PostgreSQL au service du prévisionniste météo

PGDay Toulouse

Fabien MARTY, Météo-France, juin 2017

Votre serviteur

fabien.marty@meteo.fr

Architecte SI depuis 10 ans

Responsable technique projet

Plan

Une pincée de marketing

Un soupçon de technique

Nos aventures avec PostgreSQL

Plan

Une pincée de marketing

Un soupçon de technique

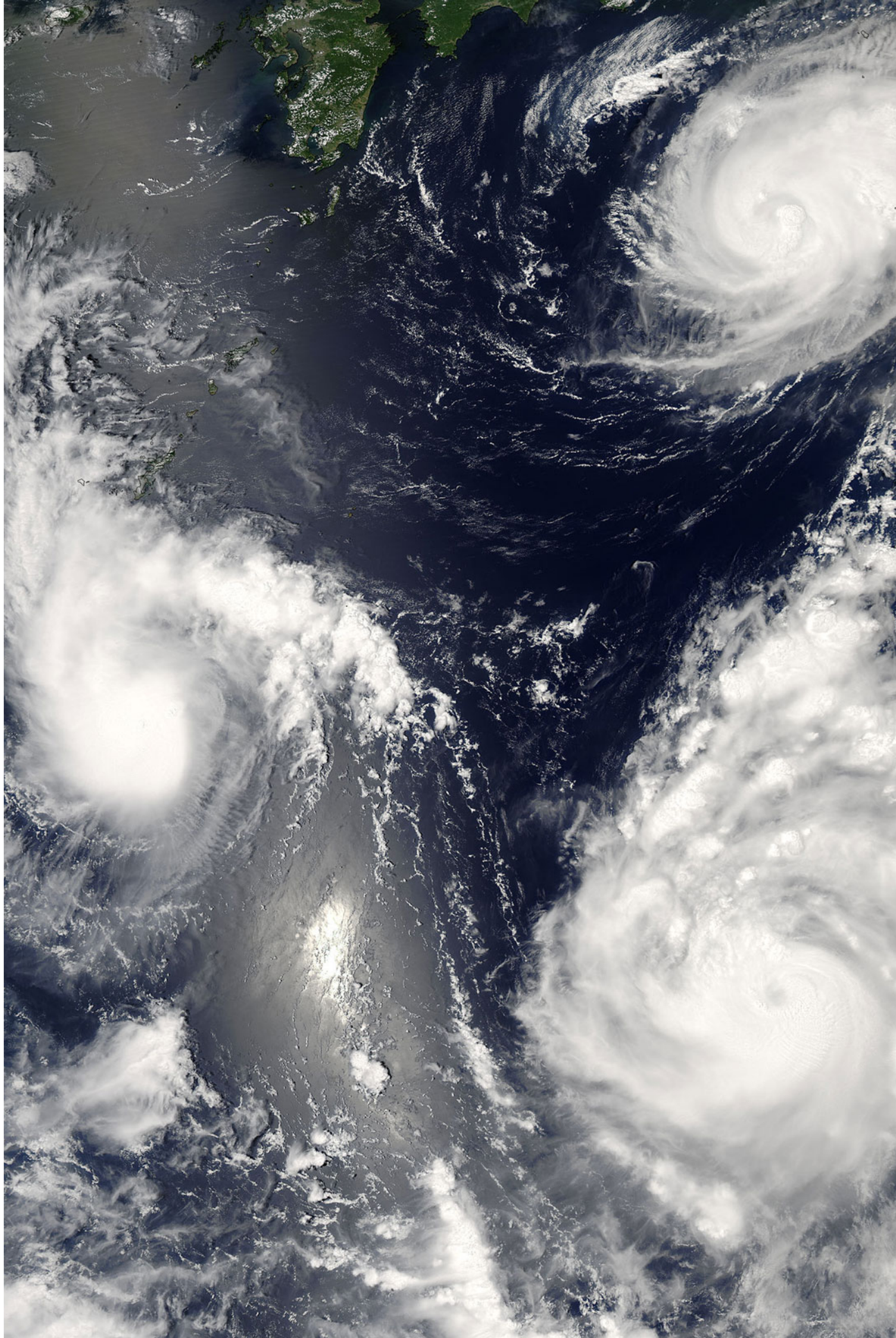
Nos aventures avec PostgreSQL

Météo-France

Un établissement **public** à caractère administratif

Sa mission principale

Alerter les autorités et les populations des phénomènes météorologiques dangereux



≈ 3000 personnes

≈ 1/3 à Toulouse

Très forte composante scientifique et technique

Financé à **50%** par l'état

Météo-France et le libre

Météo-France a une approche **pragmatique** et **non militante** vis à vis du libre

… mais on aime bien PostgreSQL ;-)

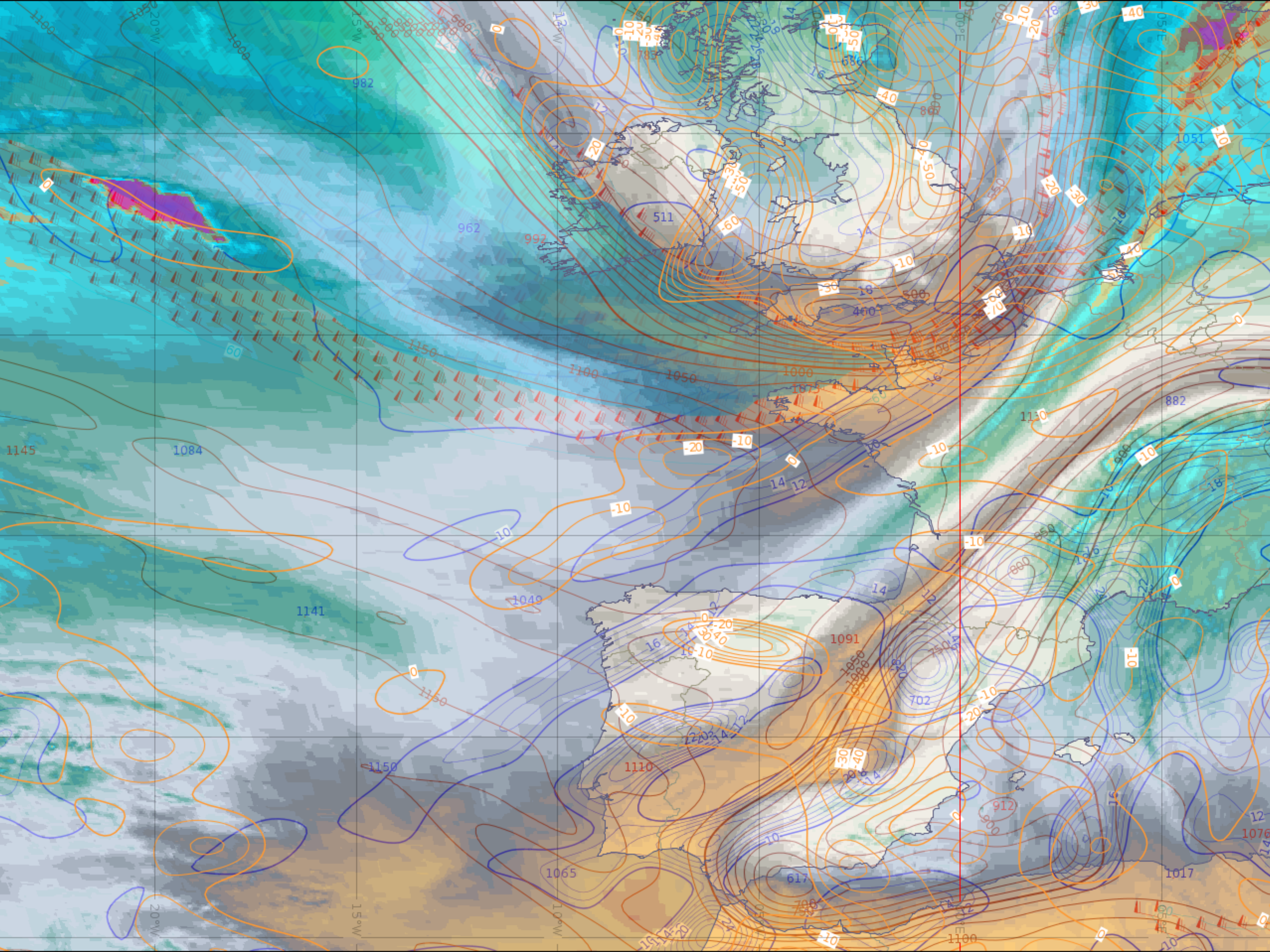
Le projet Synopsis

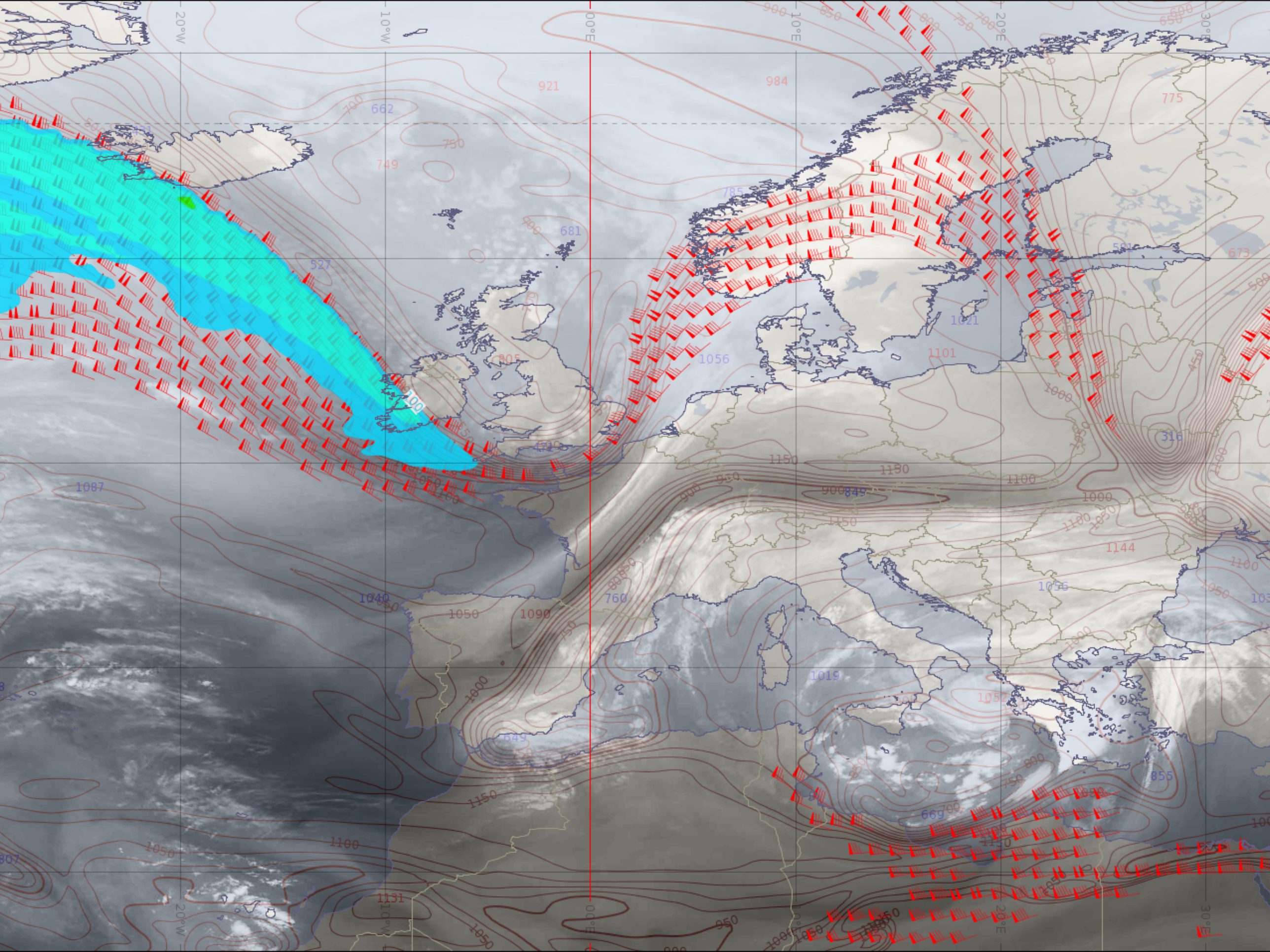
Objectif : la **nouvelle** station du travail de prévisionniste

{SYNOPSIS}

Un gros projet stratégique

- Sur > 5 ans
- > 600 hommes.mois de développement
- Au coeur du métier et du SI de Météo-France




20°W
10°W
00°E
10°E
20°E
30°E
921
984
662
749
785
681
527
805
1056
1101
1021
775
673
316
1087
1040
1050
1090
760
1019
649
1131
1144
855
669
845
1150
1100
1000
950
900
850
800
750
700
650
600
550
500
450
400
300
200
100

H] pour jeu. 13/11 à 12:00UTC × T Min/Max [...] ×

IR+HRV MSG3 pour jeu. 13/11 à 13:00UTC ×

Profil Z (hPa) ARP 0.5 13/11 00H, Ech012H pour jeu. 13/11 à 12

jeudi 13/11/2014 12:00
[lat/lon]=[46,5/0]
Iso -10: 4637 mgp
Iso 0: 2945 mgp
CAPEMax: 0 J/kg
CAPE: 0 J/kg
CIN: 0 J/kg
ADEDOKUN2: -5,5 °C
TELFER: 60
FAUST: -6,1
TTI MODifié: 48,1 °C
Psol= 990 hPa,T2m= 11,6 °C

FF (hPa) 500 ARP 0.5 13/11 00H, Ech012H pour jeu. 13/11 à 12:00UTC ×

57°42'N 24°04'E - Altitude : 0.0m

13/11/2014 12:00

Tous

Plan

Une pincée de marketing

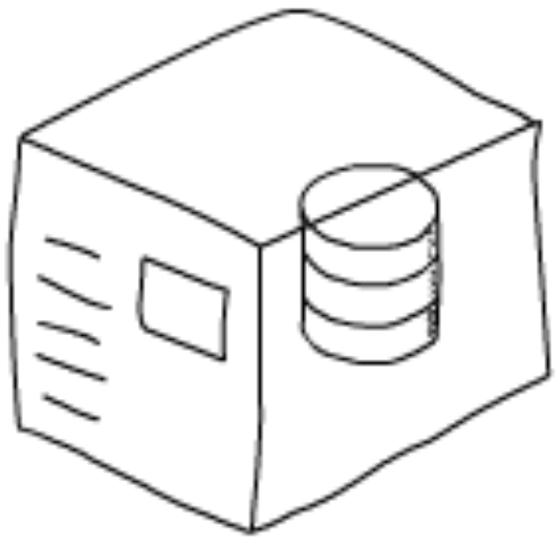
Un soupçon de technique

Nos aventures avec PostgreSQL

Schéma de fonctionnement détaillé ;-) de Synopsis

Données

FTP



Serveurs Synopsis

HTTP

IHM Synopsis

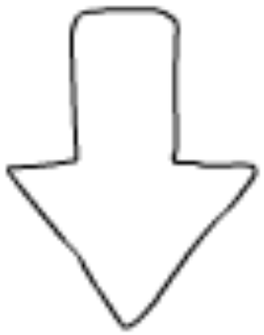
Utilisateurs heureux ;-)

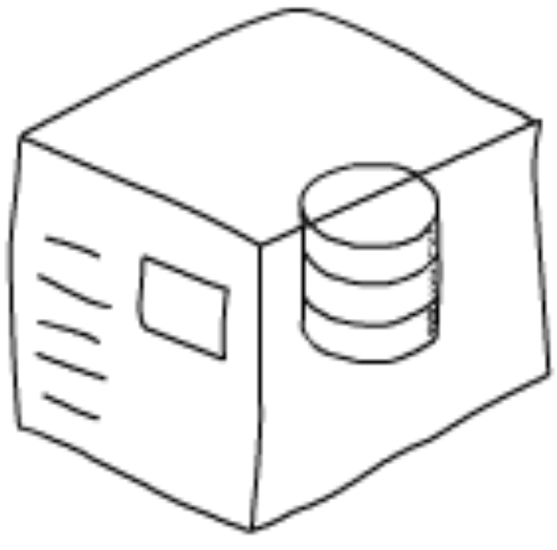
Schéma de fonctionnement détaillé ;-) de Synopsis

Données

0.5 To / jour

FTP

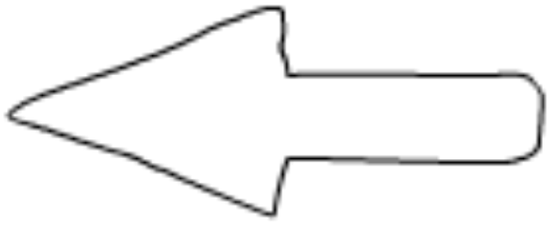




Serveurs Synopsis

x 25

250 req/s



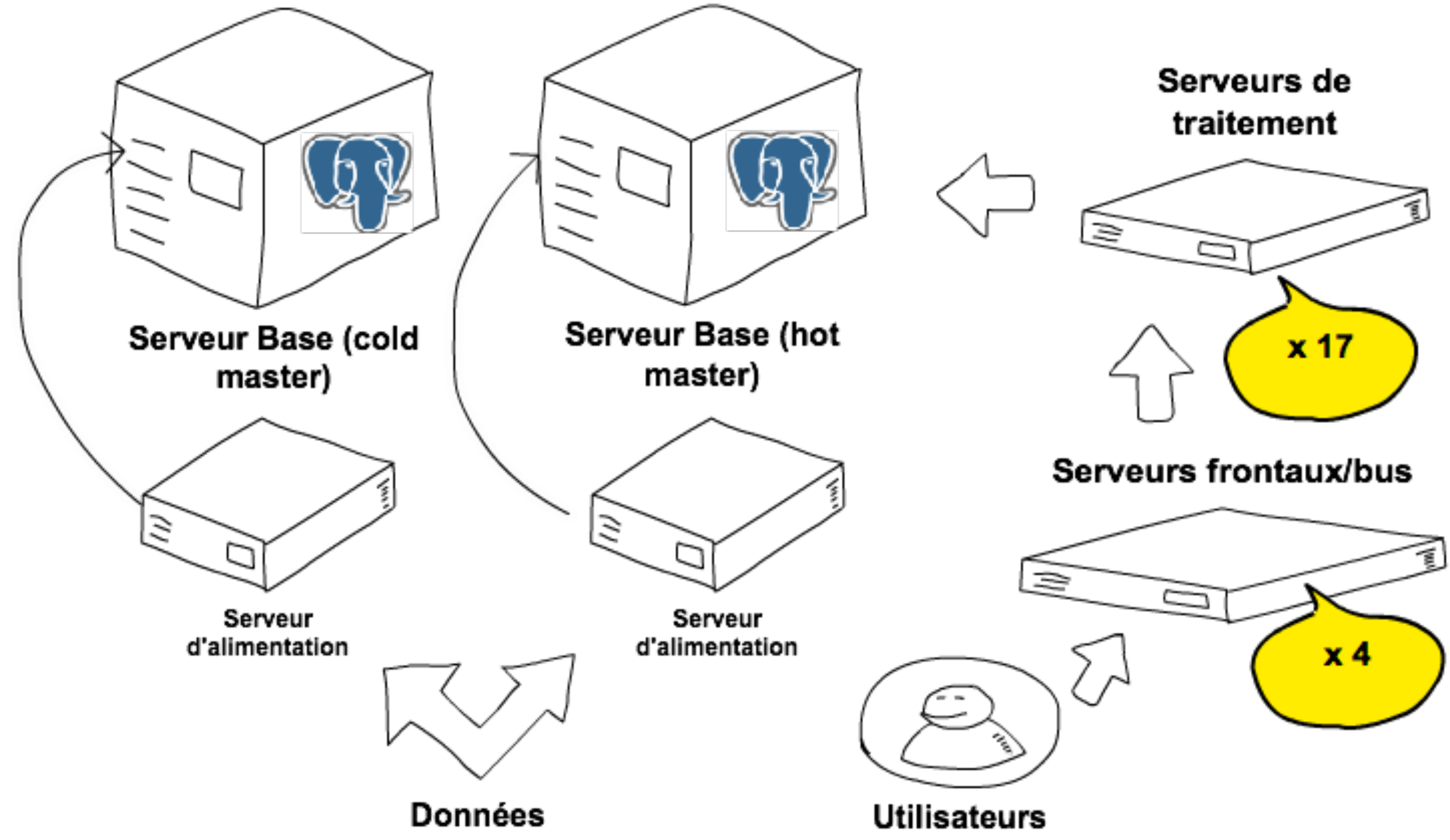
HTTP

IHM Synopsis

x 200

Utilisateurs heureux ;-)

Zoom sur les serveurs Synopsis



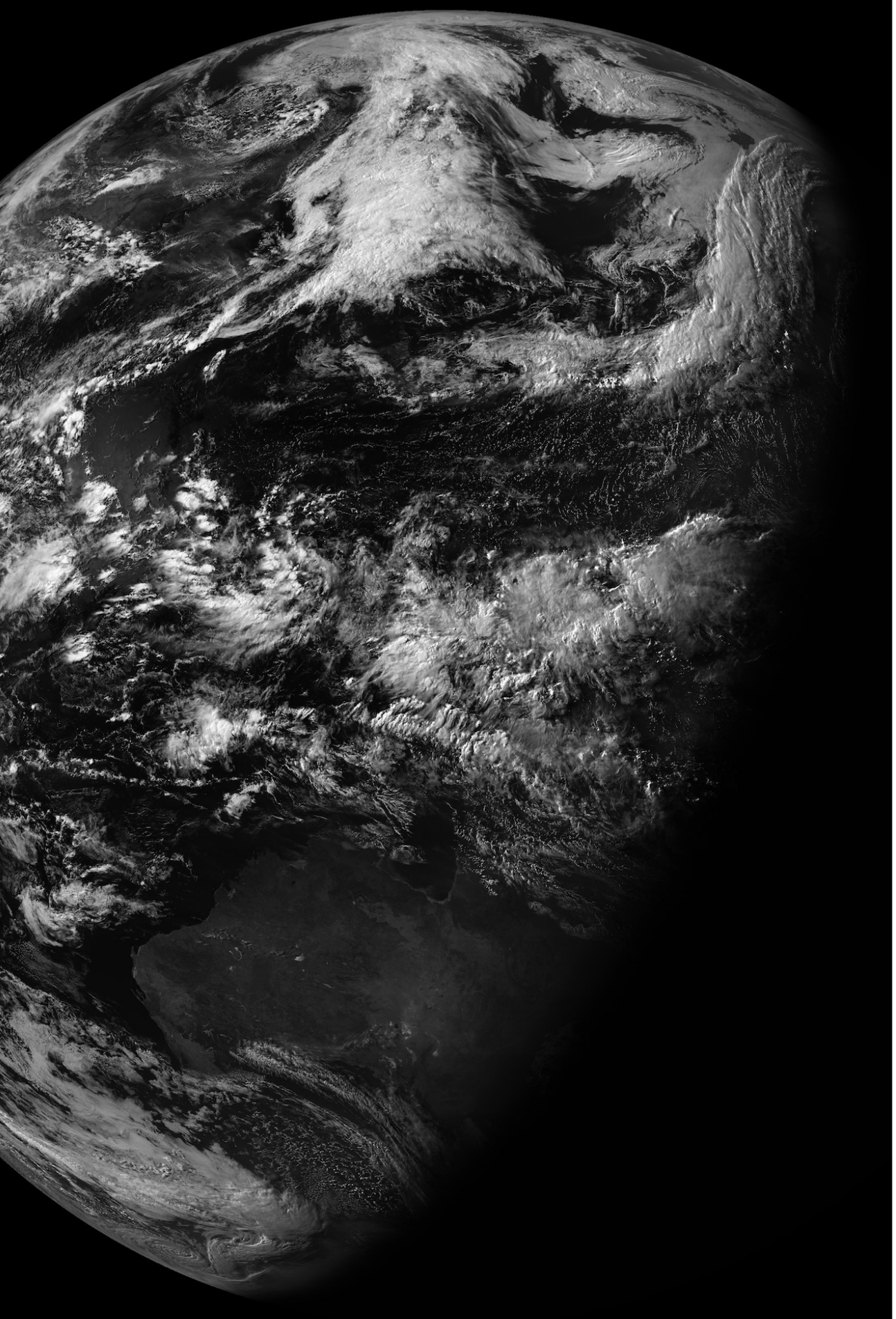
Plan

Une pincée de marketing

Un soupçon de technique

Nos aventures avec PostgreSQL

Saison 1 : stocker les gros fichiers (BLOBs) dans PostgreSQL **ou pas**



Exemple concret : une image satellite

Image géostationnaire HIMAWARI8 (Japon)

Un GeoTIFF compressé de 22000 x 22000 (pixels)

≈ 50 Mo

L'image est quasiment auto-suffisante

On stocke dans la table quelques éléments pour faciliter la recherche :

- nom de produits / fournisseurs
- dates diverses
- domaine géographique visible (polygone postgis)
- information de taille/projection

... mais reste la question de l'image elle même !

Premier épisode : la colonne BYTEA

Les +

- Dans la même ligne que les métadonnées associées (atomicité)
- Image accessible "simplement" depuis une autre machine

Les -

- Pas d'accès fichier direct
- "Mauvaises" performances (encodage / décodage)

Deuxième épisode : stocker les fichiers sur un serveur WebDAV (nginx) et stocker les URLs des BLOBs dans la table

Les +

- Légèreté (nginx) + performances (pas d'encodage/décodages)
- Accessible depuis une autre machine (via HTTP)

Les -

- Pas d'accès fichier direct
- Pas d'atomicité

Troisième épisode : WebDAV avec URLs "aléatoires" + filesystem (FUSE) maison

- URLs "aléatoires" obligent à passer par la base pour lire un BLOB (pas d'accès direct)
  - => une forme d'atomicité
- filesystem FUSE maison monté sur les serveurs de traitement pour obtenir un accès FS direct performant
  - accès en mode bloc sans télécharger la totalité de l'image
  - cache très aggressif car nos BLOBs sont en lecture seule

Saison 2 : stocker les données des stations de mesure

Plusieurs dizaines de milliers de points de mesure au niveau mondial

Plusieurs dizaines de paramètres mesurés pour une grosse station

Très grande hétérogénéité

>300 types de paramètres différents

Premier épisode : stocker les mesures "standards" en colonnes et garder le reste dans le format binaire d'origine (colonne bytea)

Les +

- Performances, simplicité pour les données "colonnes"

Les -

- Pas très souple
- Mauvaises performances pour les données "non colonnes"

Deuxième épisode : une ligne par valeur mesurée

Les +

- Simplicité
- Souplesse

Les -

- Occupation disque
- Mauvaises performances (notamment à l'insertion)

Troisième épisode : HSTORE indexé

Les +

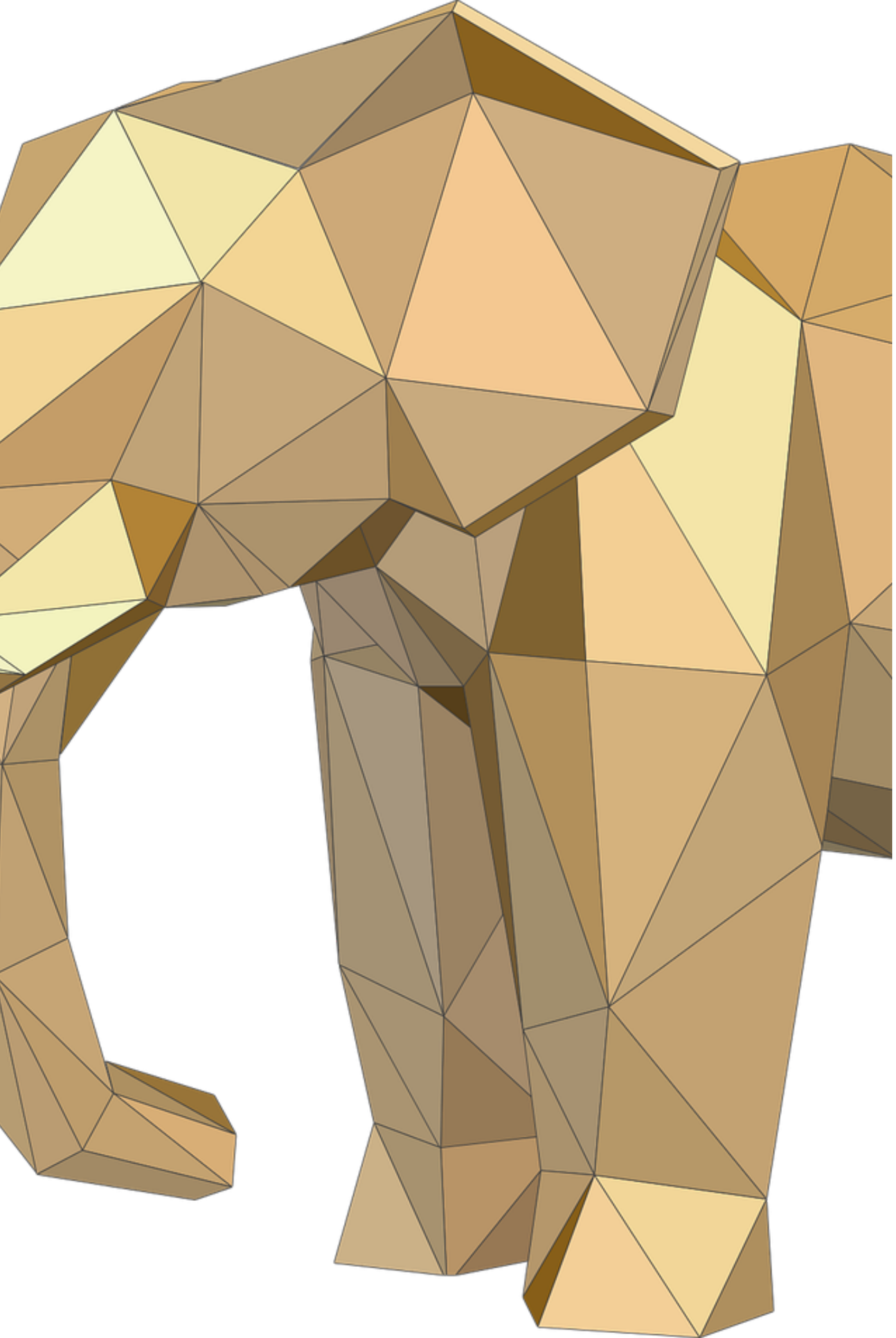
- Souplesse
- Performances

Les -

- Nécessiter de "caster" les valeurs en "string"
- Occupation disque (noms des clefs (paramètres mesurés) dupliqués d'une ligne à l'autre)

Quatrième épisode : HSTORE indexé avec clefs "compressées"

- Réduction des noms de clefs à 2 lettres
  - avec un dictionnaire dynamique de correspondance et des fonctions PLPGSQL utilitaires
- Petit regret : le support JSON n'était pas assez mature au moment de notre choix
  - ...mais si c'était à refaire...



Saison 3 : le faceting de notre catalogue dynamique de données

SELECT DISTINCT(foo)
FROM ... WHERE
whatever

(avec des millions de lignes)

Episode 1 : Utiliser PostgreSQL

Les +

- Simplicité

Les -

- SELECT DISTINCT pas très performant
- WHERE whatever nécessite quasiment de tout indexer
- Performances aléatoires selon la présence des données dans le cache

Episode 2 : Construire une sorte d'index inversé avec redis

Les +

- Performances

Les -

- Une forme de duplication
- Solution maison peu souple et difficile à maintenir

Episode 3 : ElasticSearch

- On garde les données dans PostgreSQL mais on enregistre (également) la disponibilité des données dans ElasticSearch
- On utilise le faceting natif d'ElasticSearch
- C'est performant et relativement simple mais ça reste une forme de duplication

Saison 4 : les données glissantes

Episode 1 : des tables standards

Les +

- Simplicité

Les -

- VACUUM
- Dégradation progressives des performances

Episode 2 : des tables partitionnées quotidiennes dynamiques

Les +

- Performances (y compris dans le temps)

Les -

- Une certaine complexité à la mise en oeuvre (avec du code maison)
- Nécessité d'adapter le code d'insertion (pour éviter un trigger à chaque insertion)
- Tables journalières pas adaptées pour conserver > 100 jours

Episode 3 : des tables de façon configurables avec pg_partman

Les +

- Moins de code maison
- Souplesse (tables journalières, hebdomadaires, mensuelles… selon les cas)

Les -

- Pas encore en production

En résumé / conclusion

- 2 bases PostgreSQL (9.5) sans réplication (alimentation en Y)
- Taille réduite de l'ordre de 200 Go
  - merci nginx qui héberge ≈ 5 To en webdav
- Structures des tables relativement simples quoique souples
- merci HSTORE et PostGIS

- PostgreSQL ne fait pas de faceting ou de recherches complexes
  - merci ElasticSearch
- ≈ 1500 tables et aucune administration manuelle
  - merci le partitionnement
  - et bientôt merci pg_partman !

Et jamais un crash !

Questions ?

Crédits photos / images

- Météo-France
- Louis De Funès, Gérard Oury dans La Folie des grandeurs (Photo Christophe L)
- pour-un-monde-meilleur.com
- NASA
- pixabay.com
- Toshiyuki IMAI | Flickr