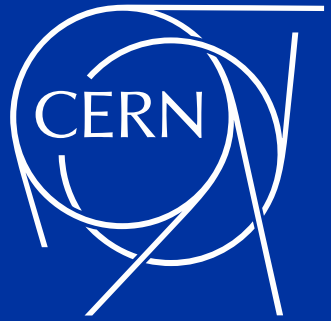


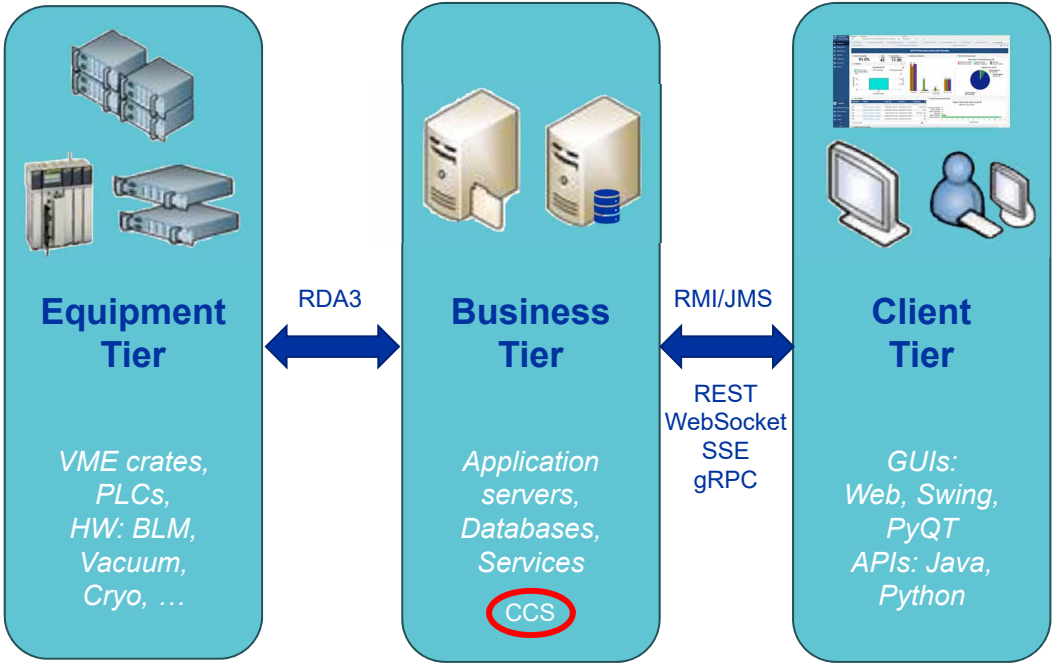

# Developing Modern High-Level Controls APIs

**Bartek Urbaniec**

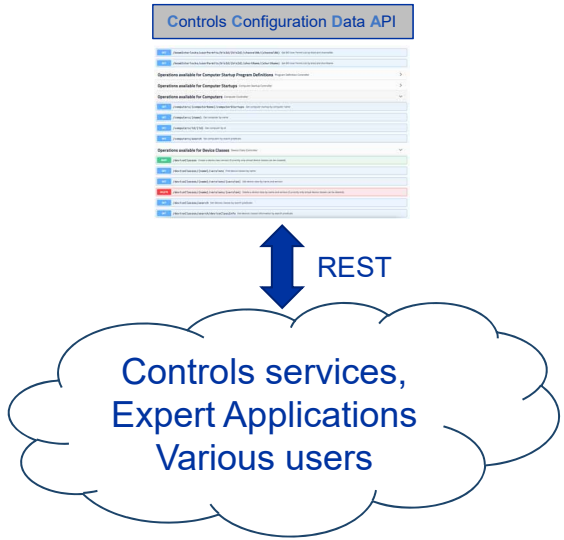
ICALEPCS 2023

# APIs in the Accelerator Control System

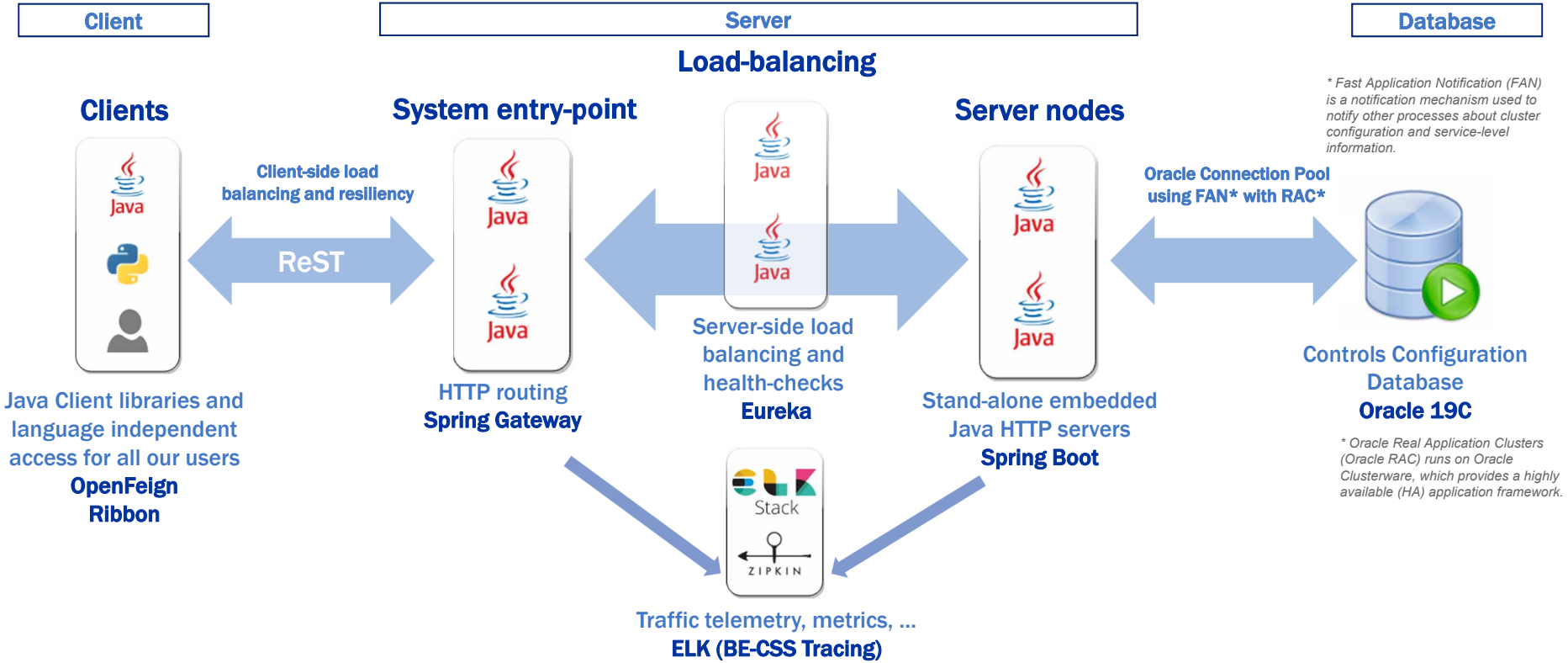
CERN Controls system services rely on a variety of APIs




The main purpose of the **Controls Configuration Service** (CCS) is to **unite and centralize** all the **information** relevant to the **Control systems** (CS) in such a way that integration between various Control sub-systems is consistent and efficient.



# CCDA - the Controls Configuration API at glance



# CCDA in numbers

- Up to **80 milion** requests per day – coming from more than **400 hundred** different users, services and applications


- Availability this year 99.999% (1 hour of unplanned downtime over 9 months) – upgrades and releases are transparent due to multiple nodes and client/server balancing
- The API provides operations for **15 different controls domains** and more than **500 DB tables**
- More than **75k lines of code** – no boilerplate code due to usage of code generation libraries

# Monitoring and alerting

- notifies us about all anomalies – helps to prevent system downtime
- allows to improve quality of service – problems are fixed before users spot them



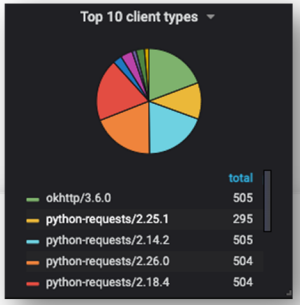


**Instance is DOWN** (0 active)

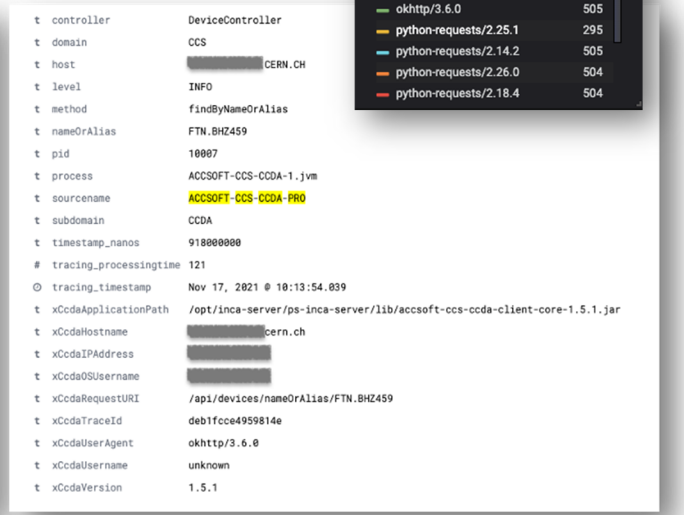
```
alert: Instance
  is DOWN
expr: up == 0
for: 1m
labels:
  severity: CRITICAL
annotations:
  description: '{{ $labels.instance }} of job {{ $labels.job }} has been down for
    more than 1 minute.'
  summary: Instance {{ $labels.instance }} is down
```

# Telemetry

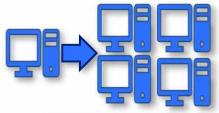
- informs us about usage of our API – who, how, when
- allows us to analyse and discover anomalies of a running system

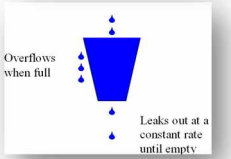






# High-throughput enablers






- **Horizontal scaling**
  *adds resources to handle more client requests*
- **Caching mechanism**
  *(more on the next slide)*

PoC **Client throttling** (e.g. Leaky bucket algorithm)
*limits misbehaving clients by rejecting requests*

PoC **Circuit breaker**
*stops requests in case of consecutive errors*

# To cache or not to cache

**The premise of a cache -** to store and provide already processed data

## Benefits:

- limits unnecessary IOs → in many systems, physical IOs are the slowest operations
- load on the server and related services is reduced to minimum → output/data is processed only once

## Drawbacks:

- increased complexity of the system (embedded cache vs standalone)
- eviction strategy* – especially difficult for complex system with mutable data
- challenge of consistency in distributed systems – every client should see the same state of cached data
  → there are solutions: e.g.: distributed cache like Apache Ignite, Redis

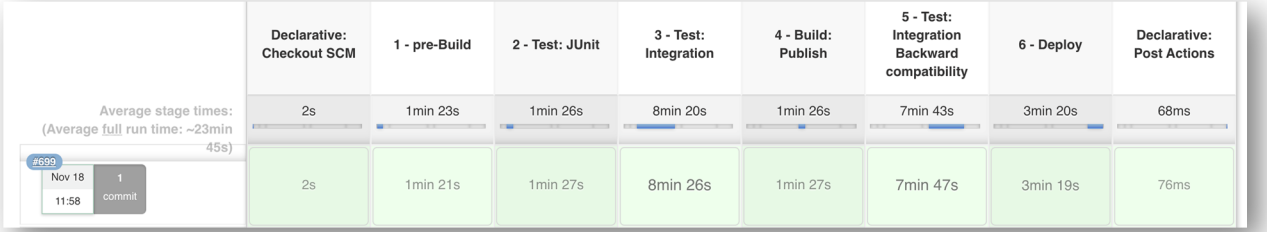
Cache eviction strategies:
- by time, e.g.: after n-seconds (TTL)
- by access frequency, e.g.: after 10k cache reads
- on-demand, e.g.: explicit cache purge
- by size, e.g.: max no. of elements in the cache

CCDA responses reduced by factor of 10
*query for an accelerator: 40ms -> 4ms*

**Caching** is not a silver bullet for performance issues

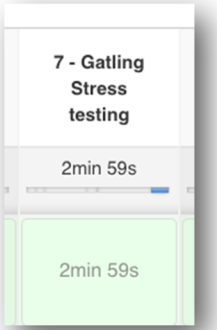
# How to verify our system - testing

- **Unit, integration, regression tests – availability and reliability**

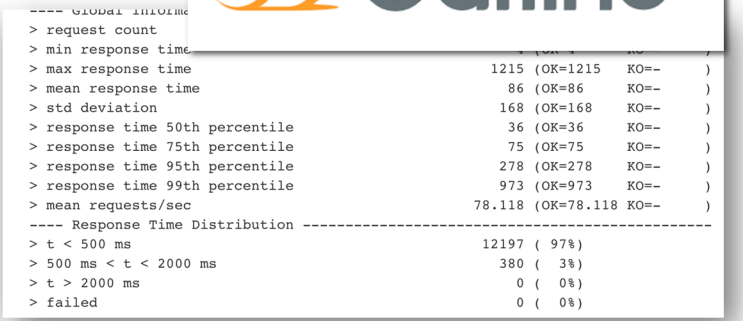


| | Declarative: Checkout SCM | 1 - pre-Build | 2 - Test: JUnit | 3 - Test: Integration | 4 - Build: Publish | 5 - Test: Integration Backward compatibility | 6 - Deploy | Declarative: Post Actions |
|---|---|---|---|---|---|---|---|---|
| Average stage times: (Average full run time: ~23min 45s) | 2s | 1min 23s | 1min 26s | 8min 20s | 1min 26s | 7min 43s | 3min 20s | 68ms |
| #699 Nov 18 11:58 1 commit | 2s | 1min 21s | 1min 27s | 8min 26s | 1min 27s | 7min 47s | 3min 19s | 76ms |

- **Stress testing – performance**



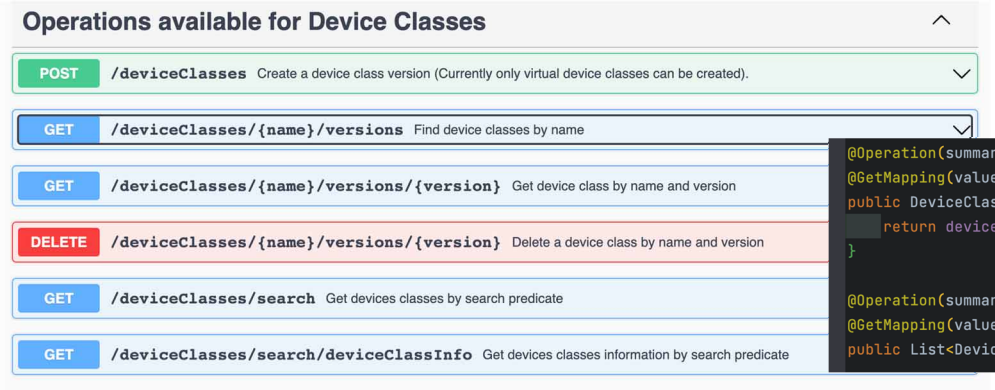

```
================================================================================
2021-10-14 10:43:10                                      161s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                  (OK=12577  KO=0     )
> findSelectors                                           (OK=611    KO=0     )
> findFesaFieldValuesByName                               (OK=566    KO=0     )
> findDevicesByQuery                                      (OK=1196   KO=0     )
> findAcceleratorByName                                   (OK=1210   KO=0     )
> findDeviceClassesByQuery                                (OK=1220   KO=0     )
> findAllAccelerators                                     (OK=1210   KO=0     )
> findDeviceByName                                        (OK=1802   KO=0     )
> findDeviceClassVersionsByName                           (OK=1781   KO=0     )
> findDeviceClassVersionsByNameAndVersion                 (OK=1781   KO=0     )
> findComputerByName                                      (OK=1200   KO=0     )
```



```
> request count                                          12577 (OK=12577  KO=-     )
> min response time                                          1 (OK=1      KO=-     )
> max response time                                       1215 (OK=1215   KO=-     )
> mean response time                                        86 (OK=86     KO=-     )
> std deviation                                            168 (OK=168    KO=-     )
> response time 50th percentile                             36 (OK=36     KO=-     )
> response time 75th percentile                             75 (OK=75     KO=-     )
> response time 95th percentile                            278 (OK=278    KO=-     )
> response time 99th percentile                            973 (OK=973    KO=-     )
> mean requests/sec                                     78.118 (OK=78.118 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 500 ms                                             12197 ( 97%)
> 500 ms < t < 2000 ms                                     380 (  3%)
> t > 2000 ms                                                0 (  0%)
> failed                                                     0 (  0%)
```

# Usability and documentation

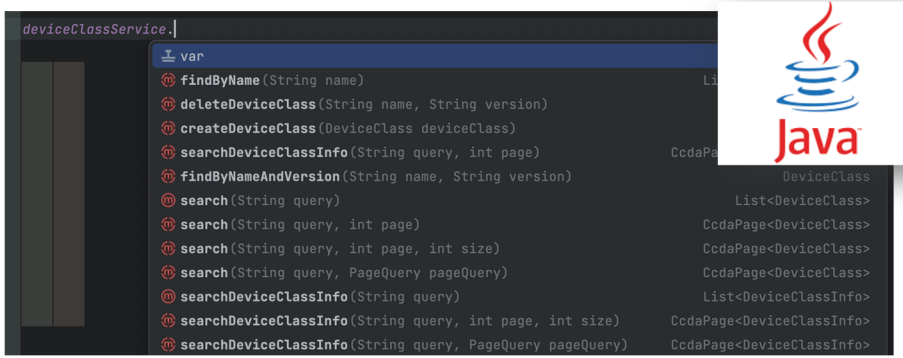
Swagger library is used to autogenerate user documentation in OpenAPI specification

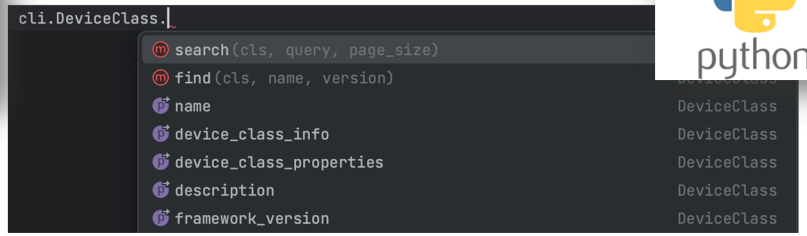



```java
@Operation(summary = "Get device class by name and version")
@GetMapping(value = "/{name}/versions/{version}")
public DeviceClass findByNameAndVersion(@PathVariable String name, @PathVariable String version) {
    return deviceClassService.findByNameAndVersion(name, version);
}

@Operation(summary = "Find device classes by name")
@GetMapping(value = "/{name}/versions")
public List<DeviceClassInfo> findByName(@PathVariable String name) { return deviceClassService.findByName(name); }
```

Client SDKs (Java + Python) with autocompletion



# Summary

- **After 5+ years of CCDA in production, we are satisfied with our choices:**
  REST, Spring Boot, Spring Cloud, stateless nodes, client/server load balancers
- **It pays off to utilize industry standards where possible:**
  focus on your domain and API usability; most technical problems have ready to use solutions off-the-shelf
- **Start with language specific SDKs and documentation as a code (Swagger):**
  limits proliferation of custom solutions and API wrappers among the users
- **Design of a robust and scalable API is a long lasting and iterative story:**
  next steps for CCDA is k8s and circuit breakers