



Cloud Native Configuration

More than just the environment

Objective / Agenda

- 01 Configuration ?
- 02 Classic approaches
- 03 Cloud native approaches
- 04 Demo



WHOAMI



[https://blog.wescale.fr/author/ismael-hommani/ \(fr\)](https://blog.wescale.fr/author/ismael-hommani/ (fr))



<https://blog.wescale.fr/author/joachim-rousseau/>



@ihommani



@_jro



Cloud Native Developer



Cloud Native Developer

WeScale: Tailor-made Cloud excellence

We help you **designing, building and mastering** your Cloud infrastructure.

To sum up



Our **expertises**



Cloud
Migration



Cloud
Architecture



DevOps
Automation



SecOps
Security



FinOps
Optimization



Site
Reliability
Engineering

Why WeScale ?

Our clients



Our training programs



Our partners



Find us

WeScale.fr
WeScaleTV
[Le blog de WeScale](#)
01.83.75.05.26 - contact@wescale.fr

1. Configuration ?



Configuration

In an applicative context

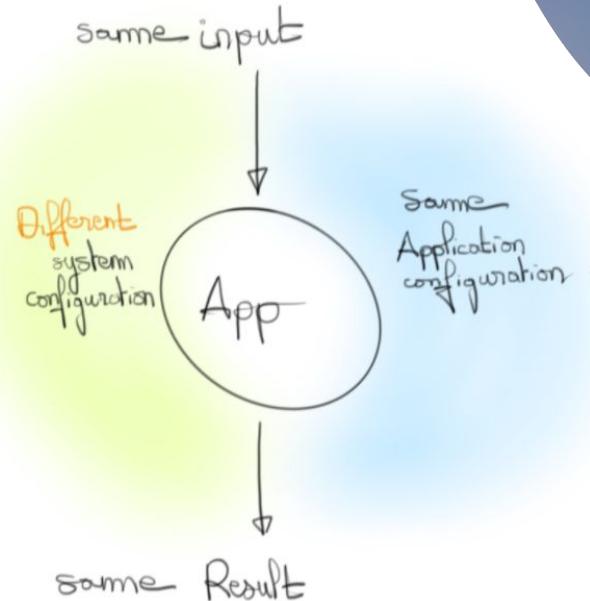
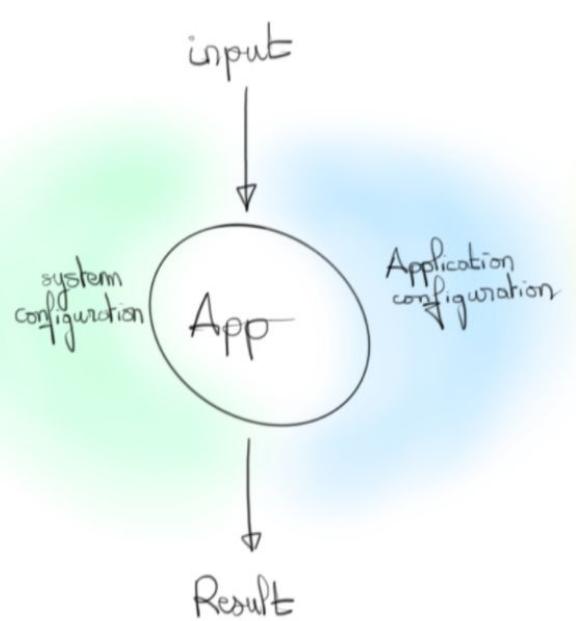


Everything that is likely to vary between deploys

- Tells how the application behaves → Application configuration under our control
 - third party services location
 - identifiers
 - algorithm configurations
 - ...
- Tells how the application is called → Set by the host system environment. Not under our control.
 - IP address
 - Port

Configuration and idempotency

A pillar of the Cloud Native



Configuration in the Cloud Native context

Issues to tackle

How to guarantee idempotency in the cloud ?

Problems:

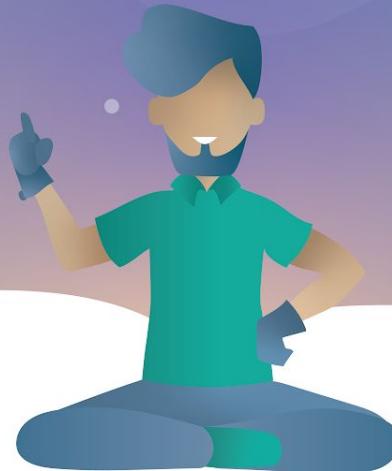
- Massively distributed
- Infinite horizontal scalability
- Ephemeral environments (update and failures are constant)

Target:

→ an automatic deployed configuration for each instance according to the targeted environment

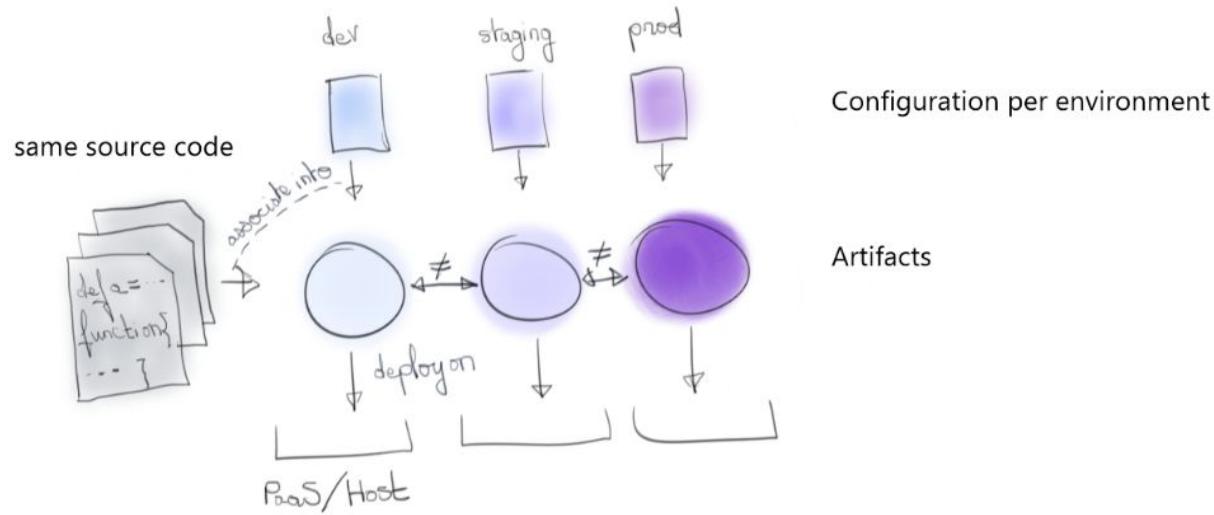
2. Classical approaches

Limitations



Code and configuration are packaged together

Configuration is embedded with the applicative code



Code and configuration are packaged together

Pros and cons



- Simple approach
- Central model for how to configure an application
- Artifact is not immutable
- An artifact per environment

Twelve factor App

The third factor



"Store the config in the environment"

- Environment variables are ubiquitous
- Standard for most languages
- unique artifact per environments
- What configuration are we talking about ?
- getEnv calls spread all over your code
- No central model of your configuration

3. The Cloud Native approach

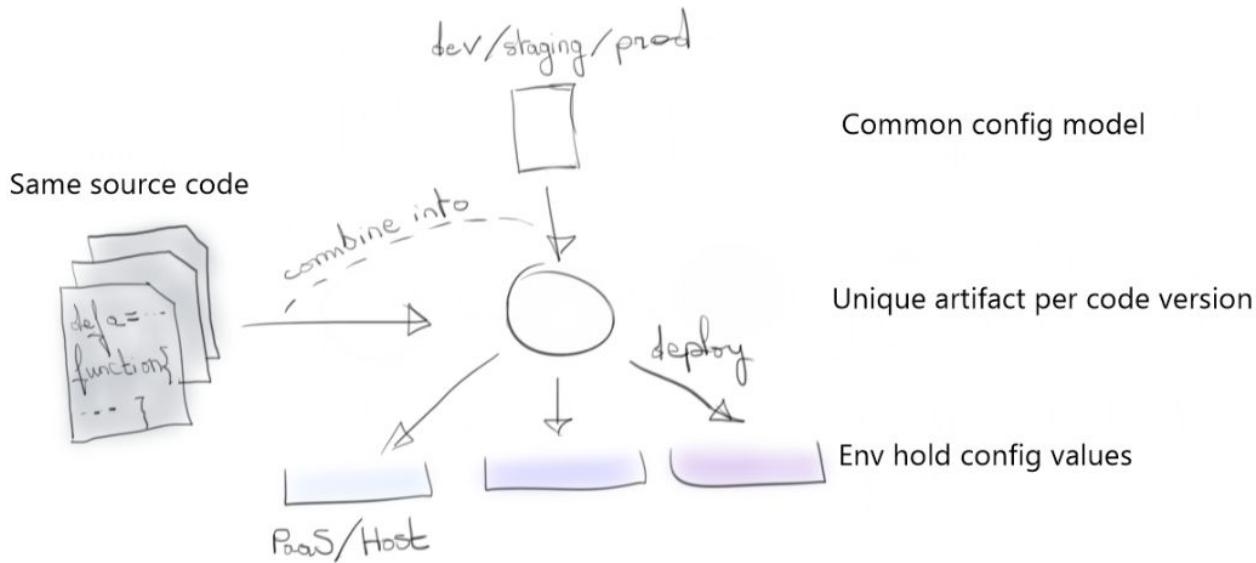
The “configuration Layer”



The Configuration Layer

What concepts?

Move a single artifact between environments

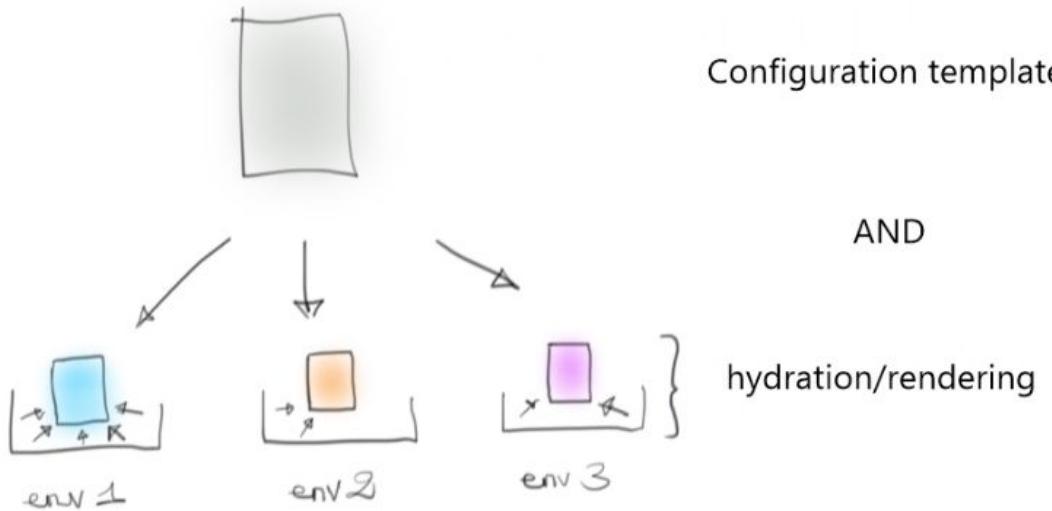


The Configuration Layer

Absorbing the context



The common model act like a sponge and “absorbs” values from the environment

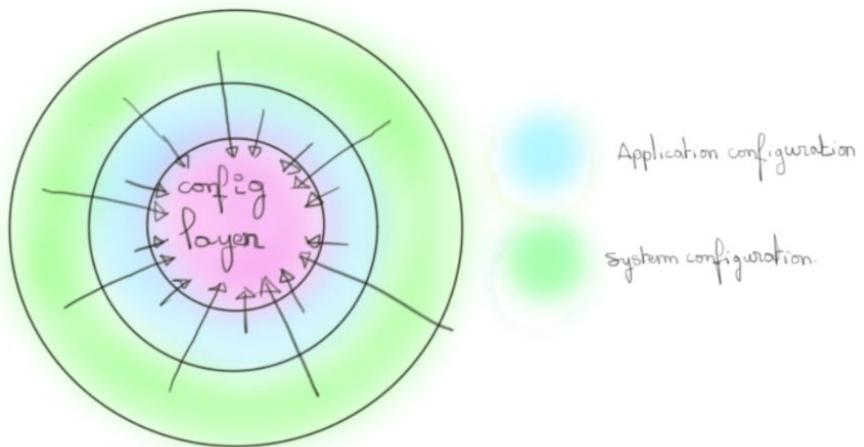
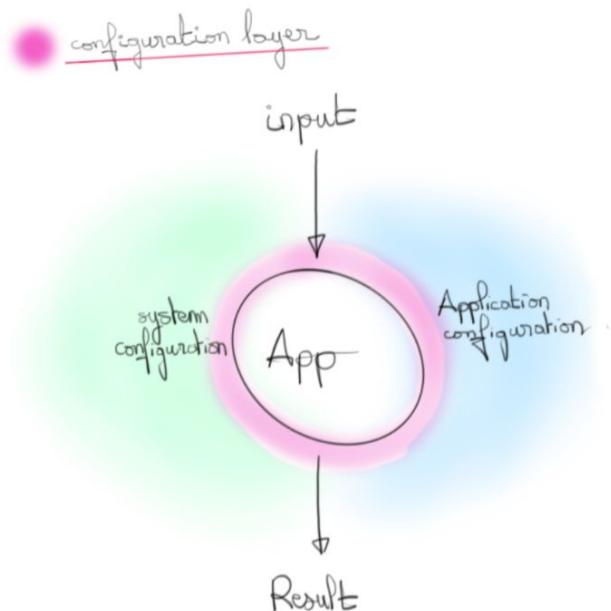


The Configuration Layer

The best of the two worlds



A single source of truth that abstract different origins



The Configuration Layer

Implementation

In the real life

- Not a particular technology but an association of libraries and patterns
- Different level of maturity depending on ecosystem (Jvm, Php, Java, DotNet)
- Conceptual but will demand more or less work considering the chosen ecosystem

4. In practise

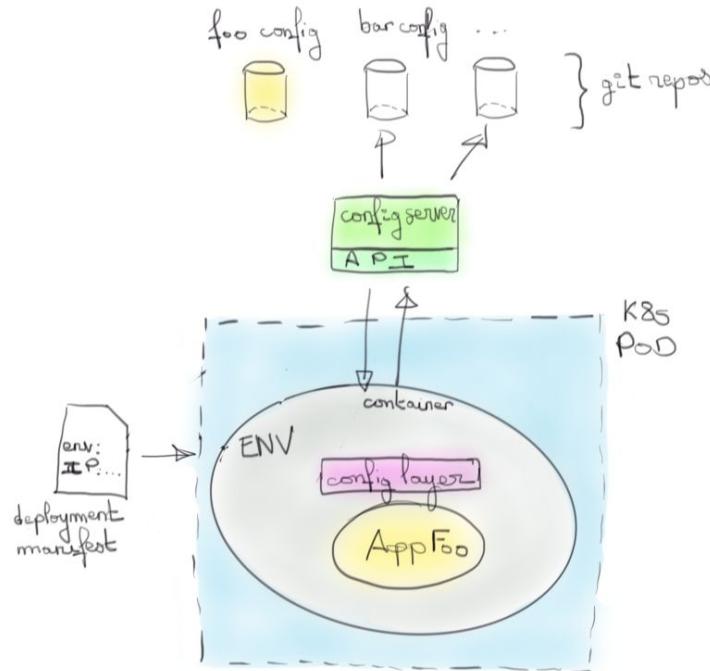
What do we have ?



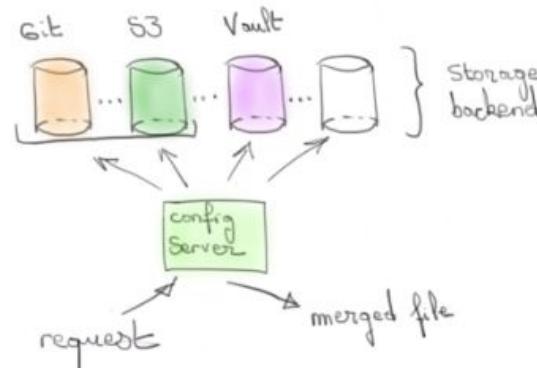
Source the configuration

Spring ecosystem example

K8S + Spring Cloud Configuration



A zoom on Spring Cloud Configuration

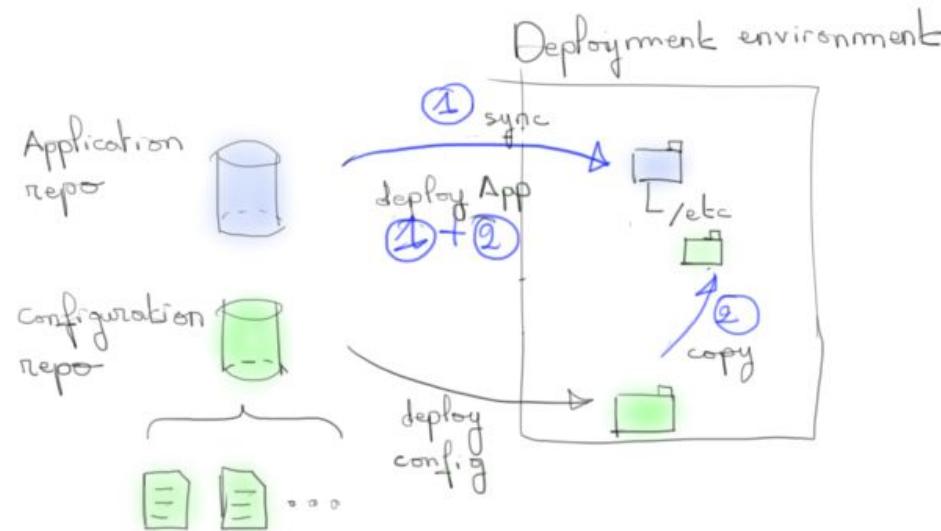


With a client (1)

Historical choices, we move step by step



Starting situation

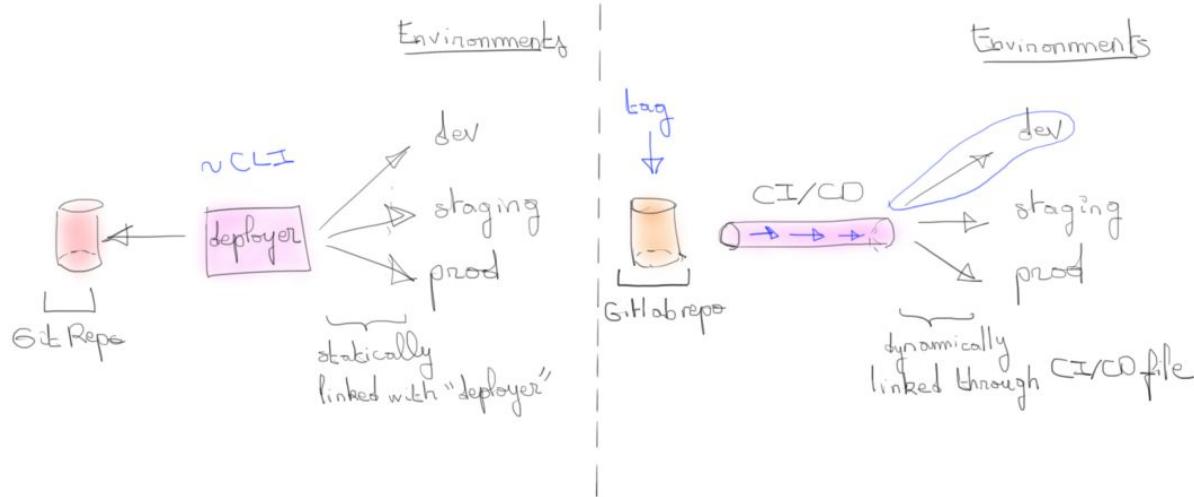


With a client (2)

First step, CI/CD with configuration layer inside the pipeline



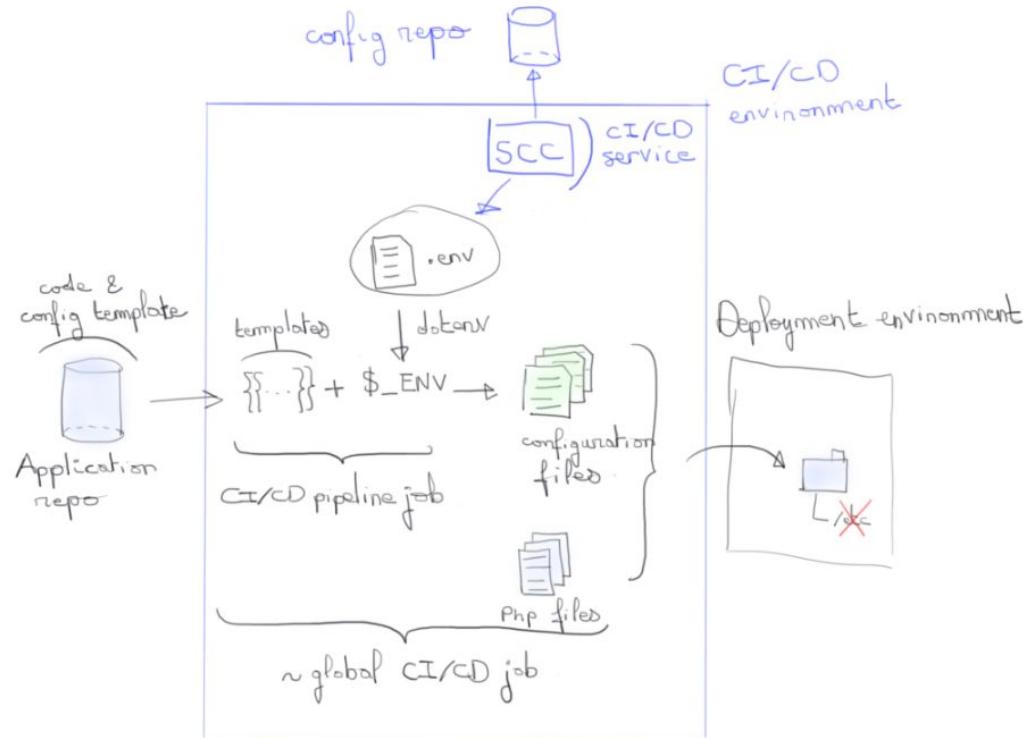
Intermediate situation



With a client (2b)

The configuration layer inside the pipeline environment

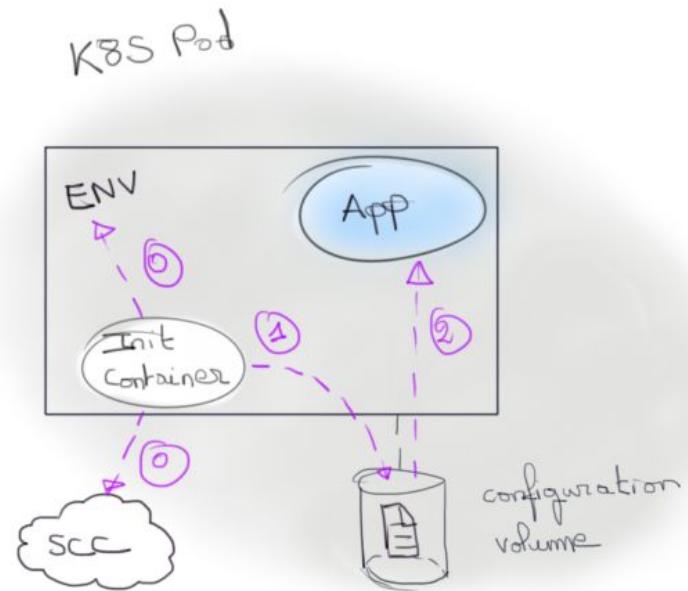
Details



With a client (3)

Translation into the Kubernetes paradigm

Finally



The Lightbend config case

The Rolls Royce of Cloud Native configuration



Human readable, yet flexible for the cloud

- HOCON format: JSON related, ease of read
 - Order matters
 - Unit for temporal measures (write “30s” instead of “30000”)
 - Typed (String, Duration, Number, Array, ConfigObject, etc.)
- Understands references
 - to variables defined in HOCON files
 - to environment variables
- Merges *.conf file with “system properties” from the JVM
- Handles *include*
- Standard orchestration of loading/merging

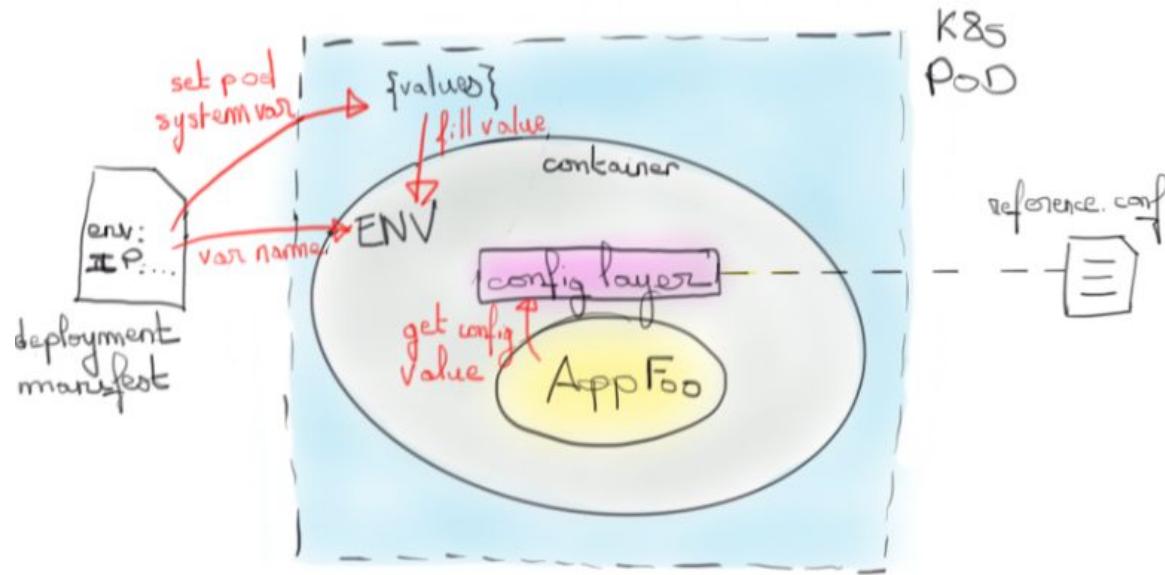
Properties allowing creation of a multi-layers configuration!

The Lightbend config case

The Rolls Royce of Cloud Native configuration



Layered model applied : 1 - reference config

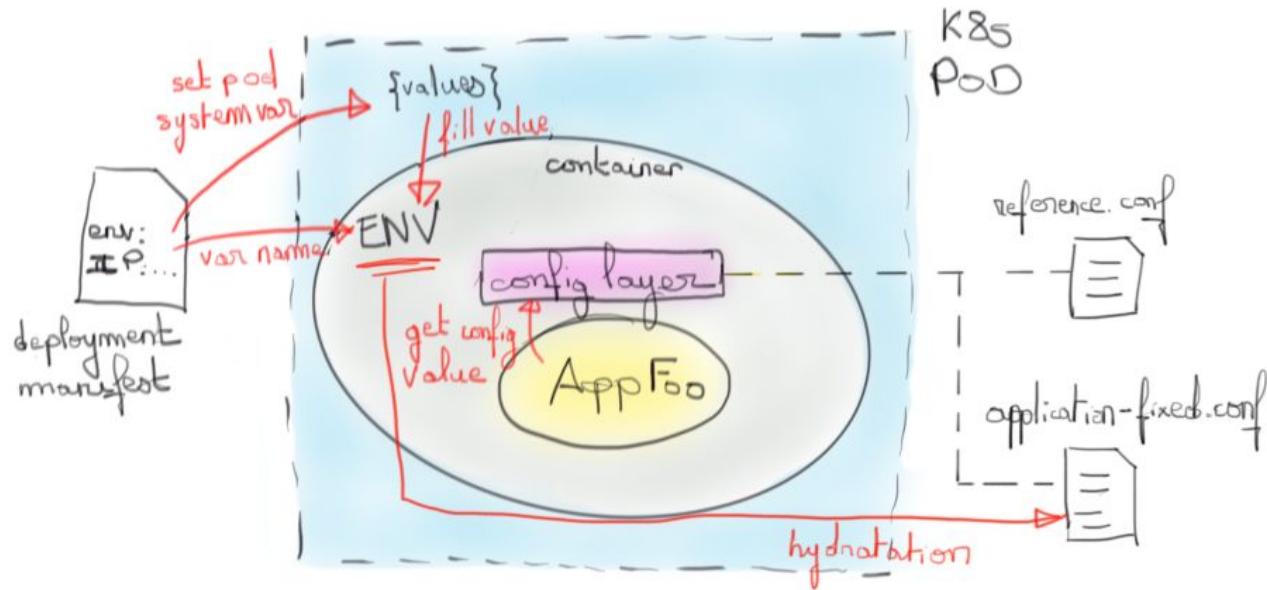


The Lightbend config case

The Rolls Royce of Cloud Native configuration



Layered model applied : 2 - Conf bound to the app lifecycle

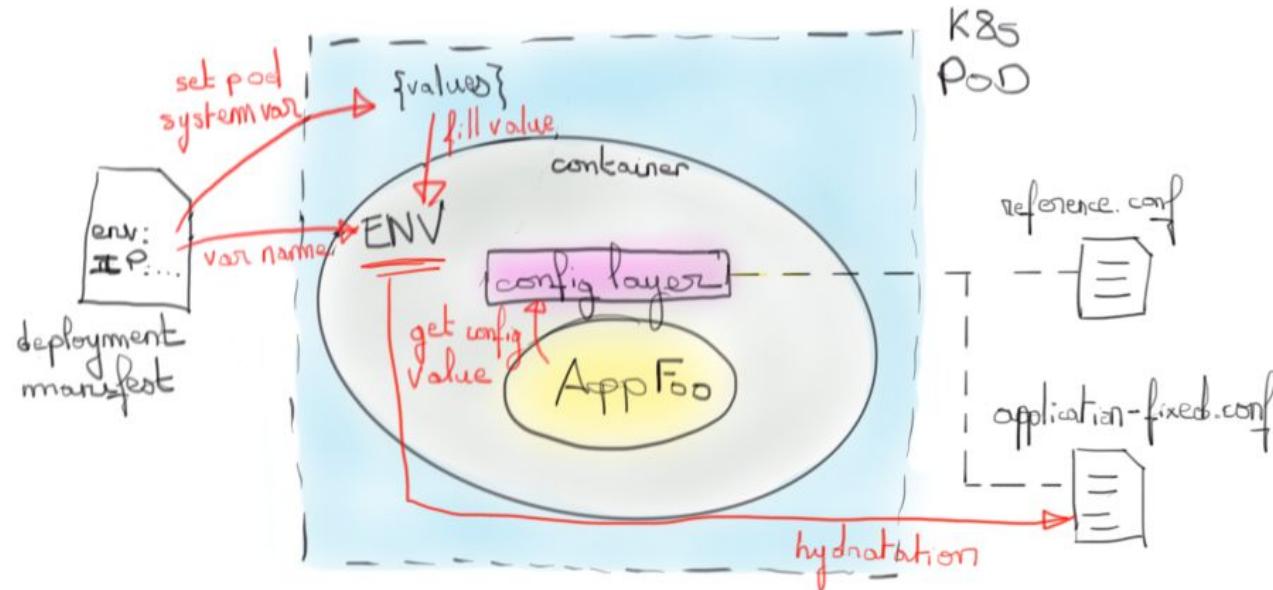


The Lightbend config case

The Rolls Royce of Cloud Native configuration



Layered model applied : 2 - Conf bound to the app lifecycle

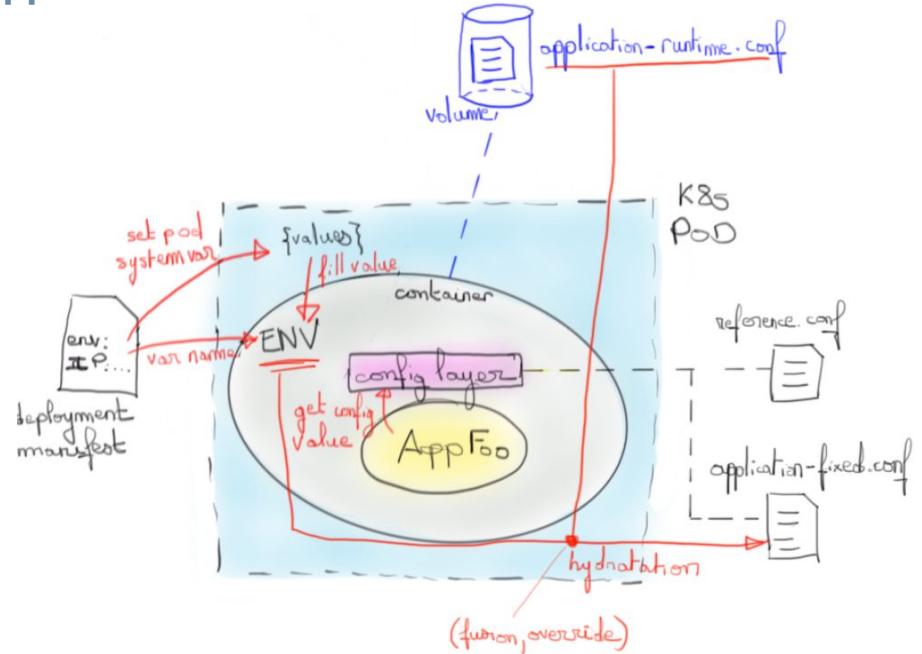


The Lightbend config case

The Rolls Royce of Cloud Native configuration



Layered model applied : 3 - Conf bound to environment

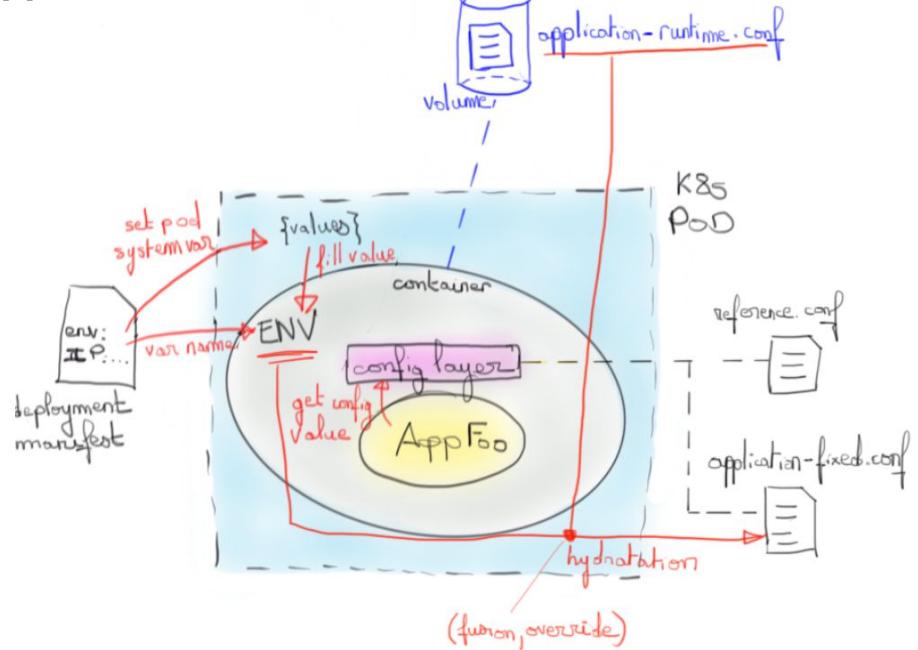


The Lightbend config case

The Rolls Royce of Cloud Native configuration



Layered model applied : 3 - Conf bound to environment



Demo

Going further

Links of interest

- <https://github.com/lightbend/config>
- <https://12factor.net/config>
- <https://cloud.spring.io/spring-cloud-config/reference/html/>
- <https://blog.wescale.fr/2020/11/06/lightbend-config-la-configuration-cloud-native/>
- <https://blog.wescale.fr/2021/02/04/le-cloud-lapplication-et-la-configuration/>

La
wetribu !
dans ton **salon** !



jobs@wescale.fr

Rejoins wescale
où que tu sois
en **France** !

En **remote** partout en **France**



jobs@wescale.fr

Thank you !