Protocol for data requirements for modelling and DSS tool

December 2023

Deliverable 3.1 (version 1)

Funded by the European Union

Project name	**Transformation for sustainable nutrient supply and management**
Project acronym	**trans4num**
Project ID	101081847
Project duration	December 2022 – November 2026
Project Coordinator	University of Hohenheim (UHOH)
Project website	https://trans4num.eu/
Work package	3
Work package leader	AU
Deliverable number and title	D.3.1: Protocol for data requirements for modelling and DSS tool
Authors	Morten Birk (Cordulus- former Fieldsense)
Version	1
Dissemination level	PU-Public

Table of Contents

List of figures

1. Introduction

This document describes the protocol for the dataflow used in trans4num task 3.1 'Monitoring of nutrient flows based on (remote) sensing in case regions', in order to ensure effective and useful collection and distribution of remote sensing data between partners, a common understanding of needs and requirements of the remote sensing data for task 3.2 'Develop a Decision Support System (including a nutrient management plan) for optimum nutrient supply at farm, regional and watershed level'. This protocol attempts to clarify what format, quality and quantity of useful data are expected to have as well as what data is to be collected and delivered in the period from month 14-42.

To do so, the data requirement protocol is based on the current knowledge we have about the data sources available, and the learnings we have made from the WP3 UX workshops for the Decision Support System tool (DSS tool). In Task 3.2 during autumn 2023, three end-user workshops were conducted among the partners involved in Work Package 3. The purpose of these workshops was to define the end-users, functionality, spatial, and temporal scales of the DSS, aiming to gather input for compiling specifications for the DSS in 2024. The outcome of these workshops has emphasized that close integration, access, and exchange of models and data between Tasks 3.1 and 3.2 are essential prerequisites for the successful development of the DSS.

With offtake in the Danish Nature Based Solutions (NBS) site, which will function as the site where the DSS prototype will be built, the workshop concluded the end users to be farmers, farmer advisors, regional planners, and the bioindustry within the NBS site. Since these end-users require outputs on varying spatial and temporal scales, the DSS must accurately provide outputs for both product, nutrient flow, and nutrient availability across these diverse scales within the region or sites where the DSS/NBS is to be applied. This implies a necessity for both real-time and historical data. Additionally, modelling of biomass outputs and locations is crucial for the DSS and can be useful for users within the Danish sites and potentially other NBS sites within a specific region.

Five key takeaways from the workshops which affect the data pipeline are the following.

1) The data pipeline must be designed to handle different types of available data sources

- Tabular data: Primarily regarding field specific information like yield and operations.
- Time series tabular data: Another form of tabular data, for instance used for weather data.
- Spatial data: Rasterized geospatial information often used to describe soil properties
- Spatiotemporal data: In the form of a series of vectorized or rasterized geospatial information like mapping of crops or collected satellite data.
- Tabular and spatial data is characterized by being static, while time series and spatiotemporal data has a static history, but often also requires continuous ingestion of new data.

2) Most modelling is performed based on the baseline, here defined as the current state of the model given current inputs, and scenarios based on what will happen if inputs are different. This enables users to understand what effect different actions/scenarios can have. The type of scenario is defined by experts (one example from the Danish site is more grass in rotation or more permanent grass), and is hence a part of a static modelling, which can be done as an offline process, and not at request time. This is important since it enables us to do large computations to pre-calculate most model parameters, such that only a limited amount of work must be done at request time by end users. Since much of the analysis is done on large scale time series data, it is expected that the analysis will be computationally intensive, and hence it is not feasible to do the analysis in real time based on user parameters.

3) It should be noted that the output of our analysis can probably be used for further analysis on the DSS tool, the only requirement from the data protocol here is that the data can be accessed very efficiently, even for batch queries of many fields.

4) It is important that the analysis is updated once more data is available. From our workshop a lot of emphasis has been put on the ability to track updates to ensure optimal practice within the season. Hence, users of the tool should be able to track development in the modelling throughout a season. This means that the data pipeline should be capable of handling updates to the dataset, which should invoke updates to the model, and redistribution of the updated parameters.

5) The DSS should function at landscapes and not only for the individual farmer or field.

A few example files (https://drive.google.com/file/d/1ODhAlsZ7ep8WIYkUyR-a5Qg44TlRW0Vn/view?usp=sharing) are distributed together with this document, to show examples of data in the pipeline.

2. Services

Based on the described properties of how modelling is to be done, and how data is supposed to be consumed, the following pipeline architecture is proposed. The aim is to allow variable scaling of ingestion and analysis, and efficient and reliable distribution of analysis results.

The pipeline is composed of 3 different services illustrated in figure 1. Every blue box illustrates individual services, which are separated by storage layers. In the following each service layer is described.

Ingester

The ingestion service is in charge of ingesting data for the DSS Analysis Engine. This means that it is in charge of processing and providing all necessary data and placing it in our object-based storage service.

For many datasets this is basically about adding new static files, but for real time datasets like satellite imagery and weather observations this also includes a live service which ingest new data once it is available for analysis.

DSS Analysis Engine

The DSS Analysis Engine is in charge of using available data from the object-based storage service, to provide the latest analysis for a given NBS site. This is where all the modelling is happening. This is done as an offline model since it is computationally intensive, and has to be performed on large amounts of data.

The results of the modelling will be saved in formats which can efficiently be distributed by the distributor service.
For tabular data and static vector data this is a NoSQL document database, and for raster data this is a WMTS tile service where rasters are saved as cloud optimized GeoTiffs.

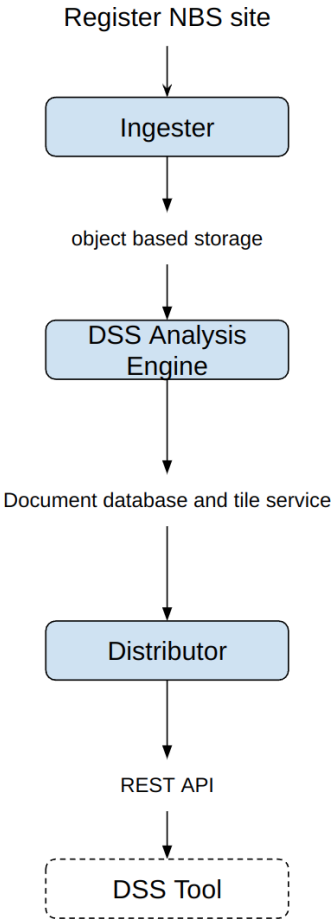


Figure 1: Three different services composing the pipeline

Distributor

The Distributor service is in charge of exposing the result of our analysis to the DSS tool used by end-users. This is done through a REST API which can efficiently access its source data from pre-calculated data in the document database or through the tile service. This means that no further calculations are done in the REST API which simply fetches the data. This allows for efficient lookups on multiple parcels, which has been shown from our workshop to be very important, since the DSS tool will have to give insights based on several parcels in an area.

All services are only connected through their storage layer which enables a loose coupling, where any service can stop without killing the other services.

The reason for this split is that it enables us to work on, and release updates to, either service in isolation. It also means that we can scale the ingester and analysis engine as needed. The Ingester will have a lot of work to do for a short period of time once new data is available (which might mean once every 5 days for satellite data).
When new data is available or if a new modelling tool is introduced, the DSS Analysis Engine will need to do a lot of work before it can basically shut down again. Since we are expecting deep learning within the analysis model, such scaling can be quite important for keeping the costs low since it allows a horizontal scaling of resources once they are needed.
The Distributor needs high availability and efficient access independent of the state of the other two services. Due to the isolation of the services, this service is a thin API which basically reads pre-calculated data. This should make it able to handle large numbers of requests even on very limited resources. From the workshop we have learned that most scenarios will be pre-calculated, allowing for this efficient pipeline, where it is not required to reprocess the analysis based on user inputs in the tool. Hence, the model is not rerun at request time.

3. NBS Sites

All analysis for the DSS tool is done for individual NBS sites in isolation and is thus only based on detailed data from the specific NBS site. This restricts users of the DSS tool to be associated with a specific test site. For instance, their fields can be specific parcels within the NBS site. Our first test site is at the Limfjord catchment in Denmark, however the protocol for data exchange does not make any assumptions about the location of the test site.

As previously mentioned, the user experience workshop with project partners concluded that it is a key requirement that our analysis is based on large scales within an NBS test site, and not only the individual farmer or field. Hence, the data collection, analysis, and serving should be designed to handle an NBS site as a whole.

The main entity of our data pipeline is an NBS site. From a data perspective all data is part of an NBS site. An NBS site contains a collection of Parcels, which are often individual fields, but can basically be any geospatial area of interest within the NBS site.

Hence, all data is ingested, in relation to a specific NBS site. Therefore, one must register an NBS site within the ingester in order for the ingester to start ingestion as illustrated in figure 2 which illustrates the internal workings of the ingester service from figure 1. All ingestion is done for individual NBS sites in isolation; there is no overlap of data between two different NBS sites.

Once an NBS site has been registered in the ingester, it will ingest all static datasets (like soil mapping) related to the site. It will also ingest all previous history of the historical datasets available. Finally, it will register for updates to all temporal datasets. Once new data is available the data will automatically be ingested and uploaded to the object-based storage service.

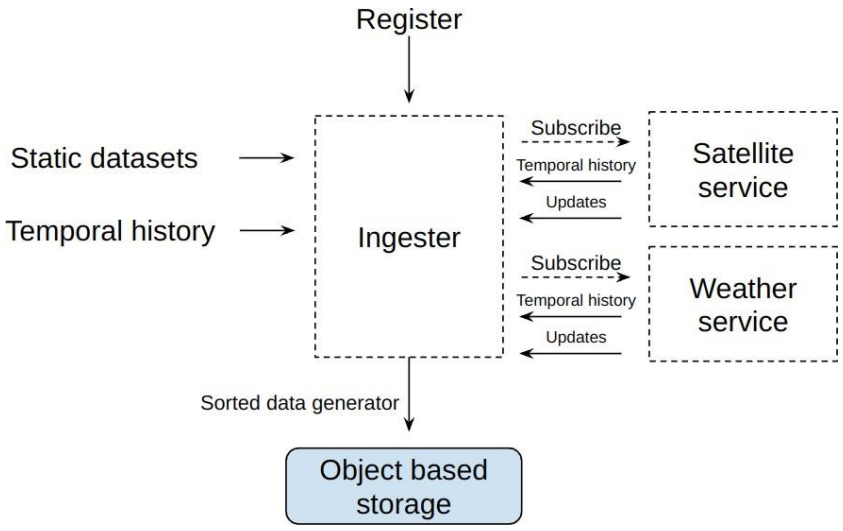


Figure 2: Ingester function

All temporal history data (like satellite or weather data) will be ingested in order. Hence, from the ingesters point of view, data is never “delivered”. It is an infinite generator, which simply

generates a lot of data at spinup, and then slows down to only generate new data, once data is available.

Creating NBS sites

An NBS test site must be registered as a whole in order for processing to be done efficiently, and especially to ensure that we can give fast access to precalculated parameters at request time for users of the DSS tool. Parts of the satellite pipeline require access to all fields within the test site in order to provide optimal data for modelling purposes which further emphasizes the requirement of batch processing of full NBS sites. Hence, it is important that great care is taken to include all geospatial regions of interest in the NBS site at registration.

A new NBS site can be created by posting a region to the /regions endpoint at the Ingester service. It is not possible to create new NBS sites from the DSS Tool since this is not something end users will do. Creation of NBS sites is something which will happen rarely and hence we are the ones submitting new regions.
The body describing the region should be a GeoJSON (RFC 7946) FeatureCollection with the following properties.

- The FeatureCollection must contain the following Foreign Members.
    - regionId: String. A unique name for the region.
- The FeatureCollection may contain the following Foreign Members.
    - since: String. A starting date (inclusive) described as an ISO 8601 string in UTC. If this is provided all temporal data before that timestamp will not be included.
    - until: String. An ending date (exclusive) described as an ISO 8601 string in UTC. If this is provided all temporal data after that timestamp will not be included.
- Every feature must contain the following properties
    - parcelId: String. A unique name for a parcel in the region
- Every feature may contain the following properties
    - satellite: (Default: true) Boolean. If true, satellite data is collected for the parcel.
- If false no satellite data is collected for the parcel.
    - growthClass: (Default: NULL) String. If specified every parcel with common growthClass will be used to generate smoother temporal satellite imagery through interpolation.
    - class: (Default: NULL) String. A class provides a common relation between parcels for overall analysis.
    - categories: (Default: []) String List. A list of strings for common relations beyond the one provided by class.
- The following is an example of a valid description of an NBS site which can be used to register an NBS site.

```
{
  "type": "FeatureCollection",
  "name": "example",
  "regionId": "efdb3833-2fb6-4db9-8c03-d0846bb4f6cb",
  "since": "2021-01-01T00:00:00Z",
  "until": "2023-01-01T00:00:00Z",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "class": "Vårbyg",
        "growthClass": null,
        "categories": ["Vårbyg", "1", "important"],
        "satellite": true,
        "parcelId": "6948c7ee-b"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[
          [ 9.0379613, 56.7515935],
          [ 9.1379613, 56.7515935],
          [ 9.1379613, 57.7515935],
          [ 9.0379613, 57.7515935],
          [ 9.0379613, 56.7515935]
        ]]
        }
      },
    {
      "type": "Feature",
      "properties": {
        "class": "Vinterhvede",
        "growthClass": "growth",
        "categories": [ "Vinterhvede", "2", "important" ],
        "satellite": true,
        "parcelId": "913a862e-8"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[
          [ 10.0379613, 56.7515935],
          [ 10.1379613, 56.7515935],
          [ 10.1379613, 57.7515935],
          [ 10.0379613, 57.7515935],
          [ 10.0379613, 56.7515935]
        ]]
      }
    }
  ]
}
```

4. Input format for DSS Analysis Engine

The ingester is in charge of ingesting data to the analysis engine which is efficient to consume. The analysis engine must handle the infinite generator from the ingester, which distributes data in a sorted fashion. Hence, it can expect that any data source is delivered sorted in time. This means that updates should hopefully not require a full recalculation of all analysis results.

The analysis engine is based on input from the object-based storage which contains the data ingested by the ingester. It has a limited set of file formats. It is expected that the object-based storage service used is AWS S3.
Input for the DSS Analysis Engine can be provided in the following formats through the object-based storage.

Tabular data

- CSV
    - For files describing the entire NBS site (or at least several parcels), rows must contain a **parcelId** column, used to identify which parcel it is associated with.
    - For files describing a single parcel, they must be named by **parcelId**.
- JSON
    - For files describing the entire NBS site (or at least several parcels), the object must be a dict, with **parcelId** as key, which should map to all data regarding that parcel.
    - For files describing a single parcel, they must be named by **parcelId**.

Raster data

- GeoTiff
    - Must contain a spatial ref and geotransform for spatial alignment.
    - If data is temporal, every entry is represented by a single file. The files must be named as an ISO 8601 string in UTC.

Vector data

- GeoJSON (RFC 7946) or Shape (Can be zipped)
    - If data is temporal two formats can be used.
      1) Every entry is represented by a single file. The files must be named as an ISO 8601 string in UTC.
      2) A single file is provided, but temporal data is available at feature attributes named by an ISO 8601 string in UTC.

All **Tabular data** will be associated with parcels based on their **parcelId**. All **Raster data** and **Vector data** is associated to specific parcels based on their overlap with a given parcel's geospatial extent.

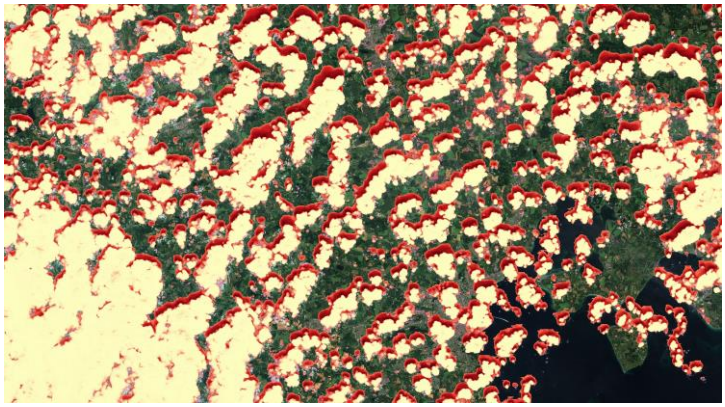
5. Collection of MultiSpectral Satellite Data

The primary spatiotemporal data source is the satellite imagery. This is the data source identified to probably have the biggest impact on temporal recalculations of the analysis model. Hence, we will go into further details with this data source in the following.

Multispectral satellite data is collected from the Sentinel-2 satellite constellation, which provides global coverage with a revisit time of 5 days at the equator (for the combined constellation). It provides access to 12 bands at varying resolution between 10 and 60 meters since November 2016.

Sentinel-2 bands	Resolution (m)
Band 1 – Coastal aerosol	60
Band 2 – Blue	10
Band 3 – Green	10
Band 4 – Red	10
Band 5 – Vegetation red edge	20
Band 6 – Vegetation red edge	20
Band 7 – Vegetation red edge	20
Band 8 – NIR	10
Band 8A – Narrow NIR	20
Band 9 – Water vapour	60
Band 10 – SWIR – Cirrus	60
Band 11 – SWIR	20
Band 12 – SWIR	20

Due to access to visible rgb bands, near infrared and infrared, common agricultural indices like NDVI (using band 4 and 8) and NDRE (using band 5 and 8) can be extracted in 10- and 20-meters resolution respectively. Further agricultural indices can of course also be extracted.
Since the multispectral bands are not capable of seeing through clouds, it cannot be expected to extract usable images every week. All pixels made unusable by clouds or cloud shadows are automatically removed in our ingester pipeline by best effort through spatial and temporal analysis. However, cloud removal is not perfect, and following analysis, should take possible noise from clouds into account.
As shown on the following image all pixels marked as yellow are cloudy, and all pixels marked as read are affected by cloud shadows, those pixels cannot be used for analysis.



Further, since the imagery is collected with a swath of 290km it is very likely that part of an NBS site is ingested at one date, while the rest is ingested at another date.

The removal of cloudy imagery and the fact that different parts of an NBS site might be ingested on different dates means that satellite imagery will be available at different dates and sparsity for individual fields within the NBS site. Our pipeline takes some steps to mitigate this effect.
However, this should be taken into account in any following analysis.

In order to handle temporal analysis of all imagery all data is ingested as L2A atmospherically corrected surface reflectance images. This conversion is done with the ESA Sen2Cor processor.

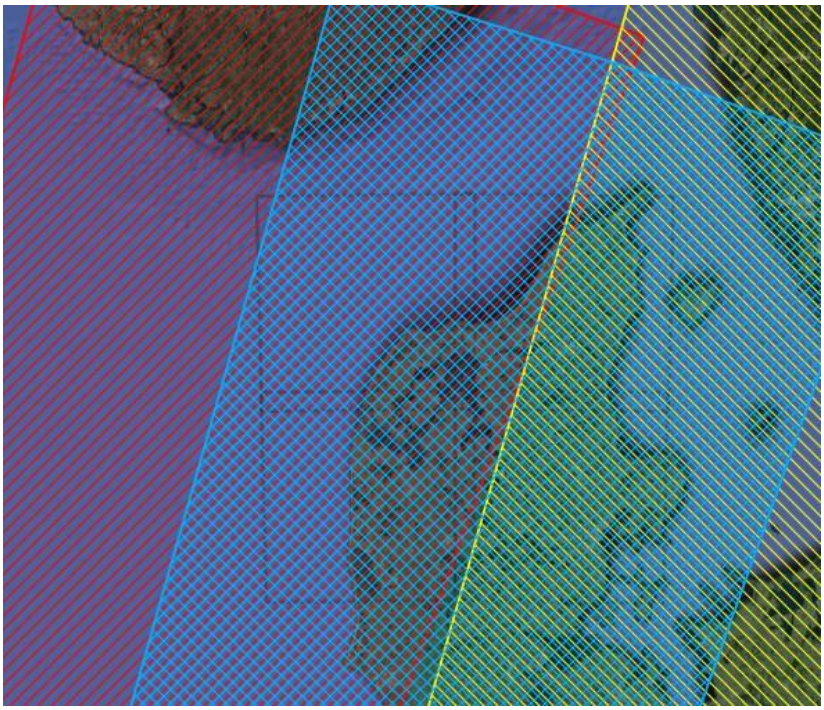
Collection of data for an NBS site

Sentinel-2 multispectral data products are distributed in tiles specified by a fixed grid of 100x100km resolution. Hence, a single NBS site might need to ingest data from several tiles.

This image illustrates how 4 different tiles cover the Danish NBS site. Hence, we must collect data from all 4 tiles in order to cover the test site.

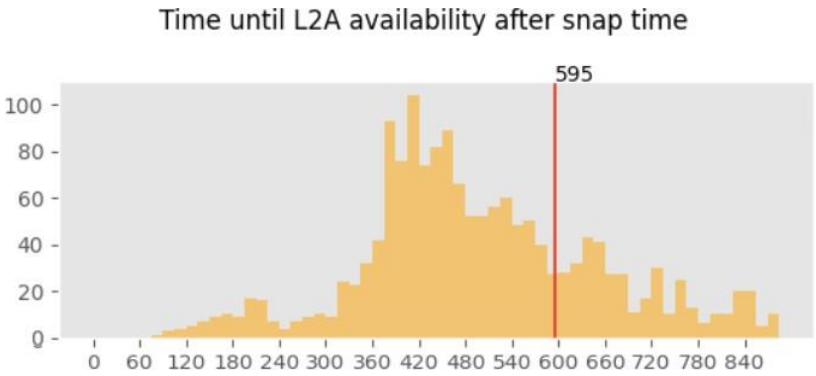
The tiles do not directly correspond with the swath (the area imaged on the surface) of the satellite.
This following image illustrates how data is collected in different areas for different dates. The

red, yellow and blue areas are collected at different times. In this example the blue swath will cover the entire NBS site, while the red and yellow swaths will result in different parts of the NBS site being collected at different dates.



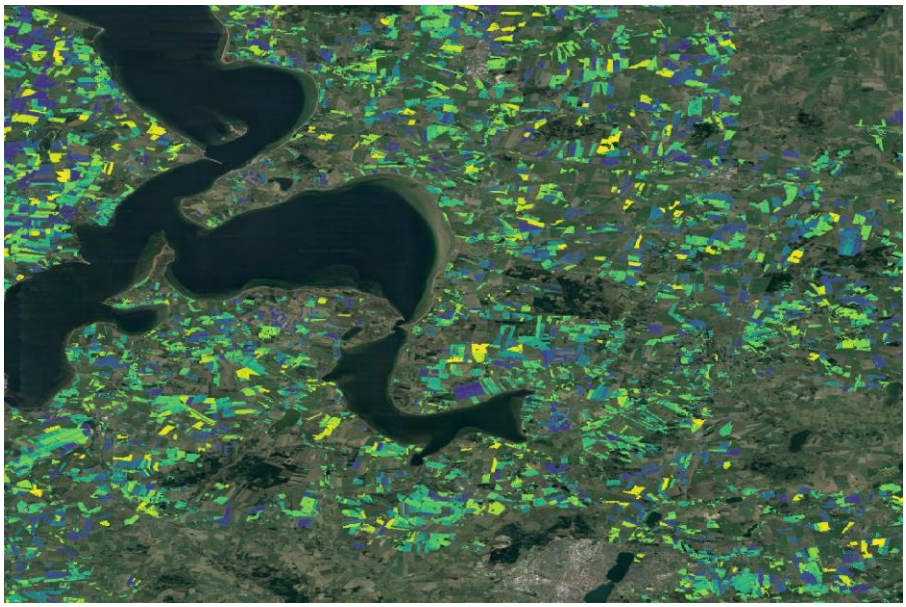
We are collecting data for all data points that overlap with a registered NBS site.

There is a delay from snap time (the time where the image is taken) until it is ingested and ready for analysis in L2A format. Through our tile ingestion pipeline, we have estimated the following delay from sampling over several tiles within the last two years, to estimate the general delay in minutes. From this we see that we generally are available to start our analysis around 10 hours after the image is taken.



However, since the delay can be different between different tiles, and we have to wait for all tiles with the same swath data to be available, we must postpone our analysis until all data is available. In practice this will mean that we should expect a delay of up to 24 hours before ingested satellite data is released for analysis. This must be taken into account for following analysis, and for users of the DSS tool.

After ingestion of a product across all tiles covering a test site, the tiles are merged, and indices for analysis are extracted, before a field mask is applied, to generate a product for the entire test site.



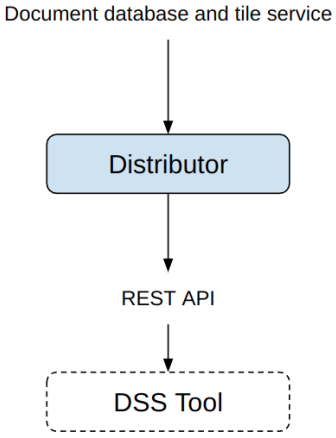
Finally, we also create a dataset where we try to extrapolate values for missing data. This enables analysis to simulate that all data is available on the same dates, however it can only be done if a growthClass is specified for the NBS site.
For every growthClass in the NBS site, an extrapolation of data is done to fill out holes for missing data.

These final products are made available for the analysis engine in GeoTiff format using ZSTD compression.

6. Distribution of data to end users

All large-scale heavy compute analysis is done in the analysis engine, and outputs are stored as tabular data in a highly scalable document database, while rasterized data is stored in a WMTS tile provider. This means that the DSS tool can access data very efficiently since it is only in charge of aggregating reads, and distributing it to clients. The distribution is done through a REST API which is an architectural style for an API that uses HTTP requests to access and use data.

From our workshop we have learned that it is important that users can efficiently extract analyzed results from several parcels at several farms, hence it is important that the REST API can handle efficient lookups for several parcels. The REST API is protected through a token which must be provided in the auth header. While the token provides access to



the API on a service level, it is the job of the DSS tool to handle any user specific authentication and authorization.

We expect to expose the following endpoints:

GET /regions

This endpoint is used to fetch the id of all registered NBS sites (regions).

Response:

{

regions: regionId[] // A list of ids for all regions

}

GET /regions/{regionId}/parcels

This endpoint is used to fetch the id of all parcels within a given region.

Path parameters:

regionId: The regionId of the region that the parcel is associated with.

Response:

{

parcels: parcelId[] // A list of ids for all parcels within the region

}

GET /regions/{regionId}/parcels/{parcelId}

This endpoint is used to fetch the info for a specific parcel within a region.

Path parameters:

regionId: The regionId of the region that the parcel is associated with. parcelId: The parcelId of the parcel.

Response:

{

feature: {}, // The feature object used while submitting the parcel in the NBS site. models: {} // A key value dictionary with modelId->modelData

}

POST /regions/{regionId}/parcels/query

This endpoint is used to batch fetch the feature info for several parcels within a region.

Path parameters:

regionId: The regionId of the region that the parcel is associated with.

Request:

{

parcels: parcelId[] // A list of ids for all parcels for which we need information.

}

Response:

{

features: {} [], // The feature object used while submitting the parcel in the NBS site.

}

GET /regions/{regionId}/models

This endpoint is used to fetch ids for all models available for a region.

Path parameters:

regionId: The regionId of the region that the parcel is associated with.

Response:

{

models: modelId[] // A list of ids for all models within the region

}

GET /regions/{regionId}/models/{modelId}

This endpoint is used to fetch region specific data for a model.

Path parameters:

regionId: The regionId of the region that the parcel is associated with.

Response:

{

model: {

modelId: str, // The identifier for the model

tilingName: str or null, // If specified provides the tilingName used to create WMTS tile requests, meta: {}, // Metadata object describing the model.

region: modelData // The models data for the region

}[] // A list of models descriptions

}

POST /regions/{regionId}/models/{modelId}/query

This endpoint is used to batch fetch the parcel specific model output for several parcels within a region.

Path parameters:

regionId: The regionId of the region that the parcel is associated with. modelId: The modelId of the model for which data should be extracted.

Request:

{

parcels: parcelId[] // A list of ids for all parcels for which we need model data.

}

Response:

{

models: modelData{} // A key value dict with parcelId -> modelData. Null if mising.

}

GET /region/{regionId}/tiles/

This endpoint is used to fetch tilingNames for all tile maps available for a region.

Path parameters:

regionId: The regionId of the region that the parcel is associated with.

Response:

{

tilingNames: tilingName[] // A list of tilingName strings.

}

GET /region/{regionId}/tiles/{tilingName}/[tilequery]

This endpoint is used to fetch WMTS tiles.

Path parameters:

regionId: The regionId of the region that the parcel is associated with. tilingName: The name of the WMTS tiles we are looking to fetch.

Response:

Return tile images based on the tileQuery.