Contributing to EDITO Platform

Tutorial, Service, Process & Data - 15-Minute Guide

Title Slide

Contributing to EDITO Platform

Tutorial, Service, Process & Data

15-Minute Guide

[Your Name]
[Your Organization]

What We'll Cover

Four Ways to Contribute:

📚 **Tutorial** - Edit tutorials.json

🔧 **Service** - Docker → Helm → Playground

⚙️ **Process** - Docker → Helm → Playground

🗺️ **Data** - Create STAC items

Prerequisites (Common to All)

✅ **EDITO Beta Tester Account**
→ Sign up at edito-infra.eu

✅ **GitLab Account**
→ gitlab.mercator-ocean.fr

✅ **Docker** installed locally

✅ **Access to playground repositories**

TUTORIAL SECTION

Tutorial - Overview

Goal: Share interactive R/Python tutorials

Key Steps:

1. Create .Rmd/.ipynb file with content
2. Push to public GitHub repo
3. Configure deployment URL in EDITO Datalab
4. Add entry to tutorials.json
5. Submit merge request

Tutorial - Structure

```
my_tutorial/
├── tutorial.Rmd      ← Your content
├── data/             ← Data files
├── README.md         ← Documentation
└── docker-compose.yml (optional)
```

Tutorial - Create R Markdown

Write your tutorial content:

- Include **Markdown** text
- Add **R code chunks**
- Show plots, tables, results inline
- Add interactivity/user interaction

Example: `stac_r_tutorial.Rmd`

Tutorial - Push to GitHub

```
# Initialize Git
git init
git add .
git config user.name username
git config user.email usermail@mail.com

# Commit
git commit -m "Initial commit"

# Add remote and push
git remote add origin https://github.com/username/stac-r-tutorial.git
git branch -M main
git push -u origin main
```

Important: Repository must be **public**

Tutorial - Configure Deployment URL

In **EDITO Datalab**:

1. Go to Service Catalog
2. Choose service (RStudio/Jupyter)
3. In **GIT section**, add your GitHub repo URL
4. Set resources (CPU, memory)
    - Example: `1600m` CPU, `5Gi` memory
5. **Save** → Copy the deployment URL from browser
6. **Launch** and test your tutorial

Tutorial - Add to tutorials.json

Clone tutorials repository:

```
git clone https://gitlab.mercator-ocean.fr/pub/edito-infra/edito-tutorials-content.git
cd edito-tutorials-content
git checkout -b my-new-tutorial-branch
git push origin my-new-tutorial-branch
```

Tutorial - Edit tutorials.json

Add your tutorial entry:

```json
{
    "name": {
        "en": "My New Tutorial"
    },
    "abstract": {
        "en": "A short description of your tutorial"
    },
    "authors": [
        "The authors and contributors"
    ],
    "types": [
        {
            "en": "Tutorial"
        }
    ],
    "tags": [
        "create",
        "stac"
    ],
    "category": "training courses in data science",
    "imageUrl": "https://www.edito.eu/wp-content/uploads/2023/09/favicon.png",
    "articleUrl": {
        "en": "https://github.com/username/stac-r-tutorial"
    },
```

Tutorial - Submit Merge Request

```
git add .
git commit -m "Added my awesome tutorial to tutorials.json"
git push origin my-new-tutorial-branch
```

Then:

1. Go to GitLab repository
2. Create merge request from your branch
3. Tag `@pub/edito-infra/codeowners`
4. Wait for review and approval

PROCESS SECTION

Process - Overview

Goal: Deploy batch processing workflows

What is a Process?

- Takes input data → transforms → output data
- Runs as batch job (not interactive)
- Examples: ML models, data pipelines, simulations

Process - Workflow Pattern

Three-Stage Pattern:

1. **Download** → Input data from S3 to `/data/input`
2. **Process** → Run scripts, output to `/data/output`
3. **Upload** → Results from `/data/output` to S3

Key: All containers share `/data` directory

Process - Dockerize Your Scripts

Example Structure:

```
my_process/
├── Dockerfile
├── requirements.txt
├── Scripts/
│   ├── 01_data_preparation.R
│   └── 02_model_analysis.R
└── README.md
```

Process - Dockerfile Example

```
FROM rocker/r-ver:4.3.0

# Install system dependencies
RUN apt-get update && apt-get install -y \
    curl \
    libcurl4-openssl-dev \
    libssl-dev \
    && rm -rf /var/lib/apt/lists/*

# Install R packages
COPY requirements.txt /requirements.txt
RUN Rscript -e "install.packages(readLines('requirements.txt'))"

# Copy scripts
COPY Scripts/ /Scripts/

# Set working directory
WORKDIR /data

# Default command
CMD ["Rscript", "/Scripts/01_data_preparation.R"]
```

Process - Build & Push Container

```
# Build the image
docker build -t ghcr.io/yourusername/my-process:1.0.0 .

# Login to registry
export CR_PAT=mycontainerregistrytoken
echo $CR_PAT | docker login ghcr.io -u yourusername --password-stdin

# Push the image
docker push ghcr.io/yourusername/my-process:1.0.0
```

Note: Use semantic versioning (semver.org)

Process - Test Container Locally

```
# Test the container
docker run -v $(pwd)/data:/data ghcr.io/yourusername/my-process:1.0.0
```

Your working process is now usable by anyone with Docker and internet connection

Process - Clone Process Playground

```
git clone https://gitlab.mercator-ocean.fr/pub/edito-infra/process-playground.git
cd process-playground
git checkout -b my-process-workflow
git push origin my-process-workflow
```

Process - Create Helm Chart Structure

```
process-playground/
└── my_process_workflow/
    ├── Chart.yaml
    ├── values.yaml
    ├── values.schema.json
    └── templates/
        ├── job.yaml
        ├── pvc.yaml
        ├── secret-s3.yaml
        └── serviceaccount.yaml
```

Process - Chart.yaml Example

```yaml
apiVersion: v2
name: my-process-workflow
description: A data processing workflow for EDITO
icon: https://example.com/icon.png
home: https://github.com/yourusername/my-process

type: application
version: 0.1.0
appVersion: "1.0.0"

dependencies:
  - name: library-chart
    version: 1.5.14
    repository: https://inseefrlab.github.io/helm-charts-interactive-services
```

Process - values.yaml Configuration

```yaml
# Image configuration
image:
  repository: ghcr.io/yourusername/my-process
  tag: "1.0.0"
  pullPolicy: IfNotPresent

# Processing configuration
processing:
  dataPreparationCommand: "Rscript /Scripts/01_data_preparation.R"
  modelAnalysisCommand: "Rscript /Scripts/02_model_analysis.R"

# Input/Output paths
inputData:
  s3Path: "my-process/input"

output:
  s3Path: "my-process/output"
```

Process - Example Job Configurations/Components

- **S3 Download Init Container** - Downloads input data
- **Processing Containers** - Run your custom commands sequentially
- **S3 Upload Container** - Uploads results
- **Shared Volume** - `/data` directory for all containers
- **Resource Management** - CPU and memory limits

Example: Copy-Output Container (MinIO)

```
- name: upload-results
  image: minio/mc
  command: ["/bin/sh", "-c"]
  args:
    - |
      echo "=== Waiting for processing to complete ==="

      # Wait for output files in /data/output
      while [ ! -d "/data/output" ] || [ -z "$(ls -A /data/output 2>/dev/null)" ]; do
        echo "Waiting for output files..."
        sleep 30
      done

      echo "Output files found. Copying to user storage..."

      # Set up MinIO client
      export MC_HOST_s3=https://$(AWS_ACCESS_KEY_ID):$(AWS_SECRET_ACCESS_KEY):$(AWS_SESSION_TOKEN)@$(AWS_S3_ENDPOINT)

      # Copy from /data/output to user storage
      mc cp --recursive /data/output/ s3/user-namespace/my-process/output/

      echo "=== Upload completed ==="
  volumeMounts:
    - name: data-volume
      mountPath: /data**Key Points:**
```

Process - Commit & Deploy

```
git add .
git commit -m "Added my awesome process"
git push origin my-process-workflow
```

Then:

1. Check pipeline in GitLab
2. Wait for deployment (5-10 min)
3. Test in process playground
4. Create merge request for production

SERVICE SECTION

Service - Overview

Goal: Deploy interactive web applications

Examples: R Shiny apps, Python Dash, Jupyter notebooks

Key Technology: Docker + Kubernetes + Helm

Service vs Process vs Tutorial

Service			
Process	Data transformation	Batch job	ML model
Tutorial			

Service - Dockerize Your App

Example Structure:

```
my_service/
├── Dockerfile
├── app/
│   ├── ui.R
│   ├── server.R
│   └── global.R
└── README.md
```

Service - Dockerfile Example

```dockerfile
FROM rocker/shiny:4.5.0

# Install system dependencies
RUN apt-get update && apt-get install -y \
    libcurl4-openssl-dev \
    libssl-dev \
    libxml2-dev \
    && rm -rf /var/lib/apt/lists/*

# Install R packages
RUN R -e "install.packages(c('shiny', 'arrow', 'leaflet'))"

# Create app folder and copy files
RUN mkdir -p /srv/shiny-server
COPY app/ui.R app/server.R app/global.R /srv/shiny-server/

# Copy the startup script
COPY app/start_app.sh /start.sh
RUN chmod +x /start.sh

# Expose port
EXPOSE 3838

# Start Shiny server
CMD ["/start.sh"]
```

Service - Build & Push Container

```
# Build and version your container
docker build -t ghcr.io/yourusername/view_parquet:1.0.1 .

# Login to registry
export CR_PAT=mycontainerregistrytoken
echo $CR_PAT | docker login ghcr.io -u yourusername --password-stdin

# Push
docker push ghcr.io/yourusername/view_parquet:1.0.1
```

Service - Test Public Image

```
docker run -p 3838:3838 ghcr.io/edito-infra/view_parquet:1.0.4
```

Open browser: `http://localhost:3838/`

Your working app version is now usable by anyone with Docker and internet connection

Service - Clone Service Playground

```
git clone https://gitlab.mercator-ocean.fr/pub/edito-infra/service-playground.git
cd service-playground

# Make your own branch
git checkout -b parquet_viewer_r
git push origin parquet_viewer_r

# Copy existing service as template
cp -r terria-map-viewer parquet_viewer_r
```

Service - Helm Chart Structure

```
service-playground/
└── my_service/
    ├── Chart.yaml
    ├── values.yaml
    ├── values.schema.json
    └── templates/
        └── deployment.yaml
```

Service - Chart.yaml Example

```yaml
name: view-parquet
description: An interactive Parquet viewer on EDITO
home: https://github.com/yourusername/view_parquet
icon: https://your.icon.url/icon.png
keywords: [shiny, r, parquet, viewer]
version: 1.0.0
appVersion: "1.0.3"
dependencies:
  - name: library-chart
    version: 1.5.16
    repository: https://inseefrlab.github.io/helm-charts-interactive-services
```

Service - values.yaml Configuration

```yaml
service:
  image:
    version: "ghcr.io/yourusername/view-parquet:1.0.3"

networking:
  service:
    port: 3838
```

Service - values.schema.json

```json
{
  "properties": {
    "service": {
      "properties": {
        "image": {
          "properties": {
            "version": {
              "listEnum": [
                "ghcr.io/yourusername/view-parquet:1.0.3",
                "ghcr.io/yourusername/view-parquet:1.0.1"
              ],
              "default": "ghcr.io/yourusername/view-parquet:1.0.3"
            }
          }
        }
      }
    }
  }
}
```

Service - Commit & Deploy

```
# Run formatting check
make check-format

# Commit your changes
git add .
git commit -m "Added my awesome service"
git push origin parquet_viewer_r
```

Then:

1. Check pipeline in GitLab
2. Wait 5-10 min for deployment
3. Launch from EDITO Datalab
4. Create merge request for production

COMMON WORKFLOW - SERVICE & PROCESS

Common Pattern: Service & Process

Both follow the same workflow:

1. **Dockerize** - Create Dockerfile
2. **Build & Push** - Push to container registry
3. **Create Helm Chart** - Configure Kubernetes deployment
4. **Clone Playground** - GitLab repository
5. **Add Your Chart** - Copy template, customize
6. **Commit & Push** - Git workflow
7. **Test in Playground** - Deploy and verify
8. **Submit MR** - Merge request for production

Common - Git Workflow

For Service & Process:

```
# Clone playground repo
git clone https://gitlab.mercator-ocean.fr/pub/edito-infra/[playground].git

# Create branch
git checkout -b my-contribution

# Add your chart/config
# ... make changes ...

# Commit & push
git add .
git commit -m "Added my contribution"
git push origin my-contribution
```

Common - Testing Before Production

✅ **Test locally** (Docker)
✅ **Test in playground**
✅ **Verify pipeline passes**
✅ **Check deployment works**

Then create merge request!

Common - Production Release

Submit Merge Request:

1. Go to GitLab repository
2. Create merge request from your branch
3. Tag `@pub/edito-infra/codeowners`
4. Wait for review
5. Once approved → Live on EDITO!

REVIEW & COMPARISON

Review: What's Different?

Tutorial				
Service	Dockerized web app	Container registry	service-playground	Docker → Helm → Git
Process				

Review: Common Elements

Service & Process share:

- ✅ Docker containerization
- ✅ Container registry (build & push)
- ✅ Helm charts
- ✅ GitLab workflow
- ✅ Playground testing
- ✅ Merge request process
- ✅ EDITO platform deployment

Tutorial is different:

- ✅ GitHub repository
- ✅ Deployment URL configuration
- ✅ Edit tutorials.json

Review: Quick Reference

Tutorial:

→ GitHub repo + tutorials.json + deployment URL

Service:

→ Container registry + Helm chart + service-playground

Process:

→ Container registry + Helm chart + process-playground

All:

→ GitLab + Merge Request

DATA SECTION

Data - Overview

Goal: Add your datasets to the EDITO Data Lake

Key Technology: STAC (SpatioTemporal Asset Catalog)

What is STAC?

→ Standardized JSON metadata format
→ Describes when, where, and what your data contains
→ Links to actual data files
→ Searchable and discoverable

Specification: stacspec.org

Data - STAC Structure

STAC Hierarchy:

📦 **Catalog** - Top-level container, links to Collections

📚 **Collection** - Groups related Items (e.g., climate forecasts)

📄 **Item** - Individual dataset with geometry, properties, assets

🔗 **Asset** - Link to actual data file (NetCDF, Parquet, etc.)

Data - Reading STAC Catalogs

```python
import pystac

# Connect to EDITO STAC catalog
stac_url = "https://api.dive.edito.eu/data/catalogs/Galicia_CCMM_catalog"
stac = pystac.Catalog.from_file(stac_url)

# Save locally for offline exploration
stac.normalize_and_save("data/mystac/", catalog_type="SELF_CONTAINED")
```

What it does:

✅ Connects to EDITO STAC catalogs

✅ Downloads metadata for offline exploration

✅ Preserves catalog structure locally

Data - STAC Item Required Fields

Required Fields:

- `id` - Unique identifier
- `type` - Must be `"Feature"`
- `stac_version` - STAC version (e.g., `"1.0.0"`)
- `geometry` - GeoJSON geometry (Polygon, Point, etc.)
- `properties` - Must include `datetime` OR `start_datetime` / `end_datetime`
- `assets` - Links to actual data files

Recommended: title, description, providers, bbox

Data - Creating STAC Item

```python
from pystac.validation import validate_dict
import pystac

metadata = {
    "type": "Feature",
    "stac_version": "1.0.0",
    "id": "example-item-001",
    "properties": {
        "datetime": "2020-01-01T12:00:00Z"
    },
    "geometry": {
        "type": "Polygon",
        "coordinates": [[[5.0, 51.0], [5.1, 51.0],
                        [5.1, 51.1], [5.0, 51.1],
                        [5.0, 51.0]]]
    },
    "bbox": [5.0, 51.0, 5.1, 51.1],
    "assets": {
        "data": {
            "href": "https://example.org/data/file.tif",
            "type": "image/tiff",
            "roles": ["data"]
        }
    }
}

# Validate
validate_dict(metadata)

# Create STAC Item object
item = pystac.Item.from_dict(metadata)
```

Data - Creating STAC Collection

```
{
  "type": "Collection",
  "stac_version": "1.0.0",
  "id": "my-collection",
  "title": "My Ocean Data Collection",
  "description": "Collection of oceanographic datasets",
  "license": "CC-BY-4.0",
  "extent": {
    "spatial": {"bbox": [[-180, -90, 180, 90]]},
    "temporal": {
      "interval": [["2020-01-01T00:00:00Z", null]]
    }
  },
  "links": [
    {
      "rel": "items",
      "href": "./items/"
    }
  ]
}
```

Data - Creating STAC Catalog

```json
{
  "type": "Catalog",
  "stac_version": "1.0.0",
  "id": "my-catalog",
  "title": "My Data Catalog",
  "description": "Catalog of marine datasets",
  "links": [
    {
      "rel": "self",
      "href": "./catalog.json"
    },
    {
      "rel": "child",
      "href": "./collections/my-collection/catalog.json"
    }
  ]
}
```

Data - Creating STAC from Data Files

```
# From NetCDF
python make_stac_from_data.py netcdf my_data.nc <data_url>

# From Zarr
python make_stac_from_data.py zarr my_data.zarr <data_url>

# From Parquet
python make_stac_from_data.py parquet my_data.parquet <data_url>
```

What it does:

✅ Extracts spatial bounds (lat/lon)

✅ Extracts temporal range (datetime)

✅ Reads metadata (title, license, etc.)

✅ Creates valid STAC item

✅ Validates output

Data - EDITO Data Lake

Three main components:

📊 **STAC Catalog** - Metadata and discovery

🗄️ **Object Storage** - Actual data files (S3-compatible)

🔗 **API Access** - `api.dive.edito.eu/data`

Your workflow:

1. Create STAC item from your data
2. Upload data to accessible storage
3. Post STAC item to EDITO API

SUMMARY & RESOURCES

Summary

What We Covered:

✅ **Tutorial** - Edit tutorials.json
✅ **Service** - Dockerized web applications
✅ **Process** - Batch processing workflows
✅ **Data** - STAC items/collections/catalogs

All use: GitLab, Git workflow, and EDITO platform

Getting Started

Next Steps:

1. **Get Account** - Sign up as beta tester at edito-infra.eu
2. **Access GitLab** - Create account at gitlab.mercator-ocean.fr
3. **Choose Your Path** - Tutorial, Service, Process, or Data
4. **Follow Documentation** - Check workshop repository

Repository: github.com/EDITO-Infra/edito-workshops-presentations

Resources

Documentation:

- **EDITO Docs:** docs.dive.edito.eu
- **Workshop Repo:** github.com/EDITO-Infra/edito-workshops-presentations
- **STAC Spec:** stacspec.org

Platforms:

- **EDITO Datalab:** datalab.dive.edito.eu
- **GitLab:** gitlab.mercator-ocean.fr/pub/edito-infra

Contact: edito-infra-dev@mercator-ocean.eu

Thank You!

Questions?

Contact:

- Email: [your-email]
- EDITO Community: edito-infra.eu

Funded by the European Union