

# DIRACX

DEMO AND TECHNICAL DETAILS

# WHAT DOES "TOKENS" EVEN MEAN?

- There are three things that we care about
  - Submitting pilots
  - Data access
  - Verifying a user's identity
- For DIRAC - we only care about verifying identity
  - Can't keep using certificates as it's getting harder to find an issuer
- Once we have an identity, it's purely internal to DIRAC
  - Should be as simple as logging in to any website

# HOW DO WE GET AN IDENTITY?

- Currently “getting identity” means:
  1. Find someone trusted to give you a certificate
  2. Import the certificate to your web browser
  3. Ask for someone to help you to figure out why it didn't work
  4. Sign AUP in VOMS
  5. Import the certificate ~/.globus
  6. Upload the certificate to DIRAC
  7. Do the whole dance again ~once a year (new PC, randomly)

# HOW DO WE GET AN IDENTITY?

- Currently “getting identity” means:
  - ~~1. Find someone trusted to give you a certificate~~
  - ~~2. Import the certificate to your web browser~~
  - ~~3. Ask for someone to help you to figure out why it didn't work~~
  4. Sign AUP in VOMS
  - ~~5. Import the certificate ~/.globus~~
  - ~~6. Upload the certificate to DIRAC~~
  - ~~7. Do the whole dance again once a year (new PC, randomly)~~
  
- Reduce this to just a login and users don't care how

# HOW DO WE MAKE THIS CHANGE?

- DIRAC is old and is filled with technical debt
  - Attempts to make major change now systematically fail (OAuth2, HTTPS)
- DIRAC was very well thought out with a solid foundation
  - Wasn't clear what a "Grid" even was when it started
- DiracX is a new approach, learning from the past 20 years
  - Learn from 20+ years of DIRAC
  - Tens of years of developer experience
- So what will using DIRAC look like during the transition?

# THE PATHWAY

- Currently we use DIRAC with only X509
- Use both X509 and tokens for a while
  - X509 == DIRAC
  - Tokens == DiracX
- **Users always have both a token and a proxy**
- Services slowly move from DIRAC -> DiracX
- Eventually tokens + DiracX are the only thing left

# THE PATHWAY

- Currently we use DIRAC with only X509
- Use both X509 and tokens for a while
  - X509 == DIRAC
  - Tokens == DiracX
- **Users always have both a token and a proxy**
- Services slowly move from DIRAC -> DiracX
- Eventually tokens + DiracX are the only thing left

**This will be a multi-year transition**

Users shouldn't need to care what proxy or token is





# HOW WILL THIS LOOK?

If the demo gods are favourable, I'll hopefully now show:

- `dirac-proxy-init`
- Submit a job
- Show in web app
- Job monitoring CLI
- Download output sandbox



# WHAT DID YOU JUST SEE?

- `dirac-proxy-init` ←  **X509+DIRAC**
  - Used certificate to authenticate
  - DIRAC also returned a DiracX token
- Submit a job ←  **X509+DIRAC**
  - Used the existing DIRAC JobManagerHandler
- Show in web app
- Job monitoring CLI ←  **Token+DiracX**
  - Transparently used DiracX via legacy adapter mechanism (see later)
- Download output sandbox ←  **Token+DiracX**
  - Stored in the new DiracX sandbox store

**Nothing changes for users**

# THE PHILOSOPHY OF THE DIRAC TOKEN MIGRATION

- We need to move to tokens for external constraints
- The migration needs to be smooth
- It's a massive amount of work
  
- The current DIRAC design is 20+ years old
  - Lots of expertise in what we (don't) need to be able to do
  - Held back from improving by what we've inherited
- For years the groundwork has been laid for modernising
- Let's take advantage of this opportunity!

# SO WHAT WOULD WE WANT TO IMPROVE?

Make authentication transparent to users (no certificate errors)

Later we'll have a talk about the security model.

For this rest of this talk we don't care how authentication is done.

# SO WHAT WOULD WE WANT TO IMPROVE?

Simpler interfaces and clearer errors

- Communicating errors is critical
  - Not sustainable to have experts solving every problem

```
Failed to create catalog objects
Traceback (most recent call last):
  File ".../bin/dirac-dms-lfn-replicas", line 8, in <module>
    sys.exit(main())
    ^^^^^^
  File ".../DIRAC/Core/Base/Script.py", line 74, in __call__
    return entrypointFunc._func()
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File ".../LHCbDIRAC/DataManagementSystem/scripts/dirac_dms_lfn_replicas.py", line 43, in main
    exit(executeLfnReplicas(dmScript))
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File ".../LHCbDIRAC/DataManagementSystem/Client/ScriptExecutors.py", line 794, in executeLfnReplicas
    return printLfnReplicas(lfnList, active=active, diskOnly=diskOnly, preferDisk=preferDisk, forJobs=forJobs)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File ".../LHCbDIRAC/DataManagementSystem/Client/ScriptExecutors.py", line 806, in printLfnReplicas
    res = dm.getReplicas(lfnList, active=active, diskOnly=diskOnly, preferDisk=preferDisk)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File ".../DIRAC/DataManagementSystem/Client/DataManager.py", line 1667, in getReplicas
    res = self.fileCatalog.getReplicas(lfnChunk, allStatus=allStatus)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File ".../DIRAC/Resources/Catalog/FileCatalog.py", line 158, in __getattr__
    raise AttributeError
AttributeError
```

*No proxy?*



# SO WHAT WOULD WE WANT TO IMPROVE?

First class Multi V0 support

- Current Multi-V0 support is an afterthought
- Single V0 = Multi V0 with only one V0 configured
  - “Could merge any two installations”

# SO WHAT WOULD WE WANT TO IMPROVE?

More flexibility (e.g. access via HTTPS without a DIRAC client)

- Also less flexibility
  - DIRAC was designed with lots of “flexibility”
  - Eventually paralyzes development
- Every community has some special needs
  - Standards like OAuth, HTTPS, REST make it easier to accommodate
- Make sure we can follow external changes

# SO WHAT WOULD WE WANT TO IMPROVE?

## More stable releases

- DIRAC itself is very stable
  - Has a lot of failover mechanisms
  - Gracefully degrades when things fail
- Changing the code is a different story (but improving)
  - Design the entire package to be testable
  - More robustness to mistakes



# SO WHAT WOULD WE WANT TO IMPROVE?

## Simpler installation and configuration

- Turn key solution
  - Trivial to run a development instance locally
  - Easy for a sysadmin to get a production instance up and running
- Guided upgrade path between versions, should tell you
  - DB changes
  - Configuration changes
  - Deployment changes
- Ideally changes are automated (or wizard-like)

# SO WHAT WOULD WE WANT TO IMPROVE?

Easier to maintain extensions (especially for the webapp)

**“We worry about catching all the changes”**

- Extensions are currently tightly coupled to DIRAC
  - Can modify just about anything
  - Sometimes even overriding private methods!
  - Not sustainable
- Need a clearer interface of what is extendable
- Make a smoother path to maintain extensions by design

# SO WHAT WOULD WE WANT TO IMPROVE?

More accessible to new developers

- Our fields have a strong bias towards
  - inexperienced developers
  - short contracts
- Needs to be as accessible as possible
- A 4 month intern should be able to do something useful

# SO WHAT WOULD WE WANT TO IMPROVE?

- Make authentication transparent to users (no certificate errors)
- Simpler interfaces and clearer errors
- First class Multi VO support
- More flexibility (e.g. access via HTTPS without a DIRAC client)
- More stable releases
- Simpler installation and configuration
- Easier to maintain extensions (especially for the webapp)
- More accessible to new developers



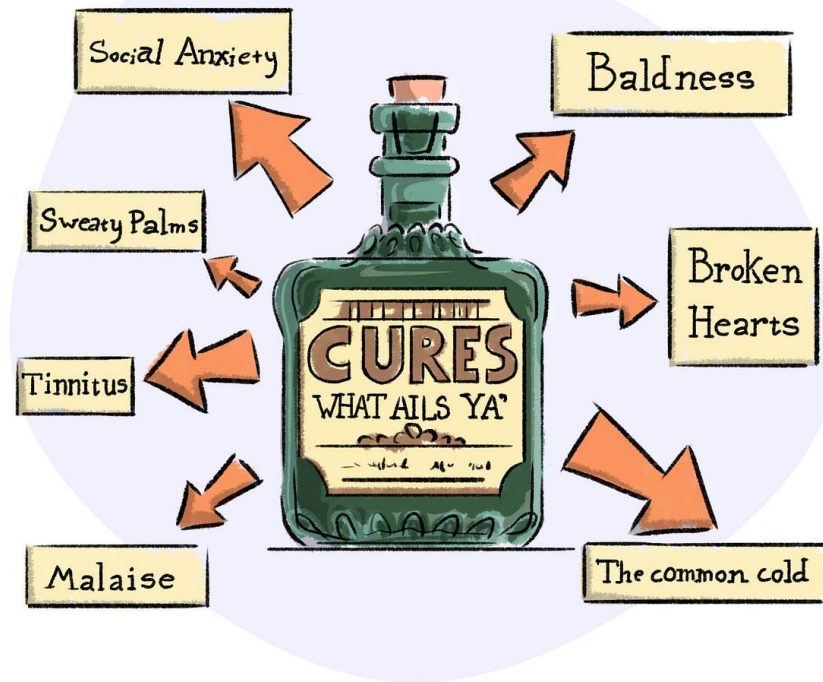
# DIRACX IS NOT THE ANSWER TO EVERYTHING

I think we can get most of these  
aforementioned desires as part  
of DiracX

But:

- It will take a while
- People need to learn new things

## PANACEA



# HOW MIGHT THE FINAL DIRACX LOOK?

Demo gods be willing, same actions as before

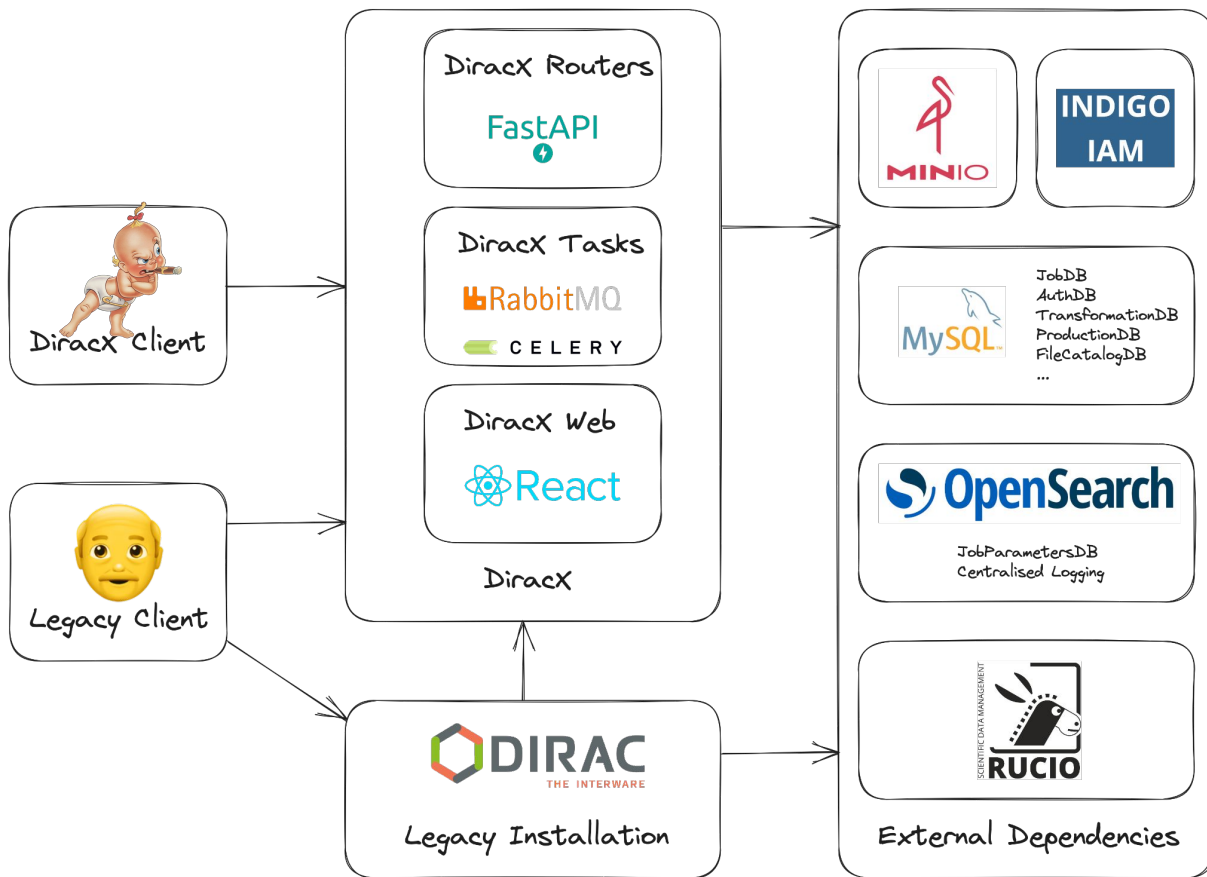
- Login
- Submit a job
- Show in web app
- Job monitoring CLI

# CURRENT STATUS

- DiracX is still immature
  - The demo can be ran standalone, but...
  - It will take a long time before it's approaching completeness
  - E.g. pilot submission isn't going to happen "soon"
- The priority is groundwork and legacy compatibility
- Second priority is user facing components
- The "interface" (CLI, web, python, REST) is experimental
  - Not yet had much effort invested



# ARCHITECTURE



# FASTAPI



<https://fastapi.tiangolo.com/>

High performance framework and widely used at scale

Designed for easy prototyping and development

Removes a lot of low level code and boilerplate

Standards based

*"[...] I'm using **FastAPI** a ton these days. [...] I'm actually planning to use it for all of my team's **ML services at Microsoft**. Some of them are getting integrated into the core **Windows** product and some **Office** products."*

Kabir Khan - **Microsoft** (ref)

*"We adopted the **FastAPI** library to spawn a **REST** server that can be queried to obtain **predictions**. [for Ludwig]"*

Piero Molino, Yaroslav Dudin, and Sai Sumanth Miryala - **Uber** (ref)

*"**Netflix** is pleased to announce the open-source release of our **crisis management** orchestration framework: **Dispatch!** [built with **FastAPI**]"*

Kevin Glisson, Marc Vilanova, Forest Monsen - **Netflix** (ref)

# SWAGGER

The screenshot displays the Swagger UI interface for the Dirac API. At the top right, there is an 'Authorize' button with a lock icon. The main content is organized into two sections: 'auth' and 'jobs'. Each section contains a list of endpoints with their respective HTTP methods, paths, and descriptions. The endpoints are color-coded: blue for GET, green for POST, and red for DELETE. Each entry has a dropdown arrow on the right side.

Method	Path	Description
GET	/auth/{vo}/device	Do Device Flow
POST	/auth/{vo}/device	Initiate Device Flow
GET	/auth/{vo}/device/complete	Finish Device Flow
GET	/auth/{vo}/device/complete/finished	Finished
POST	/auth/{vo}/token	Token
GET	/auth/{vo}/authorize	Authorization Flow
GET	/auth/{vo}/authorize/complete	Authorization Flow Complete
<b>jobs</b>		
POST	/jobs/	Submit Bulk Jobs
DELETE	/jobs/	Delete Bulk Jobs
GET	/jobs/{job_id}	Get Single Job
DELETE	/jobs/{job_id}	Delete Single Job
POST	/jobs/{job_id}/kill	Kill Single Job

Swagger generates interactive documentation from the JSON  
Included in FastAPI by default (/docs)

# REDOC

Search...

auth

- GET Do Device Flow
- POST Initiate Device Flow
- GET Finish Device Flow
- GET Finished
- POST Token
- GET Authorization Flow
- GET Authorization Flow Complete

jobs >

config >

well-known >

## auth

### Do Device Flow

This is called as the verification URI for the device flow. It will redirect to the actual OpenID server (IAM, Checkin) to perform a authorization code flow.

We set the `user_code` obtained from the device flow in a cookie to be able to map the authorization flow with the corresponding device flow. (note: it can't be put as parameter or in the URL)

#### PATH PARAMETERS

→ `vo`  
**required** string (Vo)

#### QUERY PARAMETERS

→ `user_code`  
**required** string (User Code)

#### Responses

> 200 Successful Response

### Initiate Device Flow

Initiate the device flow against DIRAC authorization Server. Scope must have exactly up to one `group` (otherwise default) and one or more `property` scope. If no property, then get default one

Offers the user to go with the browser to `auth/<vo>/device?user_code=XYZ`

#### PATH PARAMETERS

GET /auth/{vo}/device

Response samples

200

Content type  
application/json

null

Copy

POST /auth/{vo}/device

Response samples

200

Content type  
application/json

Redoc is another implementation

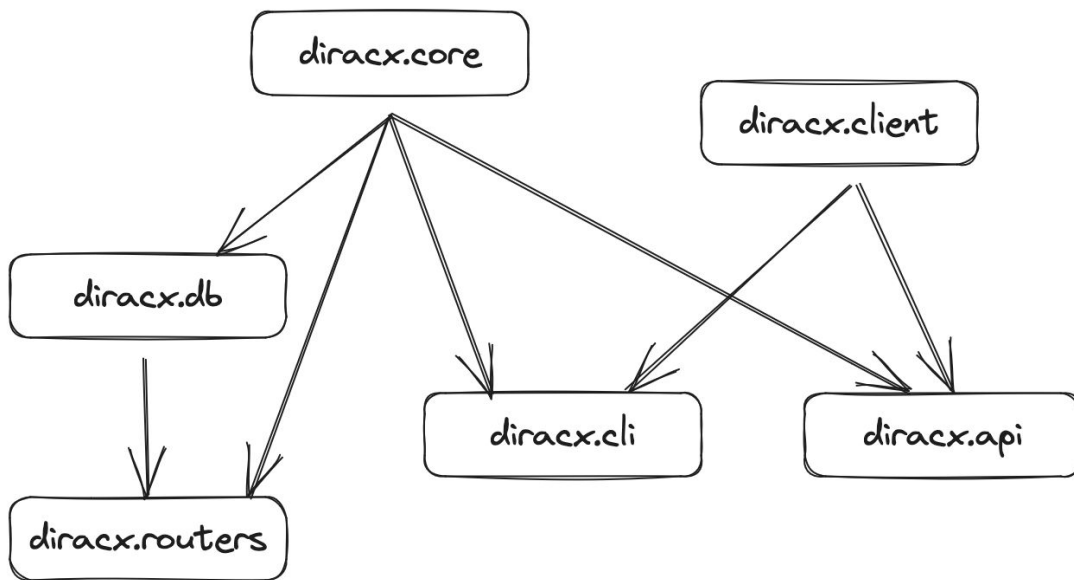
Also included in FastAPI by default (/redoc)

# CELERY

- We need more than just API calls
  - Long running “things” (seconds -> hours)
  - Covers “Agents”, “Requests” and “Executors” in DIRAC
- 
- Will be turned into asynchronous tasks
  - Celery works well for this and is widely used

# PUTTING IT TOGETHER

The Python package structure looks like



Allows us to isolate dependencies more easily

# USING DIRACX WITHOUT THE CLIENT

Demo gods be willing, with only curl!

- Submit a job
- Job monitoring

# MIGRATION

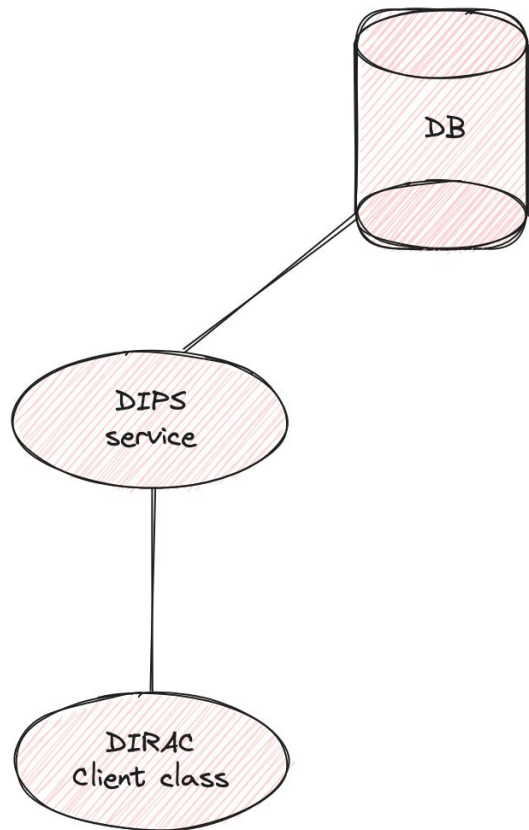
Want the least disturbance possible for installations



# SERVICE MIGRATION

Current situation has:

- MySQL database
- DIPS service using a DB class
- DIRAC Client class

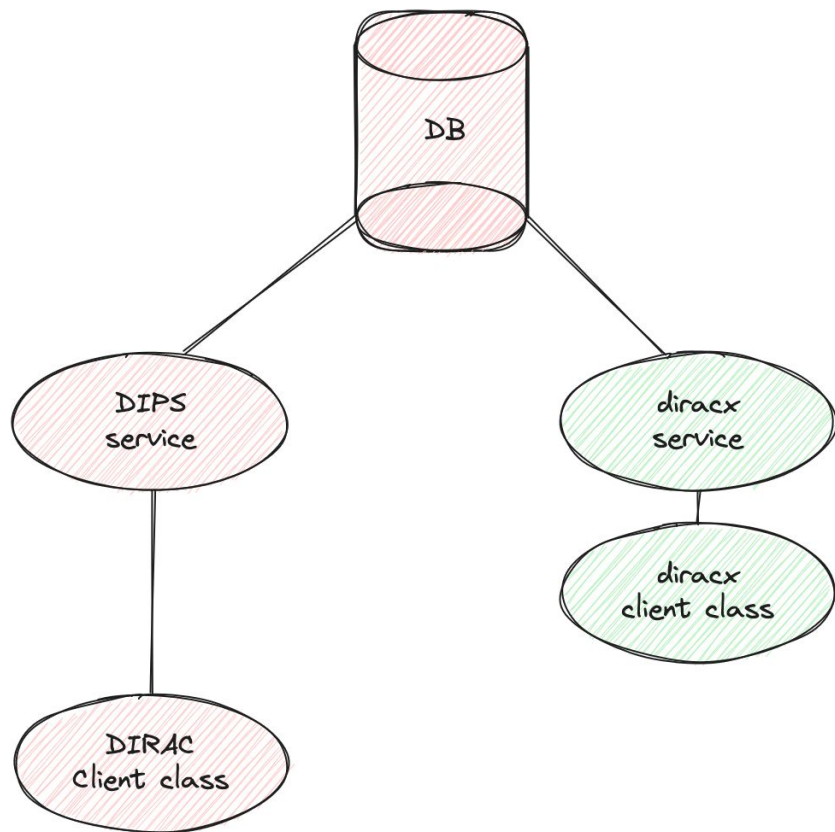


# SERVICE MIGRATION

The MySQL DB stays the same.

Develop in parallel:

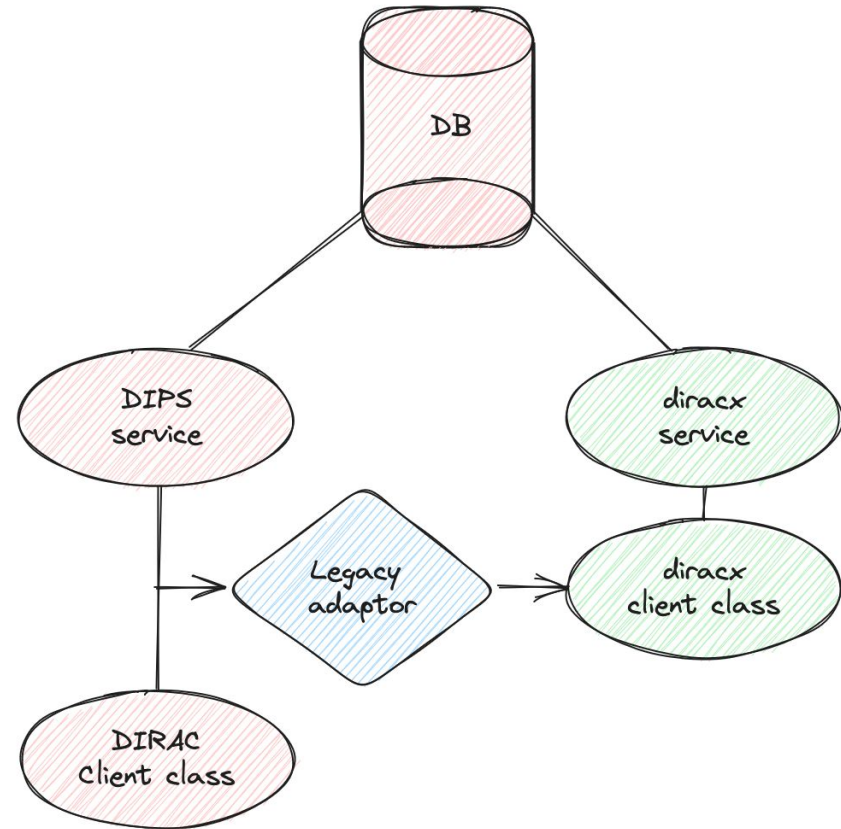
- FastAPI router
- Async SQLAlchemy DB class
- Modern API + CLI + tests



# SERVICE MIGRATION

Once diracx service is ready, add a “legacy adaptor”

Integration tests pass unmodified



# SERVICE MI

Once dirac  
a “legacy  
Integratio

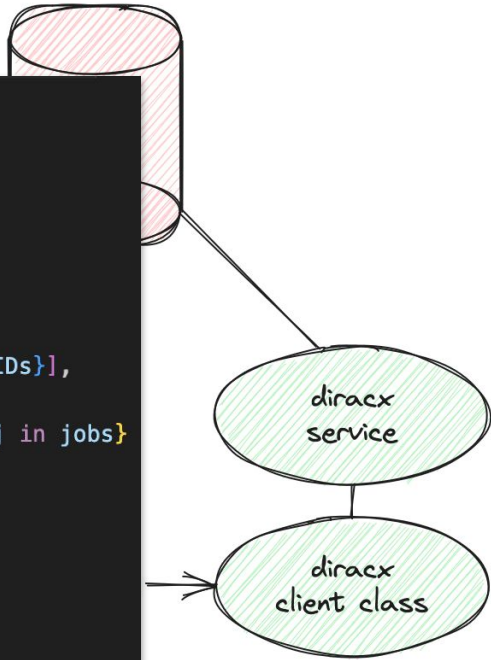
```
from dirac.client import Dirac
from DIRAC.Core.Utilities.ReturnValues import convertToReturnValue

def fetch(parameters, jobIDs):
    with Dirac(endpoint="http://localhost:8000") as api:
        jobs = api.jobs.search(
            parameters=["JobID"] + parameters,
            search=[{"parameter": "JobID", "operator": "in", "values": jobIDs}],
        )
        return {j["JobID"]: {param: j[param] for param in parameters} for j in jobs}

class JobMonitoringClient:
    @convertToReturnValue
    def getJobsMinorStatus(self, jobIDs):
        return fetch(["MinorStatus"], jobIDs)

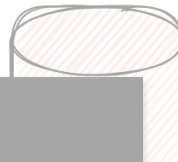
    @convertToReturnValue
    def getJobsStates(self, jobIDs):
        return fetch(["Status", "MinorStatus", "ApplicationStatus"], jobIDs)

    @convertToReturnValue
    def getJobsSites(self, jobIDs):
        return fetch(["Site"], jobIDs)
```



JLRVILL M

```
from diracx.client import Dirac
from DIRAC.Core.Utilities.ReturnValues import convertToReturnValue
```



```
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01-2023-09-01 00:00:00-condDict4] PASSED [ 96%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-None-condDict0] PASSED [ 96%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-None-None] PASSED [ 97%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-None-condDict2] PASSED [ 97%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-None-condDict3] PASSED [ 97%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-None-condDict4] PASSED [ 97%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01-condDict0] PASSED [ 97%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01-None] PASSED [ 98%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01-condDict2] PASSED [ 98%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01-condDict3] PASSED [ 98%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01-condDict4] PASSED [ 98%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01 00:00:00-condDict0] PASSED [ 99%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01 00:00:00-None] PASSED [ 99%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01 00:00:00-condDict2] PASSED [ 99%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01 00:00:00-condDict3] PASSED [ 99%]
tests/Integration/FutureClient/WorkloadManagement/Test_JobMonitoring.py::test_getStates[2023-09-01 00:00:00-2023-09-01 00:00:00-condDict4] PASSED [100%]
```

```
===== warnings summary =====
./../mambaforge/envs/chrisackaton/lib/python3.11/site-packages/pkg_resources/__init__.py:121
/Users/cburr/mambaforge/envs/chrisackaton/lib/python3.11/site-packages/pkg_resources/__init__.py:121: DeprecationWarning: pkg_resources is deprecated as an API
  warnings.warn("pkg_resources is deprecated as an API", DeprecationWarning)

./../mambaforge/envs/chrisackaton/lib/python3.11/site-packages/M2Crypto/__init__.py:24
/Users/cburr/mambaforge/envs/chrisackaton/lib/python3.11/site-packages/M2Crypto/__init__.py:24: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  version_info = StrictVersion(__version__).version

./../mambaforge/envs/chrisackaton/lib/python3.11/site-packages/M2Crypto/SSL/ssl_dispatcher.py:8
/Users/cburr/mambaforge/envs/chrisackaton/lib/python3.11/site-packages/M2Crypto/SSL/ssl_dispatcher.py:8: DeprecationWarning: The asyncore module is deprecated and will be removed in Python 3.12. The recommended replacement is asyncio
  import asyncore

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
```

```
===== 428 passed, 17 skipped, 3 warnings in 52.74s =====
```

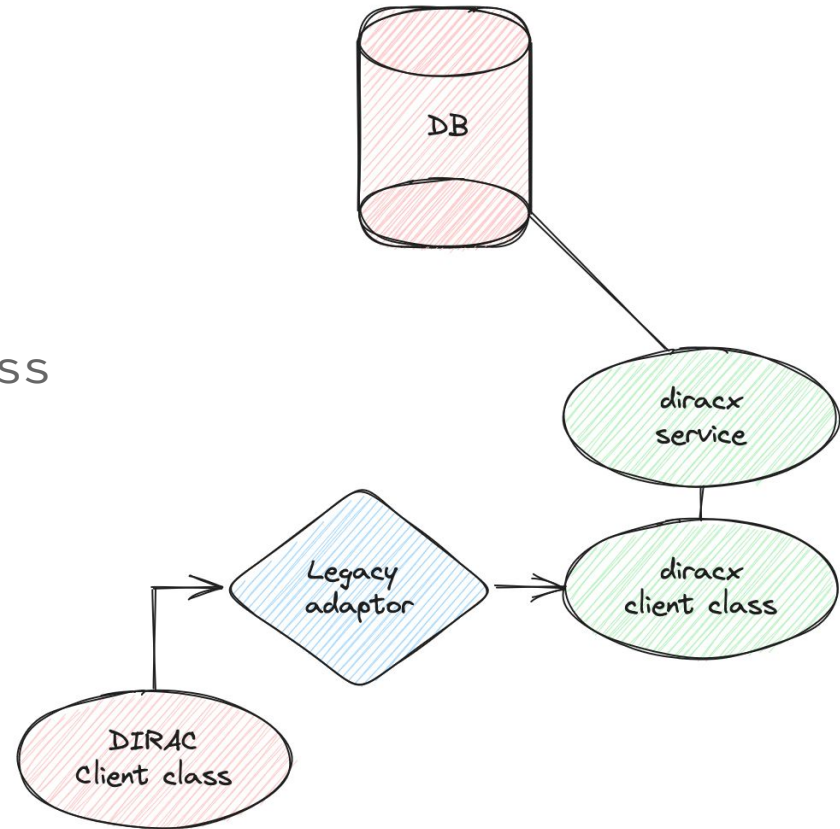
```
return fetch(["Status", "MinorStatus", "ApplicationStatus"], jobIDs)
```

```
===== 428 passed, 17 skipped, 3 warnings in 52.74s =====
```

# SERVICE MIGRATION

Promptly remove the dips service

Moving away from DIRAC client class depends on other modernisations (agents, webapp, ...)



# CONFIGURATION SERVICE

- Holds all of the configuration for what DIRAC does
  - Compute elements, storage elements, users, groups, service config
- Currently very permissive in what it accepts
  - Bad input causes downtime or hours of admin confusion
- Will keep the current service as the source of truth
  - Admins keep editing the old CS
  - DiracX automatically exports it to the new read-only CS

# THE CONFIGURATION SERVICE

## ARCHITECTURE: CONFIGURATION SERVICE

Should be typed with a known schema and comments

Read-optimised with efficient caching strategy (infinite scale)

Atomic writes with history

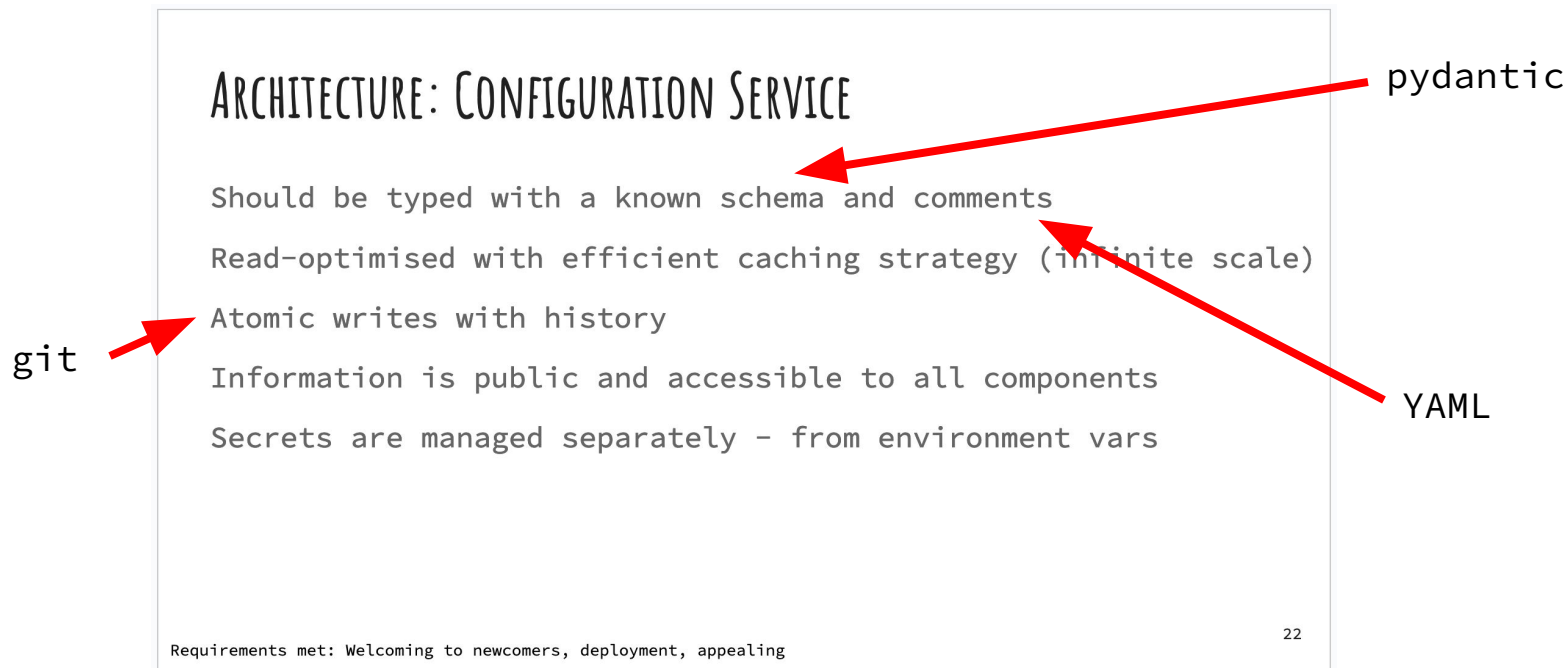
Information is public and accessible to all components

Secrets are managed separately - from environment vars

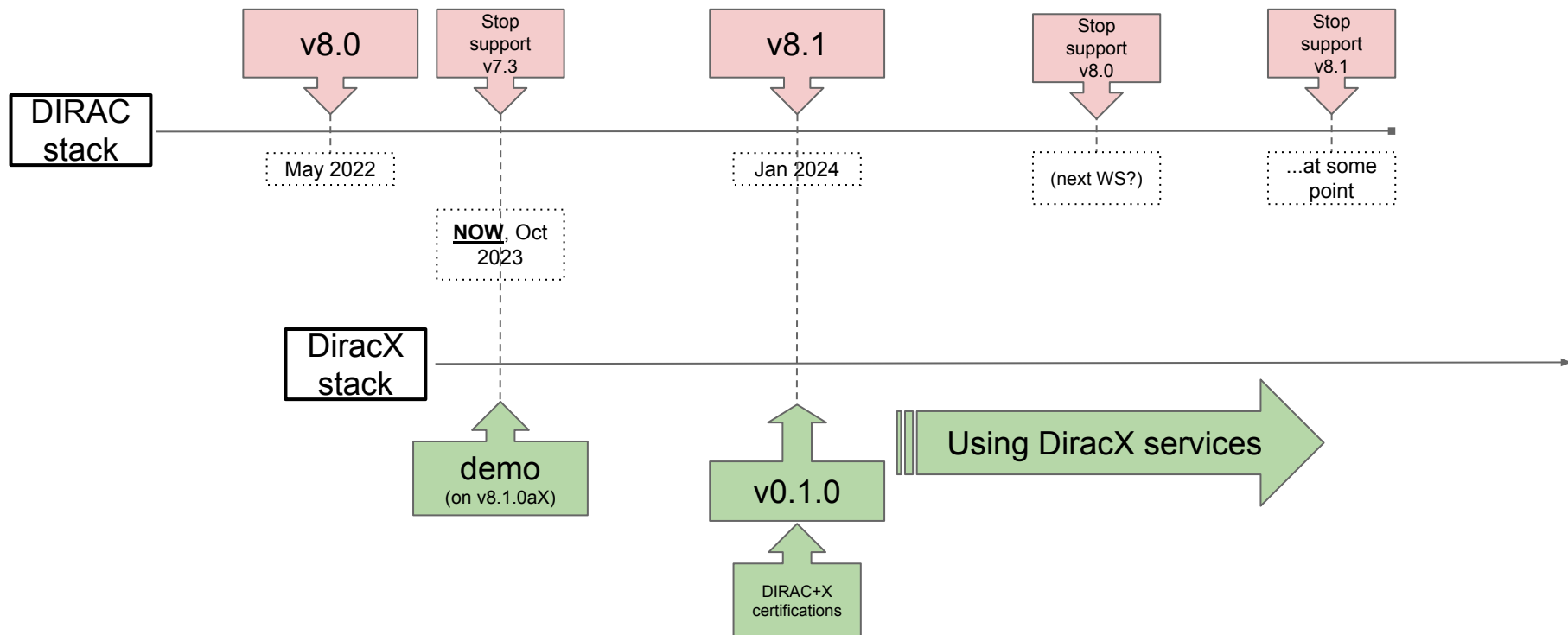


# THE CONFIGURATION SERVICE

Start from the existing CS content (details are later)



# DIRACX STATUS



# DIRACX STATUS

- We still have a lot to finish
  - “Groundwork”
  - Interoperability with legacy DIRAC
  - Deployment
  - Telemetry and monitoring
  - Documentation
  - Extensions
- DiracX will need to be installed alongside DIRAC v8.1
- DiracX won’t do much at this point
  - But all of the groundwork for a smooth transition will be ready
- Functionality will then be slowly moved to DiracX
  - Lot’s of interest from the community
  - Please materialise this effort from Q1 2024

# WHAT WE NEED FROM INSTALLATIONS

- Now: Update to DIRAC v8.0 if you haven't already
- Q1 2024: update to DIRAC v8.1 promptly
- Continuously: Give feedback, reply to discussions

<https://github.com/DIRACGrid/diracx/discussions/categories/extension-requirements>

**Especially need help understanding what extensions need to support**

- If possible, contribute developer FTEs next year

QUESTIONS?