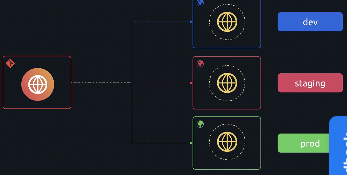
**12-factor app.**

( <https://12factor.net/> )

**1) Codebase**

One code base for whole application. No multiple applications or services in a single codebase, it’s a 12-facor app violation. For each code base we may have multiple deployments like dev, staging, prod.



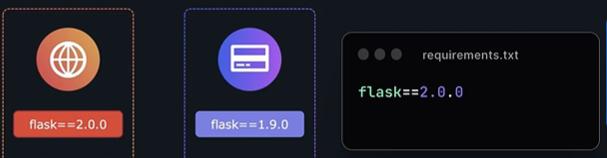


**2) Dependencies**

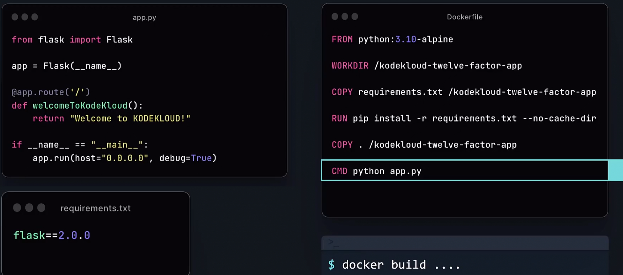
**-> 12-factor** app does not depend on systems where we run the app. So explicitly and isolate the dependencies.

Keep all the dependencies required for your application in a separate file like requirements.txt file and OPS team can download it with actual name and version. and do not link up with the codebase.

-> isolating the dependencies are like keeping the separate dependency file for each application in a isolate env where no application dependencies can interfear with other application.

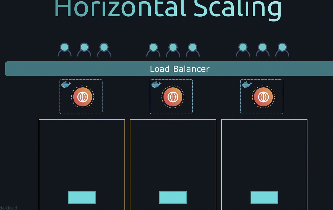


-> application which depends outside of python dependencies like system wide tools ( $curl ). For this we have docker runtime which application runs on self-contained env that is isolated from host system.



**3) Concurrency**

Since we run our application as an instance with all dependencies in a container, we can scale the instances horizontally instead of adding resources vertically whenever the traffic increases.



For this load balancer is configured to scale and distribute the load. For this to work we have to build our application as independent stateless app.

**4) Processes**

12-factor app are stateless and share nothing. Sticky sessions are violation for 12-factor app. If the process stores the session information of the user, this session data may reflect in another process when user visits next time. Even though load balancer can route to the same process to the sticky sessions it’s an violation. So, better to store all session data in a backend service called DB, cache etc.



Every process is just a process and it not storing any data in the process. For the next request the data should come from the attached resource like DB.

**5) Backing services**

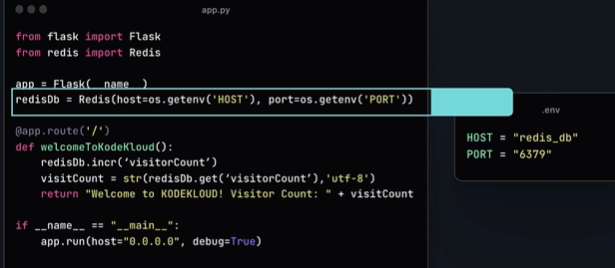
-> Have a loose coupling between applications and backend services like plsql, redis, postgresql etc.. Do not mix that service as a part of the application like configuring that in our application config file. Let it be a separate resource.



-> Backing services must be treated as attached resources. The switching of the services should be easy.

**6) Config**

To ensure our environment configuration are separate from our application code. Store the env variables and values in a separate file instead of putting in code. And we fetch those values from the file. Should not mention any of the configuration inside the code. By this even if we change the environment we no need to change the code, just changing env variables is enough.



12-factor app stores config in environment variables.

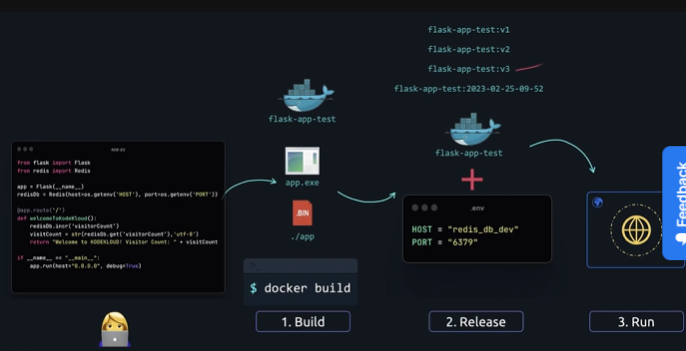
**7) Build, Release and Run**

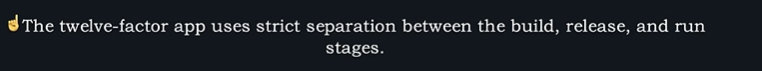
Instead of running the code directly in the production, build it and get the package of jar file and release it to the production to run. With this we get a release version of our code. The next time we want to make changes to the code, we can make changes in the build file but not in the release version. With this we achieve the history of releases and we always can go back to the previous versions and swtich.

**->** converting the code from text format to binary format or executable format is called building the code.

-> Once build is done, executable file along with config file which is suitable for your environment together becomes the release object. Every release should have a unique release ID. Any minor change in our code should create a new release.

-> In RUN phase, the exact same build is used to run in diff env which ensures we have same code base running in all diff env in a consistent fashion. Any minor change in the code will result in a new build process, new release and new deployment. By separating build and run phases, we can effectively manage our build artifacts by storing in a different location allows us to easily rollback to previous versions.





**8) port binding**

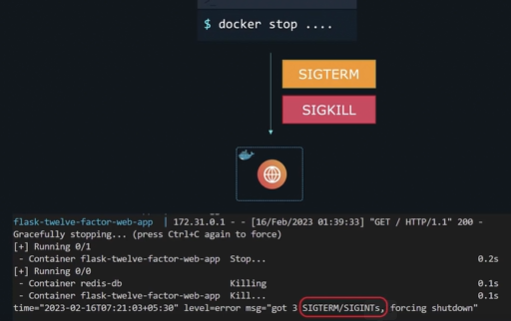
Every service we are building will have an specific port number to listen the request. We must expose all the services with their ports. We can bind the port to other services.

**9) Disposability**

12-factor app processes should be disposable where they can be started or stopped in a moment’s notice.

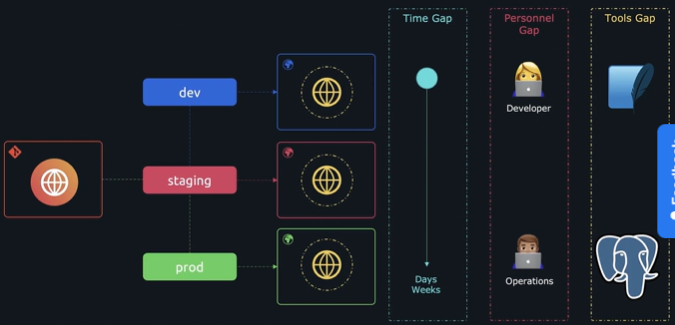
-> Just like scaling our application instances, we should delete or dispose applications gracefully when they receive SIGRERM signal from process manager. The processes should not depend on startup files.

-> after **docker stop** docker sends the **SIGTERM** signal to the process to shutdown giving the time to process the request which already made by users to this process. If it doesn’t go down after grace period it sends **SIGKILL** to kill the signal forcefully to terminate the application running in container.

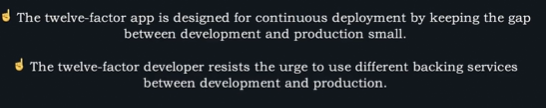


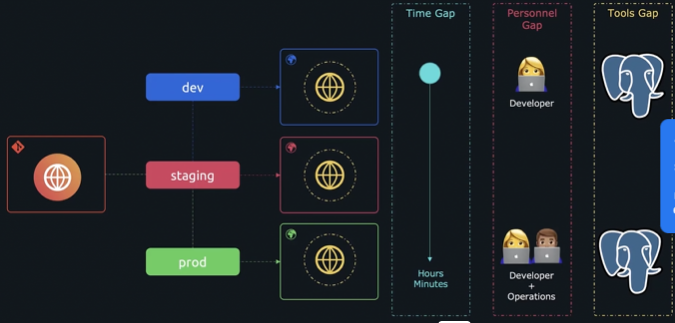
**10) dev/prod parity**

Previously there was much time gap between the development of the application to deploying the application in the prod env by OPS team. With this the cycle gap was huge and tools used in dev env and prod were different.



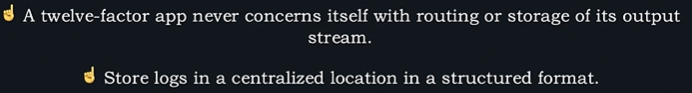
The time has shorten for the application to go from dev env to prod env because of the continuous deployment happening .



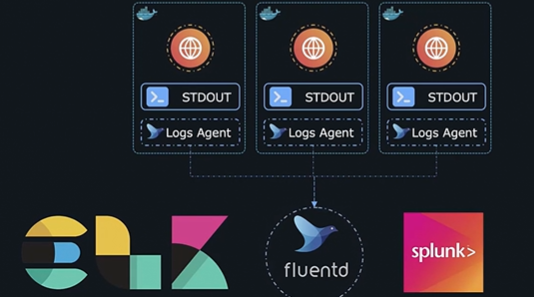


**11) logs**

In cloud env every process generate logs. Generate logs from each service and save it in one place.



All logs must go from process with the help of an agent to a different service called centralised logging solution where we can see in a structured way.



**12) Admin Processes**

Administrative tasks should be kept separate from application processes. To upgrade any server or DB data.

