

INTERFACE

Nutzerworkshop

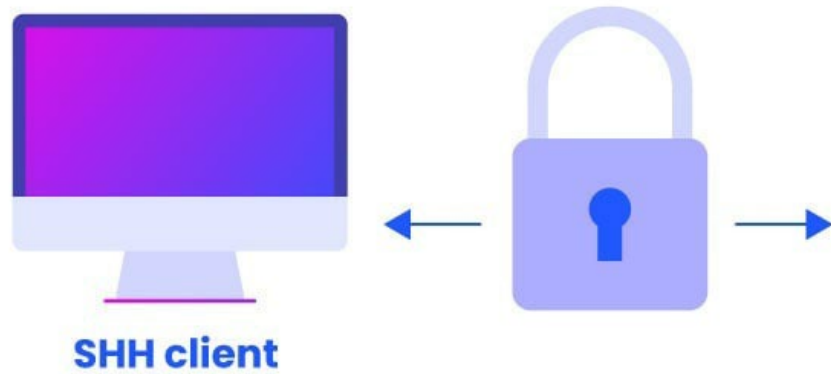
Technische Lösungen

<https://eodc.eu>

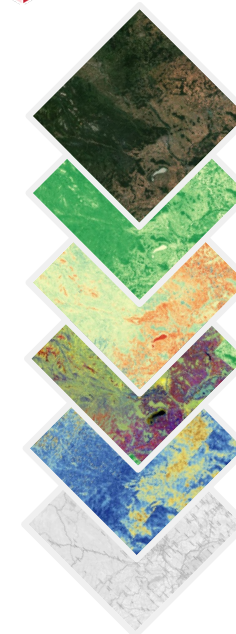
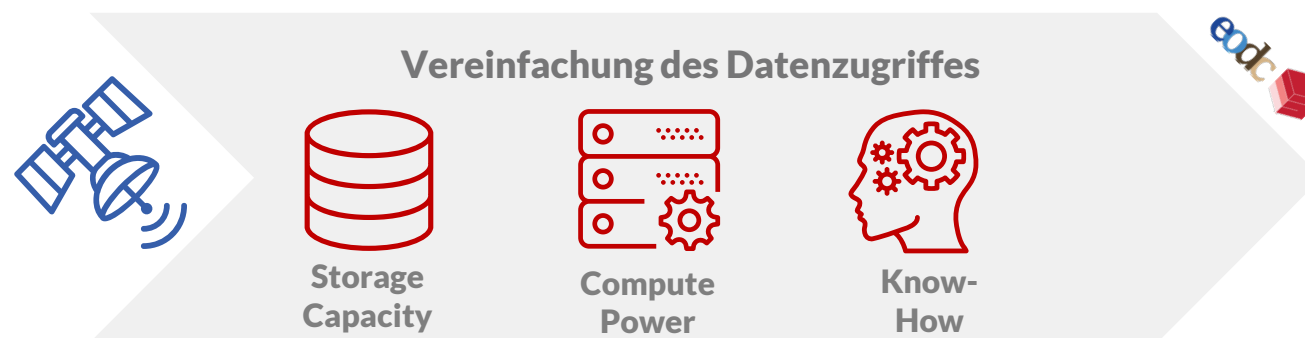


Das Problem mit Daten

Wie komme ich nun zu Daten



- Rohdaten sind ungeeignet für eigentliche Anwendungen
- Vor-prozessiert Sentinel-1 & Sentinel-2 Daten
- "Value-added data products"
- Bereitstellung von einfachen Interfaces zur direkten Daten Analyse (Data Science)



- OpenDataCube (ODC) – ein Set an open-source Python Libraries und einer Datenbank (PostgreSQL)
 - Räumliche-zeitliche Abfragen, um Daten zu laden
- Jupyter Ecosystem – JupyterHub, Lab/Notebooks
 - Keine SSH-Verbindung mehr notwendig
 - Vorinstallierte Umgebung zur Daten Analyse
- "Lessons learned"
 - Indexierung von Daten in ODC war aufwendig
 - keine Möglichkeit bestehenden Datenkatalog (CSW) einzubinden
 - Datenbank und Daten-Storage limitierten noch immer den Zugriff



Event documentation - Flood S2

Pettau am Arlberg: <https://www.express.co.uk/news/world/997981/Austria-news-flood-shock-video-flash-flood-Tyrol-Europe-weather-latest>

```
[051] import datacube
import xarray
Smartplotlib inline
dc = datacube.Datacube(app='Event Documentation')

[061] query = {
    'lon': (10.335, 10.35),
    'lat': (47.145, 47.155),
    'time': ('2018-07-10', '2018-08-20'),
}

data = dc.load(product='TCI_Sentinel_2', output_crs='EPSG:32633', resolution=(-10, 10), **query)

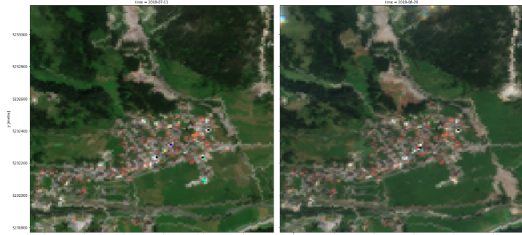
[071] datasets = []
temporal_data = ['2018-07-11', '2018-08-20']
for time in temporal_data:
    datasets.append(data.sel(time=time))

new_data = xarray.concat([dataset for dataset in datasets], dim='time')

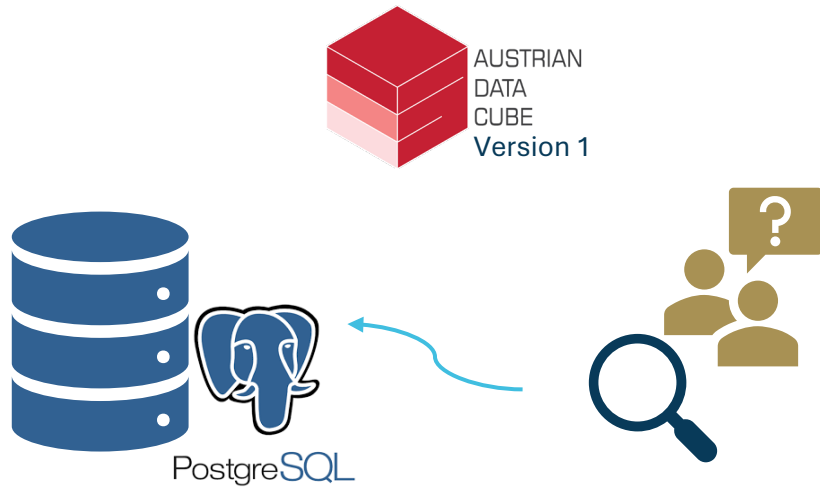
***

[091] new_data.plot.imshow(x=data.crs.dimensions[1], y=data.crs.dimensions[0], col='time', size=10, col_wrap=2)

[091] xarray.plot.facetgrid at 0x7f661b127748
```



Welche Daten sind nun verfügbar?



Data Catalog

The EODC Data Catalog includes petabytes of environmental monitoring data, in consistent, analysis-ready formats. All of the datasets below can be accessed via Azure Blob Storage, and can be used by developers whether you're working within or outside of our EODC Hub.

Featured

Featured

Orthofotos

Other

Sentinel-1

Sentinel-2



Sentinel 1

EODC provides global access to Copernicus Sentinel-1 radar data, enabling industries and researchers to seamlessly utilize high-quality satellite imagery for environmental monitoring and analysis.

Sentinel1 Imagery



Sentinel 2

EODC provides global access to Copernicus Sentinel-2 imagery, enabling industries and researchers to utilize high-resolution satellite data for applications like agriculture, forestry, and land use analysis.

Sentinel2 Imagery

Orthofotos



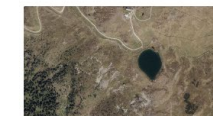
Digital Orthophotos (DOP) Austria - Land Kärnten: Orthofotos Flugblock Klagenfurt

orthofotos orthoimage carinthia klagenfurt



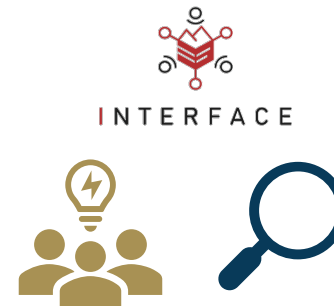
Digital Orthophotos (DOP) Austria - Land Kärnten: Orthofotos Flugblock Osttirol

orthofotos orthoimage Osttirol east.tyrol



Digital Orthophotos (DOP) Austria - Land Kärnten: Orthofotos Flugblock Tamsweg

orthofotos orthoimage tamsweg carinthia



- Spezifikation zur Beschreibung von Geoinformation via JSON
 - Statisch vs. Dynamisch (STAC API)
- STAC dient zur **Suche** und **Erkundung** von Geodaten
- Philosophie hinter STAC ist *“keep it simple, yet flexible and extensible”*, im Gegenteil zu OGC CSW
- STAC ist KEIN Metadaten Standard wie ISO 19115
- Die Entwicklung von STAC fokussiert sich ausschließlich auf Raster/Array Daten



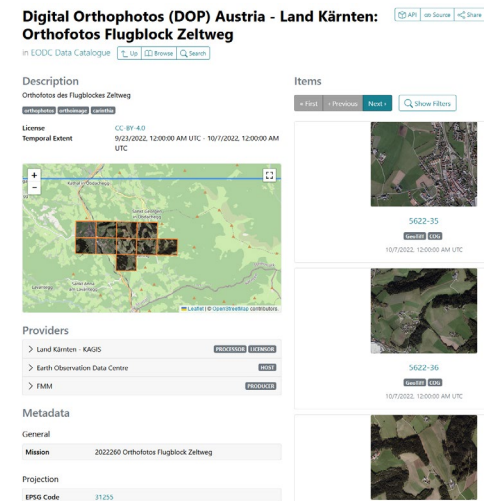
- STAC API stellt eine Anzahl an REST Endpunkten zur Verfügung um nach bestimmten STAC Items suchen zu können.
- CQL2 (OGC) als Abfragesprache zum Filtern der Daten



Storage [HA Database]



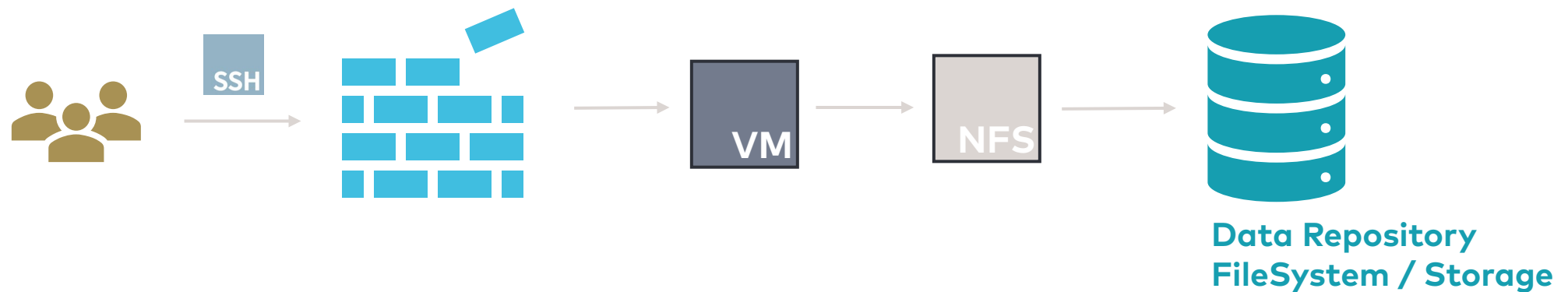
STAC API
Implementierung

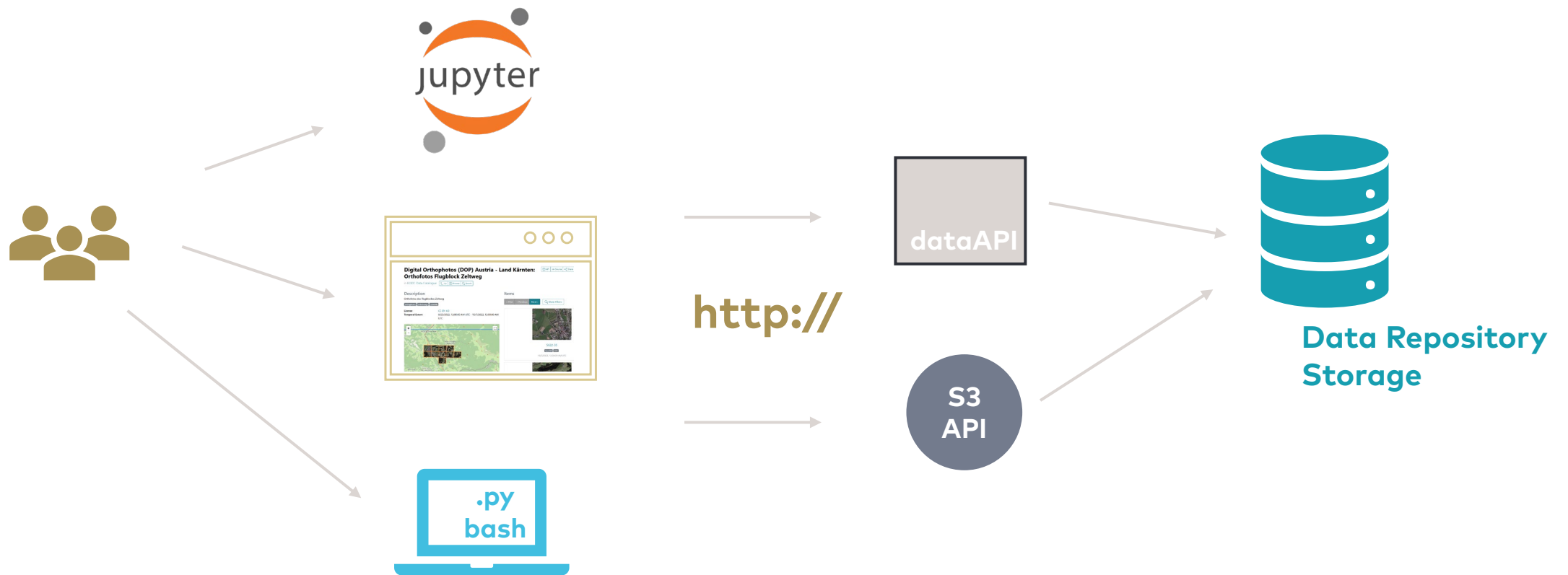


Warum muss es nun STAC sein

- STAC basiert auf JSON und RESTful APIs, einfacher als OGC CSW (XML)
- STAC (JSON) funktioniert mit allen Web Technologien und wurde speziell für Cloud Anwendungen entwickelt (cloud native)
- STAC API Query Performance, es müssen keine XML analysiert werden
- Große Community and umfangreiches Ecosystem
 - STAC API Server sind einfacher zu deployen und zu warten
 - Clients verfügbar in Python, C, R, Rust, Javascript (Web Browser), ...

- Traditionelle Systeme nicht konzipiert für die Cloud
- Zugriffsberechtigungen kompliziert zu pflegen
- Etablierte Protokolle wie NFS sind performant aber nicht benutzerfreundlich





- Einfacher und schneller Zugriff auf Daten
 - kein kompliziertes navigieren auf einem Filesystem
- eodc dataAPI: read-only Zugriff auf File basierten Storage
- S3 API: Zugriff auf ObjectStorage (CEPH)
- Zusätzliche Funktionen verfügbar
 - Granulare Zugriffskontrolle (RBAC, PBAC)
 - Teilen mit anderen Usern
 - Zugriff auf einzelne Files in einem ZIP Archiv, etc.
 - Range-Requests – "cloud native" Datenformate

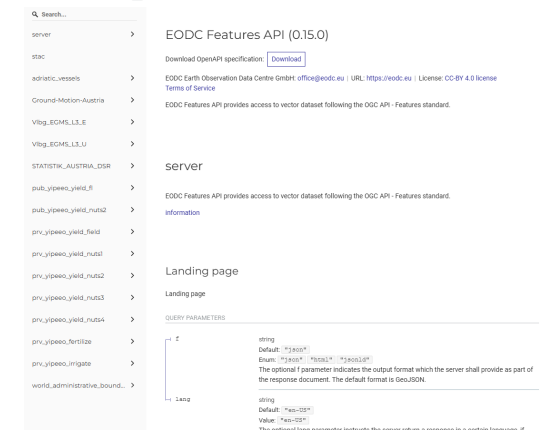


Zarr

[tile]DB GEOJSON

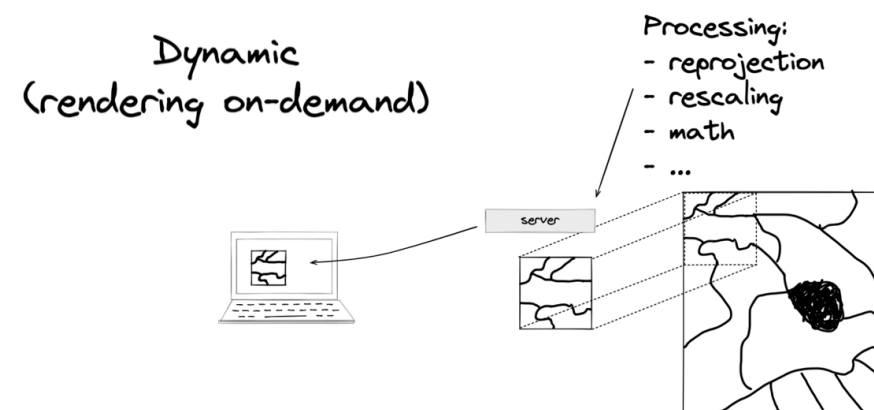


- Aufbauend auf OGC Web Services Standards (WMS, WFS, WCS, WPS, etc.)
- HTTP APIs basierend auf (Geo)JSON
 - Metadaten und Dokumentation direkt via API verfügbar (selbstbeschreibend)
- pygeoapi
 - Referenz-Implementierung in Python
 - Daten können in verschiedenen Formaten bereitgestellt werden (xarray, GeoParquet, Postgis, ...)
 - OGC API Features (WFS) für Vektor Daten





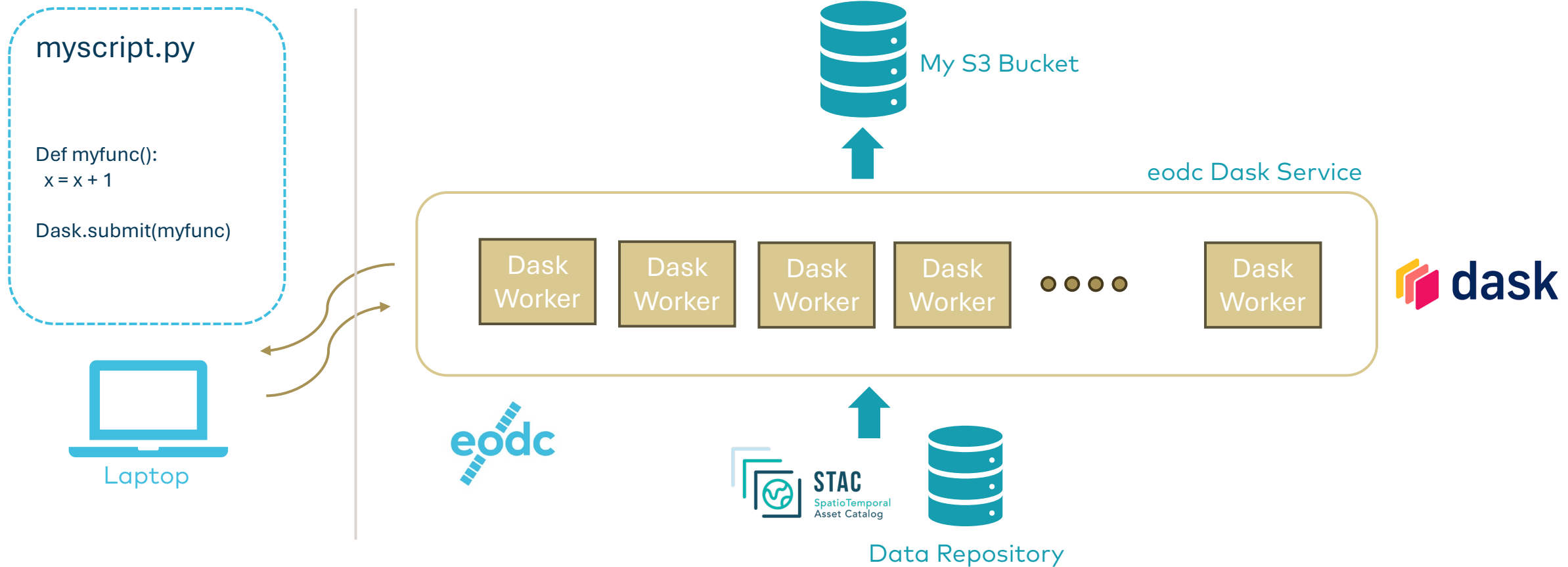
- TiTiler: Dynamisches, "on-the-fly" Rendering von Rasterdaten
- Keine Vorprozessierung notwendig, um Daten zu visualisieren (pre-rendered / static)
- Daten werden via HTTP API ausgeliefert
 - Daten können beliebig skaliert werden
 - Band-Math Operationen möglich
 - Beliebige Colormaps





- Wie kann ich meine Software/Code in einer Cloud Umgebung ausführen?
- IaaS Ansatz: Virtuelle Maschine (VM)
 - Oft zu kompliziert und skaliert nicht
- Dask: Python Framework für "Parallel and distributed computing"
 - Ausführen von Python Code in einer Cloud Umgebung
- ArgoWorkflows – Workflow Engine für Kubernetes
 - Ausführen von Arbeitsschritten, wobei jeder Arbeitsschritt in einem Container ausgeführt wird.





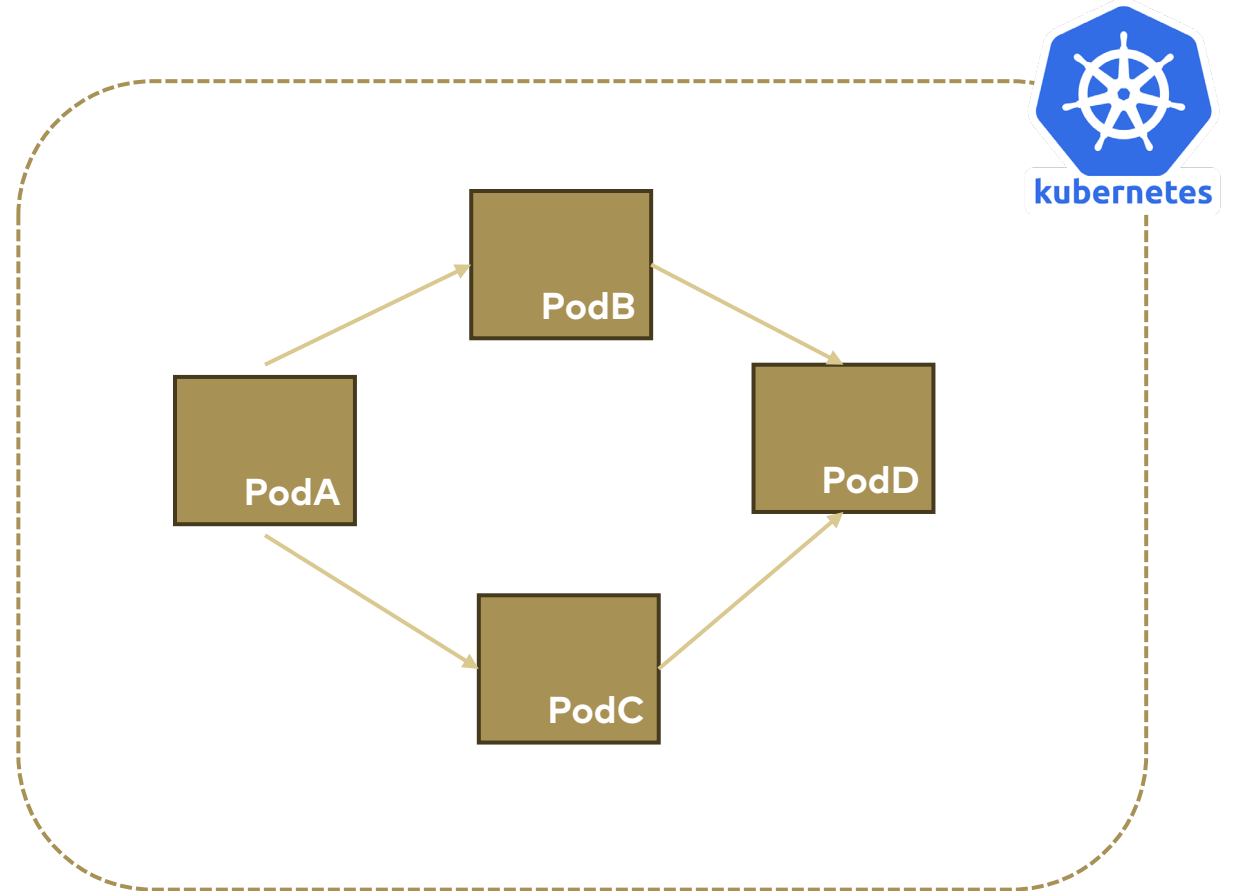
myargowf.py

```
from hera.workflows import DAG, Workflow, script

@script()
def echo(message):
    print(message)

with Workflow(generate_name="dag-diamond-", entrypoint="diamond") as w:
    with DAG(name="diamond"):
        A = echo(name="A", arguments={"message": "A"})
        B = echo(name="B", arguments={"message": "B"})
        C = echo(name="C", arguments={"message": "C"})
        D = echo(name="D", arguments={"message": "D"})

    A >> [B, C] >> D
```



Danke

Details zu den einzelnen Services im Workshop



Translating data
into knowledge