python
ON A PLANE

Abstract

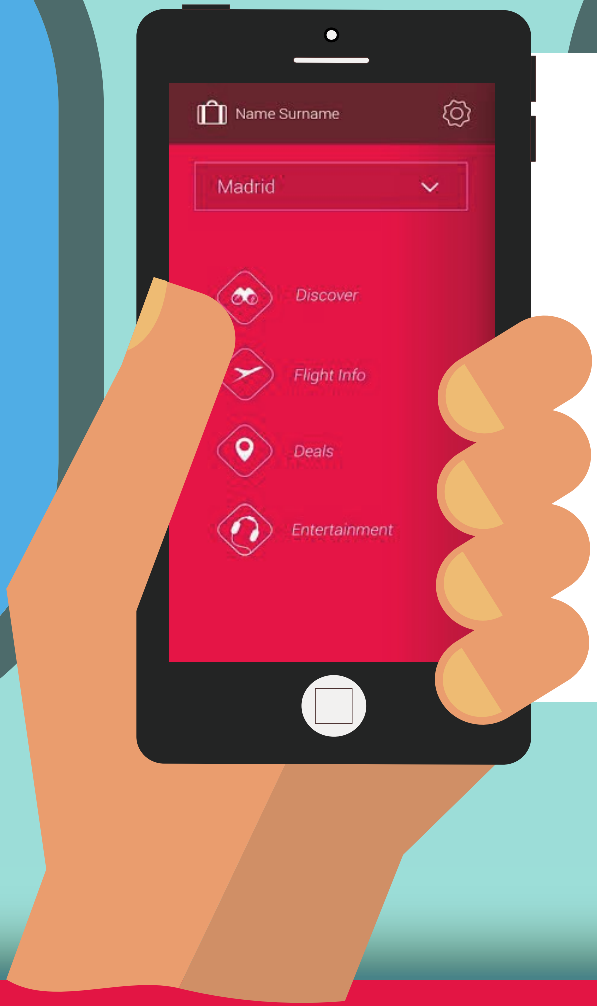
- An entertainment system at 10,000 metres
  - Embedded computer in each plane
  - Synchronized after each flight
  - And a Platform with lots of integrations

- This is how we do it

Hi!

- I'm David Arcos
- Python/Django developer since 2008
- Interested in distributed systems, databases, scalability, security
- Backend engineer at **Immfly**

Immfly: In-Flight Entertainment



Immfly is a new Entertainment, Retail and Communication platform for the in-flight experience.

Focused on the European domestic flights market, Immfly offers wireless content to passengers via their Personal Electronic Devices.

The three challenges

1) **On board the aircraft:**

- Provide a Backend with lots of features
- Will work off-line

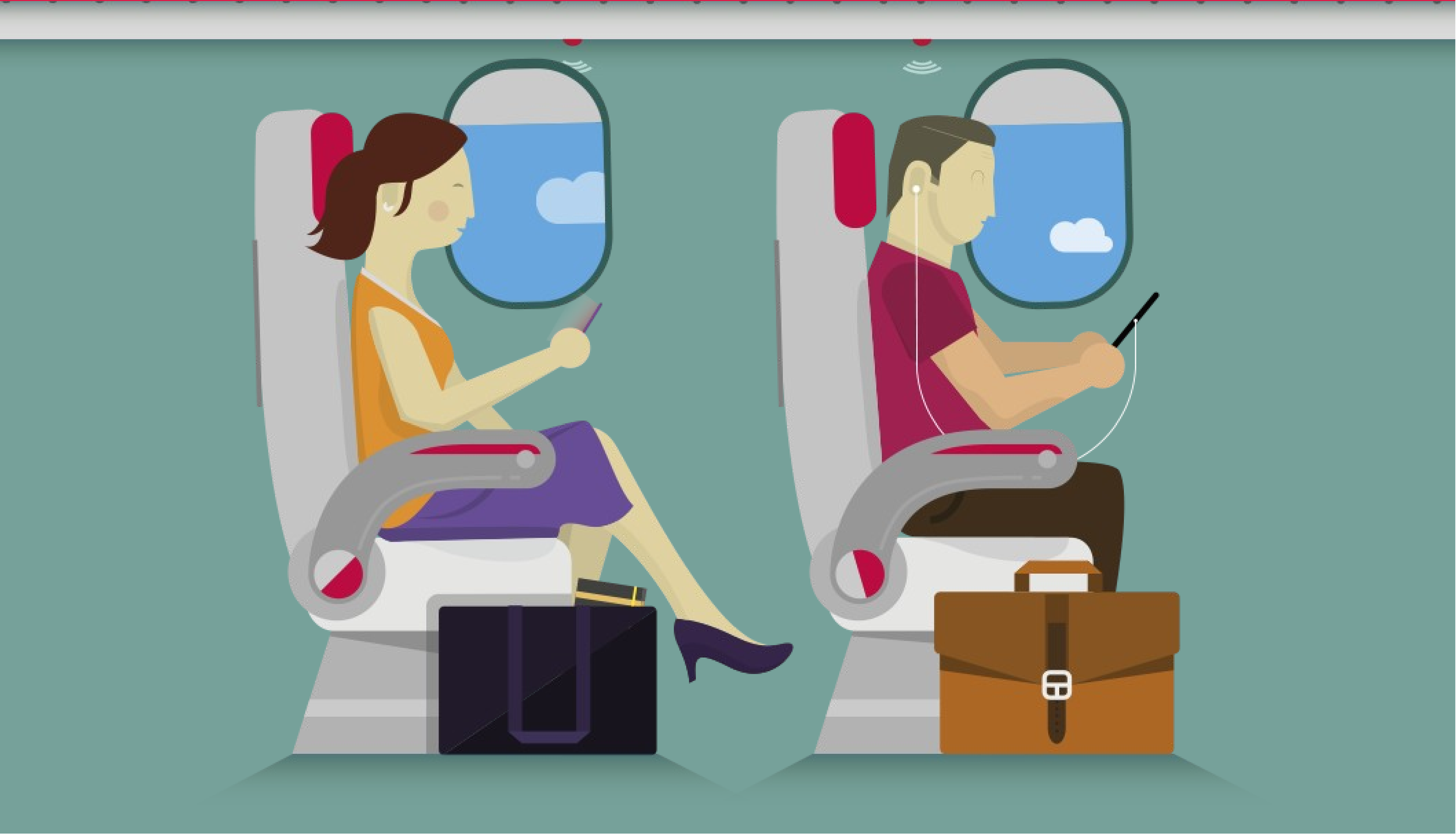
2) **Synchronize the aircrafts**

- Keep the replicas consistent
- Update the contents: *I don't want yesterday's newspaper!*

3) **Integrate with 3rd parties**

- Content providers, payments, mailing, weather, flight info...

1) On board the aircraft



Some requirements

- Users have different needs
- Common authentication
- It's offline
- Show some flight info
- Deliver static content

Users have different needs

- Apps: web, android, iOS
- Devices: laptop, smartphone, tablet
- Return different results based on language, destination, airline, schedule…

Users have different needs

- Apps: web, android, iOS
  - Same REST API for every frontend
- Devices: laptop, smartphone, tablet
  - Generate thumbnails with different sizes to allow a responsive design
- Return different results based on language, destination, airline, schedule...
  - API calls allow filtering by many parameters

Common authentication

- Same user, different apps
  - Example: web app opens mobile app
  - Example: mobile app embeds a webview

Common authentication

- Same user, different apps
  - Example: web app opens mobile app
  - Example: mobile app embeds a webview

- Django Rest Framework has **TokenAuthentication**
  - Just a HTTP header, example:

```
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

It's off-line!

- Can't use any external service:
  - No analytics, online error logging, google maps
  - No fancy SaaS integrations
  - Can't just use CDN, email or DNS servers

It's off-line!

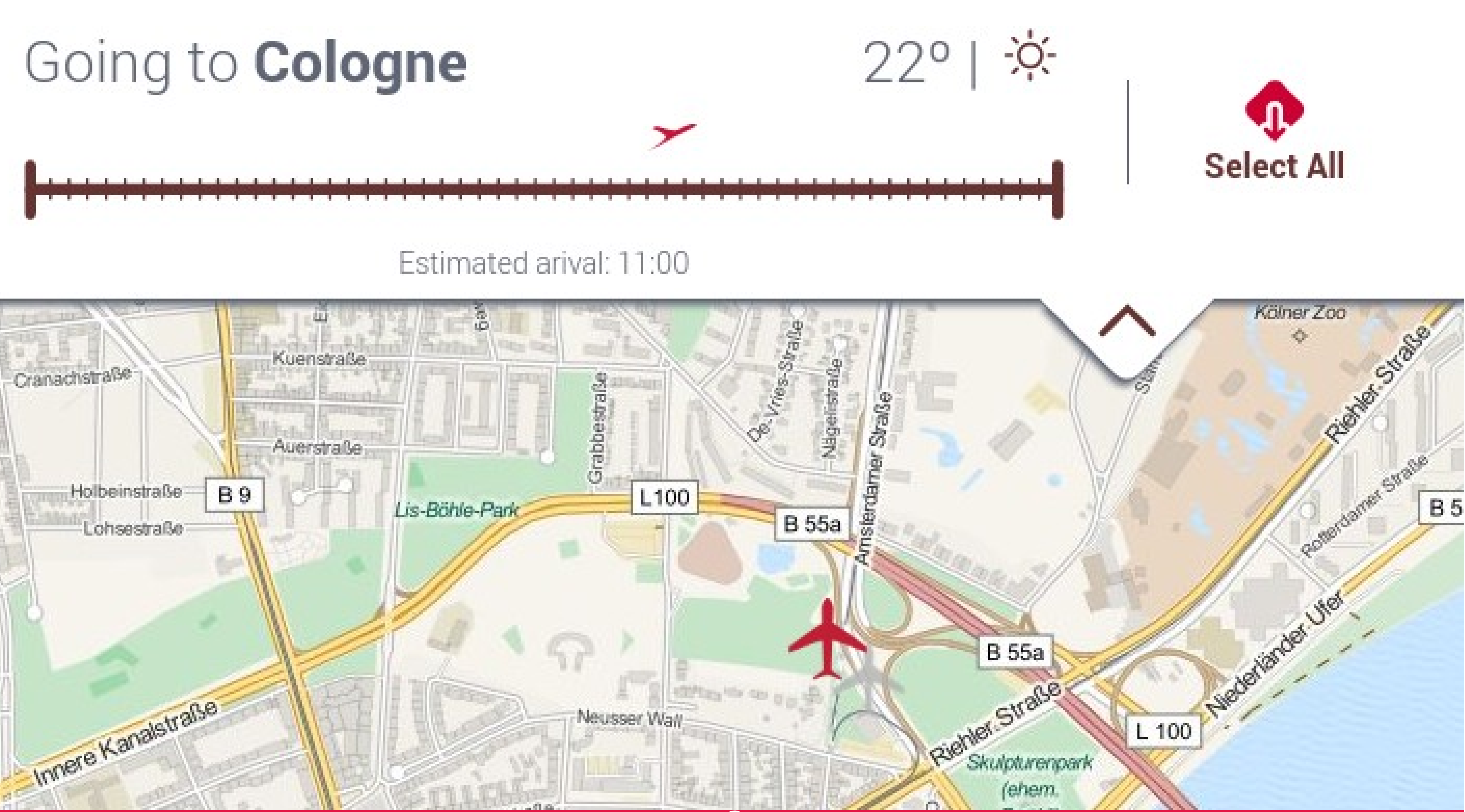
- Can't use any external service:
  - No analytics, online error logging, google maps
  - No fancy SaaS integrations
  - Can't just use CDN, email or DNS servers

- We deployed our own solutions
  - Very time-consuming
  - Can't hotfix bugs. Need to do lots of Q&A.

Show some flight info

- Estimated Time of Arrival?
- Weather at destination
- Show a nice map
- Where is the plane?

Show THIS flight info



Lots of flight info

- Estimated Time of Arrival?
  - Pre-load Flightstats API
- Weather at destination
  - Pre-load OpenWeatherMap (`pyowm`)
- Show a nice map
  - OpenStreetMap
- Where is the plane?
  - The plane tells us :)

Avionics data bus

We get data in real-time:

- `altitude`
- `flight_id`
- `ground_speed`
- `heading`
- `latitude`
- `longitude`
- `mach_speed`
- `outside_temperature`
- `pitch`
- `roll`
- `wind_speed`
- `yaw`

Discrete-time signals

- DCFAILSIG
- ACFAILSIG
- OVERTEMPSIG
- GSM_POWER_STATUS
- ENB2SIG
- ENB1SIG
- ENB0SIG
- GSMSIG_STATUS
- CPLD_REV0
- CPLD_REV1
- SYSENSIG
- ENB3SIG
- ENB4SIG
- ENB5SIG
- ALERT
- CONFIGSIG0
- CONFIGSIG1
- CONFIGSIG2
- INTTEST_OUT
- INTTEST_IN
- ISO_OUT0
- ISO_OUT1
- ISO_OUT2
- ISO_OUT3
- GPIO_DCFAILSIG
- GPIO_ACFAILSIG
- GPIO_OVERTEMPSIG
- GPIO_SYSENSIG

Deliver static content

- Provide the user with different contents:
  - TV Shows, Videos (format? size?)
  - Newspapers, Magazines
  - Images

Deliver static content

- Provide the user with different contents:
  - TV Shows, Videos
  - Newspapers, Magazines
  - Images

- Just use nginx, but:
  - Transcode video to HLS (bitrate, chunks)
  - Pre-process PDFs
  - **Load test** to verify assumptions

Other components

Our django apps:

- common
- content
- flightinfo
- images
- stats
- surveys
- users

Some external libraries:

- Django
- Django Rest Framework
- django-uuidfield
- django-redis
- django-celery
- django-extensions
- django-imagekit

Aircraft Infrastructure

HTTP server:

NGINX

+

gunicorn

+

Supervisor
hang on to your processes

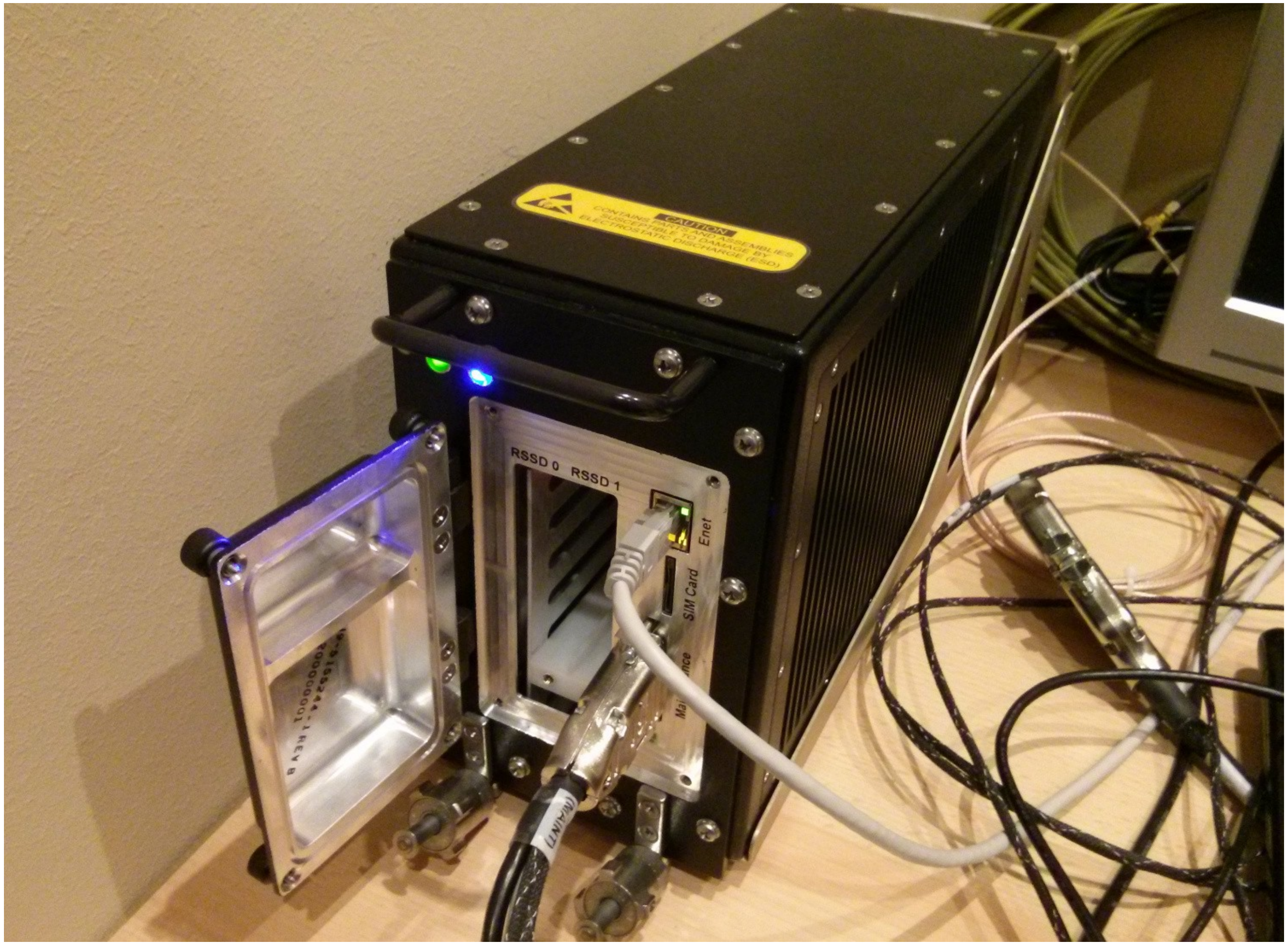
Databases:

PostgreSQL

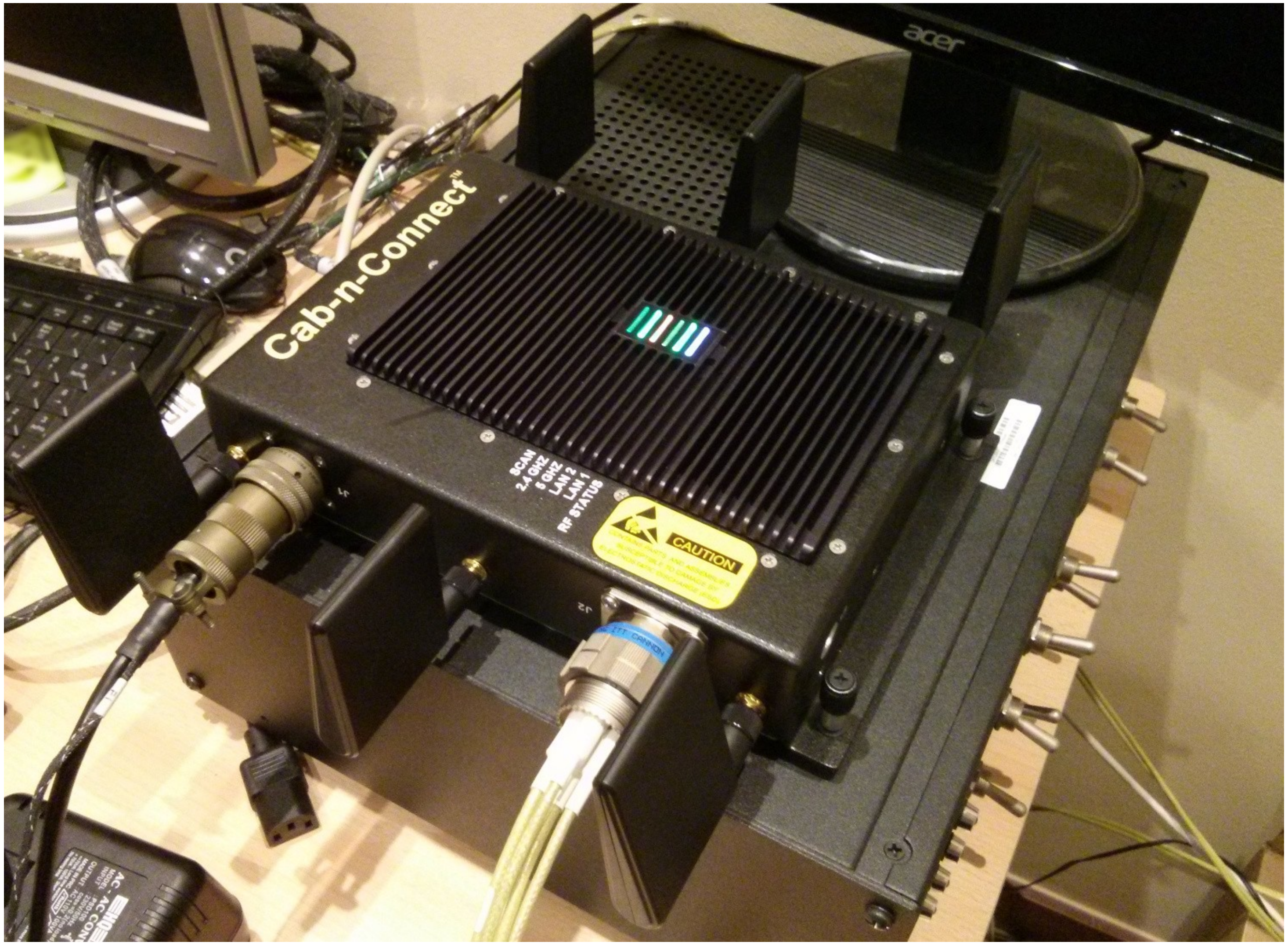
&

redis

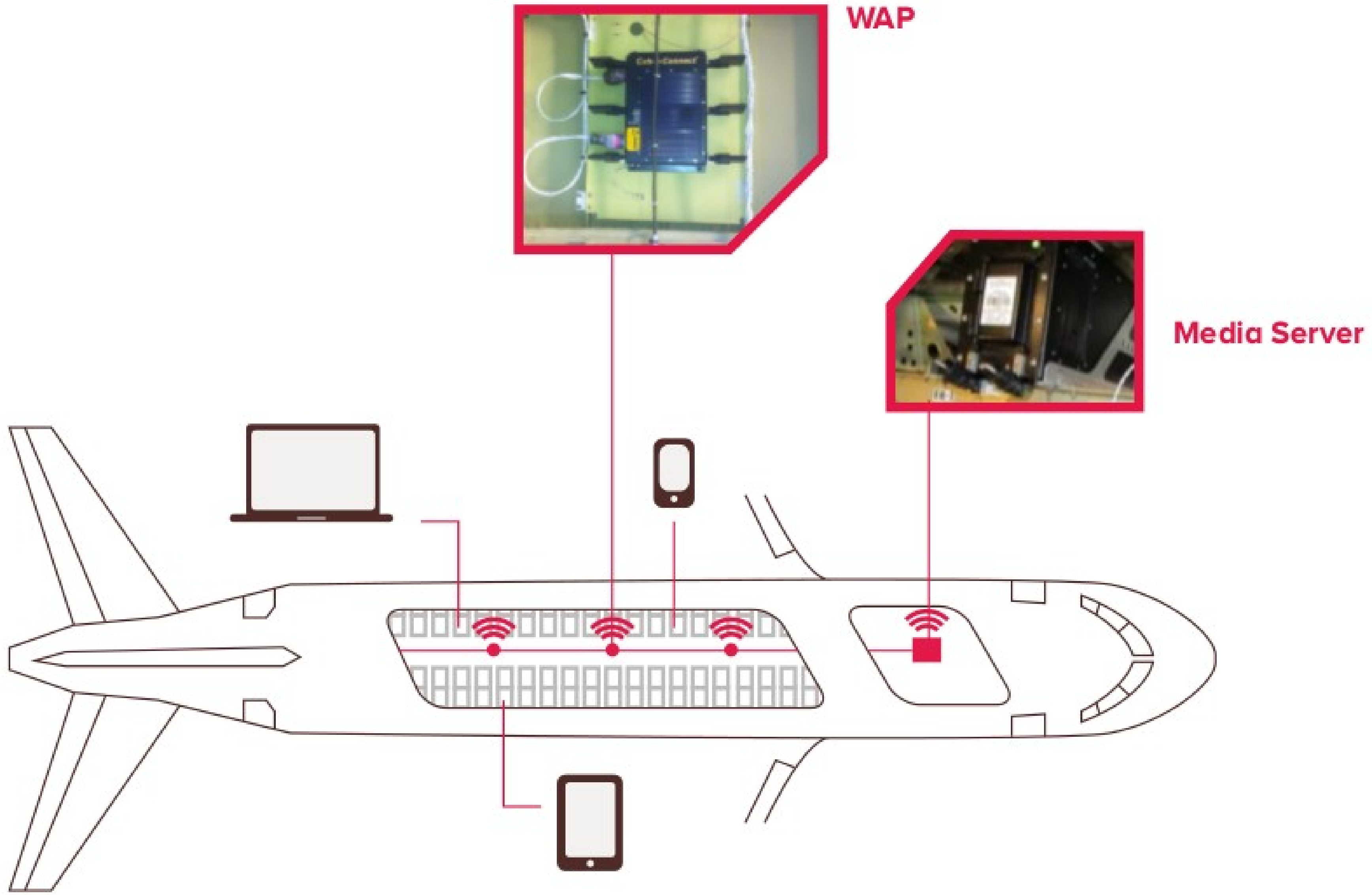
Embedded computer (lab)


CAUTION
CONTAINS PARTS AND ASSEMBLIES SUSCEPTIBLE TO DAMAGE BY ELECTROSTATIC DISCHARGE (ESD)
RSSD 0
RSSD 1
Enet
SIM Card
Maintenance

Access point + dashboard (lab)



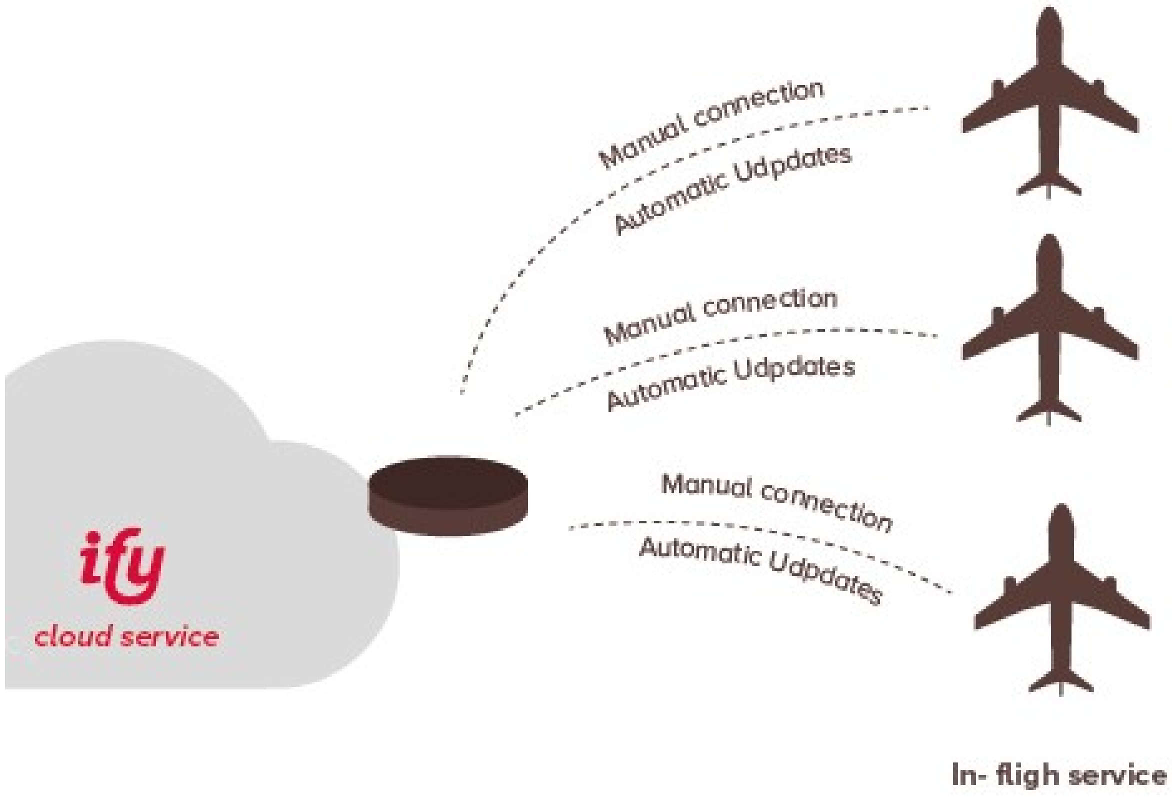
Hardware schema



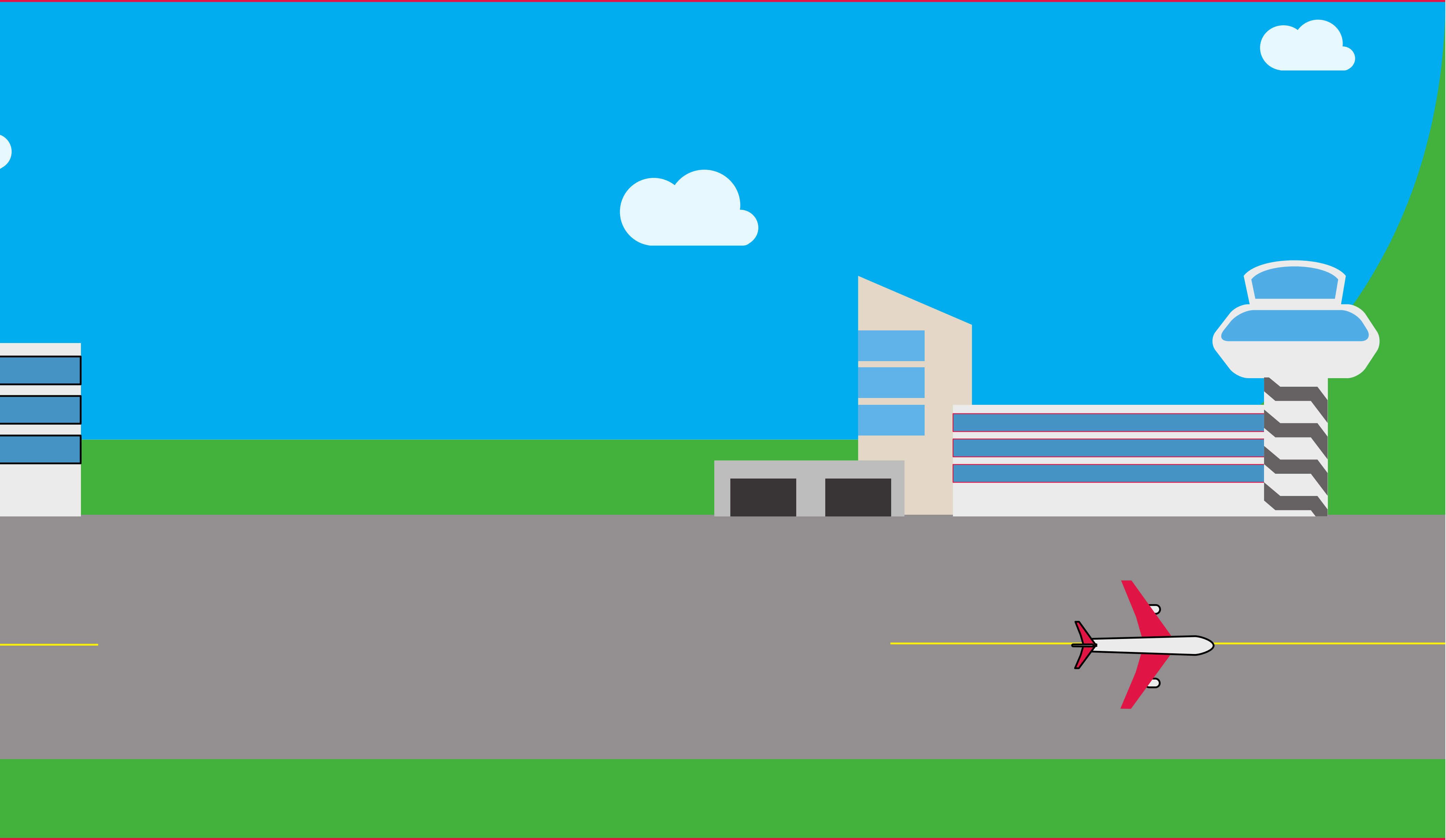
The three challenges

2) **Synchronize the aircrafts**

- Keep the replicas consistent
- Update the contents: *I don't want yesterday's newspaper!*



2) Synchronize the aircrafts



Context

- An Aircraft needs to update contents and usage
- During the flight, the Aircraft is **off-line**
- Once landed, it gets **connectivity**
- Then it tries to **synchronize**

Connectivity

- Internal 3G data card
  - Enabled when grounded
  - Disabled when about to flight
- VPN ("control tower")
  - SSH to the Hangar
  - Hangar can SSH back

The Hangar

- **Central point** of the platform
- Special instance, in the cloud
  - Same software (db schema)
  - Different apps and settings
- Used as a CMS, to manage all resources
  - `django.contrib.admin`
- All other instances just replicate some data from the Hangar

The Heartbeat

- An Aircraft sends **Heartbeats** when it's online
  - It's just a “`ping`” to the Hangar
  - A simple `POST`, with some details

- The **Hangar** stores the heartbeats
  - Knows what planes are online, grounded, at a given moment
  - Will send commands to those planes
    - In example: synchronize

Aircraft synchronization

- An Aircraft:
  - Sends usage data
    - New/updated users, surveys, payments, stats, etc
  - Gets usage data
    - New/updated users
  - Gets updated contents
    - New/updated resources, contents, offers, destinations, misc data...

Trouble?

- For some models (like Users), the Primary Keys **will collide**
- Can get conflicts
- May need to merge resources, edited in several Aircrafts at the same time

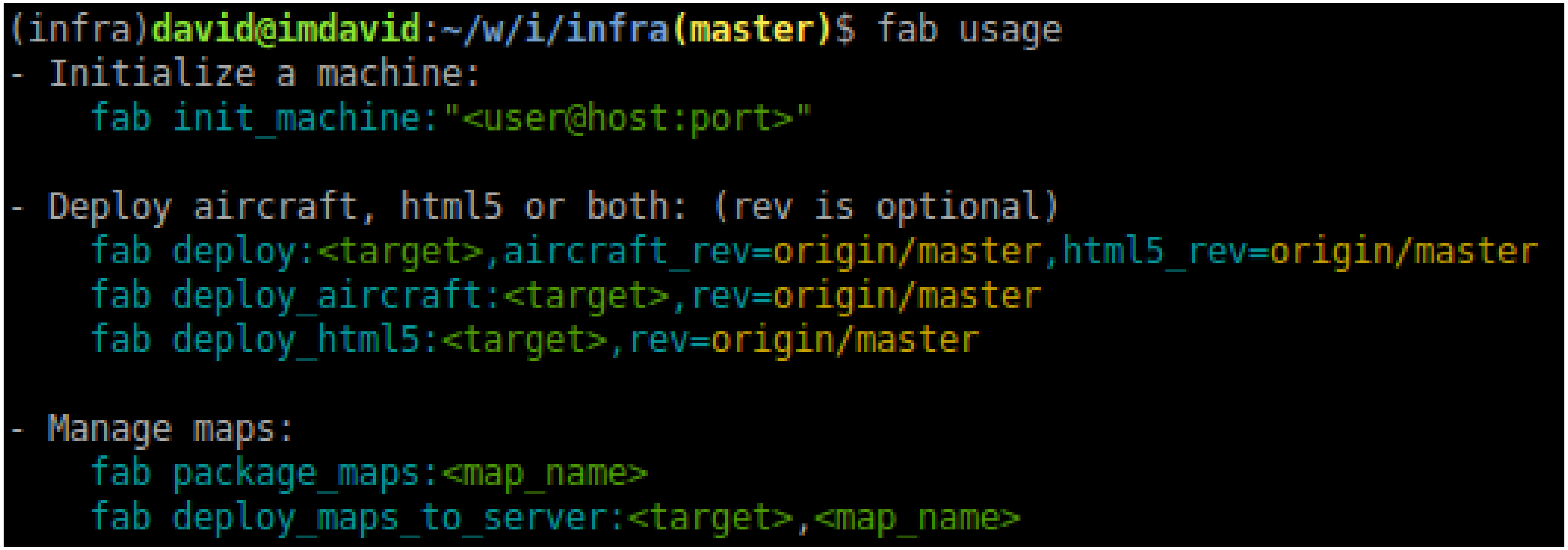
Prevent the trouble

- For some models (like Users), the Primary Keys **will collide**
  - Those PKs must be UUIDs
- Can get conflicts
  - Each model should be synchronized only in one direction: A->H or H->A
- May need to merge resources, edited in several Aircrafts at the same time
  - Make sure our logic doesn't allow that

fabric

Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks

http://fabfile.org



```
(infra)david@imdavid:~/w/i/infra(master)$ fab usage
- Initialize a machine:
    fab init_machine:"<user@host:port>"

- Deploy aircraft, html5 or both: (rev is optional)
    fab deploy:<target>,aircraft_rev=origin/master,html5_rev=origin/master
    fab deploy_aircraft:<target>,rev=origin/master
    fab deploy_html5:<target>,rev=origin/master

- Manage maps:
    fab package_maps:<map_name>
    fab deploy_maps_to_server:<target>,<map_name>
```

fabtools.require

fabtools includes useful functions to help you write your Fabric files.

Using fabtools.require allows you to use a more declarative style, similar to Chef or Puppet.

http://fabtools.readthedocs.org

Example:

```
# Require a PostgreSQL server
require.postgres.server()
require.postgres.user(name=user, password=password, createdb=True)
require.postgres.database(name=db, owner=user)
```

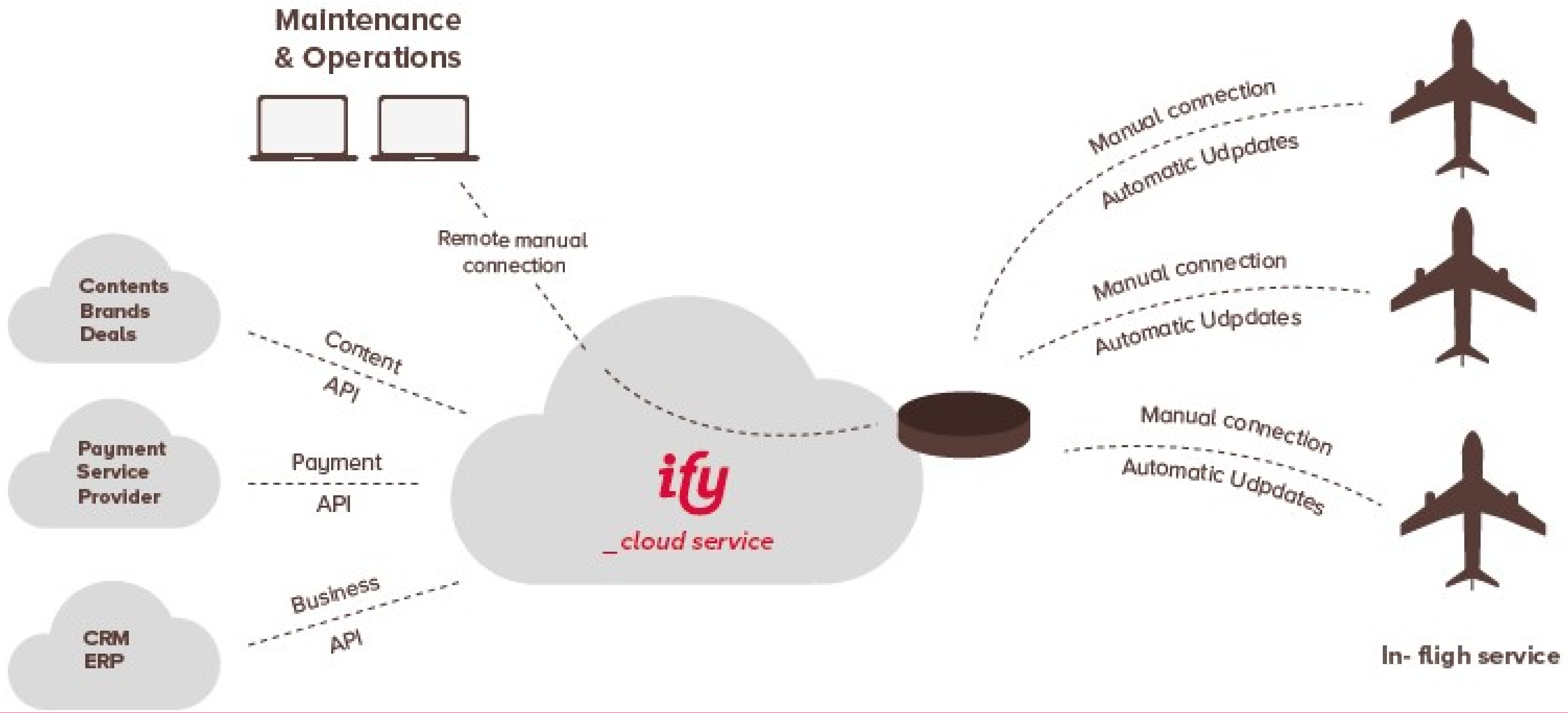
Other tools

- Python
  - `boto`: upload/download from S3
  - `requests`: API calls
  - `django.db.migrations`
- Unix
  - openssh
  - rsync
  - bash script

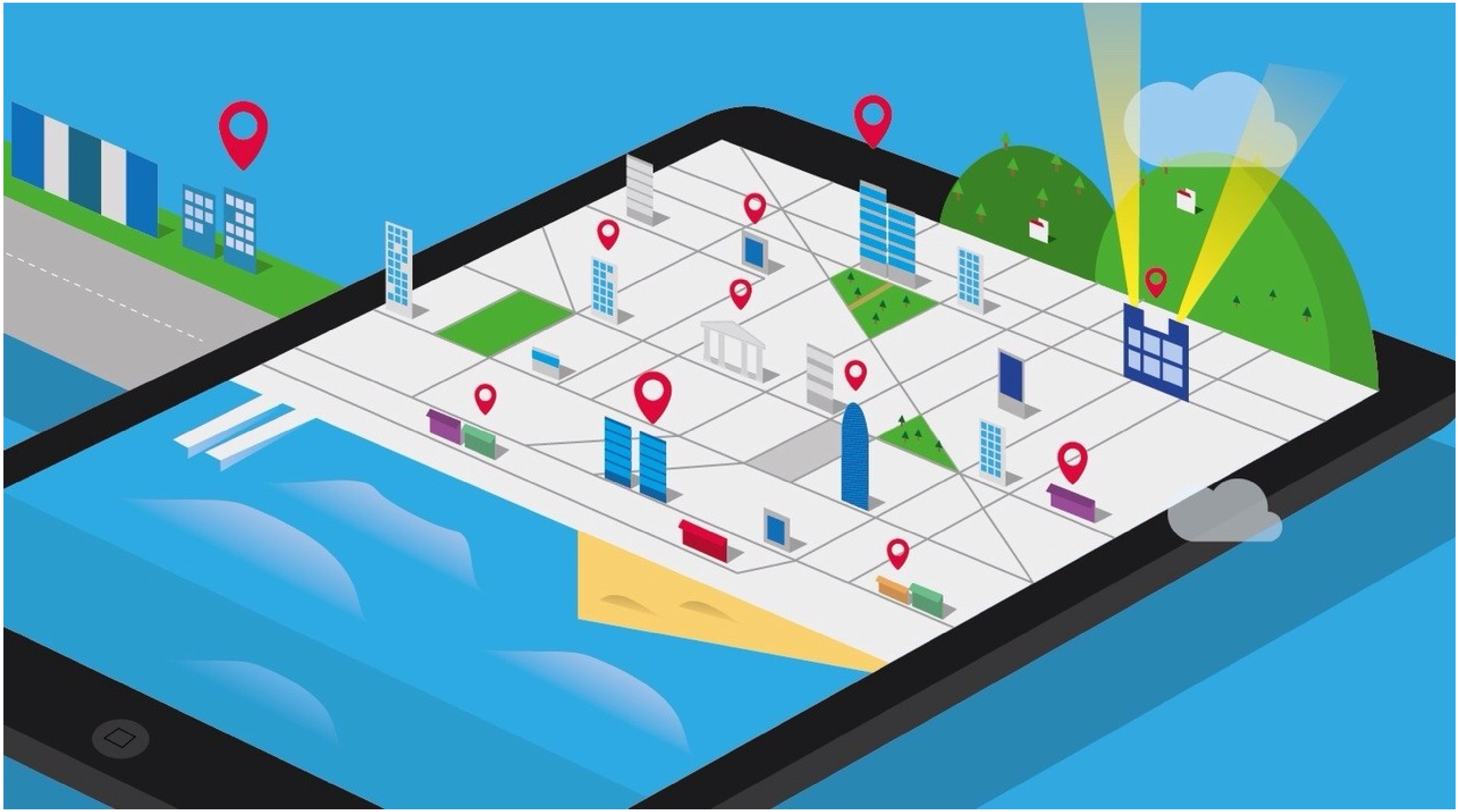
The three challenges

3) **Integrate with 3rd parties**

- Providers, payments, mailing, weather, flight info…



3) Integrate with 3rd parties



Highlights

- Resource Ingest
- API diversity
- AWS
- Celery
- Other APIs

Resource ingest

- Entertainment
  - Newspapers
  - Magazines
  - Videos
  - TV Shows
- Deals
  - Airport
  - Transport
  - Destination

- Lots of resources with lots of providers
  - Per language, country, etc

API diversity...

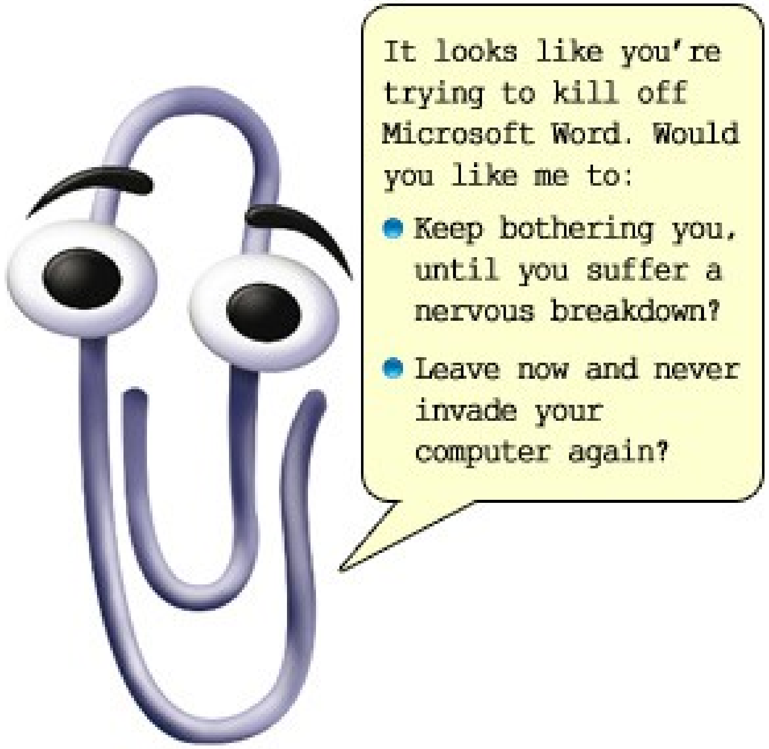
- Best case
  - Python API :)
- Good enough
  - REST API, lots of metadata, well documented
- Acceptable
  - S3 bucket or FTP, import all, but no metadata
- Worst case:
  - ?

Beware of the Intern-API!

https://www.flickr.com/photos/reidrac/2387432357/

Is this even a API?

- Excel document
  - (no format, of course)
- Word document
  - (why?)


It looks like you're trying to kill off Microsoft Word. Would you like me to:
Keep bothering you, until you suffer a nervous breakdown?
Leave now and never invade your computer again?

AWS, using boto

- ElasticTranscoder
  - *Media transcoding in the cloud*
- Simple Notification Service (SNS)
  - *A fast, flexible, fully managed push messaging service*
- Simple Storage Service (S3)
  - *Secure, durable, highly-scalable object storage.*

Celery task queue

- Hangar tasks use **Celery**:
  - *Celery is an asynchronous task queue/job queue based on distributed message passing*
  - Async
  - I/O intensive tasks: crawling APIs
  - CPU intensive tasks: thumbnails, billing pdfs

Flower: Celery monitoring tool

- *Flower is a web based tool for monitoring and administrating Celery clusters*
  - http://flower.readthedocs.org
- Real time
- Tip: a queue per task type

Other APIs

- Bookings
- Payments
- Billing
- Emails
- Weather
- Flight info
- Statistics
- Slack

Other libraries

- Images:
  - Pillow
  - django-imagekit
  - pilkit
  - PyPDF2
  - Wand
- PDFs:
  - django-easy-pdf
  - xhtml2pdf
- Other:
  - django-yubin
  - libsaas

Recap

- Solved the three challenges!
  - In-flight API with tons of features
  - When grounded, they get synchronized
  - The Hangar manages the full platform

Conclusions

- Python made it possible!
- Very versatile, covers all our use cases
- *"We stand on the shoulders of giants"*
- Developed in a short time

Thanks for attending!

Get the slides at http://slideshare.net/DZPM

Any questions?

Questions?

Thanks for attending!

Get the slides at http://slideshare.net/DZPM

Enjoy your flight! http://immfly.com