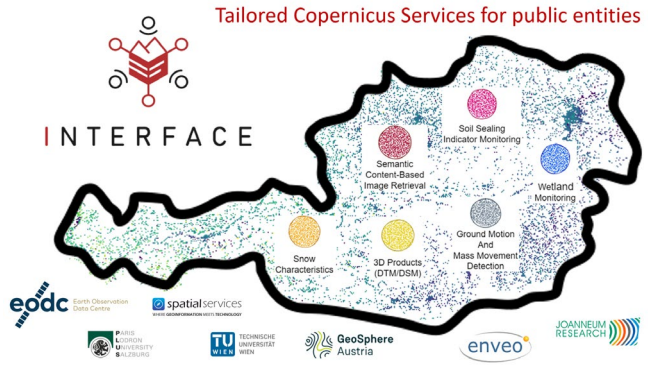
Earth Observation Data Centre



INTERFACE Nutzerworkshop

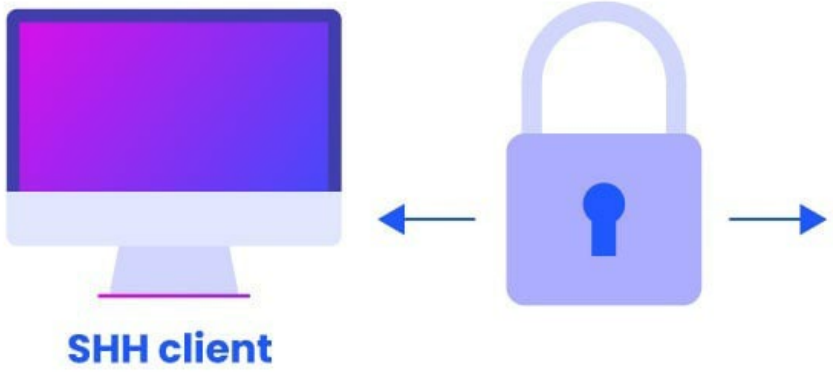
Technische Lösungen

https://eodc.eu

INTERFACE

Das Problem mit Daten

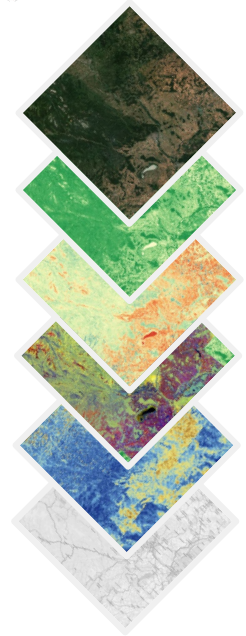
Wie komme ich nun zu Daten

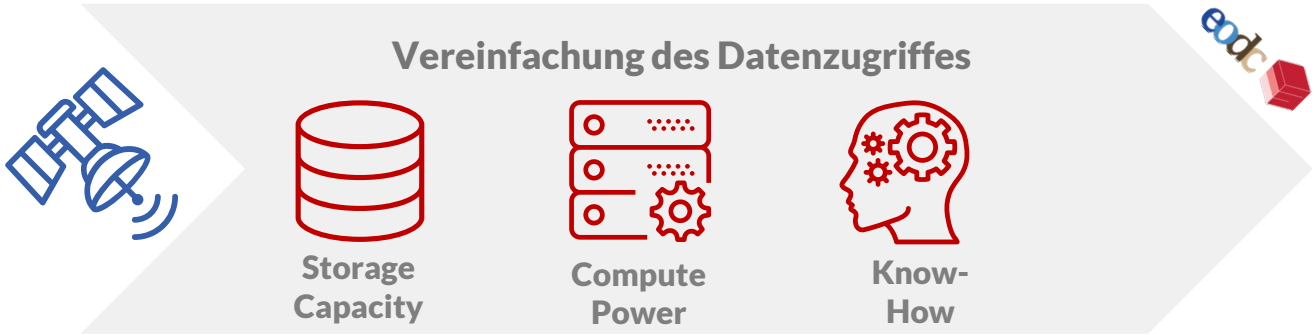


Austrian Data Cube (version 1)

- 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)

AUSTRIAN DATA CUBE

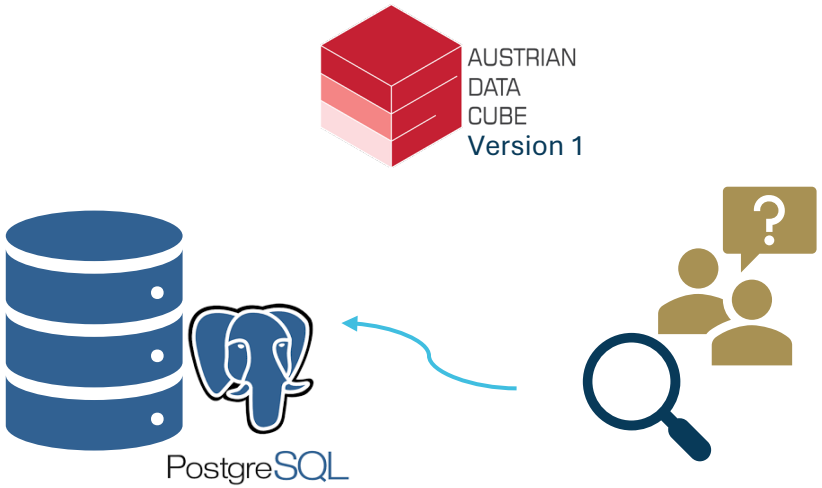


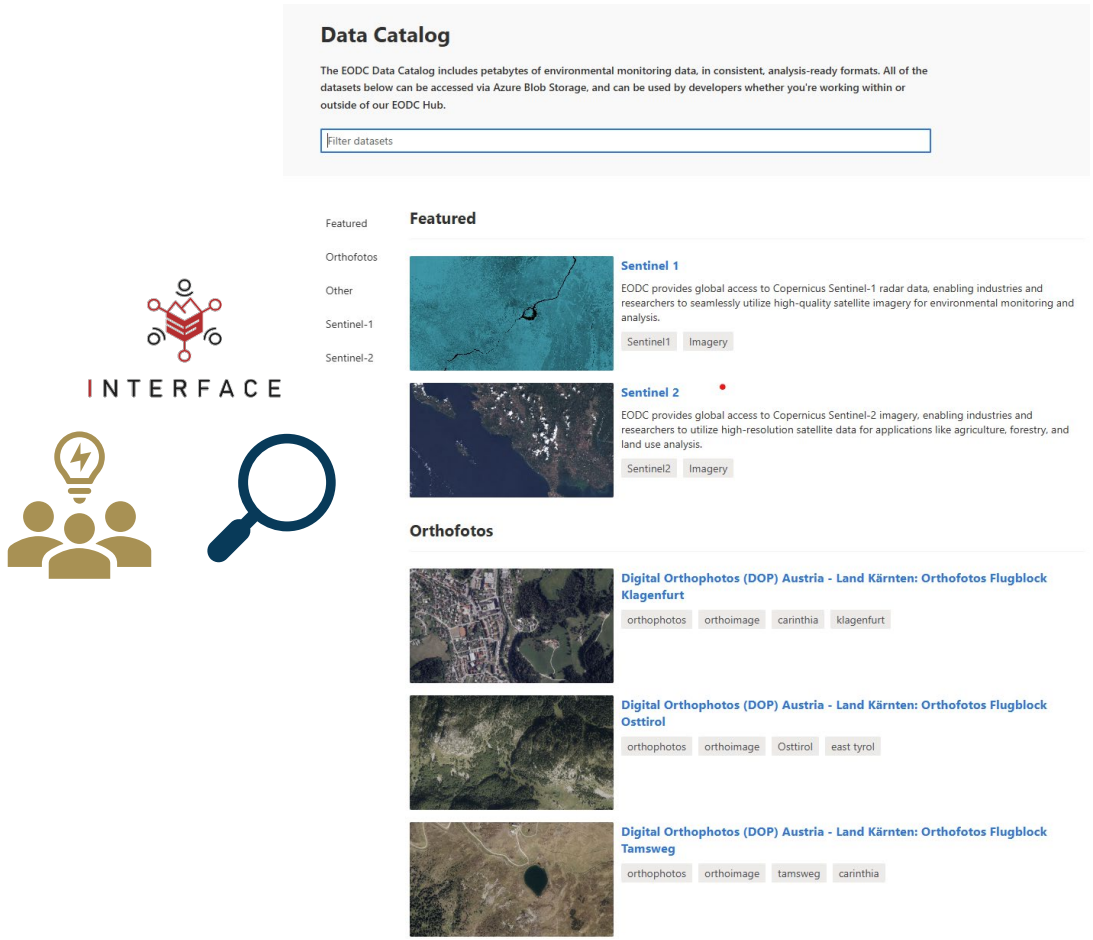


Technologien – Acube version 1

- 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

Welche Daten sind nun verfügbar?



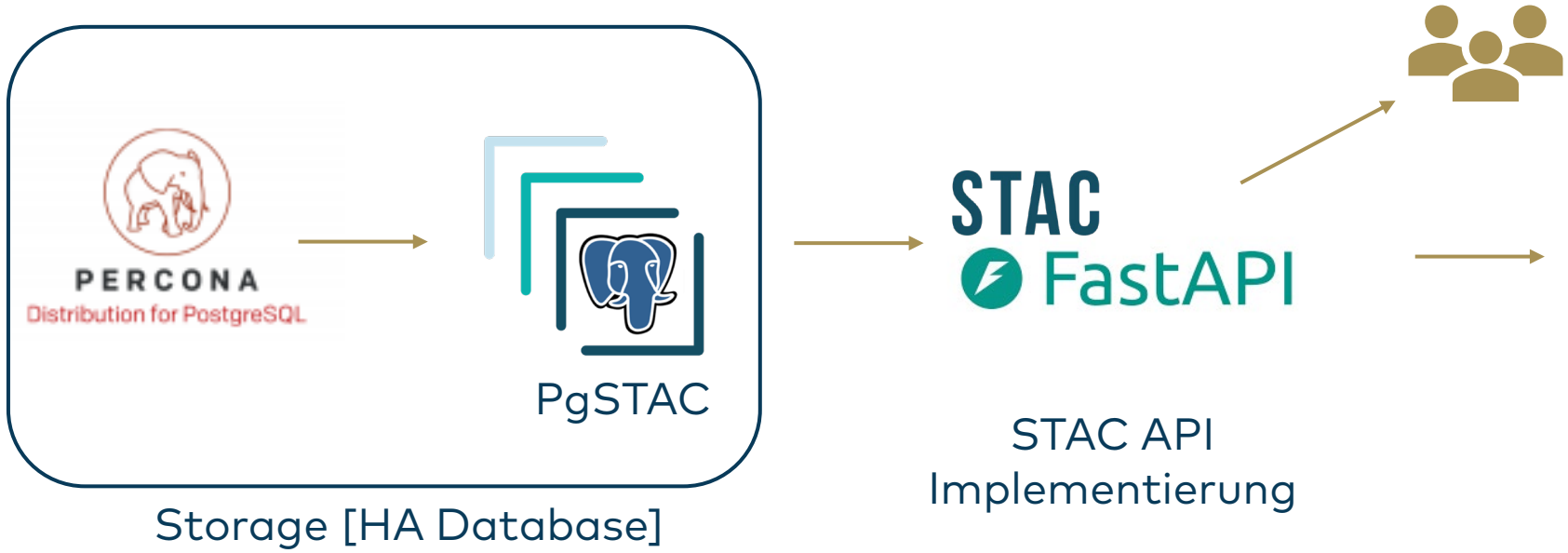


STAC – Spatial Temporal Asset Catalog

- 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 als neuer Index für Acube

- 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

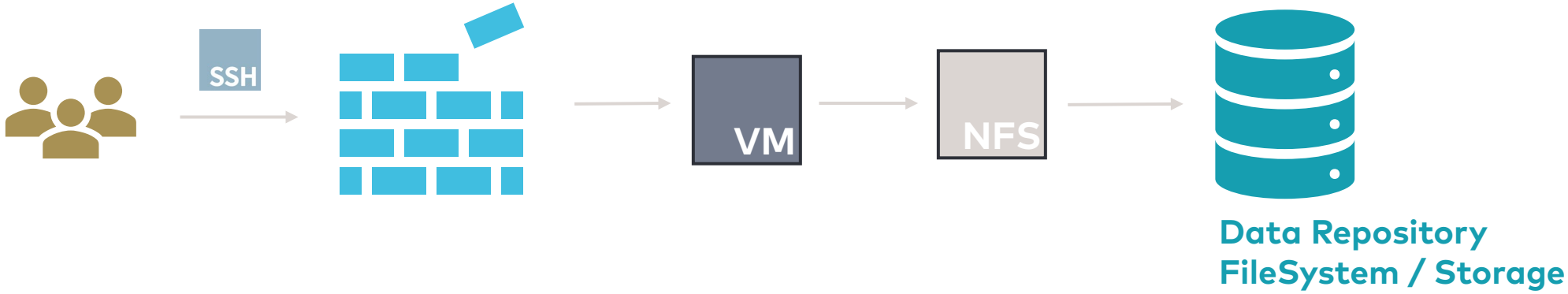


Warum muss es nun STAC sein

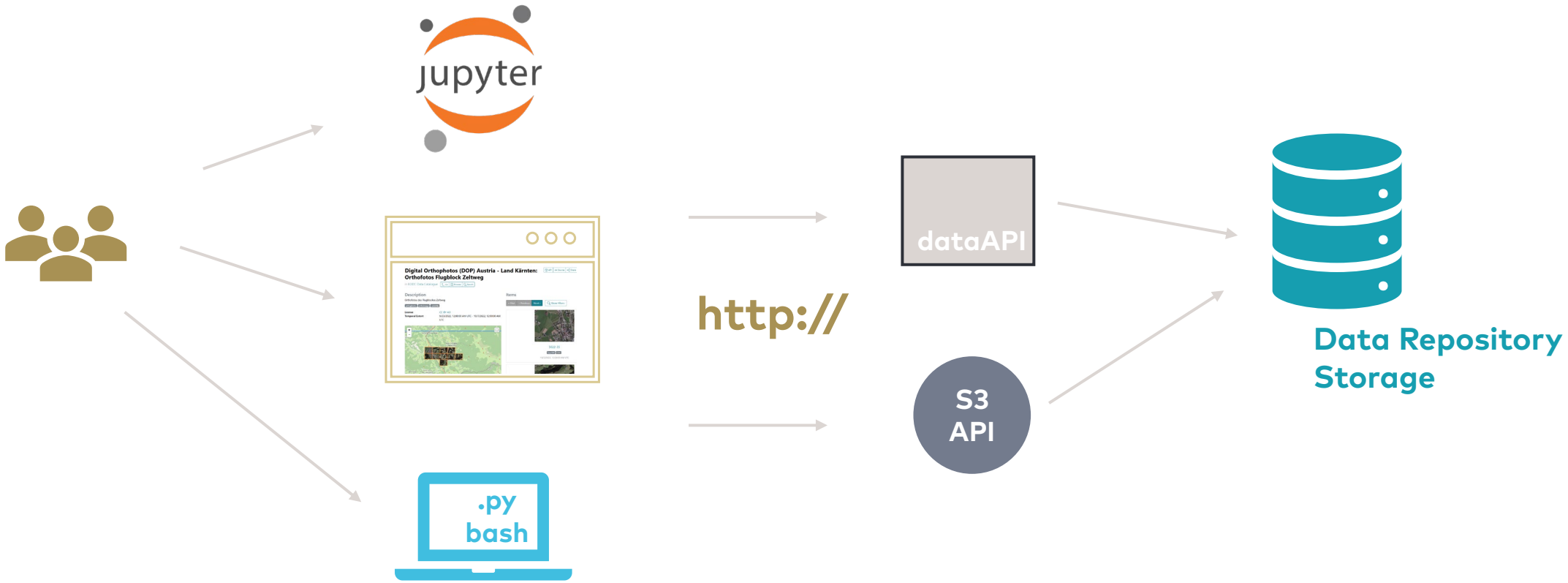
- STAC basiert auf JSON und RESTful APIs, einfacher als OGC CSW (XML)
- STAC (JSON) funktioniert mit allen Web Technologien und wurde speziell für Cloud Anwendungen entwickelt (cloud native)
- STAC API Query Performance, es müssen keine XML analysiert werden
- Große Community and umfangreiches Ecosystem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

Wie erfolgt nun der Datenzugriff

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



Daten APIs –Cloud Native

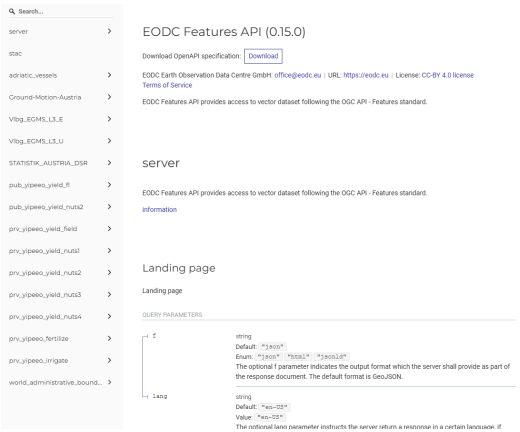


Daten APIs – Cloud Native

- 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

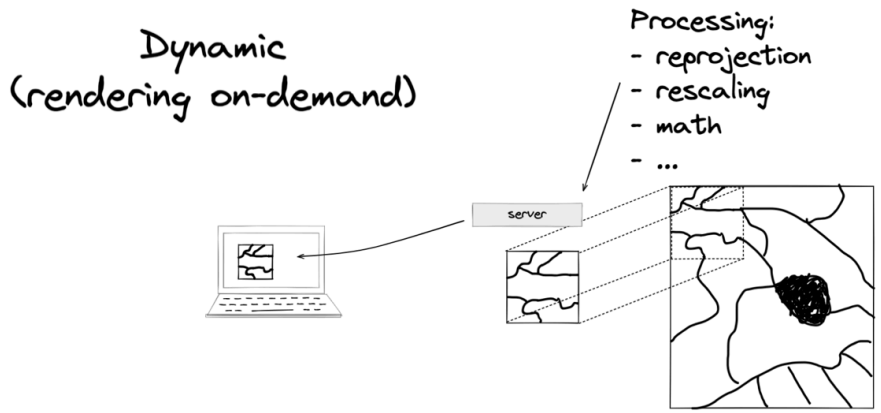
OGC API: Der "neue" OGC Standard

- 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



Visualisierung von Daten

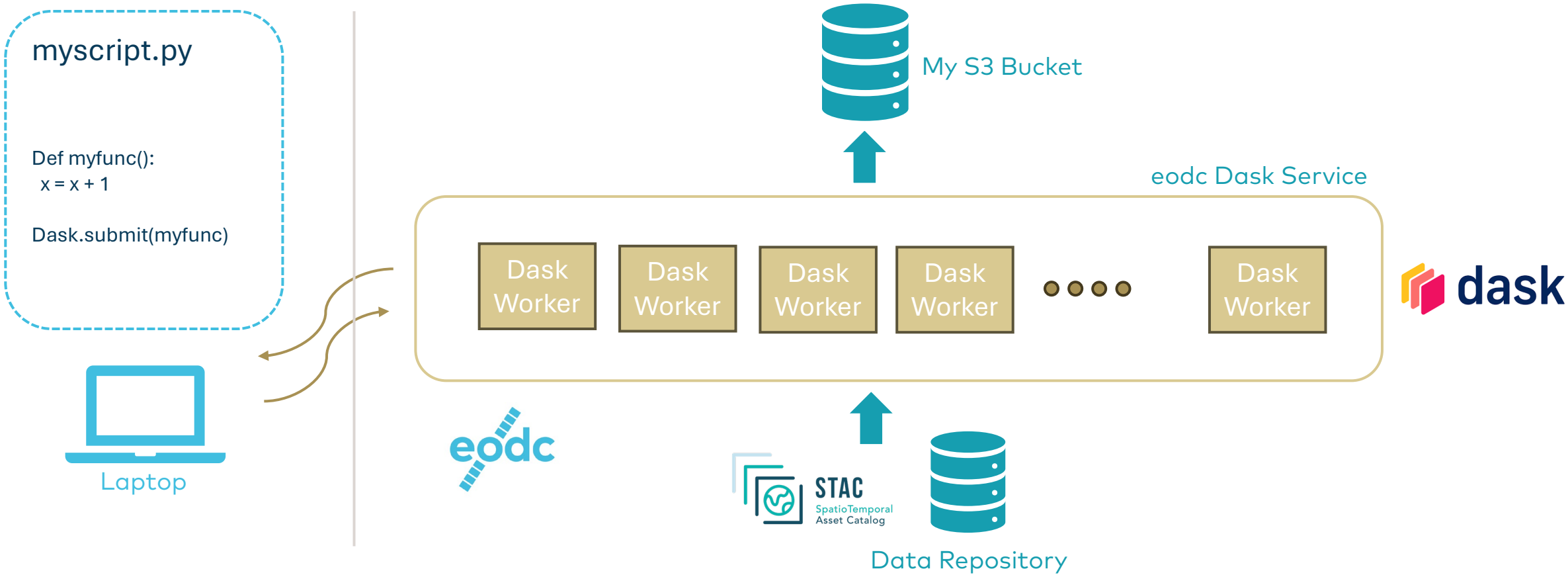
- TiTiler: Dynamisches, “on-the-fly” Rendering von Rasterdaten
- Keine Vorprozessierung notwendig, um Daten zu visualisieren (pre-rendered / static)
- Daten werden via HTTP API ausgeliefert
  - Daten können beliebig skaliert werden
  - Band-Math Operationen möglich
  - Beliebige Colormaps



Datenanalyse in der Cloud

- 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.

Anwendung von Dask

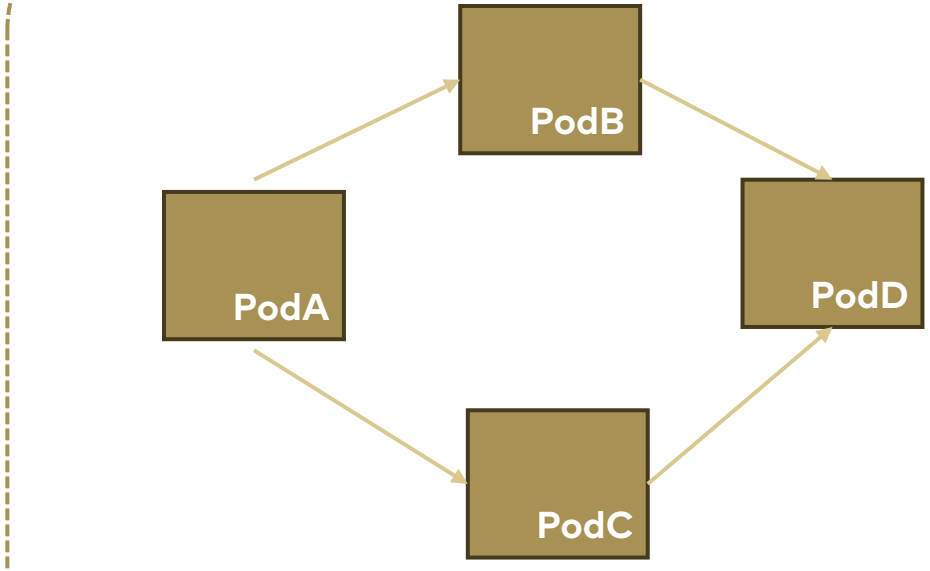


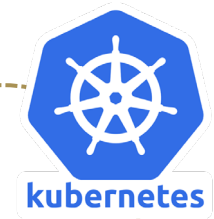
ArgoWorkflows Use Case

myargowf.py

```python
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
```

Laptop





Danke

Details zu den einzelnen Services im Workshop


eodc
Translating data
into knowledge