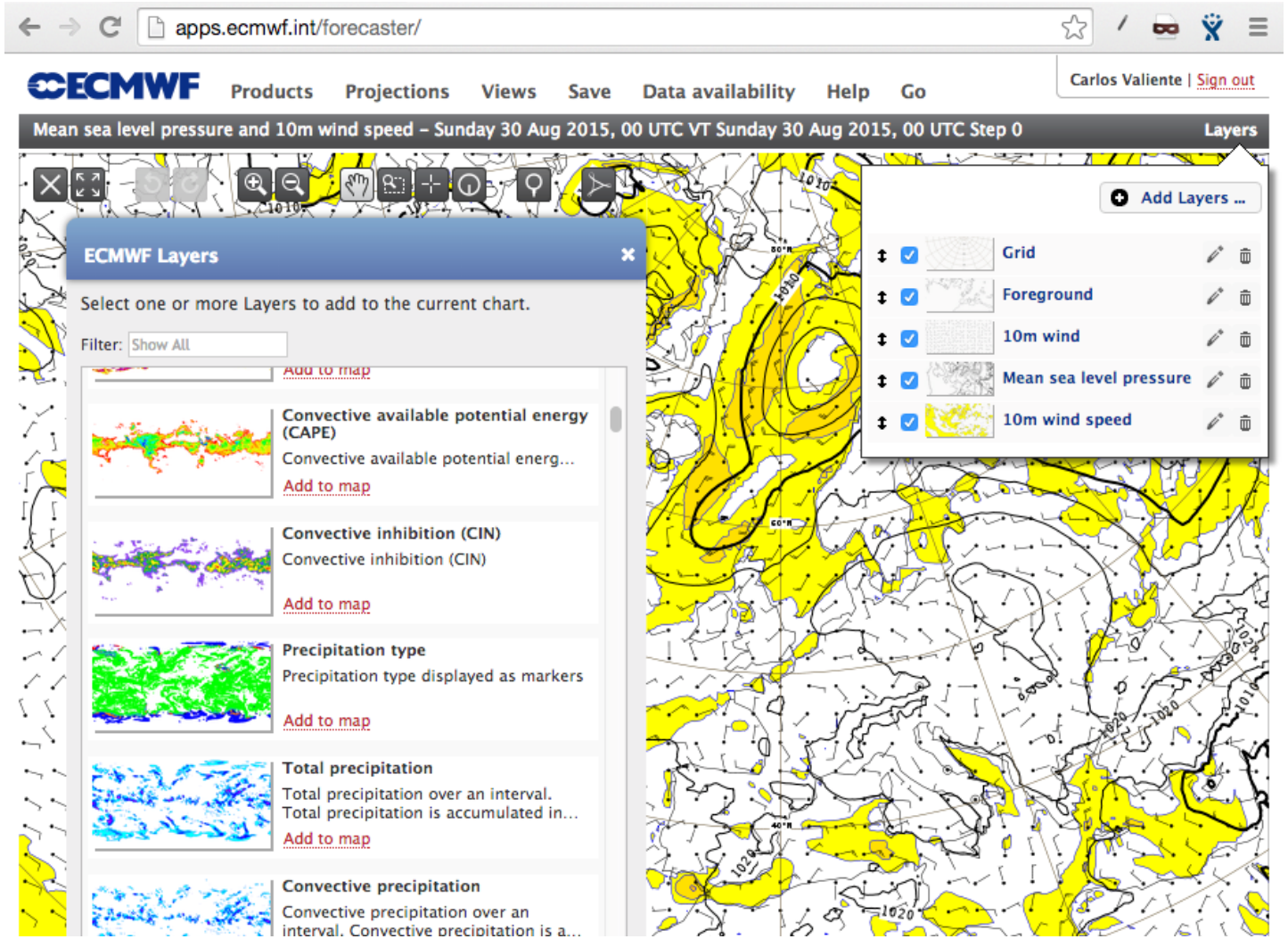
ecCharts: Behind the scenes

The Web people at ECMWF <eccharts-support@lists.ecmwf.int>

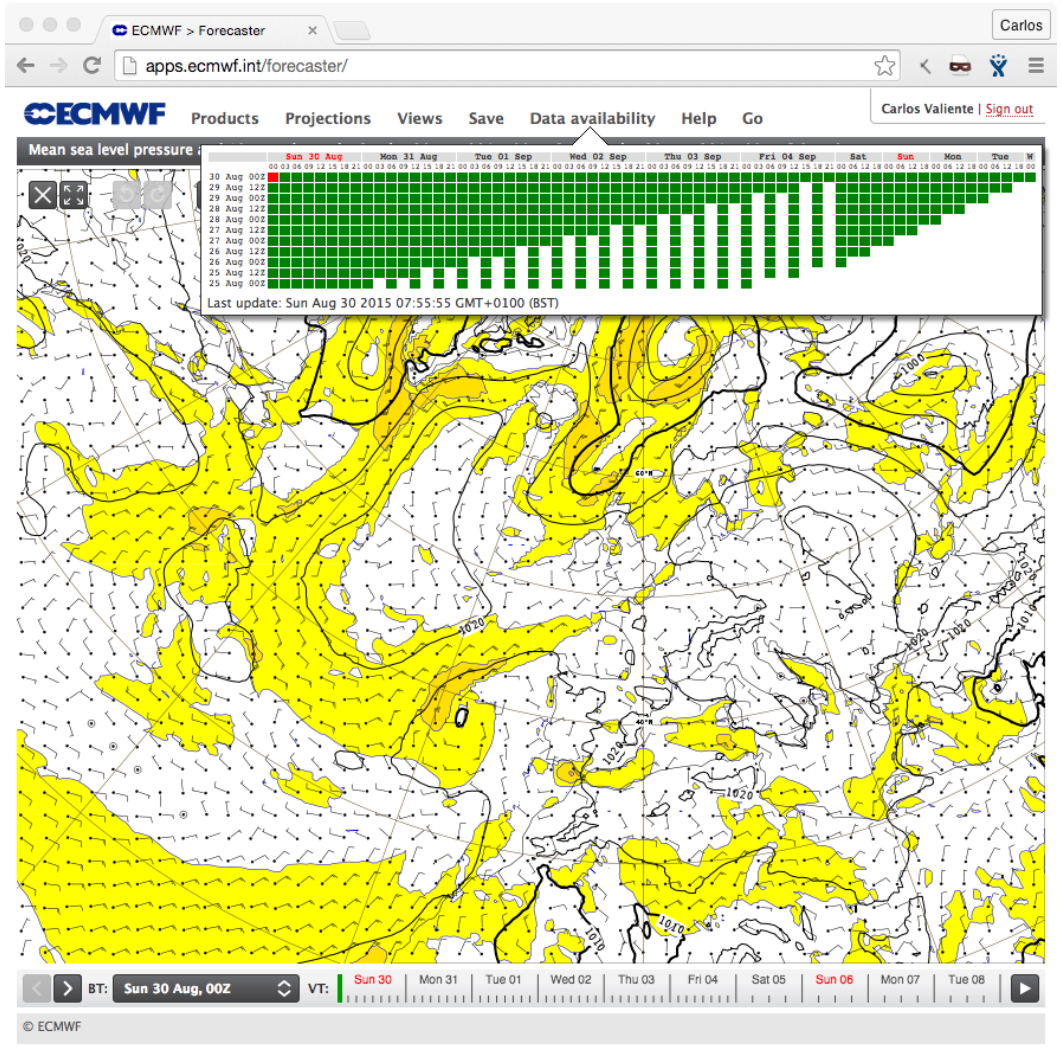
ECMWF

© ECMWF October 1, 2015

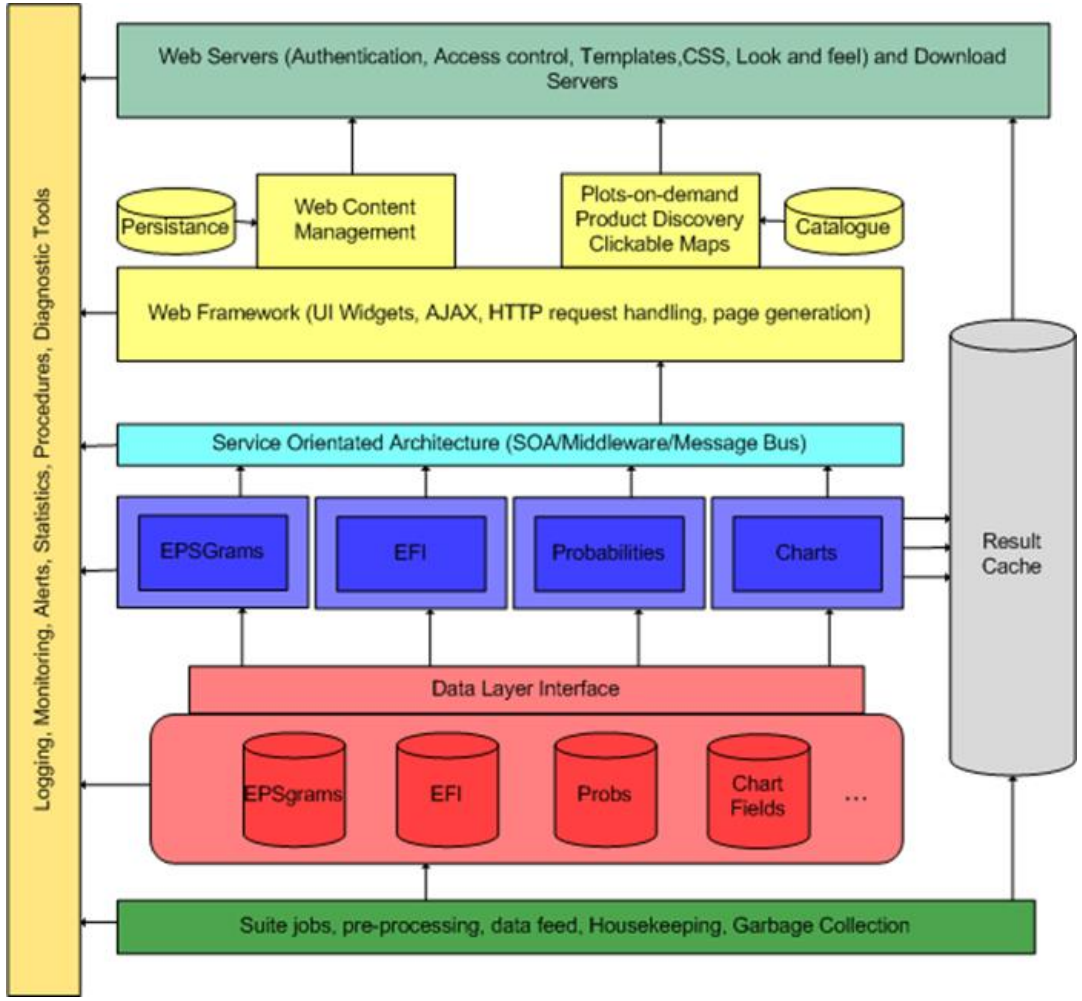
ecCharts: What users see (1)



ecCharts: What users see (2)



ecCharts: What users do **not** see (when we get it right!)



ecCharts: Servers and operating systems

~500 CPUs

~1.5 TB memory

~200 TB disk

All running Linux (SuSE SLES 11 Service Pack 3)

Python 2.7 everywhere

ecCharts: The front-end layer

- Two **hardware load balancers** for distributing HTTP/HTTPS requests to ..
- … several **Varnish** HTTP caches (HTTP traffic)
  - Very robust, used mainly for load-balancing to lower layers, helps with caching
- … and several **Apache** instances (HTTPS traffic)
- Several **Nginx** HTTP servers for streaming large static files (i.e. plots)

ecCharts: The web applications layer

- Several **Django** instances, taking care of:
  - Access control
  - Parsing user requests from the Javascript
  - Dispatching requests to the services layer below (see next slides)
- Our usage of Django is very lightweight
  - Most UI work is done in the Javascript code above, and most data processing is done in the services layer below.

ecCharts: The (micro)services layer

- A **microservices** architecture, based on **Celery**.
- Using **RabbitMQ**, an AMQP request broker for dispatching jobs …
- … and **Redis**, a key-value store, for storing job results.
- **Services** written in **Python**.
- Lots of **caching** everywhere, using **Memcached**.

ecCharts: The (micro)services layer (2)

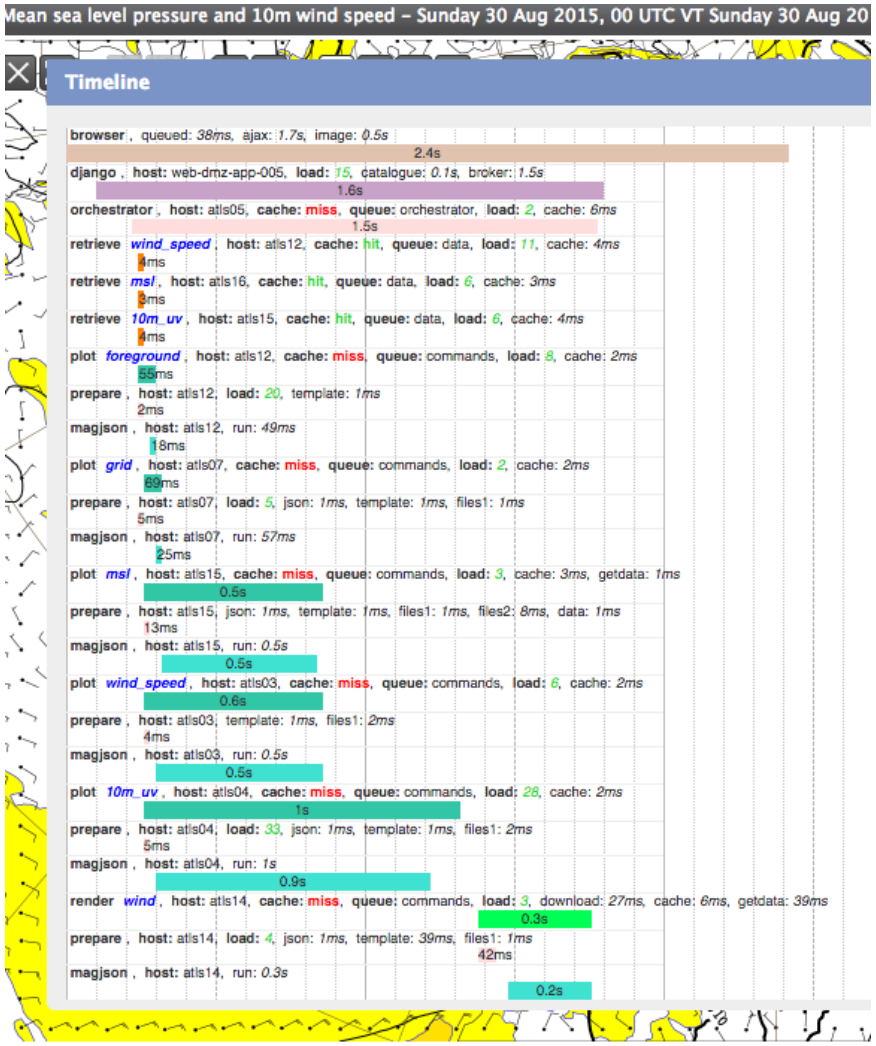
```
# The inevitable `echo` and `sum` services
from servicelib import errors, start_services

def echo_service(context, *args):
    context.log.debug("Executing echo() request from: %s", context.user)
    return " ".join(args)

def sum_service(context, *args):
    try: args = [(a) for a in args]
    except:
        raise errors.BadRequest("Invalid args: %s" % (args,))
    return sum(args)

if __name__ == "__main__":
    start_services({"name": "sum", "execute": "sum_survice"},
                   {"name": "echo", "execute": "echo_service"})
```

ecCharts: The (micro)services layer (3)



ecCharts: The (micro)services layer (4)

```
from metview.macro import retrieve, sqrt

def wind_speed(r):
    if r['levtype'] == 'sfc':
        u = '165.128‘
        v = '166.128‘
    else:
        u = '131.128‘
        v = '132.128‘
    r['param'] = u
    u = retrieve(r)
    r['param'] = v
    v = retrieve(r)
    return sqrt(u * u + v * v)
```

ecCharts: The data layer

- A **distributed file storage** service written in **Python**
  - Content is addressable as HTTP URLs by all services
  - New content pushed with HTTP PUT requests
- Content is indexed in a **MongoDB** cluster

```
> db.fields.findOne()
{ "param" : "msl",
  “base_time": ISODate("2013-04-08T00:00:00Z"),
  ..
  "locations" : [
    {"url": "http://host42.ecmwf.int/data0000.grib",
     "offset": 0, "length": 4158 }
]}
```

ecCharts: The data layer (2)

- Twice a day:
  - We push **~250 GB** of **GRIB data**
  - We insert **~250,000 records** in the **MongoDB** indexes
  - We remove ~250 GB of old GRIB data
  - We remove ~250,000 old records from the MongoDB indexes

Life gets interesting at times …

ecCharts: The release process

We do releases **every two weeks or so**.

The more frequent, the better!

We have **two identical clusters** of servers: production and pre-production

Release candidate **tested** in a **pre-production** cluster

On release day**, production traffic** is **redirected** to pre-production cluster, and vice-versa.

Usual **downtime**: **15 minutes**

Should be transparent!

ecCharts: DevOps

Two different teams: Development and Operations

We **talk** to each other

Questions?