cdisc®

2026 Europe Interchange

THE FUTURE IS CONNECTED: STANDARDS AND AI POWERING DIGITAL TRANSFORMATION

MILAN, ITALY | MAIN CONFERENCE: 20-21 MAY | TRAININGS & WORKSHOPS: 18, 19, & 22 MAY

Operationalizing CDISC CORE as a Scalable REST API Using Docker and Azure Container Apps

Sandeep Juneja, Clinical Solutions Lead, argenx

Speakers

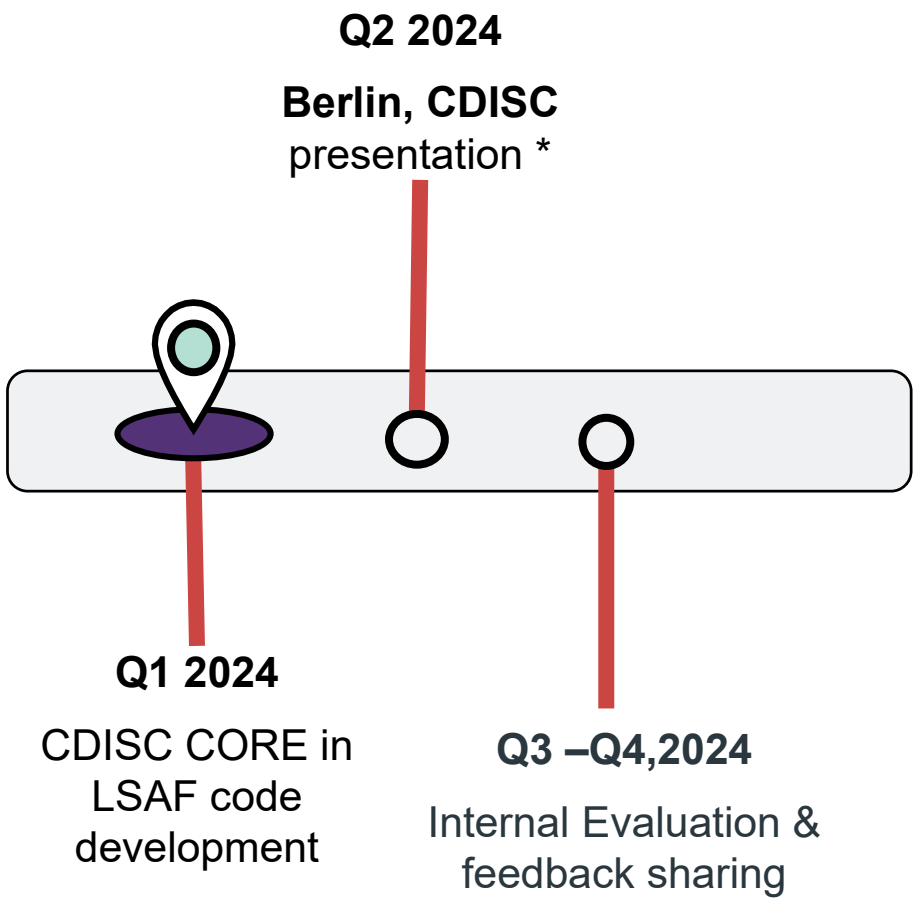
Sandeep Juneja
Clinical Solutions Lead
argenx

- Sandeep Juneja is a Clinical Solutions Lead with deep expertise in architecting and delivering GxP-compliant clinical data and advanced analytics platforms in cloud environments. With extensive hands-on experience across clinical data management, system validation, and regulated analytics, he focuses on translating complex R&D requirements into scalable, production-ready solutions.

Agenda

- Journey so far.
- Design, Architecture, Implementation
- How we built REST endpoints
  - I – Code development
  - II – Docker containerization
  - III – Cloud Deployment
  - IV – Execution automation
- Benefits
- Next Steps

The Journey so far - Language-Specific to Language-Agnostic



The Challenge

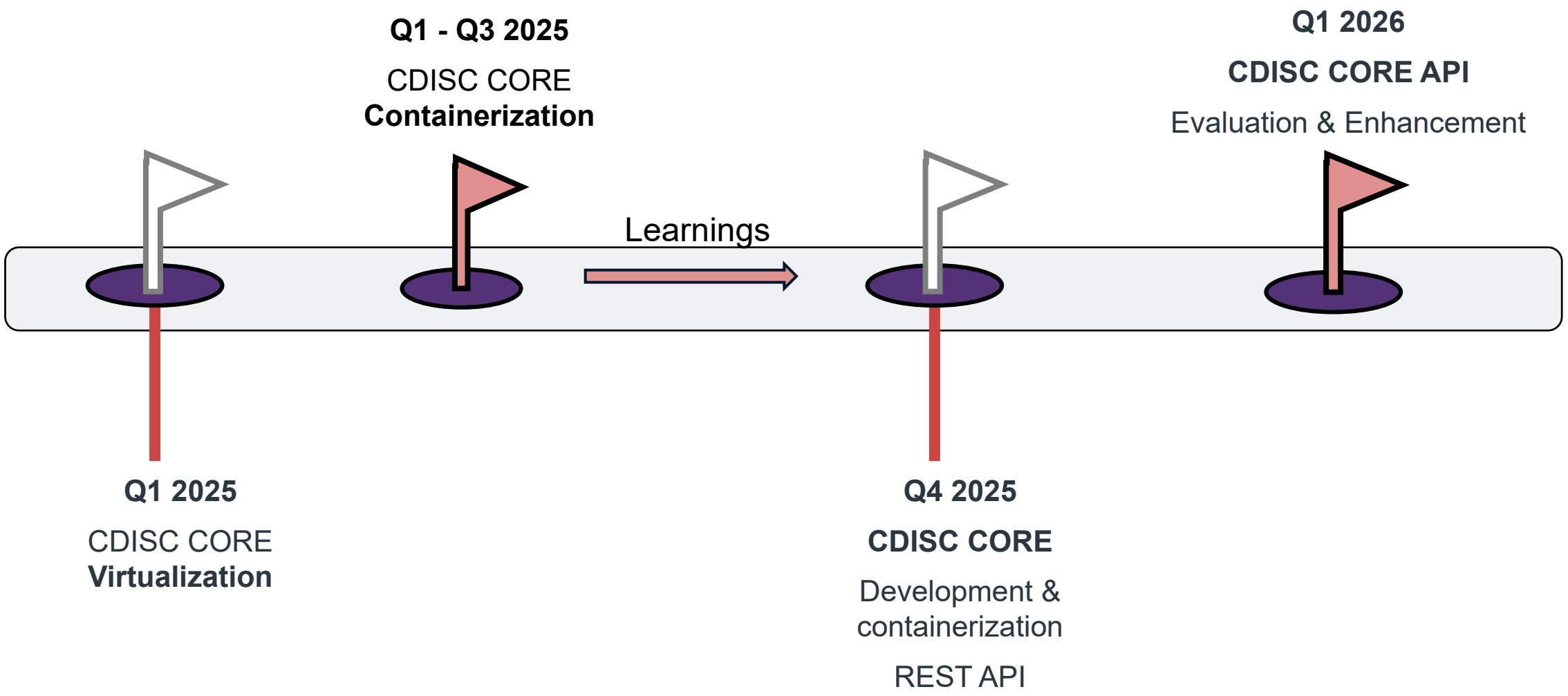
- ❑ Frequent CDISC Core releases
- ❑ Closed systems like SAS LSAF limit machine access
- ❑ Dependency on SAS team due to hosted solution
- ❑ Validation challenges due to OS level updates
- ❑ Longer qualification timelines

The Need

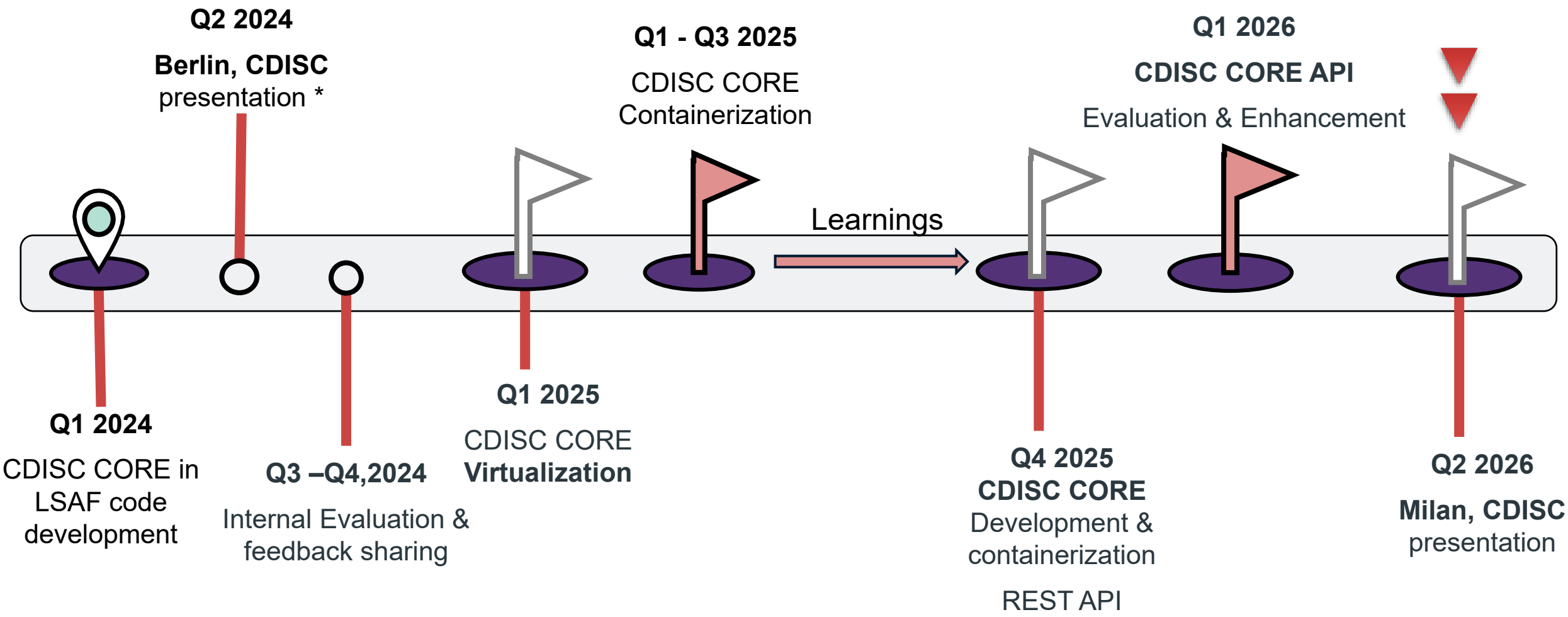
Develop mechanism to deploy frequently updated CORE engine and execute CDISC CORE rules outside validated environment (LSAF) in secured and controlled way

*** Presentation Title -** Implementing CDISC CORE in a cloud native, regulated environment supporting Rare Diseases submissions

The Journey so far - Language-Specific to Language-Agnostic



The Journey so far - Language-Specific to Language-Agnostic



*** Presentation Title -** Implementing CDISC CORE in a cloud native, regulated environment supporting Rare Diseases submissions

CORE API – Design, Architecture, Implementation

- Design

1 Client Upload

`HTTP POST /validate`

standard, version, ZIP file

Client sends data (POST request)

2 Input Processing

Save to tmpdir

extract & list dataset files

FastAPI receives & parses payload

3 CORE CLI Execution

`subprocess.run(...)`

capture stdout / stderr

CDISC CORE engine executes rules

4 Report Generation

Scan for .xlsx

return 500 if not found

Results captured in Excel file

5 Response + Cleanup

`FileResponse · stream`

BackgroundTask deletes tmpdirs

XLSX response returned to client

CORE API – Design, Architecture, Implementation

- **Architecture** A thin *wrapper* around an open-source engine.

CDISC CORE Engine

Open-source rules engine validates SDTM, ADaM & Define-XML datasets against published rulesets

REST API Layer

FastAPI wrapper exposes Endpoints — accepts a ZIP of datasets / dataset, returns an Excel conformance report

Container + Cloud

Docker image deployed to Azure Container Apps enables on-demand, auto-scaling

FEATURES

- Stateless & Scalable
- Standard-Agnostic
- CI/CD Ready

- Modular Architecture

routers/

HTTP handlers only

validate_zip · validate_xpt
validate_sas · report_download

services/

Business logic

file · runner · report
report_store · conversion

middleware/

Cross-cutting

X-Process-Time header on every response

metrics/

Observability

timestamp · command
execution time · status

auth/

Future scaffold

UserContext dataclass
3-line activation

Future

api.py — entry point

cdisc®

CORE API – Design, Architecture, Implementation

Packages · Files · Responsibilities

A look *inside* the codebase.

Each module owns one concern. Routers expose endpoints. Services hold the work. Middleware times every request. Metrics log it. Auth is scaffolded for the future.

routers/

HTTP handlers

validate_zip.py
POST /validate · POST /validate/stream

validate_xpt.py
POST /validate/xpt

validate_sas.py
POST /validate/sas · /sas/zip

report_download.py
GET /report/{token}

services/

Business logic

file_service.py
Temp dirs, ZIP/XPT/SAS staging

runner_service.py
Builds & runs core.py via subprocess

report_service.py
Find xlsx → FileResponse

report_store.py
Token store, 5-min auto-expiry

conversion_service.py
SAS7BDAT → XPT via pyreadstat

middleware/

Cross-cutting

timing.py
X-Process-Time header on every response

metrics/

Observability

logger.py
JSON lines → metrics.log
timestamp · command · status · duration

auth/

JWT-ready scaffold

dependencies.py
UserContext dataclass
get_current_user() · 3-line activation

`api.py — entry point`

CORE API – Design, Architecture, Implementation

- Implementation

REST API END-POINTS			
Request Type	**Endpoint**	**Description**	**Output**
GET	/	Home	
GET	/health	Container health check endpoint	200 OK
GET	/docs	Swagger UI (interactive testing)	HTML

FEATURES

Execution timing

X-Process-Time on every response

Metrics log

Structured JSON lines

Console logging

Real-time stream from container

Max report rows

Configurable safety cap

ERROR HANDLING & CLEANUP

→ No file → HTTP 400 → Bad ZIP → HTTP 400 → CORE non-zero exit → HTTP 500 + stderr → No report found → JSON message → finally: cleanup input_dir always runs

CORE API – Design, Architecture, Implementation

- Implementation

What *lives where* on disk.

```
CDISC_CORE_REST_API/
├── cdisc-rules-engine/   # git clone the engine
├── rest_api/   # Python source
│   ├── auth/
│   ├── middleware/
│   ├── routers/
│   ├── services/
│   ├── metrics/
│   └── api.py   # entry point
├── .venv/   # python deps
├── Dockerfile   # 7-stage build
└── docker-compose.yml   # one-command run
```

Step 1

Clone the engine

```
git clone https://github.com/cdisc-org/cdisc-rules-engine.git
```

Step 2

Add the API wrapper

```
rest_api/  ← Python code
```

Step 3

Wrap in Docker

```
Dockerfile  +  docker-compose.yml
```

Phase I – Code development

- Six commands *to a working API.*

Before any container, before any cloud — get the API running on a developer machine.

1 **Clone the open-cdisc-rules repo**

```
git clone https://github.com/cdisc-org/cdisc-rules-engine.git
```

2 **Install the requirements**

```
python -m pip install -r ./cdisc-rules-engine/requirements-dev.txt
```

3 **Activate the virtual environment**

```
./venv/Scripts/activate
```

4 **Test core.py execution**

```
python ./cdisc-rules-engine/core.py validate -s sdtmig -v 3-4 -d ./data/ -o ./reports/
```

5 **Install REST API packages**

```
python -m pip install uvicorn fastapi python-multipart
```

6 **Start the app**

```
python -m uvicorn api:app --host 0.0.0.0 --port 8000 --reload
```

```
INFO: Started server process · Uvicorn running on http://0.0.0.0:8000  ·  Application startup complete.
```

Phase I – Testing Endpoints with Postman

Does it *actually work?*

Quick smoke tests against the local server. Each call returns either a JSON status or an Excel report.

GET `localhost:8000/health`

Health check

```
↳ Response  { "status": "ok" }
```

GET `localhost:8000/docs`

Swagger UI

```
↳ Response  CDISC Validation API — interactive endpoint browser
```

POST `localhost:8000/validate/xpt`

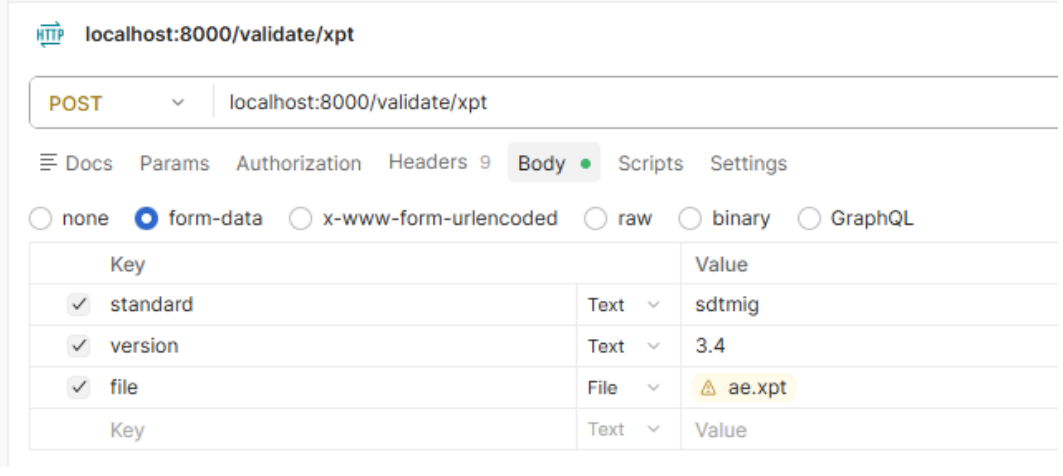
Single XPT dataset validation

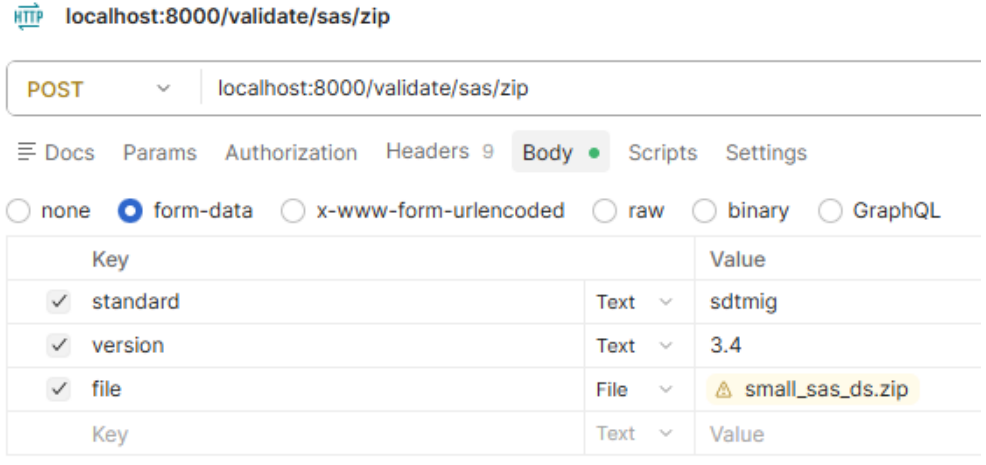
```
↳ Response  ae_validation.xlsx · 14 KB
```

POST `localhost:8000/validate/sas/zip`

ZIP of sas7bdat datasets

```
↳ Response  validation_report.xlsx · 14 KB
```





Phase II – Docker Containerization

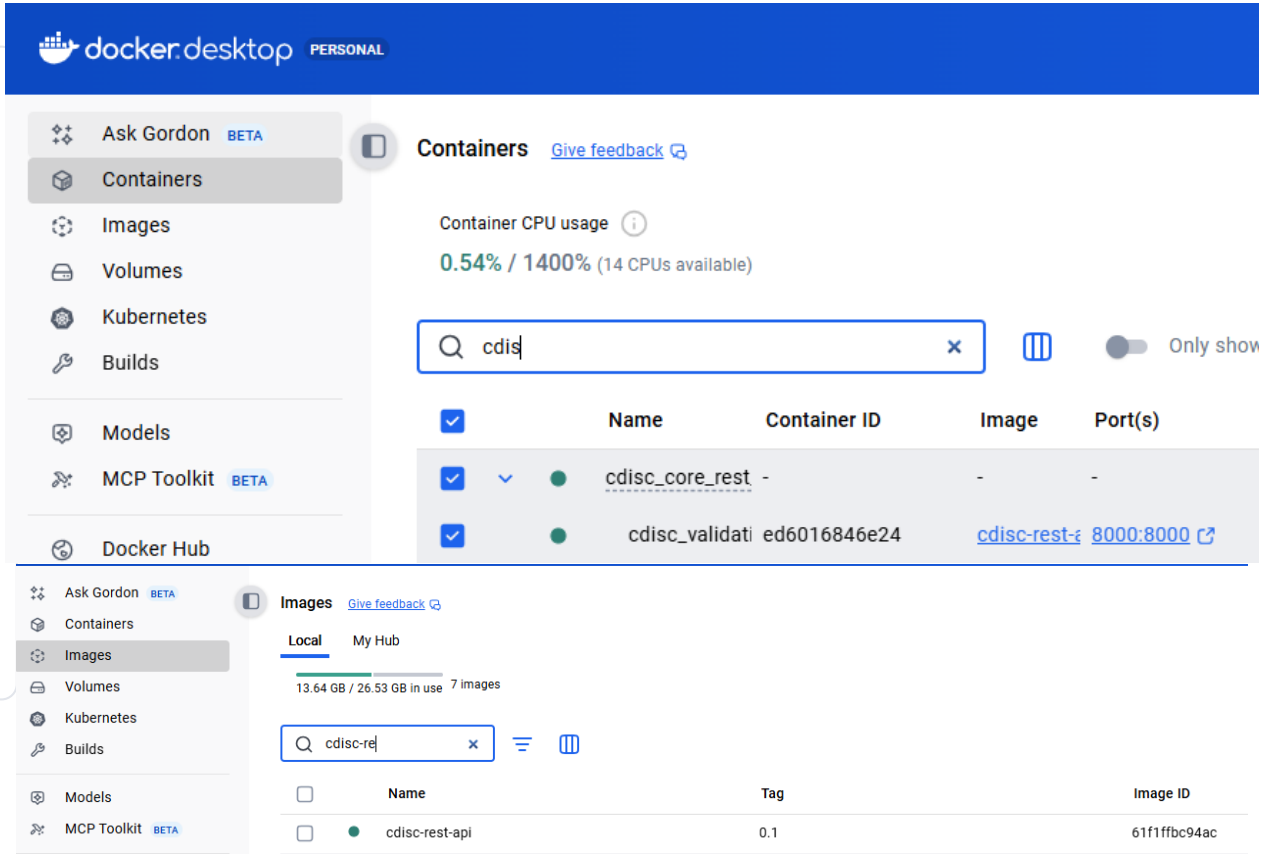
- A clean, repeatable build. Same image on every machine, every time.

DOCKERFILE · 7 STAGES

1. **Base image:** `python:3.12-slim-bookworm`
2. **System deps:** `git, curl, build-essential`
3. **Clone:** `cdisc-rules-engine from GitHub`
4. **pip install:** `engine requirements.txt`
5. **pip install:** `api requirements.txt`
6. **COPY:** `rest_api/ source into image`
7. **Expose:** `8000` · `start uvicorn`

BUILD & RUN

```
$ docker build -t cdisc-rest-api .
$ docker-compose up --build
$ docker run -p 8000:8000 \
    -v $(pwd)/logs:/app/logs cdisc-rest-api
```



⚠ Build Failure → Fixed

Cause: add-apt-repository ppa:deadsnakes called Launchpad — network timeout in Docker build.

Fix: switched to `python:3.12-slim-bookworm` · Python 3.12 built in, no PPA needed.

Phase II – Docker Containerization

Scan first. *Patch what we can.*

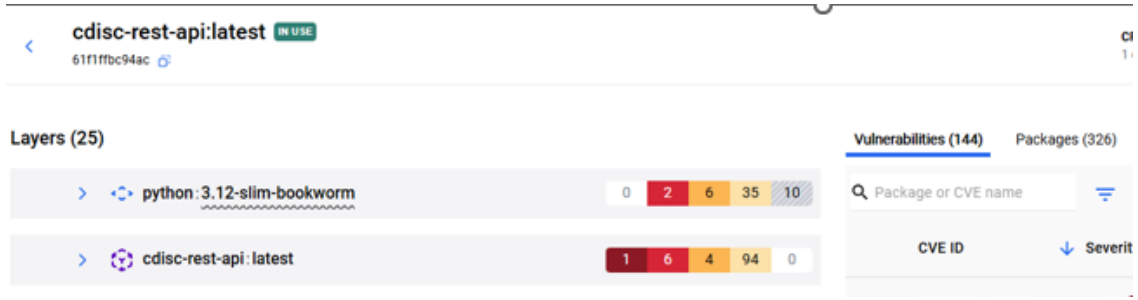
Docker Scout flagged Common Vulnerabilities and Exposures (CVEs) in our base image. We compared base candidates and switched to a cleaner one.

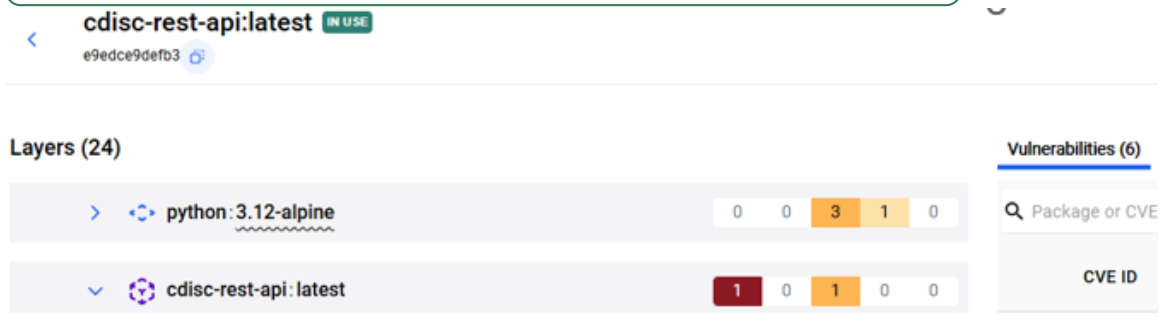
BEFORE · python:3.12-slim-bookworm

1.97 GB	25
Image size	Layers
144	2
Vulnerabilities	Critical CVEs

AFTER · python:3.12-alpine

2.10 GB		24	-1
Image size		Layers	
6	-138	1	-1
Vulnerabilities		Critical CVEs	





One CVE we couldn't fix yet · CVE-2024-10096 (dask 9.8) — still need `from dask.dataframe import dd`. Stuck on dask 2024.6.0 until upstream fix.

Phase II – Docker Containerization

Same API. *Now portable.*

A POST to /validate/sas, then watch the structured log stream live from the running container.

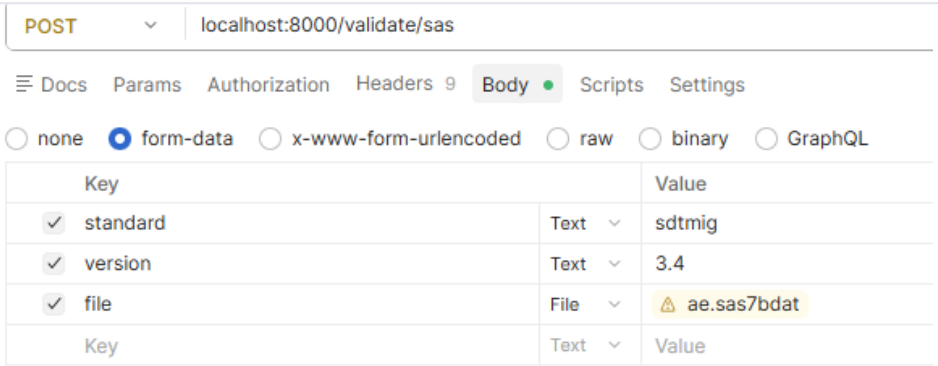
REQUEST

POST localhost:8000/validate/sas

Form-data body

standard	sdtmig
version	3.4
file	ae.sas7bdat (1600 KB)

200 OK · ae_validation.xlsx · 18.94 s · 14.07 KB



Docker container log

```
============================================================
[/validate/sas] Request received
  Start time : 2026-05-03 16:39:47.481
  Standard   : sdtmig
  Version    : 3.4
  File       : ae.sas7bdat
============================================================

  SAS datasets to validate (1 found):
    • ae.sas7bdat  (1600.0 KB)


  Converting 1 SAS7BDAT file(s) to XPT:
    ↳ Converting ae.sas7bdat → ae.xpt ...
       ✓ ae.xpt written (552.3 KB)
  Conversion complete — 1 XPT file(s) ready.

  Command    : /usr/local/bin/python /app/cdisc-rules-engine/core.py validate
p percents -mr 10

============================================================
[/validate/sas] Request completed successfully
  End time   : 2026-05-03 16:40:06.364
  Duration   : 18.882s
============================================================
```

```
  Command    : /usr/local/bin/python /app/cdisc-rules-engine/core.py validate -s sdtmig -v 3.4 -d /tmp/tmppkufpyzk -o /tmp/tmptlm93y6w/results -rt /app/cdisc-rules-engine/resources/templates/report-template.xlsx -
p percents -mr 10
```

Phase II – Docker Containerization

Three things to *know* as an operator.

BUILD & RUN	`$ docker build -t cdisc-core-api:latest .`	*Build image from current directory*
	`$ docker run -d -p 8000:8000 --name cdisc-core-api …`	*Start container · detached, port-bound*
	`$ docker run --restart unless-stopped …`	*Auto-restart on host reboot*
INSPECT & DEBUG	`$ docker ps -a`	*List all containers · running and stopped*
	`$ docker logs -f <container-id>`	*Stream application logs*
	`$ docker exec -it <id> /bin/bash`	*Open interactive terminal inside container*
	`$ env`	*Print environment variables (run inside)*
MANAGE & CLEAN	`$ docker images`	*List all local images*
	`$ docker stop <container-id>`	*Gracefully stop a running container*
	`$ docker rm -f cdisc-core-api`	*Force-remove container*
	`$ docker network ls`	*List Docker isolated networks*

Tip · Use `docker-compose.yaml` to combine all run parameters — port binding, env vars, networks, restart policy — into one file.

cdisc®

Phase III – Cloud Deployment (Azure)

Same container. *Now running on Azure.*

Docker Image
Local Build

→

Azure Container Registry
ACR
(cdisccoreapi)

→

Azure Container Apps
Managed Runtime
+ Autoscaling

→

Public HTTPS Endpoint
REST API
Port 443

Step-by-Step ACR Push

1 Azure Login

```
az login
```

2 ACR Login

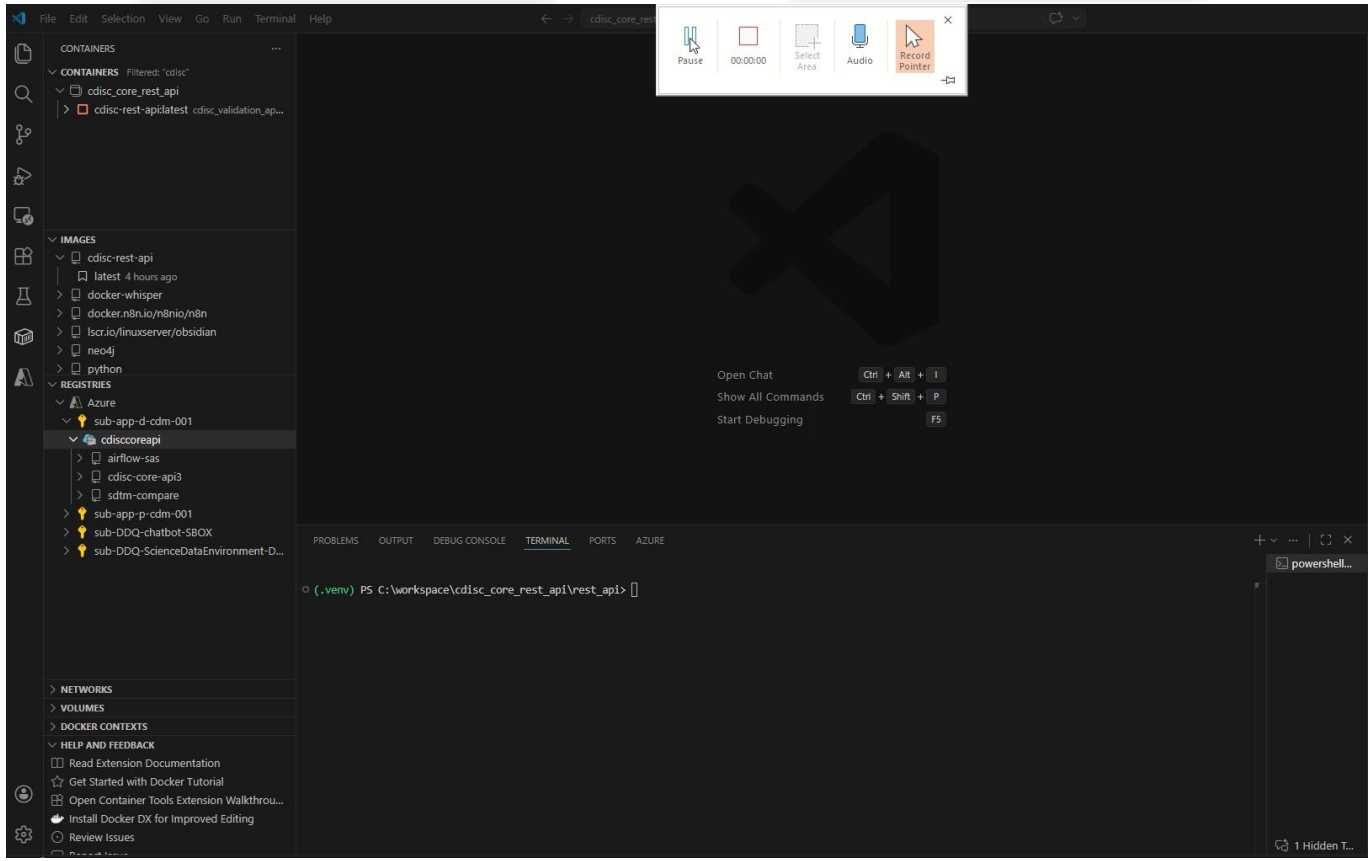
```
az acr login --name cdisccoreapi
```

3 Tag Image

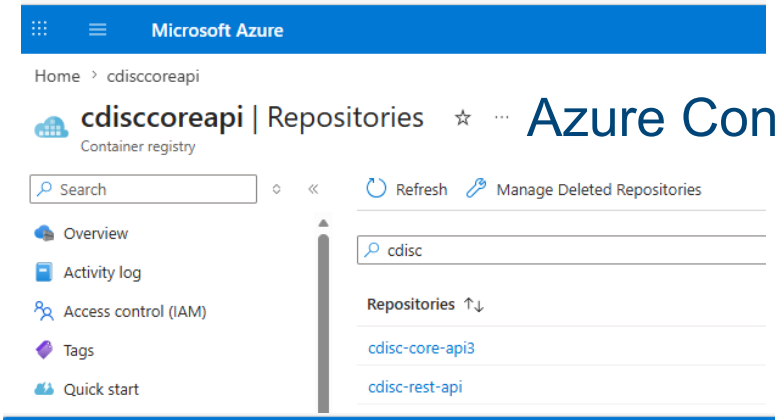
```
docker tag cdisc-core-api3
  cdisccoreapi.azurecr.io/
  cdisc-core-api:latest
```

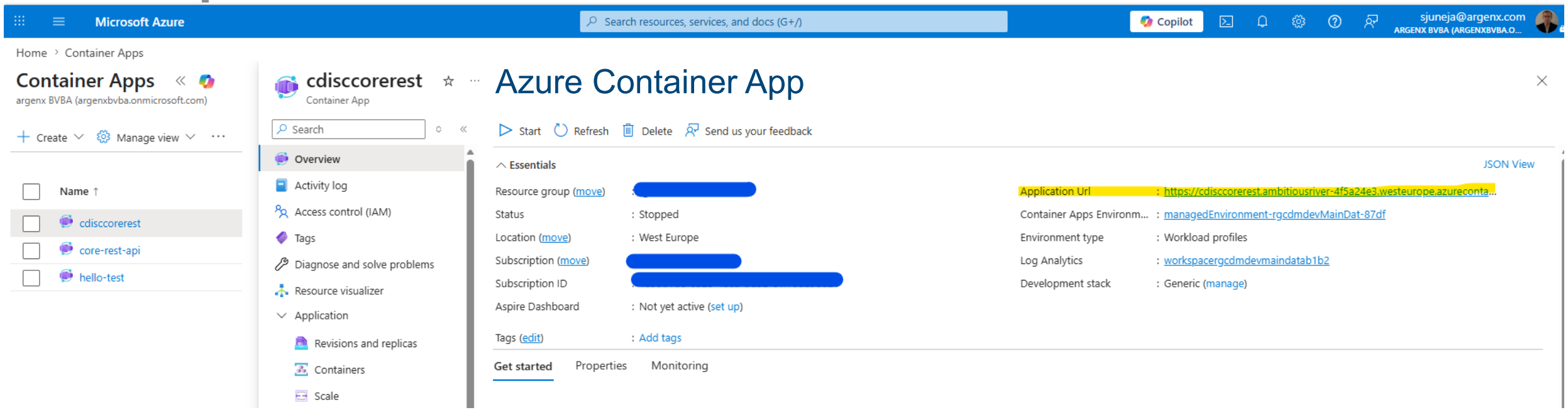
4 Push Image

```
docker push
  cdisccoreapi.azurecr.io/
  cdisc-core-api:latest
```



Phase III – Cloud Deployment (Azure)





Phase III – Cloud Deployment (Azure)

Same Postman. *New URL.*

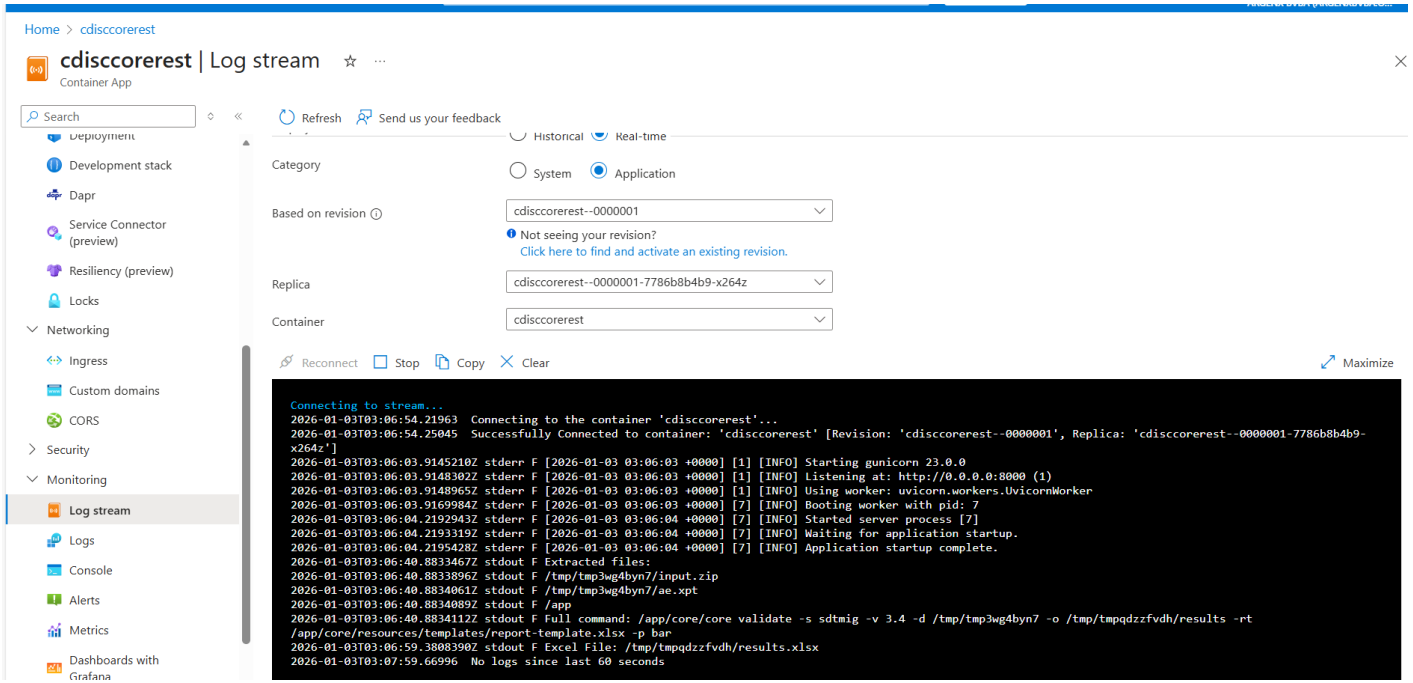
```
POST  https://cdisccorerest.ambitiousriver-4f5a24e3.westeurope.azurecontainerapps.io/validate/xpt
```

POST /validate/xpt

200 OK

```
<binary>  validation_results.xlsx

– Single .xpt validated against
   sdtmig 3.4 ruleset
```



Phase IV – Execution Automation

SAS Code

send the dataset ZIP along-with the rule set name and version(`sdtmig/3.4`) to the API and writes the returned Excel report straight to disk.

```
filename _h_out temp;
filename _cdxpt  "&base_path./data/small_package.zip" recfm=f lrecl=1;
filename _cdresp "&base_path./reports/validation_results.xlsx";
proc http url="http://localhost:8000/validate"
          method="POST"
          out=_cdresp
          headerout=_h_out
          in=MULTI "form-data"
                  ( "sdtmig" header="Content-Disposition: form-data; name=""standard""",
                    "3.4" header="Content-Disposition: form-data; name=""version""",
            _cdxpt header="Content-Disposition: form-data; name=""file""; filename=""small_package.zip"""
            header="Content-Type: application/octet-stream" ) ;
run;
```

```
557   filename _h_out temp;
558   filename _cdxpt  "&base_path./data/small_package.zip" recfm=f lrecl=1;
559   filename _cdresp "&base_path./reports/validation_results.xlsx";
560   proc http url="http://localhost:8000/validate"
561             method="POST"
562             out=_cdresp
563             headerout=_h_out
564             in=MULTI "form-data"
565                     ( "sdtmig" header="Content-Disposition: form-data; name=""standard""",
566                       "3.4" header="Content-Disposition: form-data; name=""version""",
567               _cdxpt header="Content-Disposition: form-data; name=""file""
567 ! filename=""small_package.zip"""
568               header="Content-Type: application/octet-stream" ) ;
569   run;

NOTE: 200 OK
NOTE: PROCEDURE HTTP used (Total process time):
      real time           23.05 seconds
      cpu time            0.14 seconds
```

Phase IV – Execution Automation

Code – Python, R, Java, Curl

curl

```
curl -X POST http://localhost:8000/validate \
  -F "standard=sdtmig" \
  -F "version=3.4" \
  -F "file=@/path/to/data/small_package.zip;type=application/octet-stream"
```

Python

```python
import requests;

url = "http://localhost:8000/validate"
file_path = "path/to/data/small_package.zip"

# The 'data' dict represents the text fields
payload = {
    'standard': 'sdtmig',
    'version': '3.4'
}

# The 'files' dict handles the binary stream and filename
with open(file_path, 'rb') as f:
    files = {
        'file': ('small_package.zip', f, 'application/octet-stream')
    }

    response = requests.post(url, data=payload, files=files)

print(f"Status Code: {response.status_code}")
print(response.text)
```

Java

```java
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Map;
import java.util.ArrayList;
import java.util.List;

public class SasHttpEquivalent {
    public static void main(String[] args) throws Exception {
        String url = "http://localhost:8000/validate";
        Path file = Paths.get("data/small_package.zip");
        String boundary = "SAS_BOUNDARY_" + System.currentTimeMillis();

        // Build the multipart body
        byte[] body = createMultipartBody(boundary, Map.of(
            "standard", "sdtmig",
            "version", "3.4"
        ), file);

        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .header("Content-Type", "multipart/form-data; boundary=" + boundary)
                .POST(HttpRequest.BodyPublishers.ofByteArray(body))
                .build();

        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

        System.out.println("Status: " + response.statusCode());
        System.out.println("Response: " + response.body());
    }

    private static byte[] createMultipartBody(String boundary, Map<String, String> fields, Path file) throws Exception {
        var byteList = new ArrayList<byte[]>();
        String separator = "--" + boundary + "\r\n";

        // Add text fields
        for (var entry : fields.entrySet()) {
            byteList.add(separator.getBytes());
            byteList.add(("Content-Disposition: form-data; name=\"" + entry.getKey() + "\"\r\n\r\n").getBytes());
            byteList.add((entry.getValue() + "\r\n").getBytes());
        }

        // Add file field
        byteList.add(separator.getBytes());
        byteList.add(("Content-Disposition: form-data; name=\"file\"; filename=\"" + file.getFileName() + "\"\r\n").getBytes());
        byteList.add("Content-Type: application/octet-stream\r\n\r\n".getBytes());
        byteList.add(java.nio.file.Files.readAllBytes(file));
        byteList.add(("\r\n--" + boundary + "--\r\n").getBytes());

        // Flatten to single byte array
        int totalLength = byteList.stream().mapToInt(b -> b.length).sum();
        byte[] result = new byte[totalLength];
        int currentPos = 0;
        for (byte[] b : byteList) {
            System.arraycopy(b, 0, result, currentPos, b.length);
            currentPos += b.length;
        }
        return result;
    }
}
```

R

```r
library(httr)

url <- "http://localhost:8000/validate"
file_path <- "path/to/data/small_package.zip"

response <- POST(
  url,
  body = list(
    standard = "sdtmig",
    version  = "3.4",
    file     = upload_file(file_path, type = "application/octet-stream")
  ),
  encode = "multipart"
)

print(status_code(response))
print(content(response, "text"))
```

cdisc

Note: *Code generated using Google Gemini (not tested)*

Benefits - What This Enables for your organization

Validate Any Time

Run conformance checks on-demand — not on a scheduled batch. Get same-day feedback during dataset development, not at submission crunch.

Language-Agnostic

SAS, Python, R, Java, curl — any tool that can make a web request can trigger validation. No dependency on a single programming team.

Always on Latest Rules

On-demand updates to the latest CDISC CORE release. No qualification cycle, no waiting for IT to deploy an update.

Secure & Controlled

Runs outside validated environment (LSAF) in a hardened Azure container. Fully auditable. No data leaves the organization's cloud boundary.

Consistent Reporting

Every run produces the same structured Excel report. Easy to feed into your QC dashboards or submission readiness tracking.

Future-Ready

Authentication, metrics capture, and multi-user support are already scaffolded. Switch them on with minimal effort.

Next Steps - What's Already Scaffolded for the Future

No refactor needed — just uncomment and fill in

Authentication (JWT/OAuth2)

- Implement get_current_user() in auth/dependencies.py
- Uncomment user: CurrentUser in each router
- Uncomment user_id/username in metrics.record()

User Metrics Capture

- user_id, username, client_ip already in metrics.record()
- Fields emit to metrics.log once auth is wired
- Log ingestion: ELK / Splunk / CloudWatch ready

Adding New Endpoints

- Copy any router as template
- Register one line in api.py: app.include_router(...)
- All services reusable — no duplication needed

Production Deployment

- docker-compose WORKERS=4 for multi-process
- Volume mount /app/logs for metrics persistence
- Health check endpoint: GET /health wired in

cdisc®

Thank You!

Sandeep Juneja <sjuneja@argenx.com>