Training documentation for Geoscience Australia in relation to the Digital Atlas Project

1. Modelling, Ontologies, Dataset and Feature Collection definitions

1.1. Ontologies and associated modelling

Modelling related to FSDF is in the GA Supermodel, as documented here.

1.2. Feature collection and Dataset definitions

Feature collection and dataset definitions are available in the <DA-ETL-Processing repo>

2. ETL processing

2.1. RDF generation

Python based row processing utilising python pandas, SQLAlchemy to connect to postgres. Python's pandas library provides a good wrapper for ingestion of data from a range of relational sources, including Postgres databases as currently used at GA. It is a widely used well supported library. Once read in to a pandas Dataframe (an in memory table), the data may be iterated over row by row with any data manipulation performed using Python functions; for example built in string functions, and geospatial conversion functions through third party libraries such as Shapely. This provides a high degree of flexibility for any necessary data manipulation, including mapping of codes, logic, spelling corrections etc. which can be difficult or expensive operations with other methods. The Python code is readable for new users, and can easily be updated to suit any new requirements, or fix bugs.

RDFLib is utililsed to create valid RDF from each row of data, this ensures no syntax errors are introduced into the RDF graph. The data is serialised using RDFLib, to a set of RDF files. The exception to this is the GeoFabric and GNAF datasets, for which record by record processing is done from database dumps. This is due to the size of these datasets, which could not be loaded in to memory on most desktop computers.

The ETL scripts are collected in the <digital atlas etl repo>

2.1.1. Local development, testing

To make updates to the ETL scripts, the python scripts in `app/processing/{dataset}_block_convert_to_rdf.py` within the Digital Atlas ETL repo should be updated. The changes can then be tested locally by running the python code directly (option 1 below), and inspecting the output. Local integration testing can be performed using Docker Compose, both for the ETL process, and to confirm the output displays correctly in Prez.

Options:

1. Run python script directly, either through the terminal, or an IDE run configuration.
2. Docker container - allows testing of the processing code only
3. Docker Compose - allows testing of the processing code, loading it to Jena Fuseki, and displaying it through Prez. This facilitates rapid iteration over the processing code, allowing issues to be identified and resolved early in the development process. *Detailed instructions for these methods of running the ETL process and testing are described in the repository's Readme*.

2.1.2. Cloud based processing

The ETL Process utilises ECS and EFS when deployed in AWS via Terraform, see AWS infrastructure and Infrastructure management below. The Terraform script creates a Task Definition for each dataset to be processed, which can be run manually through the AWS console by navigating to "ECS", and finding the relevant task under "Task Definitions". The tasks are configured to write to the S3 Bucket specified using the "OUTPUT_LOCATION" environment variable. This environment variable is currently set in the task definition JSON template (used for all processing tasks), to `s3://digital-atlas-rdf`. The ECS task has permission to write to this bucket through a statement attached to the `ecs_task_definition_role_policy_document` data definition in `data.tf`. The processing code will prepend the `DATASET` environment variable to the key of each object stored in this bucket, effectively creating a 'folder' structure when viewed through the AWS Console. This structure also allows easy pattern matching when attempting to read specific datasets from S3 to process in to TDB2 datasets, as detailed below.

2.2. TDB2 generation and spatial indexing

TDB2 is the second generation on-disk database format utilised by Apache Jena. It provides high scalability, indexation, and works in conjunction with extensions including GeoSPARQL and Full Text Search. TDB2 datasets can be created directly, by utilising the `tdb2.tdbloader` command line utility packaged with Apache Jena; through the Fuseki UI; or through Assembler files, which are parsed on Jena startup.

2.2.1. Local development, testing

To test "end to end" ETL processing, in a similar manner to as it would occur on AWS, it is recommended to utilise the Docker Compose scripts, as the outputs of one container process can be reused in another. This allows the outputs of the RDF generation steps to be loaded (via a common volume) to the TDB2 generation scripts in a dependent docker container. NB the ETL compose file utilise a `Jena` image, which includes the `tdb2.tdbloader` command line utility, but does not include the Fuseki UI.

Should you wish to test RDF that has already been generated, or is from some source other than the

RDF generation scripts, you can directly run the `Jena Fuseki` container image and create a TDB2 dataset through the Fuseki UI, by going to "manage", then "new dataset". NB This TDB2 dataset will not persist beyond the life of the docker container, unless an external mount or volume has been configured, as docker containers are ephemeral.

The Jena spatial index is created through a jar file, repackaged from Apache Jena by Zazuko. This has been added to a multi-stage docker container by Surround, so as to include TDB2 generation along with spatial indexation in one image. This was done as for spatial datasets, the two processes both need to occur, and the output of the TDB2 generation (a TDB2 dataset) is the input of the spatial indexer.

In addition to the TDB2 generation and spatial indexation, a prior step is to validate the input data is valid RDF. For clarity, this validation is to determine whether the files are valid RDF, not whether they conform to any profiles (e.g. the OGC Linked Data API profile), meaning the validation will only provide a guarantee that a TDB2 dataset can be created from them, not that the data itself conforms to any useful model. The validation is completed utilising the Apache Jena RIOT command line utility. Surround has written a short bash script which calls the validator on all nquads files which are to be loaded, and should any of them contain any invalid RDF, prints out the problematic lines and errors, and renames the files with an extension of `.error`. This renaming prevents `tdb2.tdbloader` attempting to load them to a TDB2 dataset. Successfully validated files will also write a short message confirming their valid format. Should a file fail validation, you should try to identify the problematic code which generated invalid RDF, fix this, and the invalid RDF files (or parts of them, and apply a patch). The TDB2 dataset can either be appended to by re-running processing on just the problematic files, or the whole TDB2 dataset can be regenerated if this is easier.

2.2.2. Cloud based processing

Task definitions have been created to allow running the TDB2 generation on AWS. This is useful (and arguably necessary) for larger datasets. A task has been created for each TDB2 dataset, which reads data from the relevant RDF Datasets (as RDF files in the output RDF bucket), and writes the output TDB2 dataset to a mounted EFS volume. These are:

TDB2 Dataset	RDF Datasets (dcat:Dataset)
cache	power_substations, power_stations, transmission_lines, facilities, placenames, gnaf, asgs, geofabric
fsdf	power_substations, power_stations, transmission_lines, facilities, placenames
gnaf	gnaf
asgs	asgs
geofabric	geofabric

To run the TDB2 generation task on AWS, the task must specify and EFS volume that can be mounted. Either an existing TDB2 dataset can be used, or a new one must be created. These scenarios are described in detail below:

1. There is an existing TDB2 dataset on an existing EFS volume which you would like to add

additional data to

   - Ensure any Jena Fuseki instances currently using the volume are stopped. You can do this by setting the "Desired tasks" count to zero on the relevant "Service" in the `ld-digitalatlas-nonprod` cluster on AWS.
   - Remove the network mount points for the EFS volume. Go to the EFS homepage in AWS, select the relevant EFS volume (as it already exists, it should be mapped to the relevant TDB2 dataset in the API Terraform) navigate to network, click edit, and remove the mount points.

2. A new TDB2 dataset is required, requiring an EFS volume to be created

   - Go in to the AWS console (or through the CLI or otherwise) and create an EFS volume in the appropriate region (ap-southeast-2)
   - AWS will automatically generate network mount points, delete these in the console so terraform can create its own network mounts in the right subnets for the task to use. To do this, navigate to network, click edit, and remove the mount points.

Specify the EFS volume in the da-etl-terraform repository's `terraform.tfvars` file, under `efs_id`. This will set the task up to utilise the appropriate volume when deployed. Run Terraform apply, and wait for the changes to propagate.

Log in to the AWS console, navigate to EFS, select the relevant EFS volume, and make a note of the Subnet IDs the EFS mount targets are in (under the Network tab).

The task can now be run and will mount the correct volume, as specified in Terraform. To run the task, navigate to ECS in the AWS console, click on "Task Definitions", find the relevant task (they are all prepended with "tdb2_generation_*"), select "Deploy" and then "Run task". Select the relevant ECS Cluster (currently figured as `da-etl-nonprod`), and select only the relevant subnets that you made a note of above. You can now deploy the task.

To view the status of the task, click on the task ID, and then click on "logs". It can take a minute for the task to be registered/deployed. Once the task has finished, logs will not be available for viewing under the ECS pages, however logs have been configured to be sent to cloudwatch logs, so they can be viewed here until the retention period ends, currently set to 1 day.

2.3. Display of linked data through a web interface

2.3.1. Web application

- Surround proprietary code with a license
- fastapi based
- direct SPARQL queries to triplestore
- utilises labels from ontologies, dataset and feature collection definitions and instance data to create the human readable displays in HTML
- is available on Github and integrated with Docker Hub to build / push new images on updates. Bugs can be reported in Github issues.

2.3.2. Triplestore

- The triplestore used is a combination of three open source Apache Jena related technologies:
    1. Jena (Java triplestore)
    2. TDB2 (Persistent store for Jena)
    3. Fuseki (Webserver providing UI and SPARQL endpoints)
- Public docker images for Jena and Jena with Fuseki have been created by a Jena user, Stain, and are available on Docker Hub
- The Jena Fuseki image includes both these components (and the ability to work with TDB2)
- The Jena image includes a set of TDB2 command line utilities, which can be used to load RDF data to TDB2, and then query/update/delete directly in TDB2. This is the preferred approach for creating large datasets, or performing updates across large numbers of triples.

2.4. Source code management

Bitbucket for private code repositories Github for public projects A small (2MB) container image based on BusyBox to create Jena configs

2.5. Continuous integration and continuous deployment

- Bitbucket pipelines for application packaging as docker images, and pushing to container registries.
- Updates to application processing and API code will be automatically built in to new images and **available** for deployment, however manual deployment is required, in order to facilitate User Acceptance testing prior to deployment. Learning Resources: bitbucket-piplines.yml file at https://bitbucket.org/geoscienceaustralia/digital-atlas-etl/addon/pipelines/home

2.6. AWS infrastructure

ECS services and ECS scheduled tasks are used to run the docker containers.

- Elastic Container Service (ECS) - Runs the docker containers as services (for Prez and Jena) or as one off jobs (ETL processing).
- Elastic File System (EFS) - provides persistent storage for the docker containers running on ECS.
- Elastic Load Balancer (ELB) - provides load balancing for the ECS services.
- Elastic Container Registry (ECR) - provides a registry for the docker images used by the ECS services.
- Eventbridge - listens for events from xxx to / periodically, triggers the ECS tasks to run.
- Simple Storage Service (S3) - Data backup. Learning resources:
- AWS documentation

2.7. Infrastructure management

Terraform was used as this is the GA infrastructure management tool. This contains: - AWS infrastructure (ECS, EFS, ELB) - Definitions of the docker images used by the ECS services - AWS security groups and rules

Conversion of data from RDF to the