



EVENTUALLY COMING WITH SOME DOCUMENTATION

19/06/2024

10TH DIRAC USERS' WORKSHOP

ALEXANDRE BOYER

NAVIGATING DIRACX ARCHITECTURE AND DEPLOYMENT TOOLS

DIRACX 0.1.0: REQUIREMENTS

- Stable underpinnings
 - interfaces for services, dbs, auth
- No schema changes beyond what Dirac v9 requires
- Support for extensions
- Support for Legacy Adapter (i.e Dirac -> DiracX interactions)
 - One fully supported service: JobStateUpdate
- Complete administration documentation
 - Including how to run with K3s

We are not
so far:

The screenshot displays a Kanban board with three columns: 'Todo' (6 items), 'In Progress' (5 items), and 'Done' (24 items). Each column contains task cards with titles and status indicators.

Column	Count	Description
Todo	6	This item hasn't been started
Todo	1	diracx #31: Exchange token for proxy
Todo	1	diracx-charts #15: Report logs if ingress controller fails to start
Todo	1	diracx-charts #45: Prevent running the demo with sudo
In Progress	5	This is actively being worked on
In Progress	1	diracx #25: Investigate OpenTelemetry
In Progress	1	diracx #27: Example extension
In Progress	1	diracx #30: Finish auth router
Done	24	This has been completed
Done	1	diracx-charts #67: OpenTelemetry integration
Done	1	diracx-charts #55: Try K3S
Done	1	diracx-charts #56: Try RKE2

PLAN

> Focus on the foundational aspects essential for developing and deploying DiracX.

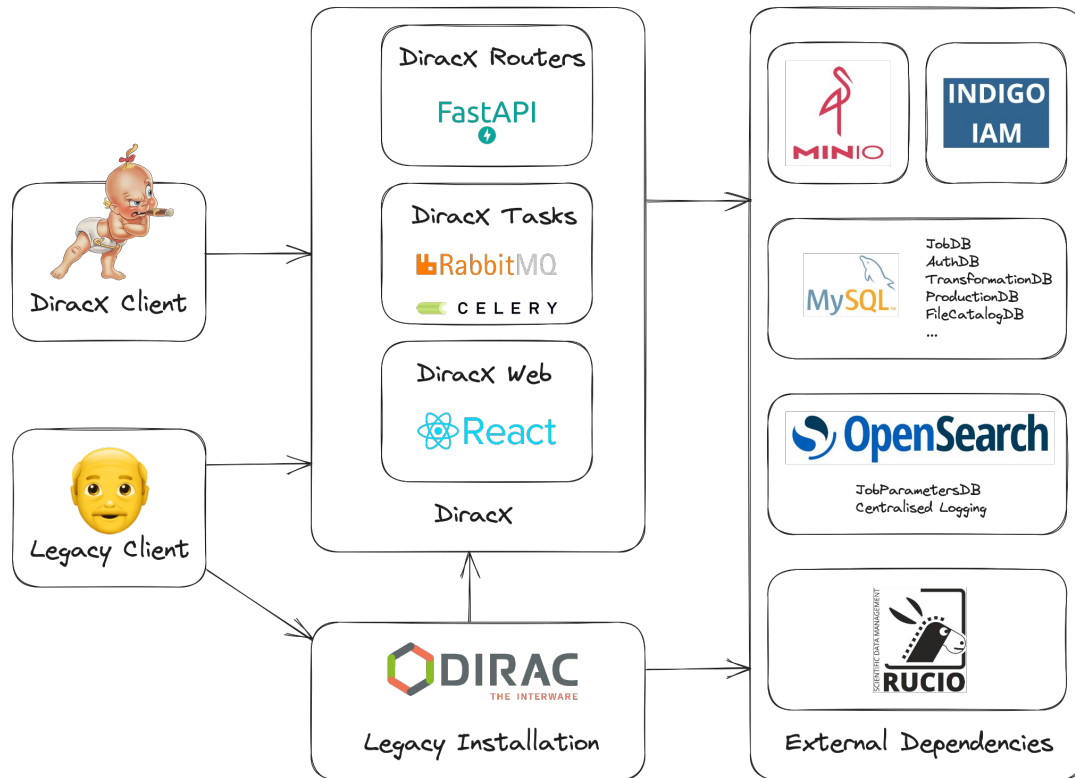
> Serves as an initial step to assist you in creating a DiracX extension tailored for your community.

1. DBs
2. Services
3. Clients
4. Extensions
5. Deployment

ARCHITECTURE

Repositories:

- **diracx:** backend
- **diracx-web:** frontend
- **diracx-charts:** helm charts
- **container-images:** docker images to run diracx

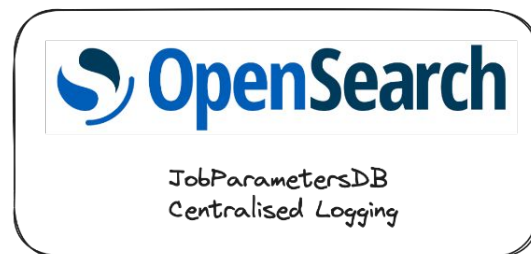


1. INTERFACING WITH DBS

diracx-db: data access layer of diracx.

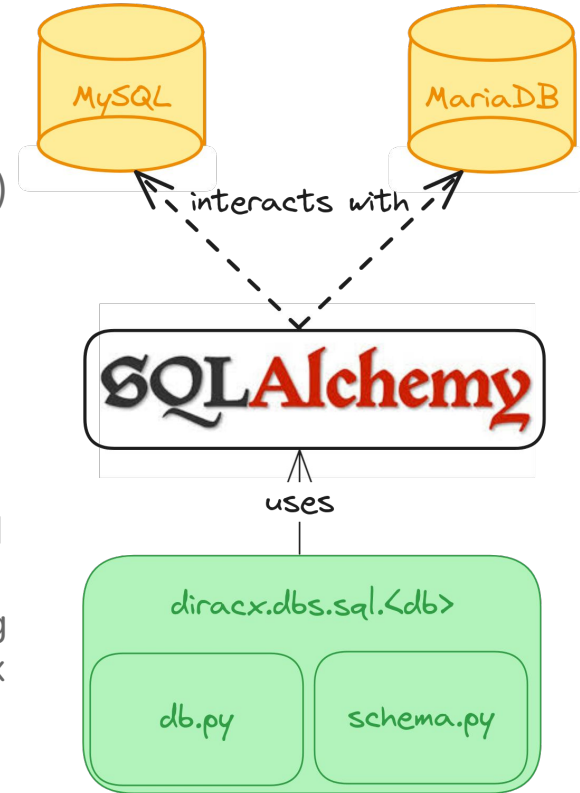
Supports:

- **SQL DBs:** Most of the operational data.
- **OpenSearch:** medium-term data about jobs and pilots, Open Telemetry data.



1.1.1 GENERATING THE INTERFACE: SQL DBS

- **SQLAlchemy:**
 - Python SQL toolkit and Object-Relational Mapping (ORM) library.
- Supported SQL implementations in diracx:
 - MySQL
 - MariaDB
 - SQLite (only for testing)
- **diracx-db SQL structure:**
 - **schema.py:** <db> schema based on SQLAlchemy API (tables, attributes). No more .sql file within the code.
 - **db.py:** Methods to interact with <db> using schema.py. <db> should inherit from diracx BaseSQLDB.



1.1.2 EXAMPLE: SQL DBS

schema.py:

```
from sqlalchemy.orm import declarative_base

JobDBBase = declarative_base() # define a table

class JobJDLs(JobDBBase):
    # table name
    __tablename__ = "JobJDLs"
    # primary key
    JobID = Column(Integer, autoincrement=True,
primary_key=True)
    # other columns
    JDL = Column(Text)
    JobRequirements = Column(Text)
    OriginalJDL = Column(Text)
```

db.py:

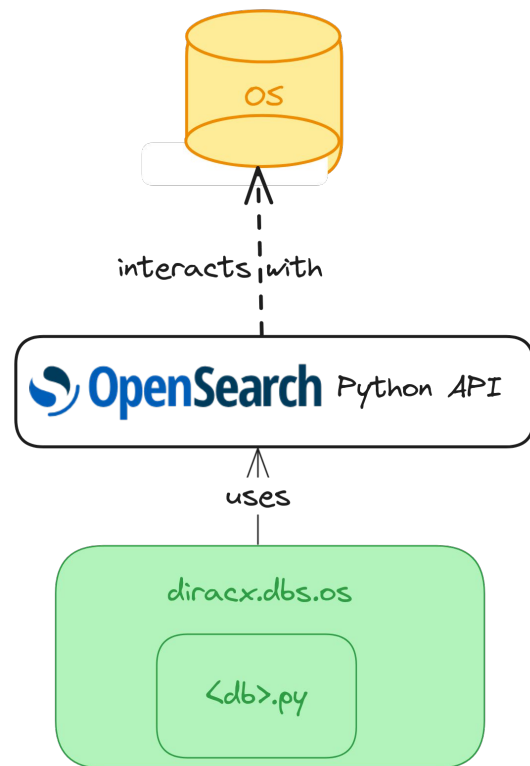
```
from .schema import JobJDLs
from sqlalchemy import delete

# inherits from BaseSQLDB
class JobDB(BaseSQLDB):
    metadata = JobDBBase.metadata

    # uses sqlalchemy to build SQL requests
    async def delete_jobs(self, job_ids: list[int]):
        """Delete jobs from the database."""
        stmt = delete(JobJDLs).
            where(JobJDLs.JobID.in_(job_ids))
        await self.conn.execute(stmt)
```

1.2.1 GENERATING THE INTERFACE: OS DBS

- Interface depends on the OpenSearch API.
- `diracx-db OS` structure:
 - `<db-name>.py` contains the fields and the index name.
 - Class should inherit from `BaseOSDB`.



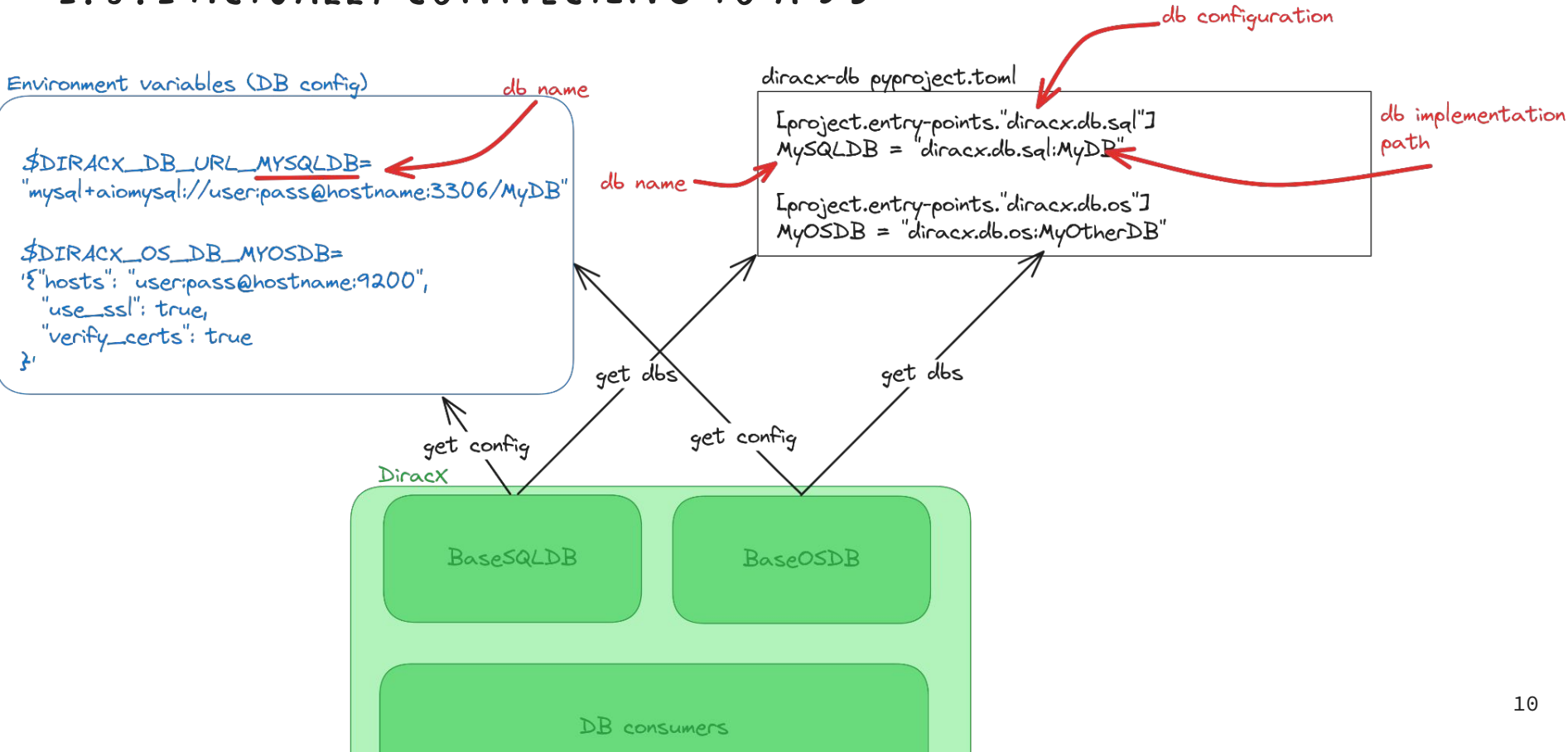
1.2.2 EXAMPLE: OS DBS

job_parameters.py:

```
class JobParametersDB(BaseOSDB):
    fields = {
        "JobID": {"type": "long"},
        "timestamp": {"type": "date"},
        "HostName": {"type": "keyword"},
        ...
    }
    index_prefix = "elasticjobparameters_index_"

    def index_name(self, doc_id: int) -> str:
        return f"{self.index_prefix}_{doc_id // 1e6:.1f}m"
```

1.3.1 ACTUALLY CONNECTING TO A DB



1.3.2 ACTUALLY CONNECTING TO A DB: FURTHER DETAILS

- Configuration is no longer specified in *dirac.cfg* file but through environment variables.
- SQL DBs: configuration must follow the [SQLAlchemy connection URL format](#).
 - The driver part of the URL is always specified and must refer to an async-compatible backend.
- OS DBs: configuration must be defined as a JSON mapping (more details in the [opensearch documentation](#))

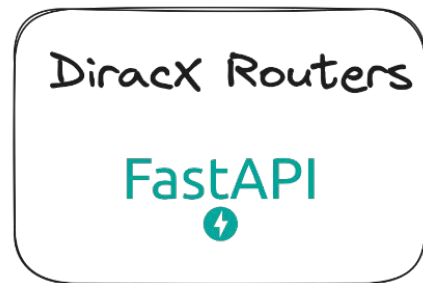
2. DEALING WITH SERVICES

diracx-routers: business logic layer of diracx, based on FastAPI.

- Services are now contained within a single FastAPI application:

```
diracx.routers.create_app()
```

- What was previously a Dirac handler (service) is now an API router.



2.1.1 ROUTERS

- Each service is associated with a `DiracxRouter`
 - Services are served under `/api/<service>`:
 - Service names are defined in the `dirac.services` entrypoint of `pyproject.toml`.

```
[project.entry-points."diracx.services"]
```

```
jobs = "diracx.routers.job_manager:router"
```

- Routes & Operations (HTTP methods) are the equivalent of the `export_<method>()` methods from `Dirac` handlers:

```
@router.<operation>("/<route>")  
async def ...
```

```
@router.delete("/{job_id}")  
async def delete_single_job(...)
```

```
curl -X <operation> <diracxurl>/<route>
```

```
curl -X DELETE <diracxurl>/api/jobs/<job_id>
```

jobs

GET	/api/jobs/sandbox	Get Sandbox File
POST	/api/jobs/sandbox	Initiate Sandbox Upload
DELETE	/api/jobs/sandbox	Unassign Bulk Jobs Sandboxes

2.1.2 ROUTERS: EXAMPLE

- Deleting a job in Dirac:

```
class JobManagerHandler(RequestHandler):
```

```
    def export_deleteJob(self, jobIDs):
```

```
        """Delete jobs.
```

```
        """
```

```
        ...
```

- Deleting a job in DiracX:

```
# if router is defined as "jobs" in
```

```
pyproject.toml
```

```
router = DiracxRouter()
```

```
# then the following operation is available
```

```
through DELETE /api/jobs/
```

```
@router.delete("/")
```

```
async def delete_bulk_jobs(
```

```
    job_ids: Annotated[list[int], Query()],
```

```
    ...
```

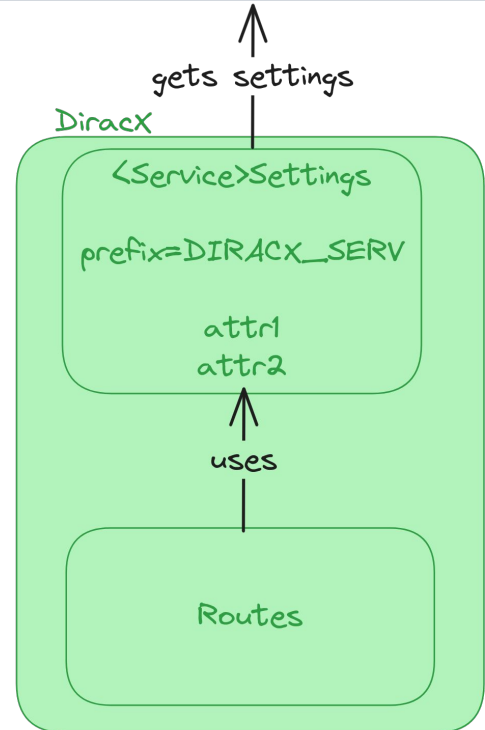
2.2.1 INJECTING SETTINGS IN SERVICES

- Secrets are no longer specified using the `dirac.cfg` file.
- `create_app()` uses environment variables to set them.
- Refer to these as “settings”:
 - Based on [pydantic settings management](#)
- Examples:
 - Signing key for tokens
 - OIDC client credentials
 - Service specific things (token lifetimes, sandbox store options...)
- Note: there exist environment variables to disable specific services :

`DIRACX_SERVICE_<service-name>_ENABLED=false`

Environment variables

```
$DIRACX_SERV_ATTR1=something  
$DIRACX_SERV_ATTR2=somethingelse
```



2.2.2 INJECTING SETTINGS IN SERVICES: EXAMPLE

- Settings class for the Authentication router:

```
@add_settings_annotation
class AuthSettings(ServiceSettingsBase):
    """Settings for the authentication
    service."""
    model_config =
SettingsConfigDict(env_prefix="DIRACX_SERVI
CE_AUTH_")
    dirac_client_id: str = "myDIRACClientID"
```

- Defining settings as environment variables:

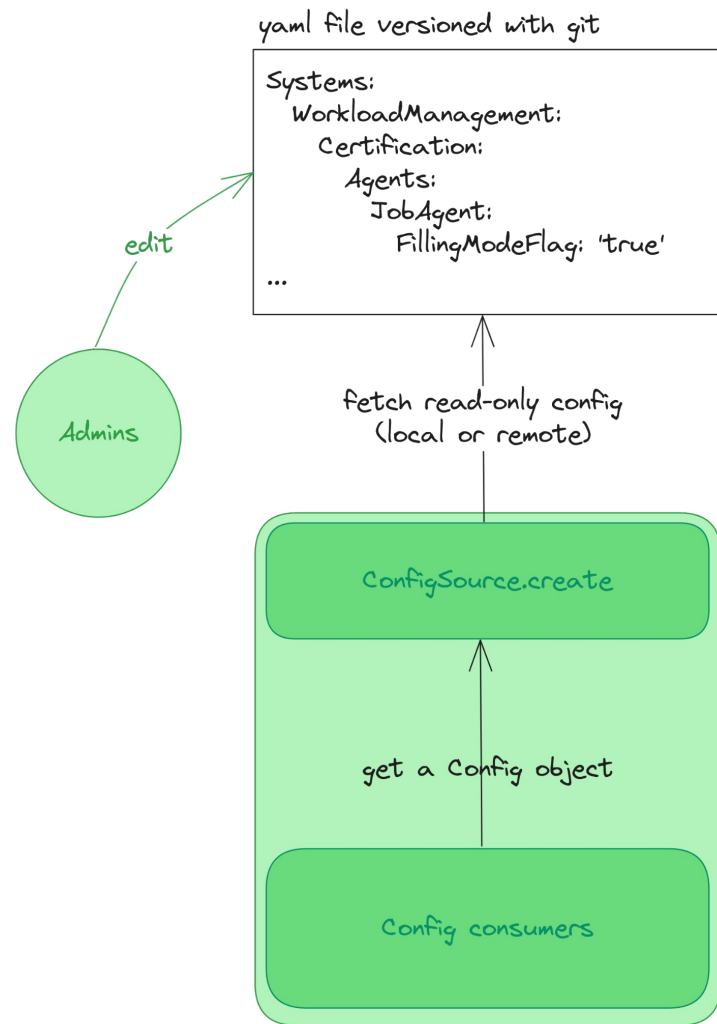
```
DIRACX_SERVICE_AUTH_DIRAC_CLIENT_ID =
"myNewDIRACClientID"
```

- Get values from a route:

```
@router.get("/openid-configuration")
async def openid_configuration(settings:
AuthSettings):
    ...
    client_id = settings.dirac_client_id
```


2.3.1 GETTING CONFIGURATION

- DiracX only has a read-only view of the CS.
 - Will be the last thing to be migrated
- Updates are made to the legacy CS and synchronized.
- Structure of the DiracX Configuration is not the same.
 - Truly multi-VO
 - Strictly typed
 - Well defined schema



2.3.2 GETTING CONFIGURATION: EXAMPLE

- Get configuration from a route:

```
from diracx.routers.dependencies import Config
```

```
@router.post("/summary")
```

```
async def summary(
```

```
    config: Config,
```

```
)...
```

```
    if not config.Operations["Defaults"].Services.JobMonitoring.GlobalJobsInfo:
```

```
        ...
```

2.4 GETTING DATA THROUGH DIRACX-DB

- SQL and OS DBs are available through the same module:
`diracx.routers.dependencies`

```
from diracx.routers.dependencies import JobDB
```

```
@router.get("/{job_id}")
```

```
async def get_single_job(job_db: JobDB):
```

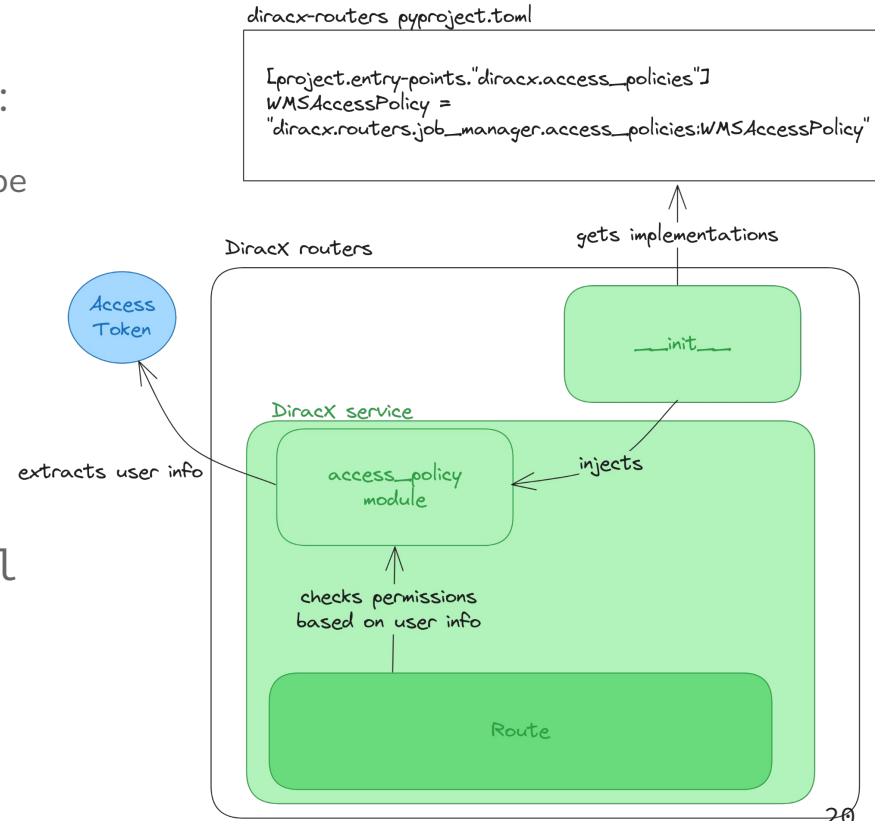
- SQL DBs: transactions are opened for the duration of the request.
 - Successful requests: commit transaction.
 - Bad request (HTTP status code ≥ 400): roll back the transaction.
- No such transaction/rollback mechanism for OS DBs.

2.5.1 PERMISSION MANAGEMENT: USING ACCESS POLICIES

- Managed by `access_policies` modules:
 - Every route should rely on a given policy
 - Open routes (requiring no authN/authZ) should be explicitly decorated with `@open_access`.
- Rely on the access token payload.

```
@router.get("/{job_id}")  
async def get_single_job(check_permissions:  
CheckWMSPolicyCallable ...
```

- Implementation of policies can be injected within the code through `pyproject.toml`



2.5.2 PERMISSION MANAGEMENT: EXAMPLE

- PolicyAccess:

```
class SandboxAccessPolicy(BaseAccessPolicy):

    @staticmethod
    async def policy(
        policy_name: str,
        user_info: AuthorizedUserInfo,
        /,
        *,
        action: ActionType | None = None,
        job_db: JobDB | None = None,
        ...): ...

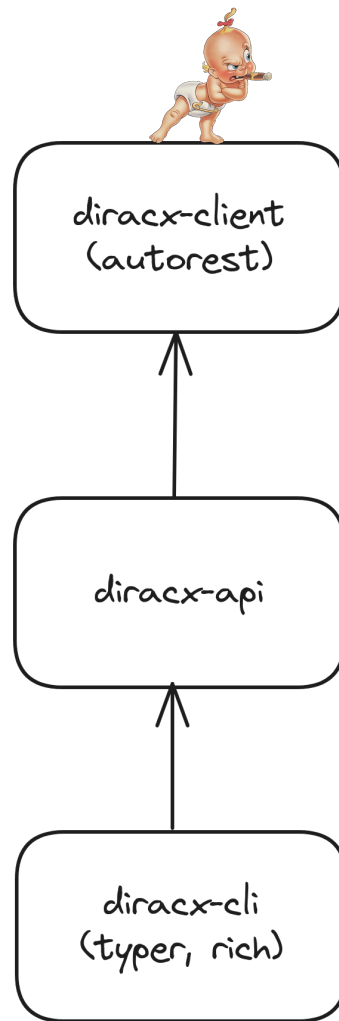
    if action == ActionType.CREATE:
        if NORMAL_USER not in user_info.properties:
            raise HTTPException(status.HTTP_403_FORBIDDEN)
        return
```

- Calling it from a route:

```
await check_permissions(
    action=ActionType.CREATE,
    sandbox_metadata_db=sandbox_metadata_db, pfn=[pfn]
)
```

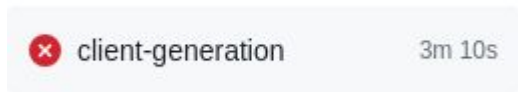
3. CLIENTS: OVERVIEW

- **diracx-client:** generated by autorest from the OpenAPI json file generated by FastAPI.
- **diracx-api:** Python API to interact with services using diracx-client methods(business logic).
- **diracx-cli:** a CLI for direct interaction with the services.



3.1.1 GENERATING/UPDATING DIRACX-CLIENT

- Each time there is a PR targeting the main branch, a CI/CD job aims at detecting breaking changes in the API.



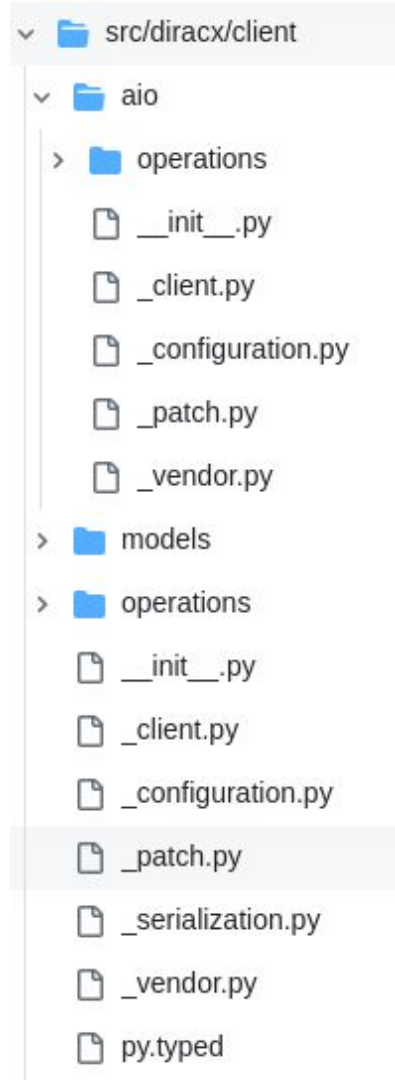
- If approved by the repo admins, you can try to regenerate it following the [“documentation”](#).
 - If you don't manage, admins can also regenerate a client on your behalf within your PR.

3.1.2 CUSTOMISING DIRACX-CLIENT

- Structure:
 - `models`: data structures
 - `operations`: methods to interact with the services
 - `aio`: asynchronous clients (constains async operations)
- **`_patch.py`**: allows customising the generated client/operations/models.

Note1: modifications should be avoided as much as possible.

Note2: any modifications in the sync client should be ported to the async client, and vice-versa.



3.2.1 CREATING A DIRACX-API METHOD

- Import `DiracClient`
- Decorate the method with `@with_client` to handle client configuration.
- Pass the client as a keyword argument

```
from diracx.client.aio import DiracClient
```

```
from .utils import with_client
```

```
@with_client
```

```
async def create_sandbox(paths: list[Path], *, client: DiracClient) -> str:
```

```
    ...
```

3.2.2 USING A DIRACX-API METHOD

- Not passing a `DiracClient` to the API method:
 - Can be provided by `@with_client`.
 - Useful for quick work requiring a single call to a service.

```
result = await create_sandbox(paths)
```

- Passing a `DiracClient` to the API method:
 - For optimised performances: calls to multiple services.

```
async with DiracClient() as client:
```

```
    result = await create_sandbox(paths, client)
```

3.3 CREATING A DIRACX-CLI (REPLACEMENT FOR DIRAC SCRIPTS)

- Import `DiracClient` and/or a `diracx-api`
- Import `AsyncTyper` (custom `async Typer`)
- Decorate the method with `@app.async_command()`

Note: `Typer` allows generating commands and subcommands such as:

```
$dirac jobs search <parameters>
```

```
from diracx.client.aio import
DiracClient
from .utils import AsyncTyper
app = AsyncTyper()

@app.async_command()
async def login():
    async with DiracClient() as client:
```



```
$dirac login
```

3.4 CONFIGURING CLIENTS: PREFERENCES

- DiracXPreferences: configuration is loaded from the environment variables (Similar to the service settings).
- Options (environment variables):
 - (Required)\$DIRACX_URL: pointing to the diracx services.
 - (“Required”)\$DIRACX_CA_PATH: CA path to interact with the services.
 - \$DIRACX_CREDENTIALS_PATH: path where access and refresh tokens are stored.
 - \$DIRAC_LOG_LEVEL: log level.
 - \$DIRAC_OUTPUT_FORMAT: output format

4. GENERAL WORD ABOUT EXTENDING DIRACX

- DiracX provides many Python “entrypoints” (pyproject.toml)
- Used to override databases/services/auth policies
- No support for setting at runtime
 - Extensions are configured at install time based on the extension code
 - Changes require making a new release of your extension

```
[project.entry-points."diracx.db.sql"]
```

```
AuthDB = "diracx.db.sql:AuthDB"
```

```
JobDB = "<extension>.db.sql:ExtendedJobDB"
```

5. DEPLOYMENT



kubernetes



- Kubernetes (k8s) has become the defacto way:
 - allows you to deploy containerized applications
 - underlying infrastructure is abstracted.
 - configuration of the application and how it should run is communicated to k8s via yaml files.
- Helm: allows templating these yaml files.
 - a templated description of an application like DiracX is called **chart**.
 - also allows managing dependencies between charts.

(e.g. the DiracX application needs a database to run, so the DiracX charts has a dependency on the mysql charts.

5.1.1 DIRACX-CHARTS: PRESENTATION

- Contains the deployment for diracx and diracx-web, as well as dependencies:
 - MySQL, OpenSearch databases
 - Dex and IAM as identity provider
 - Minio as an object store for the SandboxStore
 - OpenTelemetry to manage traces, monitoring and logging (experimental).

5.1.2 DIRACX-CHARTS: INSTALLATION TYPE

- 4 types of installation:
 - **demo/dev:** install everything and configure everything with pre-configured values.
 - **prod:** a DIRAC installation with it's own DBs and everything already exist. Create a cluster, but bridge on existing external resources (like DBs).
 - **new:** start from absolutely nothing (no DIRAC), and install all the dependencies.
 - **new without dependencies:** start with nothing, but use externally managed resources (like DB provided by your IT service).

5.1.3 DIRACX-CHARTS: VALUES

- DiracX environment variables are provided through a yaml file.
- Settings and DB credentials are loaded as secrets.

diracx:

```
hostname: <hostname>
settings: # Service settings and Config location
  DIRACX_SANDBOX_STORE_AUTO_CREATE_BUCKET: "true"
sqlDbs: # SQL DB credentials and configuration
  dbs:
    JobDB:
osDbs: # OS DB credentials and configuration
  dbs:
    JobParametersDB:
```

5.2.1 RUNNING THE DEMO INSTALLATION (LOCALLY)

- Useful for demo or testing purposes.
- Simply start it with: `run_demo.sh`
- Once ready, you will get some information on how to interact with the installation:
 - Set environment variables to interact with the cluster:
KUBECONFIG, HELM_DATA_HOME, PATH
 - Set environment variables to configure the DiracX client:
DIRACX_URL, DIRACX_CA_PATH
 - URL and credentials to access the demo from a web browser (diracx-web).

5.2.2 RUNNING THE DEMO INSTALLATION: A FEW TIPS

- Python and Node modules can be mounted within the containers.
 - Example: DiracX, Dirac, Diracx-Web
 - Code can be edited and applied within the cluster in real time.
 - `run_demo.sh /path/to/diracx /path/to/diracx-web ...`
- Configuration is also mounted within the containers.
 - You can access it locally, edit it and `git commit`.

5.3.1 RUNNING THE PROD INSTALLATION

1. Does your institute provide a managed k8s service?
 - E.g. Rancher, Openshift, Tanzu, public clouds
 - -> If yes, use it!
2. Else, we recommend k3s?
 - A lightweight kubernetes distribution (single or multi node)
 - [Installation docs](#)
 - DIRAC certification will run with k3s

5.3.2 RUNNING THE PROD INSTALLATION: A FEW TIPS

- Custom branches (diracx, diracx-web) can be deployed:
 - Use cases: test features on certification instances, hotfix in production

diracx:

```
pythonModulesToInstall:
```

- "git+https://github.com/USERNAME/diracx.git@BRANCH_NAME#egg=diracx_core&subdirectory=diracx-core"
- "git+https://github.com/USERNAME/diracx.git@BRANCH_NAME#egg=diracx_db&subdirectory=diracx-db"

diracx-web:

```
repoURL: "https://github.com/USERNAME/diracx-web.git"
```


```
branch: "feat-custom-branch"
```

A WORD ABOUT DIRACX-WEB

Foundations are almost there:

- Generic table and filters
 - JobMonitor
- Dashboard with draggable instances of applications
- Possibility to create extensions

Further details in a presentation dedicated to the web interface at the next BiLD meeting.



Dashboard ^

- Dashboard ::
- Job Monitor ::

Other v

+ Add application

Documentation

Select Virtual Organization

Select a Group

[LOGIN THROUGH YOUR IDENTITY PROVIDER](#) [ADVANCED OPTIONS](#)

Need help? Please contact system administrator

Job Monitor Job Monitor

ADD FILTER APPLY FILTERS CLEAR ALL FILTERS

Edit Filter				Status	Minor Status	Submission Time
Column	Operator	Value				
<input type="checkbox"/>	8246	00000290_00000003	LCG.GRIDKA.de	Failed	Job forced to Failed	2024-03-01T14:40:41
<input type="checkbox"/>	8421	00000292_00000001	LCG.CERN.cern	Waiting	Pilot Agent Submission	2024-04-18T08:58:07
<input type="checkbox"/>	8422	00000292_00000002	LCG.UKI-LT2-IC-HEP.uk	Failed	Job forced to Failed	2024-04-18T08:58:07
<input type="checkbox"/>	8423	00000292_00000003	LCG.CERN.cern	Waiting	Pilot Agent Submission	2024-04-18T09:04:07
<input type="checkbox"/>	8545	jobName	ANY	Deleted	Checking accounting	2024-05-28T15:49:15
<input type="checkbox"/>	8546	helloWorld	LCG.NCB3.pl	Deleted	Checking accounting	2024-05-30T08:40:40
<input type="checkbox"/>	8547	helloWorldNCB3	LCG.NCB3.pl	Deleted	Checking accounting	2024-05-30T08:40:41
<input type="checkbox"/>	8548	jobWithOutput	LCG.NCB3.pl	Deleted	Checking accounting	2024-05-30T08:40:41
<input type="checkbox"/>	8549	jobWithOutputs	LCG.NCB3.pl	Deleted	Checking accounting	2024-05-30T08:40:41
<input type="checkbox"/>	8550	helloWorld	ANY	Deleted	Checking accounting	2024-05-30T09:23:26
<input type="checkbox"/>	8551	helloWorldNCB3	LCG.NCB3.pl	Deleted	Checking accounting	2024-05-30T09:23:26
<input type="checkbox"/>	8552	jobWithOutput	LCG.NCB3.pl	Deleted	Checking accounting	2024-05-30T09:23:27
<input type="checkbox"/>	8553	jobWithOutputs	LCG.NCB3.pl	Deleted	Checking accounting	2024-05-30T09:23:27
<input type="checkbox"/>	8554	helloWorldCloudCE	LCG.UKI-LT2-IC-HEP.uk	Deleted	Checking accounting	2024-05-30T13:10:21

Rows per page 25 1-25 of 220 < >

QUESTIONS?