DataONE Operations Documentation

Release 2.0.1

DataONE

Jul 01, 2020

Contents

1 Contents **3**
- 1.1 Project Resources . . . 3
- 1.2 Member Nodes Resources . . . 23
- 1.3 Coordinating Nodes . . . 61
- 1.4 Development . . . 77
- 1.5 DataONE Environments . . . 81
- 1.6 Glossary . . . 82

2 Indices and Tables **87**

Index **89**

Operations documentation for the DataONE project and deployed infrastructure.

CHAPTER 1

Contents

1.1 Project Resources

1.1.1 Communications

Real Time

DataONE uses Slack for messaging between project participants. Anyone may join the DataONE Slack organization by requesting an invite at: https://slack.dataone.org/

Video Conferencing

DatONE uses Zoom for video conferencing within the project and for webinars.

Project participants needing Zoom for project purposes should contact *support @ dataone.org*.

Discussion Lists

DataONE uses Google Groups for email list management. The DataONE.org Google Groups are managed as a *dataone.org* subdomain of *nceas.ucsb.edu* and includes the following lists:

Address	Description
administrator@dataone.org	For contacting administrators of the DataONE infrastructure.
community@dataone.org	A voluntary participation mailing list for disseminating information to the community with interest in DataONE.
coredev@dataone.org	Internal communications for the DataONE core development team.
mnforum@dataone.org	Member Node forum, for discussion related to Member Nodes and disseminating information to DataONE Member Node operators.
developers@dataone.org	General development related discussion.
DUGcommittee@dataone.org	Private list for the DataONE User Group Committee members.
DUGmembers@dataone.org	Open list for the DataONE Users Group.
leaders@dataone.org	Private list for members of the DataONE Leadership Team.
team@dataone.org	Active project participants.
mn-coordinators@dataone.org	Private list for the Member Node coordination team.
operations@dataone.org	List for notification to stakeholders of the DataONE infrastructure. Any outages, maintenance, or other perturbations to normal operations should be posted here.
support@dataone.org	For requesting support on any aspect of the DataONE project.
sysadmin@dataone.org	Project participants with systems administration responsibilities.

1.1.2 Documents

DataONE Project Documents are stored in various locations.

Google Drive https://drive.google.com/	The primary location for project documents in Phase II of the project.
Etherpad https://epad.dataone.org	Open access collaborative notepad.
Hackmd https://hpad.dataone.org	More advanced collaborative editor that uses markdown syntax and authentication through GitHub.
Subversion Repository https://repository.dataone.org	A subversion repository that contains technical documentation and source code.
GitHub https://github.com/DataONEorg	A collection of Git repositories hosted by GitHub contains some documentation sources. This resource is limited by the size of content that may be stored.
Redmine https://redmine.dataone.org/	The redmine issue tracker can be used to store documents as attachements, or to add simple documents using the wiki (Markdown syntax).
docs.dataone.org (DEPRECATED) https://docs.dataone.org/	Setup with Phase I of the DataONE project. This is a plone site that is maintenance challenged. New content should not be added here, and where appropriate, existing content should be migrated to another location.

Google Drive Notes

DataONE uses the public, generally available version of docs, and as a consequence there are some inconveniences to be aware of.

Duplicate files appear like some sort of duality paradox.

Managing File Permissions

1.1.3 Issue Tracking

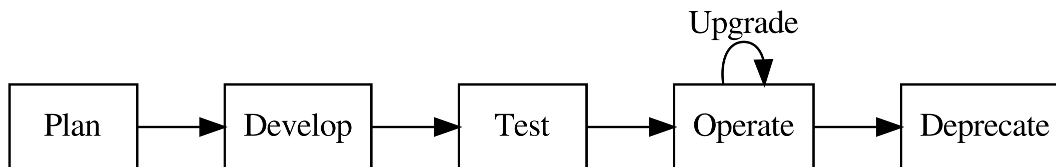
Redmine.dataone.org is the primary issue tracker used by the DatONE project. Some other components used by DataONE are maintained elsewhere and use different issue trackers, including:

Product	Tracker
Infrastructure	redmine.dataone.org/projects/d1
Member Nodes	redmine.dataone.org/projects/mns
www.dataone.org	redmine.dataone.org/projects/d1ops
Metacat	projects.ecoinformatics.org/ecoinfo/projects/metacat-5.
MetacatUI	github.com/NCEAS/metacatui

Member Node Issue Tracking

Member Node (MN) related issues are tracked in the Member Nodes project in Redmine. Each MN is assigned to a *MNDeployment* issue type and tasks associated with the MN are attached to the MNDeployment

Member Nodes transition through several stages during their lifetime (Figure 1).



Redmine.dataone.org

Redmine is currently (2018-01-02) setup on an Ubuntu 16.04 server running at UNM. The installation uses the redmine distribution available from the standard Ubuntu `apt` repositories. Redmine is using a Postgresql database, converted from the previous MySQL installation using pgloader. Details of the installation can be found at redmine.dataone.org/admin/info which provides:

```
Environment:
  Redmine version                  3.2.1.stable
  Ruby version                     2.3.1-p112 (2016-04-26) [x86_64-linux-gnu]
  Rails version                    4.2.6
  Environment                      production
  Database adapter                 PostgreSQL
SCM:
  Subversion                       1.9.3
  Git                              2.7.4
  Filesystem
Redmine plugins:
  clipboard_image_paste            1.12
  plantuml                         0.5.1
  redmine_checklists               3.1.10
  redmine_custom_css               0.1.6
```

(continues on next page)

(continued from previous page)

```
  redmine_wiki_extensions          0.7.0
  redmine_wiki_lists               0.0.7
  scrum                            0.18.0
```

Scripting Redmine

REST API reference: http://www.redmine.org/projects/redmine/wiki/Rest_api

User API key: https://redmine.dataone.org/my/account

Given:

```
KEY="my-api-key-from-redmine"
URL="https://redmine.dataone.org"
```

List project names and their IDs:

```
curl -s "${URL}/projects.xml?key=${KEY}&limit=100" | \
xml sel -t -m "//project" -v "id" -o ": " -v "name" -n

43: DUG
12: Infrastructure
18: Animations
34: Java Client
37: Log Reporting
40: MN Dashboard
...
```

List issue trackers:

```
curl -s "${URL}/trackers.xml?key=${KEY}" | \
xml sel -t -m "//tracker" -v "id" -o ": " -v "name" -n

4: Story
5: Task
1: Bug
2: Feature
...
```

List issue statuses:

```
curl -s "${URL}/issue_statuses.xml?key=${KEY}" | \
xml sel -t -m "//issue_status" -v "id" -o ": " -v "name" -n

1: New
12: Planning
13: Ready
2: In Progress
...
```

List custom fields:

```
curl -s "${URL}/custom_fields.xml?key=${KEY}" | \
xml sel -t -m "//custom_field" -v "id" -o ": " -v "name" -n
```

(continues on next page)

(continued from previous page)

```
7: Estimatedhours
10: Impact
14: Remaining time
15: Risk cat
16: Risk prob
...
```

List issues of *tracker id* = 9, in *project id* = 20, with *status id* = 9 status (MNDeployment tickets in Member Nodes project that are operational):

```
curl -s "${URL}/issues.xml?key=${KEY}&limit=100&project_id=20&status_id=9&tracker_id=9
↪" | \
xml sel -t -m "//issue" -v "id" -o ": " -v "custom_fields/custom_field[@name='MN URL']
↪" -n

7969: http://www.uvm.edu/vmc
7956: http://environmentaldatainitiative.org/
7842: https://researchworkspace.com/intro/
7629: https://arcticdata.io/
...
```

Upgrade Notes, redmine 2.6 -> 3.2

Note: These notes are not relevant to general use of redmine, but are kept here for future reference.

The old version of redmine, running on Ubuntu 14.04 with MySQL:

```
Environment:
  Redmine version                 2.6.1.stable
  Ruby version                    2.0.0-p598 (2014-11-13) [x86_64-linux]
  Rails version                   3.2.21
  Environment                     production
  Database adapter                Mysql2
SCM:
  Subversion                      1.8.8
  Git                             1.9.1
  Filesystem
Redmine plugins:
  redmine_checklists              3.1.5
  redmine_questions               0.0.7
  redmine_wiki_extensions         0.6.5
  redmine_wiki_lists              0.0.3
```

On Ubuntu 16.04, latest maintained redmine is:

```
$apt-cache showpkg redmine
Package: redmine
Versions:
3.2.1-2 (/var/lib/apt/lists/us.archive.ubuntu.com_ubuntu_dists_xenial_universe_binary-
↪amd64_Packages) (/var/lib/apt/lists/us.archive.ubuntu.com_ubuntu_dists_xenial_
↪universe_binary-i386_Packages)
 Description Language:
                 File: /var/lib/apt/lists/us.archive.ubuntu.com_ubuntu_dists_xenial_
↪universe_binary-amd64_Packages
```

(continues on next page)

(continued from previous page)

```
                  MD5: 3a216a1439e1b07aad3aecd0c613d53b
 Description Language: en
                 File: /var/lib/apt/lists/us.archive.ubuntu.com_ubuntu_dists_xenial_
↪universe_i18n_Translation-en
                  MD5: 3a216a1439e1b07aad3aecd0c613d53b


Reverse Depends:
  redmine-plugin-custom-css,redmine 2.3.1~
  redmine-sqlite,redmine 3.2.1-2
  redmine-plugin-recaptcha,redmine 2.0.0
  redmine-plugin-pretend,redmine
  redmine-plugin-pretend,redmine 2.3.1~
  redmine-plugin-local-avatars,redmine
  redmine-plugin-local-avatars,redmine 2.3.1~
  redmine-plugin-custom-css,redmine
  redmine-mysql,redmine 3.2.1-2
  redmine-pgsql,redmine 3.2.1-2
Dependencies:
3.2.1-2 - debconf (0 (null)) dbconfig-common (0 (null)) redmine-sqlite (16 (null))␣
↪redmine-mysql (16 (null)) redmine-pgsql (0 (null)) ruby (16 (null)) ruby-
↪interpreter (0 (null)) ruby-actionpack-action-caching (0 (null)) ruby-actionpack-
↪xml-parser (0 (null)) ruby-awesome-nested-set (0 (null)) ruby-bundler (0 (null))␣
↪ruby-coderay (2 1.0.6) ruby-i18n (2 0.6.9-1~) ruby-jquery-rails (2 4.0.5) ruby-mime-
↪types (2 1.25) ruby-net-ldap (2 0.3.1) ruby-openid (0 (null)) ruby-protected-
↪attributes (0 (null)) ruby-rack (2 1.4.5~) ruby-rack-openid (0 (null)) ruby-rails␣
↪(2 2:4.2.5) ruby-rails-observers (0 (null)) ruby-rbpdf (0 (null)) ruby-redcarpet (0␣
↪(null)) ruby-request-store (0 (null)) ruby-rmagick (0 (null)) ruby-roadie-rails (0␣
↪(null)) debconf (18 0.5) debconf-2.0 (0 (null)) redmine-plugin-botsfilter (1 1.02-
↪2) redmine-plugin-recaptcha (1 0.1.0+git20121018) passenger (0 (null)) bzr (0␣
↪(null)) cvs (0 (null)) darcs (0 (null)) git (0 (null)) mercurial (0 (null)) ruby-
↪fcgi (0 (null)) subversion (0 (null))
Provides:
3.2.1-2 -
Reverse Provides:
```

Plan:

1. Create new server, ubuntu 16.04

   Created at UNM CIT, 8GB RAM, 4 CPU, 1TB disk. VM is d1-redmine5.dataone.org running on 64.106.40.38

2. Update, install mariadb-server, redmine via apt

   ```
   sudo apt-get install mariadb-server
   sudo apt-get install apache2
   sudo a2enmod ssl
   sudo a2enmod headers
   sudo a2ensite default-ssl
   sudo apt-get install passenger
   sudo apt-get install libapache2-mod-passenger
   sudo chown -R www-data:www-data /usr/share/redmine/public/plugin_assets
   sudo apt-get install imagemagick
   sudo apt-get install libmagickwand-dev
   sudo apt-get install ruby-rmagick
   sudo ufw allow 443
   ```

3. Make redmine readonly

4. Copy across attachments, mysql database dump, load database
5. Upgrade the database
6. Check operations
7. Migrate database to Postgresql
8. Verify operation
9. Install plugins
10. Switch DNS, make new redmine the current one

Plugins to install:

- scrum https://redmine.ociotec.com/projects/redmine-plugin-scrum
- redmine_checklists (free version) https://www.redmineup.com/pages/plugins/checklists
- Clipboard_image_paste http://www.redmine.org/plugins/clipboard_image_paste
- redmine_custom_css http://www.redmine.org/plugins/redmine_custom_css
- redmine_wiki_extensions http://www.redmine.org/plugins/redmine_wiki_extensions
- redmine_wiki_lists http://www.redmine.org/plugins/redmine_wiki_lists

Needed to adjust permissions to allow bundler to run without root (running with root really messes things up). Some help here: https://www.redmineup.com/pages/help/installation/how-to-install-redmine- plugins-from-packages

In `/usr/share/redmine`:

```
chmod -R g+w public/plugin_assets
sudo chmod -R g+w public/plugin_assets
sudo chmod -R g+w tmp
chown -R www-data:www-data db
sudo chmod -R g+w www-data db
sudo chmod -R g+w  db
```

```
cd /usr/share/redmine
bundle install --without development test
```

Transferred to Postgresql using `pgloader`:

```
pgloader mysql://redmine:<<password>>@localhost/redmine_default pgsql:///redmine_
↪default
```

After the transfer, needed to adjust table etc ownership:

```
for tbl in `psql -qAt -c "select tablename from pg_tables where schemaname = 'public';
↪" redmine_default` ; do  psql -c "alter table \"$tbl\" owner to redmine" redmine_
↪default ; done

for tbl in `psql -qAt -c "select sequence_name from information_schema.sequences␣
↪where sequence_schema = 'public';" redmine_default` ; do  psql -c "alter table \"
↪$tbl\" owner to redmine" redmine_default ; done

for tbl in `psql -qAt -c "select table_name from information_schema.views where table_
↪schema = 'public';" redmine_default` ; do  psql -c "alter table \"$tbl\" owner to␣
↪redmine" redmine_default ; done
```

and set defaults for new objects:

```
alter database redmine_default owner to redmine;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO redmine;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO redmine;
GRANT ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public TO redmine;
alter default privileges grant all on functions to redmine;
alter default privileges grant all on sequences to redmine;
alter default privileges grant all on tables to redmine;
```

Installed scrum plugin from https://redmine.ociotec.com/projects/redmine-plugin-scrum/wiki:

```
bundle exec rake redmine:plugins:migrate --trace NAME=scrum RAILS_ENV=production
```

1.1.4 Metrics

DataONE Contributors

Examine the solr index to determine the number of contributors to content exposed by DataONE. The count provided here is likely higher than the actual number of contributors because the indexing process only performs minimal pre-processing of names added to the index. For example, in some cases names appear with both ASCII and Unicode variants and are treated as separate.

[1]:
```
import requests
import json
import pprint
from datetime import datetime
import dateutil

SOLR_TIME_FORMAT = "%Y-%m-%dT%H:%M:%SZ"

T_NOW = datetime.utcnow()
T_START = datetime(2012,7,1)
```

[2]:
```
def getContributors(t_start=None, t_end=None):
    url = "https://cn.dataone.org/cn/v2/query/solr/"
    params = {
        "q":"*:*",
        "facet":"on",
        "rows":"0",
        "facet.limit":"-1",
        "facet.field":"investigator",
        "wt":"json",
             }
    dq = None
    if t_start is not None:
        st = f"{t_start:{SOLR_TIME_FORMAT}}"
        if t_end is None:
            dq = f"dateUploaded:[\"{st}\" TO \"{T_NOW}\"]"
        else:
            dq = f"dateUploaded:[\"{st}\" TO \"{t_end:{SOLR_TIME_FORMAT}}\"]"
    else:
        et = f"{t_end:{SOLR_TIME_FORMAT}}"
        dq = f"dateUploaded:[* TO \"{et}\"]"
    if dq is not None:
        params["q"] = dq
```

(continues on next page)

(continued from previous page)

```
    response = requests.get(url, params=params)
    data = json.loads(response.text)
    investigators = data["facet_counts"]["facet_fields"]["investigator"]
    names = []
    counts = []
    for i in range(0, len(investigators), 2):
        n = investigators[i+1]
        if n > 0:
            names.append(investigators[i])
            counts.append(investigators[i+1])
    return names, counts
```

[6]:

```
c_date = T_START
name_count = []
columns = ["date","contributors"]
print(",".join(columns))
while c_date < T_NOW:
    names, counts = getContributors(t_end = c_date)
    entry = (c_date, len(names))
    print(f"{entry[0]:%Y-%m-%d},{entry[1]}")
    name_count.append(entry)
    c_date = c_date + dateutil.relativedelta.relativedelta(months=+1)
```

```
date,contributors
2012-07-01,10741
2012-08-01,11900
2012-09-01,12372
2012-10-01,12784
2012-11-01,13562
2012-12-01,14145
2013-01-01,14594
2013-02-01,15405
2013-03-01,16250
2013-04-01,16926
2013-05-01,17479
2013-06-01,18434
2013-07-01,19077
2013-08-01,19717
2013-09-01,20488
2013-10-01,21481
2013-11-01,22236
2013-12-01,22899
2014-01-01,23468
2014-02-01,24402
2014-03-01,25338
2014-04-01,26142
2014-05-01,27010
2014-06-01,27738
2014-07-01,28675
2014-08-01,29670
2014-09-01,30339
2014-10-01,31801
2014-11-01,32896
2014-12-01,34765
```

(continues on next page)

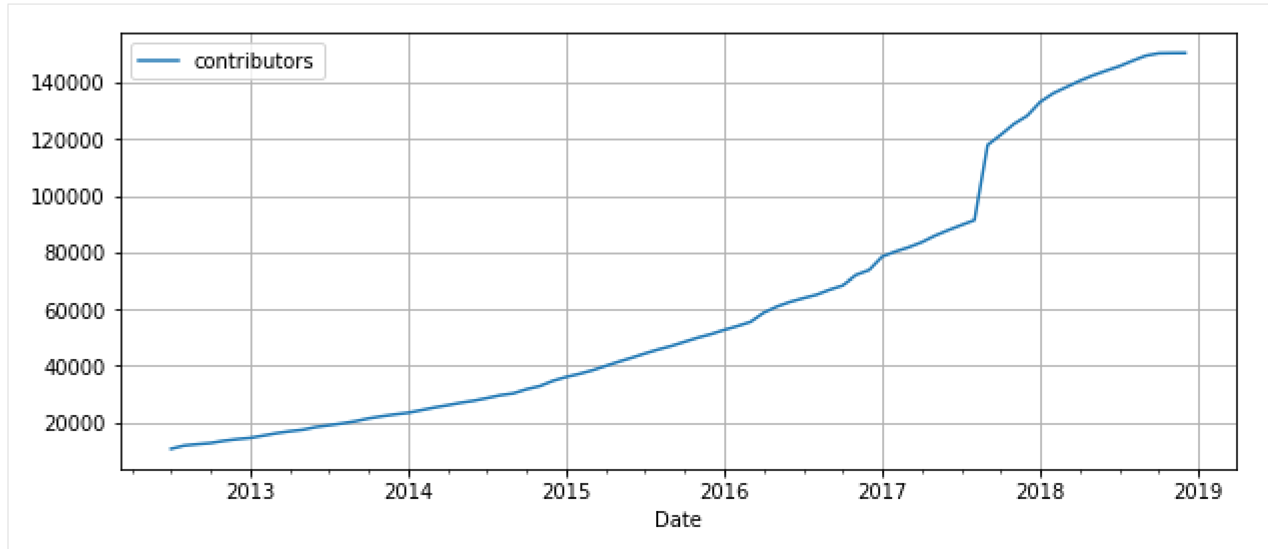
(continued from previous page)

```
2015-01-01,36120
2015-02-01,37216
2015-03-01,38473
2015-04-01,40018
2015-05-01,41530
2015-06-01,42940
2015-07-01,44434
2015-08-01,45835
2015-09-01,47079
2015-10-01,48547
2015-11-01,49998
2015-12-01,51250
2016-01-01,52730
2016-02-01,54083
2016-03-01,55565
2016-04-01,58823
2016-05-01,60980
2016-06-01,62622
2016-07-01,63926
2016-08-01,65132
2016-09-01,66912
2016-10-01,68409
2016-11-01,71985
2016-12-01,73708
2017-01-01,78495
2017-02-01,80239
2017-03-01,81732
2017-04-01,83531
2017-05-01,85831
2017-06-01,87768
2017-07-01,89567
2017-08-01,91315
2017-09-01,117874
2017-10-01,121462
2017-11-01,125302
2017-12-01,128160
2018-01-01,133103
2018-02-01,136162
2018-03-01,138344
2018-04-01,140542
2018-05-01,142453
2018-06-01,144095
2018-07-01,145664
2018-08-01,147594
2018-09-01,149380
2018-10-01,150243
2018-11-01,150317
2018-12-01,150348
```

[7]:

```
import pandas as pd
import matplotlib.pyplot as plt
data_frame = pd.DataFrame(name_count, columns=columns)
data_frame.set_index('date', inplace=True)

plot = data_frame.plot(figsize=(10,4))
plot.set_xlabel("Date")
plot.grid(True)
```




```
[ ]:
```

Listing Member Nodes

The authoritative list of Member Nodes is provided by the `/node` endpoint of the Coordinating Nodes. This notebook shows how to retrieve that information using the command lines tools `curl` and `xml starlet`.

The response from the `/node` endpoint is an XML document with multiple `node` elements, each containing details of a node. For example:

```
<node replicate="false" synchronize="false" type="mn" state="up">
        <identifier>urn:node:PANGAEA</identifier>
        <name>PANGAEA</name>
        <description>Data publisher for Earth &amp; Environmental Science</description>
        <baseURL>https://pangaea-orc-1.dataone.org/mn</baseURL>
        <services>
            <service name="MNCore" version="v1" available="true"/>
            <service name="MNRead" version="v1" available="true"/>
            <service name="MNAuthorization" version="v1" available="true"/>
            <service name="MNStorage" version="v1" available="true"/>
            <service name="MNReplication" version="v1" available="true"/>
            <service name="MNCore" version="v2" available="true"/>
            <service name="MNRead" version="v2" available="true"/>
            <service name="MNAuthorization" version="v2" available="true"/>
            <service name="MNStorage" version="v2" available="true"/>
            <service name="MNReplication" version="v2" available="true"/>
        </services>
        <synchronization>
            <schedule hour="*" mday="*" min="11" mon="*" sec="0" wday="?" year="*"/>
            <lastHarvested>2018-05-03T03:01:02.868+00:00</lastHarvested>
            <lastCompleteHarvest>1900-01-01T00:00:00.000+00:00</lastCompleteHarvest>
        </synchronization>
        <subject>CN=urn:node:PANGAEA,DC=dataone,DC=org</subject>
        <contactSubject>CN=M I A213106,O=Google,C=US,DC=cilogon,DC=org</contactSubject>
        <property key="CN_node_name">PANGAEA Data Publisher for Earth and␣
↪Environmental Science</property>
```

(continues on next page)

(continued from previous page)

```
        <property key="CN_operational_status">operational</property>
        <property key="CN_logo_url">https://raw.githubusercontent.com/DataONEorg/
↪member-node-info/master/production/graphics/web/PANGAEA.png</property>
        <property key="CN_date_upcoming">2017-11-14T22:00:00</property>
        <property key="CN_info_url">https://www.pangaea.de/</property>
        <property key="CN_location_lonlat">8.8506,53.1101</property>
        <property key="CN_date_operational">2018-03-20T17:46:00.000Z</property>
    </node>
```

This information can be processed using an XML parser such as python's `ElementTree` to retrieve specific values of interest.

[38]:

```
import requests
import xml.etree.ElementTree as ET
from pprint import pprint

#The /node document endpoint
url = "https://cn.dataone.org/cn/v2/node"

node_document = requests.get(url).text
node_tree = ET.fromstring(node_document)
nodes = []

#Extract the node entry items of interest
for node in node_tree.iter("node"):
    node_id = node.find("identifier").text
    node_coords = node.find("property[@key='CN_location_lonlat']")
    if not node_coords is None:
        entry = {"node_id":node_id,
                 "name":node.find("name").text,
                 "type":node.get("type"),
                 "state":node.get("state"),
                 "status":node.find("property[@key='CN_operational_status']").text
                }
        node_coords = node_coords.text.split(",")
        node_coords = list(map(float, node_coords))
        # reverse coords since leaflet wants latitude first
        entry["location"] = (node_coords[1], node_coords[0])
        nodes.append( entry )

# Display the node list
for n in nodes:
    print(f"{n['node_id']:20} {n['type']:3} {n['state']:4} {n['status']:14} {n['name
↪']:40}")
```

```
urn:node:CNUNM1      cn  up   operational    cn-unm-1
urn:node:CNUCSB1     cn  up   operational    cn-ucsb-1
urn:node:CNORC1      cn  up   operational    cn-orc-1
urn:node:KNB         mn  up   operational    KNB Data Repository
urn:node:ESA         mn  up   operational    ESA Data Registry
urn:node:SANPARKS    mn  up   operational    SANParks Data Repository
urn:node:ORNLDAAC    mn  down operational    ORNL DAAC
urn:node:LTER        mn  up   operational    U.S. LTER Network
urn:node:CDL         mn  up   operational    UC3 Merritt
urn:node:PISCO       mn  up   operational    PISCO MN
urn:node:ONEShare    mn  up   operational    ONEShare DataONE Member Node
```

(continues on next page)

(continued from previous page)

```
urn:node:mnORC1      mn  up   replicator     DataONE ORC Dedicated Replica Server
urn:node:mnUNM1      mn  up   replicator     DataONE UNM Dedicated Replica Server
urn:node:mnUCSB1     mn  up   replicator     DataONE UCSB Dedicated Replica Server
urn:node:TFRI        mn  up   operational    TFRI Data Catalog
urn:node:USANPN      mn  down contributing   USA National Phenology Network
urn:node:SEAD        mn  up   operational    SEAD Virtual Archive
urn:node:GOA         mn  up   operational    Gulf of Alaska Data Portal
urn:node:KUBI        mn  down operational    University of Kansas - Biodiversity␣
↪Institute
urn:node:LTER_EUROPE mn  up   operational    LTER Europe Member Node
urn:node:DRYAD       mn  up   operational    Dryad Digital Repository
urn:node:CLOEBIRD    mn  up   operational    Cornell Lab of Ornithology - eBird
urn:node:EDACGSTORE  mn  up   operational    EDAC Gstore Repository
urn:node:IOE         mn  up   operational    Montana IoE Data Repository
urn:node:US_MPC      mn  up   operational    Minnesota Population Center
urn:node:EDORA       mn  down operational    Environmental Data for the Oak Ridge␣
↪Area (EDORA)
urn:node:RGD         mn  down operational    Regional and Global biogeochemical␣
↪dynamics Data (RGD)
urn:node:GLEON       mn  down contributing   GLEON Data Repository
urn:node:IARC        mn  up   operational    IARC Data Archive
urn:node:NMEPSCOR    mn  up   operational    NM EPSCoR Tier 4 Node
urn:node:TERN        mn  up   operational    TERN Australia
urn:node:NKN         mn  up   operational    Northwest Knowledge Network
urn:node:USGS_SDC    mn  up   operational    USGS Science Data Catalog
urn:node:NRDC        mn  up   operational    NRDC DataONE member node
urn:node:NCEI        mn  up   operational    NOAA NCEI Oceanographic Data Archive
urn:node:PPBIO       mn  up   operational    PPBio
urn:node:NEON        mn  up   operational    NEON Member Node
urn:node:TDAR        mn  up   operational    The Digital Archaeological Record
urn:node:ARCTIC      mn  up   operational    Arctic Data Center
urn:node:BCODMO      mn  up   operational    Biological and Chemical Oceanography␣
↪Data Management Office (BCO-DMO)
urn:node:GRIIDC      mn  up   operational    Gulf of Mexico Research Initiative␣
↪Information and Data Cooperative (GRIIDC)
urn:node:R2R         mn  up   operational    Rolling Deck to Repository (R2R)
urn:node:EDI         mn  up   operational    Environmental Data Initiative
urn:node:UIC         mn  up   operational    A Member Node for University of Illinois␣
↪at Chicago.
urn:node:RW          mn  up   operational    Research Workspace
urn:node:FEMC        mn  up   operational    Forest Ecosystem Monitoring Cooperative␣
↪Member Node
urn:node:OTS_NDC     mn  up   operational    Organization for Tropical Studies -␣
↪Neotropical Data Center
urn:node:PANGAEA     mn  up   operational    PANGAEA
urn:node:ESS_DIVE    mn  up   operational    ESS-DIVE: Deep Insight for Earth Science␣
↪Data
urn:node:CAS_CERN    mn  up   operational    Chinese Ecosystem Research Network (CERN)
```

Now display the nodes on a map using the `ipyleaflet` extension.

First group nodes that are close so they can be drawn with a marker cluster.

[39]:
```
def computeGroupCentroid(nodes):
    sx = 0
    sy = 0
    for n in nodes:
```

(continues on next page)

(continued from previous page)

```
        sx += n["location"][1]
        sy += n["location"][0]
    return (sy/len(nodes), sx/len(nodes))

def computeDistance(a, b):
    dx = (a[1]-b[1]) ** 2
    dy = (a[0]-b[0]) ** 2
    return (dx+dy) ** 0.5

#Initialize the groups with the first node.
#Each entry in the node_groups is a list of nodes that are close to the centroid of␣
↪those nodes.
node_groups = [
                [nodes[0],],
              ]
for node in nodes[1:]:
    added = False
    for gnodes in node_groups:
        gc = computeGroupCentroid(gnodes)
        dist = computeDistance(node["location"], gc)
        if dist < 5.0:
            gnodes.append(node)
            added = True
    if not added:
        node_groups.append([node, ])
print(f"Grouped {len(nodes)} nodes to {len(node_groups)} groups")
```

```
Grouped 50 nodes to 24 groups
```

Now render the nodes using `ipyleaflet`.

[40]:
```
from ipyleaflet import Map, Marker, CircleMarker, MarkerCluster

m = Map(center=(30, -40), zoom=2)
for ng in node_groups:
    if len(ng) == 1:
        node = ng[0]
        marker = None
        if node["type"] == "mn":
            marker = Marker(location=node["location"], draggable=False, title=node[
↪"name"])
        else:
            marker = CircleMarker(location=node["location"])
        m.add_layer(marker)
    else:
        markers = []
        for node in ng:
            marker = None
            if node["type"] == "mn":
                marker = Marker(location=node["location"], draggable=False,␣
↪title=node["name"])
            else:
                marker = CircleMarker(location=node["location"])
            markers.append(marker)
        marker_cluster = MarkerCluster(markers=markers)
        m.add_layer(marker_cluster)
```

(continues on next page)

(continued from previous page)

```
m
```

```
Map(basemap={'url': 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', 'max_zoom':␣
↪19, 'attribution': 'Map ...
```

[ ]:

Object Counts Over Time

Show the number of objects accessible through DataONE over time.

This process uses the solr index to identify the number of different types of objects available

[1]:
```
import requests
import json
import pprint
from datetime import datetime
import dateutil

SOLR_TIME_FORMAT = "%Y-%m-%dT%H:%M:%SZ"

T_NOW = datetime.utcnow()
T_START = datetime(2012,7,1)
```

[2]:
```
import sys
print(sys.version)
```

```
3.7.0 (default, Jun 28 2018, 07:39:16)
[Clang 4.0.1 (tags/RELEASE_401/final)]
```

[3]:
```
def getObjectCounts(t_start=None, t_end=None):
    results = {
        "metadata": 0,
        "data": 0,
        "resource": 0,
    }
    url = "https://cn.dataone.org/cn/v2/query/solr/"
    params = {
        "q":"-obsoletedBy:[* TO *]",
        "rows":"0",
        "wt":"json",
        "indent":"on",
        "facet":"on",
        "facet.field": "formatType",
        "facet.mincount": 1,
            }
    dq = None
    if t_start is not None:
        st = f"{t_start:{SOLR_TIME_FORMAT}}"
        if t_end is None:
            dq = f"dateUploaded:[\"{st}\" TO \"{T_NOW}\"]"
        else:
            dq = f"dateUploaded:[\"{st}\" TO \"{t_end:{SOLR_TIME_FORMAT}}\"]"
    elif t_end is not None:
        et = f"{t_end:{SOLR_TIME_FORMAT}}"
```

(continues on next page)

(continued from previous page)

```
        dq = f"dateUploaded:[* TO \"{et}\"]"
    if dq is not None:
        params["q"] = params["q"] + " AND " + dq
    response = requests.get(url, params=params)
    data = json.loads(response.text)
    ftcounts = data["facet_counts"]["facet_fields"]["formatType"]
    for i in range(0, len(ftcounts),2):
        ft = ftcounts[i].lower()
        results[ft] = ftcounts[i+1]
    return results
```

[4]:
```
getObjectCounts()
```

[4]:
```
{'metadata': 804196, 'data': 1172369, 'resource': 198233}
```

[5]:
```

c_date = T_START
object_count = []
columns = ["date", "metadata", "data", "resource"]
print(",".join(columns))
while c_date < T_NOW:
    counts = getObjectCounts(t_end = c_date)
    entry = (c_date, counts["metadata"], counts["data"], counts["resource"])
    print(f"{entry[0]:%Y-%m-%d},{entry[1]},{entry[2]},{entry[3]}")
    object_count.append(entry)
    c_date = c_date + dateutil.relativedelta.relativedelta(months=+1)
```

```
date,metadata,data,resource
2012-07-01,53935,37983,33271
2012-08-01,56013,38337,33636
2012-09-01,57153,38734,33935
2012-10-01,58170,39212,34219
2012-11-01,59935,39850,34667
2012-12-01,61498,40576,35250
2013-01-01,63165,40849,36256
2013-02-01,64988,41559,37858
2013-03-01,67136,42468,38513
2013-04-01,78131,43175,48342
2013-05-01,79531,43720,48726
2013-06-01,81426,44486,49083
2013-07-01,82835,45198,49456
2013-08-01,84211,45827,49855
2013-09-01,85626,46494,50189
2013-10-01,101524,91431,65308
2013-11-01,102953,93025,65718
2013-12-01,104262,106207,65993
2014-01-01,105829,107285,66378
2014-02-01,107460,108266,66858
2014-03-01,109182,109013,67379
2014-04-01,110895,110098,68106
2014-05-01,112443,110996,68415
2014-06-01,113896,111644,68789
2014-07-01,115695,112643,69204
2014-08-01,117569,113481,69751
```

(continues on next page)

(continued from previous page)

```
2014-09-01,119238,114091,70542
2014-10-01,121160,147466,71030
2014-11-01,139784,225202,88338
2014-12-01,141679,502607,88847
2015-01-01,144159,508124,89710
2015-02-01,146250,558404,90352
2015-03-01,170810,620899,110496
2015-04-01,174242,649892,111793
2015-05-01,177645,660905,113413
2015-06-01,185625,720123,119226
2015-07-01,192192,790147,123704
2015-08-01,200084,836458,128511
2015-09-01,208453,849627,134790
2015-10-01,213515,855994,138041
2015-11-01,216298,857573,138842
2015-12-01,220646,864802,141069
2016-01-01,223302,868777,141986
2016-02-01,226123,870414,142989
2016-03-01,231065,873424,144912
2016-04-01,260480,878181,147397
2016-05-01,267149,880579,151778
2016-06-01,272268,882501,153953
2016-07-01,277060,885499,156556
2016-08-01,282571,887481,158336
2016-09-01,290742,891994,162117
2016-10-01,294872,893848,163689
2016-11-01,307446,900447,165220
2016-12-01,312058,915351,167228
2017-01-01,318793,927216,169008
2017-02-01,322898,937484,170277
2017-03-01,326970,947631,171784
2017-04-01,334848,953745,173059
2017-05-01,342194,956625,173989
2017-06-01,349928,960240,175346
2017-07-01,354602,971766,176519
2017-08-01,359828,1057937,177739
2017-09-01,581688,1088075,180015
2017-10-01,592941,1094996,181647
2017-11-01,605424,1109878,182794
2017-12-01,613787,1112706,183967
2018-01-01,642924,1114663,184872
2018-02-01,674201,1120909,187740
2018-03-01,698868,1125055,189670
2018-04-01,707816,1127915,191504
2018-05-01,791613,1141153,192761
2018-06-01,794866,1150686,194455
2018-07-01,796735,1155786,195219
2018-08-01,799284,1161092,196260
2018-09-01,802460,1163023,197085
2018-10-01,803570,1170337,197600
2018-11-01,803734,1171411,197761
2018-12-01,804158,1172252,198195
```

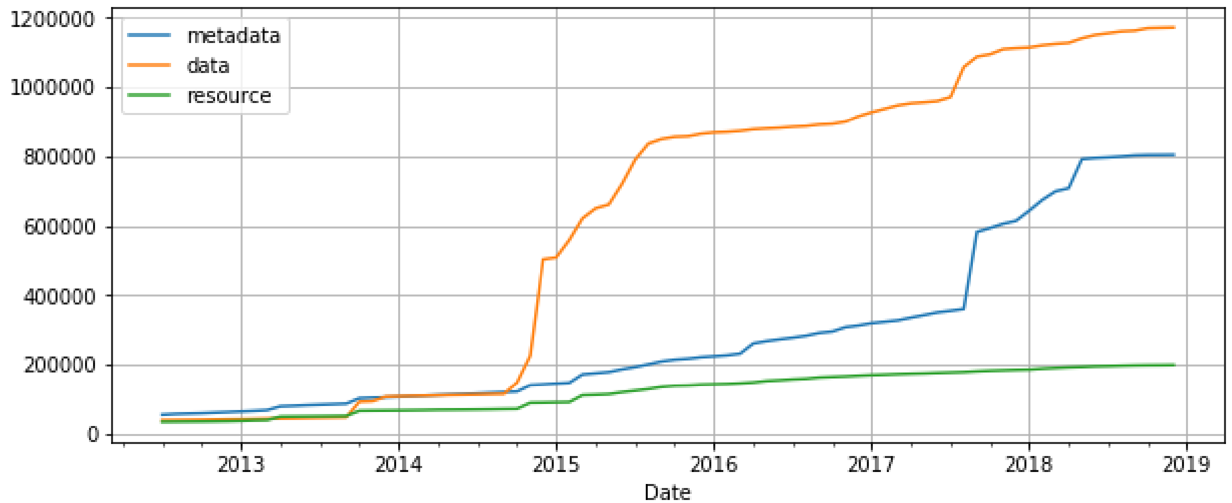
[7]:
```
import pandas as pd
import matplotlib.pyplot as plt
data_frame = pd.DataFrame(object_count, columns=columns)
```

(continues on next page)

(continued from previous page)

```
data_frame.set_index('date', inplace=True)

plot = data_frame.plot(figsize=(10,4))
plot.set_xlabel("Date")
plot.grid(True)
```



[ ]:

Amount of Content

Number of Contributors

A contributor is a subject that is identified as an author of a dataset.

Authors can be:

- The subject that added the content to DataONE. This is expressed in the system metadata as the `submitter`. However, the submitter may not be the author or creator of the content. For example, a data manager who had no involvement in the creation of the dataset may have uploaded the content.
- The authors identified in the science metadata associated with the dataset. The DataONE indexing service extracts this information from the metadata and populates the `author` and related fields of the search index.

Submission Contributors

Submission contributors is expressed as a single system metadata property which is indexed to the `submitter` field.

The unique values of `submitter` can be determined by examining the solr index with a facet query:

```
https://cn.dataone.org/cn/v2/query/solr/
  ?q=*%3A*
  &facet=on
  &rows=0
  &facet.limit=-1
  &facet.field=investigator
```

which can be processed with using `xmlstarlet` and `wc` as small bash script to show the number of unique values:

```
#!/bin/bash

SOLR_FIELD="${1}"
URL="https://cn.dataone.org/cn/v2/query/solr/?q=*%3A*"
URL="${URL}&facet=on&rows=0&facet.limit=-1"
URL="${URL}&facet.field=${SOLR_FIELD}"
echo "URL: ${URL}"
curl -s "${URL}" | xml sel -t -m "//lst/lst/int" -v "@name" -n | wc -l
```

As of this writing, the number of unique `submitters` is:

```
./field_count submiter
  16340
```

Authorship Contributors

Authorship information is expressed in several fields in the solr index:

Field	Count
`author`	24636
`authorGivenName`	11908
`authorLastName`	5863
`authorSurName`	24648
`authorSurNameSort`	24219
`investigator`	111826
`investigatorText`	64246
`origin`	108693
`originText`	64209
`originator`	0
`originatorText`	64209
`prov_generatedByUser`	0

In order to provide a realistic count, it would be necessary to retrieve the values from the fields of interest, normalize them to a standard representation, then count the resulting unique values.

1.1.5 Virtual Machine Setup

cn.server.publiccert.filename=/etc/letsencrypt/live/cn-stage-2.test.dataone.org/cert.pem environment.hosts=cn-stage-2.test.dataone.org cn-stage-unm-2.test.dataone.org

Kernel Purging

`/etc/cron.daily/purge-old-kernels`:

```
#!/bin/bash
/usr/local/bin/purge_old_kernels.py -q
```

Ensure to `chmod a+x /etc/cron.daily/purge-old-kernels`.

s Use Xenial Kernel ————–

Due to:

https://bugs.launchpad.net/ubuntu/+source/linux/+bug/1787127

it was necessary to upgrade many servers to use the 4.x kernel version.

```
sudo apt-get install linux-generic-lts-xenial
```

then restart to pick up the new kernel.

LVM Resize

Add new device.

Recognize new device:

```
apt-get install scsitools
rescan-scsi-bus.sh
```

Format the new disk:

```
fdisk /dev/sdb
  n (for new partition)
  p (for primary partition)
  1 (partition number)
  (keep the other values default)
  w (write changes)

fdisk /dev/sdb
  t (change the partition type)
  8e (for Linux LVM)
  w (write changes)
```

Initialize LVM physical volume:

```
pvcreate /dev/sdb1
```

Determine name of volume group:

```
vgdisplay
```

Add physical volume to volume group:

```
vgextend name_of_volume_group /dev/sdb1
```

New free space should appear in:

```
vgdisplay
```

Resize logical volumes to use free space:

```
lvdisplay
```

Add space to the logical volume:

```
lvresize --resizefs --size +931GB /dev/name_of_volume_group/Name
```

or more manually, add space to a logical volume:

```
lvextend -L +100G /dev/name_of_volume_group/Name
```

then resize the filesystem (ext4):

```
resize2fs /dev/name_of_volume_group/Name
```

1.2 Member Nodes Resources

The DataONE website has links to several documents which describe DataONE and questions for data entities to address when they are considering becoming a Member Node. A potential MN should review the following information to help decide if a partnership with DataONE is appropriate.

- The Member Nodes page at DataONE.
- Member Node Fact Sheet
- Member Node Partnership Guidelines
- The Member Node Description Document, which is submitted as part of the proposal process.

Still have questions? Send an email to mninfo.dataone@gmail.com or submit your question via the Contact Us form.

Member Node Operations

1.2.1 Adjusting Custom Node Properties

Member Nodes generally control the content available in the Member Node document provided through the /node service. This document describes custom properties that may be set by the Coordinating Nodes to augment the information provided by the Member Nodes. Such custom properties are not over-written by a Member Node when the Member Node registration information is updated.

Custom properties set by the Coordinating Nodes MUST be prefixed by `CN_`. Such properties will be preserved through updates to the node registration. Note that the string "CN_" has no meaning other than to act as a mechanism for distinguishing custom properties from other properties that may be set by the MN.

The value of the custom property `${PROPERTY}` for the Member Node `${NODE_ID}` can be determined from the DataONE list nodes API with the following XPath statement:

```
//node[identifier/text()='${NODE_ID}']/property[@key='${PROPERTY}']
```

For example, using `curl` and `xmlstarlet`:

```
NODE_ID="urn:node:KNB"
PROPERTY="CN_date_operational"
curl -s "https://cn.dataone.org/cn/v2/node" | \
  xml sel -t \
  -m "//node[identifier/text()='${NODE_ID}']/property[@key='${PROPERTY}']" -v "."
2012-07-23T00:00:0.000Z
```

Note that in order to set any properties, the node must be registered. Furthermore, for the node to appear in the node list, it is necessary for the node to be approved. This is potentially problematic for nodes that are "upcoming" (i.e. `CN_operational_status = upcoming`) since such nodes should initially not be synchronizing. As such, it is necessary for an upcoming Member Node to be registered and approved, but with services flagged as being off. In order to prevent a node from accidentally setting node state to allow active harvesting, it is recommended that the initial node registration is created by a different subject to the node operator. This issue will be addressed in a later release of the infrastructure #8058.

Preferred Custom Properties

The following custom properties are used by systems such as the DataONE search interface and Member Node dashboard.

For properties that express a date, these MUST be in UTC and formatted as ISO-8601 (`YYYY-MM-DDTHH:mm:ss.sssZ`). If the time portion is unknown, then substitute `00:00:00.000`. Example:

```
2017-03-20T15:25:53.514Z
```

CN_node_name

Provides the same information as the ` `name` <https://releases.dataone.org/online/api-documentation-v2.0/apis/Types.html#Types.Node.name>`__ element of the node document though may optionally be augmented by DataONE to provide additional or clarifying details.

Example:

```
<property key="CN_node_name">node name text string</property>
```

CN_operational_status

Indicates the current operational status of the Member Node. The value can be one of:

`operational`: The Member Node is operational and contributing content

`replicator`: The Member Node is operational but acts only as a replication target

`upcoming`: The Member Node is anticipated to be operational "soon".

`developing`: The Member Node is under active development and testing, should not be shown on dashboard or UI

`deprecated`: The Member Node has been deprecated and is no longer actively participating in the DataONE environment

Example:

```
<property key="CN_operational_status">operational</property>
```

CN_date_operational

The date that the Member Node became an operational participant in the DataONE environment. This should be the same time as when `CN_operational_status` is set to `operational` or `replicator`.

Example:

```
<property key="CN_date_operational">2017-03-20T15:25:53.514Z</property>
```

CN_date_upcoming

The date that the Member Node became designated as `upcoming`, meaning it is expected to soon be participating in an operational capacity. This should be the same time as when `CN_operational_status` is set to `upcoming`.

Example:

```
<property key="CN_date_upcoming">2017-03-20T15:25:53.514Z</property>
```

CN_date_deprecated

The date that the Member Node became deprecated from further particpation in the DataONE environment. This should be the same time as when `CN_operational_status` is set to `deprecated`.

Example:

```
<property key="CN_date_deprecated">2017-03-20T15:25:53.514Z</property>
```

CN_logo_url

The URL of the logo that is to be shown for the Member Node in user interfaces. Note that the protocol SHOULD be `https`.

Example:

```
<property key="CN_logo_url">https://raw.githubusercontent.com/DataONEorg/member-node-
↪info/master/production/graphics/web/KNB.png</property>
```

CN_info_url

A URL that provides additional information about the Member Node and its host. This may for example, point to a landing page describing the data repository being represented by the Member Node.

Example:

```
<property key="CN_info_url">https://knb.ecoinformatics.org/</property>
```

Getting and Setting Custom Properties

Custom Member Node properties are typically set directly in the Coordinating Node LDAP service where node information is stored.

A python script, d1nodeprops is available to get and set custom node properties. It is necessary for the CN administrator to create an `ssh` tunnel to a Coordinating Node forwarding the LDAP connection for the script to work. For example, to set the `CN_logo_url` for the node `urn:node:KNB`:

```
ssh -L 3890:localhost:389 cn-ucsb-1.dataone.org

#in another terminal
d1nodeprops -I "urn:node:KNB" -k "CN_logo_url" -p "SOME PASSWORD" -o update \
  "https://raw.githubusercontent.com/DataONEorg/member-node-info/master/production/
↪graphics/web/KNB.png"
```

To list the custom properties that have been set for a Member Node:

```
d1nodeprops -I "urn:node:KNB" -p "SOME PASSWORD"
```

[ ]:

```

```

Member Node Deployment

1.2.2 Managing Display of Upcoming Nodes

Upcoming Member Nodes are optionally added to the upcoming Member Nodes list

Adding a MN to the Upcoming Node List

Figure 1. Procedure for listing a MN in the list of upcoming Member Nodes without registering the node in the production environment.

1. There must be agreement within DataONE and by the MN that they can be listed as upcoming.
2. For the MN to display properly a number of custom properties must be set, and the values for these should be recorded in the corresponding MN Deployment ticket in readmine.
3. The logo for the MN must be ready for display and present in the web folder on in the member-node-info repository on GitHub.
4. Adding an `node` entry in the upcoming.xml document and committing the change to GitHub will trigger the listing of the node as upcoming in the dashboard.
5. The node should appear in the dashboard within 15 minutes.

Registering an Upcoming Member Node

The process of registering a node will add it to the production node list reported by the Coordinating Nodes.

Figure 2. Procedure for listing a Member Node as upcoming when it is registered with the production Coordinating Nodes.

1.2.3 Deployment to Production Environment

After a Member Node instance is successfully passing tests it can be deployed to the production environment. The steps for deployment are:

1. Add the MN logo to the Github repository
2. Add the MN to the list of upcoming Member Nodes
3. Verify that the MN instance is populated with production content, not test content
4. Prepare announcement for the MN release
5. Obtain a client certificate for the MN. This certificate must be signed by the Production Certificate Authority.
6. Register the MN in the production environment
7. Update the custom "CN_" node registration properties
8. Approve MN
9. Content starts synchronizing

10. Verify content has synchronized and appears in the search UI
11. Announce the new MN

1. Add MN Logo to GitHub

See the Member Node Info repository on GitHub.

2. Add to Upcoming

When a MN is getting close to deployment, it may be added the "upcoming" MN list that is shown on the DataONE Member Nodes dashboard.

3. Verify MN Content

It is important that no test content is present on the MN when it is being added to the production environment. Accidental content can be removed, but it is a process that should be avoided where possible.

Work with the MN operator to ensure no test content remains (if repurposing a test instance) and that the node is appropriately populated.

4. Prepare Announcement

The public announcement of a new MN requires a bit of back and forwards between DataONE and the MN, so it is best t start this process early to ensure the announcement document is ready.

5. Obtain Client Certificate

The client certificate is needed for the MN to register with the CN, and for any other actions that require the node to authenticate.

See the CA project in subversion at: https://repository.dataone.org/software/tools/trunk/ca/

6. Register MN

Registering the MN will record the presence of the MN in the Production environment node registry. The CNs will not interact with the MN until the registration is approved.

7. Add `CN_` Properties

After the node is registered

8. Approve MN

9. Content Synchronization

10. Sync and Search Verification

11. Announcement

1.2.4 Baseline Member Node Implementation Requirements

Note: 2018-02-19

This document is in draft status, copied mostly verbatim from subversion at: https://repository.dataone.org/documents/Projects/cicore/operations/source/member_node_deployment

All Member Nodes, regardless of Tier, share the same baseline requirement of managing their object collection such that:

1. All objects are identified by a globally unique identifier.

   Identifiers provide the primary mechanism for users to access a specific item. Identifiers must conform to the DataONE persistent identifier specification.

2. The original bytes of submitted objects are always available using the original identifier.

   To satisfy DataONE's preservation policy, the content bound to an identifier does not change and is thus immutable. Organizations starting with systems that allow overwrites will need additional mechanisms to ensure access to older versions (via *MN_Read.get()*)

3. Content access policies are respected.

   Tier 2 and higher nodes satisfy this requirement by implementing the API and following the functional requirements for access control implementation. By implication, Tier 1 nodes can only host and expose public objects.

4. Updates are treated as a separate objects.

   An update to an existing object causes creation of a new object and a new identifier, and the object being replaced is archived. (Archiving removes the object from the search index, but leaves it available via *MN_Read.get()*. The *isObsoletedBy / obsoletes* relationship is established in the system metadata of both objects to allow navigation through the version chain.)

5. Voluntary deletes are handled as archive actions.

   Objects published to DataONE are not deleted, but archived. This helps ensure DataONE's preservation requirements are upheld. An archived object has its *archived* field set to *true* in its systemMetadata, and the object remains accessible by its identifier. This applies to all objects ever exposed through the DataONE API regardless of how the content is added to a Member Node. For example, if a Member Node manages their collection with external tools, the system should behave such that deleted content is treated as archived, rather than actually deleted.

6. Compulsory (or administrative) deletes, are coordinated and executed via the DataONE Coordinating Nodes.

   These are true deletes where the content is removed from all nodes, and is done ONLY in cases when illegal or inappropriate content is discovered and needs to be removed. Because content may be replicated to other nodes, a Member Node performing a compulsory delete must notify DataONE of the action to ensure any replicas are also removed. Otherwise the content may remain discoverable and accessible at other nodes.

Note that the Tier 3 MN_Storage API implements requirements 4, 5, and 6, and exposes that functionality to authorized users. Tier 1 and 2 member nodes still implement the same functionality, however they do not expose the functionality through the DataONE MN_Storage API. Instead, creates, updates, and archives are handled through non-DataONE mechanisms.

1.2.5 Introduction to SSL Certificates

Note: 2018-02-19

This document is in draft status, copied mostly verbatim from subversion at: https://repository.dataone.org/documents/Projects/cicore/operations/source/member_node_deployment

DataONE uses SSL certificates, specifically X.509 certificates, to secure communication between Clients, Member Nodes, and Coordinating Nodes. All three of these actor-types should have certificates to offer for an SSL handshake, although Clients making DataONE requests without certificates are allowed.

Certificates are files that do two things:

1. assert that you are who you say you are, by establishing a chain-of-trust back to an authority your system already trusts. This is done through inclusion of verifiable digital signatures of the certificate issuer and signers of the certificate issuers certificate, etc.
2. contain encryption keys to be used in the connection for securing the data passed back and forth.

In the case of CILogon-issued certificates for DataONE, certificates have a third function:

3. CILogon certificates include additional associated Subjects in DataONE that can be used for Authorization.

In very general terms, both sides of the socket connection have to trust one another before a connection can be established. Both sides (peers) exchange their certificates, and if each side trusts the otherÕs signing entity contained in the certificate, the connection is established, and encryption keys are exchanged to allow secure transfer of information.

In the case of CILogon-issued certificates, Tier 2 and higher DataONE CNs and MNs extract the mapped identity and group Subjects from the certificates when making Authorization decisions.

Each of your systemÕs browsers maintain their own set of trusted certificate authority certificates (ÒCA certsÓ), largely overlapping, but not completely. Browsers also maintain sets of additional CA certs per user that become trusted upon the OK of the user. Your system also maintains a set of root certificates, as does your Java installation - these are used by programmatic SSL connections.

1.2.6 Development Contacts

Implementers needing clarification or help using the tools or having technical problems have several ways to get their questions answered. DataONE has set up a ways to contact the development team.

See also: Communications

Slack

The developers maintain a virtual office of sorts, in that they join a dedicated DataONE *Slack* channel for asking each other questions. Implementers are welcome to join the discussion, following the usual etiquette for online communication. Visit the `#ci` channel for technical topics.

ask.dataone.org

ask.dataone.org is a searchable compendium of questions and answers, based on the same model as stackoverflow. Especially if working off-hours, this could be the most direct way to get an answer, as it might already be answered, and if not, allows you to post the question. Developers try to answer all questions within a 48 hours.

1.2.7 Development iteration

Document Status

Status	Comment
DRAFT	(CSJ) Initial draft
DRAFT	(Dahl)

To enable a repository to communicate using the DataONE Member Node APIs, most groups take an iterative approach, implementing the lowest level APIs first, and working up through the *Tier*s. Implementing the DataONE APIs also involves managing identifiers, System Metadata, content versioning, and data packaging. We touch on these issues here.

Note: The DataONE Common and DataONE Client libraries provide serialization and deserialization of the DataONE types and language-specific wrappers for the DataONE APIs.

Implementing the Tier 1 Member Node API

MN developers will need to first implement the *MNCore API*, which provides utility methods for use by *CN*s, other *MN*s and ITK components. The `ping()` method provides a lightweight 'are you alive' operation, while the `getCapabilities()` provides the Node document so software clients can discover which services are provided by the MN. The CNs also use the `getLogRecords()` method to aggregate usage statistics of the Node and Object levels so that scientists and MN operators have access to usage for objects that they are authorized to view.

The bulk of the work within the system is provided by *MNRead API* methods. The CNs use the `listObjects()` method to harvest (or synchronize) Science Metadata from the Member Nodes on a scheduled basis. They also call `get()`, `getSystemMetadata()`, and `getChecksum()` to verify and register objects in the system. They will occasionally call `synchronizationFailed()` when there is an error, and MN operators should use these events to log these exceptions for `getLogRecords()`, and should inform the MN operator of the problems in a way they see fit (email, syslog, monitoring software, etc). Lastly, the `getReplica()` call is distinguished from the `get()` call in that it is only used between Member Nodes that are participating in the DataONE replication system, and that these events are logged differently to not inflate download statistics for objects. Implementing the MNRead API requires identifier management, System Metadata management, versioning, and packaging, described below.

Handling identifiers

To support as many repository systems as possible, DataONE puts few constraints on identifier strings that are used to identify objects. See the details for the Identifier Type. This flexibility requires that identifiers with special characters that would affect *REST* API calls, XML document structure, etc. will need to be escaped appropriately in serialized form. See the documentation on Identifiers for the details. MN operators will need to ensure that identifiers are immutable, in that there is a one to one relationship between an identifier and an object (byte for byte). This allows the DataONE system to use the checksum and byte count declared in System Metadata documents to verify and track objects correctly. Calling `get()` on an identifier must always return the same byte stream. See the documentation on mutability of content for more details.

System Metadata

Each object (*Science Data*, *Science Metadata*, or *Resource Maps*) that is exposed to the DataONE system via `MNRead.listObjects()` must have an accompanying *System Metadata* document. These documents provide

basic information for access, synchronization, replication, and versioning of content. MN operators will need to either store or generate System Metadata.

One notable component of System Metadata is the formatId field. To support as many repository systems and object types as possible, DataONE assigns format information to objects according to an extensible list of object formats. The Object Format List holds a list of types akin to MIME types and file formats. The definitive list is found on the production CNs. When a new MN, holding a new type of object, joins DataONE, the new Object Formats should be added to the Object Format List before they are used in the System Metadata formatID field. DataONE has been involved in the Unified Digital Format Registry project to establish a comprehensive registry of formats, and development is ongoing. When MN objects are synchronized, objects that are tagged in the Object Format List as holding metadata (i.e. Science Metadata), will be parsed and indexed on the CN for search and discovery services. The automatic parsing and indexing is not supported for all types of objects that contain Science Metadata. For objects that are not currently supported, MN developers should coordinate with the DataONE developers to provide mappings between metadata fields and fields in DataONE's search index. See the Content Discovery documentation for details.

Content versioning

Objects in DataONE are immutable. When a scientist or MN operator wants to update an object (Science Data, Science Metadata, or Resource Map), the process involves the following sequence:

1) Minting a new identifier
2) Creating a new System Metadata document for the new version of the object, and setting the 'obsoletes' field to the previous object identifier
3) Updating the previous object's System Metadata, setting the 'obsoletedBy' field to the new object's identifier, and setting the 'archived' field to 'true'.
4) Updating the Member Node data store to include the new object (and the old one). For Tier 1 and 2 MNs, this will happen outside of the DataONE API. For Tier 3 and Tier 4 MNs, it can be done through the MNStorage.update() call.

Since one of the main goals of DataONE is to provide a preservation network with citable data to enable reproducible science, this sequence is critical to the system's success. There are times when a Member Node won't be able to store earlier versions of objects indefinitely, in which case MNs should set a replication policy in their object's System Metadata to 'true' so that replicas can be made and the system will act as a persistent store of the versioned objects. However, Member Nodes have suggested that DataONE support mutable objects, and possible ways to support this without impeding the preservation goals of the federation are currently under investigation. DataONE is open for input on this. If you are facing issues with versioning, please contact support@dataone.org.

Data packaging

Scientists often work with separate objects (files) containing data and metadata, and want to access them as a collection. However, communities use different packaging technologies that are often specific to their data types. To support collections across a federated network, DataONE chose to represent data packages using the *OAI-ORE* specification. Also known as *Resource Map*s, these documents use *RDF* to describe relationships among objects (resources). DataONE has chosen a limited vocabulary to represent associations between objects. Curently, the associations are:

- describes / describedBy
- aggregates / isAggregatedBy
- documents / isDocumentedBy

For instance, a Resource Map *describes* an aggregation, and the aggregation *aggregates* a Science Metadata document and a Science Data object. In turn, the Science Metadata document *documents* a Science Data object. These relationships are captured in the Resource Map as 'triple statements', and provide a graph of associations. Resource Maps

may also be more complex, where one Science Metadata document *documents* many Science Data objects. Some repositories may find the need to create hierarchical collections, where one Resource Map *aggregates* another.

Member Node operators will need to store or generate Resource Maps that will get harvested during scheduled synchronization. The CNs will parse the maps and index these data/metadata relationships, to be used in content discovery. Note that Resource Maps will also need associated System Metadata documents. during this phase of development, MN operators can make use of the DataONE Common and DataONE Client libraries for building Resource Maps. Details about *data packaging* can be found in the architecture documentation.

1.2.8 Development Testing

Contents

- *Development Testing*
  - *Member Node Tester*
    - *Testing Requirements*
      - *Tier 1 Nodes*
      - *Tier 2 Nodes*
      - *Tier 3 & 4 Nodes*
    - *Anatomy of tester failure messages*
    - *Identifier Encoding Tests*
    - *Authentication and Authorization Tests*
  - *Alternate Testing Approaches*

Development testing is done to make sure that the future Member Node is ready for real-world testing in the staging environment. This includes testing for correct method responses under different circumstances, making sure that metadata and resource maps parse and are usable, and that the formats used by the node are registered.

To accomplish this, DataONE has developed a suite of tests that are used to test our own Member Node implementations. For convenience, we have deployed these tests as a web-based Member Node testing service. Those already familiar with Maven and JUnit may instead wish to download the tests from our code repository and run the tests locally. Passing all of the required tests will signal readiness to enter the deployment phase.

While Member Node developers are encouraged to run the MN testing service at any point in their internal development process, the primary source of guidance on *how* to implement a method should be the DataONE architecture documentation and API reference. When troubleshooting failed tests, in general, careful rereading of the method documentation in the context of the failure message is the best way to understand what exactly went wrong, as well as looking at the source code of the test. Developers are also welcome to contact the DataONE development team for more information and explanation.

Member Node Tester

Multiple versions of the testers are maintained, corresponding to the major.minor versions of the published API. Within each tester, the testing service is organized by Tier, so that Tier1 API methods are tested before Tier2 methods, and so on. Results are also organized by tier, and a summary for each subset of tests is given. If you are planning to deploy a Tier 1 node, then of course only worry about testing and passing Tier 1 methods.

Testing Requirements

Requirements for testing vary from tier to tier. Most tests rely on existing content, except MN_Storage methods, which need to test the create, update, and archive methods. Below are listed the general content related requirements for each Tier.

Tier 1 Nodes

Most of the Tier1 tests rely on listObjects to get an identifier that can be used to run Tier 1 tests. So, Tier 1 nodes need to have at least one object. Tier 1 also tests that resource maps can be parsed, so if there is only one object, it should be a resource map. Ideally, a member node would have a representative object for all of the data formats it has objects of.

Tier 2 Nodes

The bulk of Tier 2 testing is testing data security, and so the member node needs to load a set of objects with the pre-defined access policies.

TODO: make the test objects available as a download.

Tier 3 & 4 Nodes

If you are implementing a Tier 3 node, you do not need to start with any content in the node to test Tier 1 and 2 methods. Instead, the testers will create the data objects they need, provided that the create method is working properly.

You will probably, however, wish to clear out the test objects if the node under test is going into production.

TODO: instructions on how to identify these objects (they all have the same owner, I believe)

Tier 4 tests are limited to testing exception handling, since most of this functionality requires interaction with Coordinating nodes and other member nodes.

Anatomy of tester failure messages

Tests can fail for many reasons, so the best a failure message can hope to do is be clear about the circumstances that caused the failure, and provide relevant diagnostic information. Each failure message is composed of the following parts:

1. The failure description
2. REST call details
3. stack trace

Of the three, the **failure description** is the most useful for understanding the what failed. In some cases the description is the exception thrown (Class name:: detail code: http status: exception message), and in other cases it is just a message explaining the failure.

The **REST call details** are useful for recreating the call that resulted in a failure. Remembering that many tests (especially in Tier1) first call listObjects to get an identifier to work with for the main test, do not be surprised to see a problem in listObjects even if the test is for another method. Also note that in cases where it is known that the failure is client-side (not server side), the REST call is usually not provided.

The **stack trace** is there to debug client-side exceptions, or to find the spot in the *client* code that where an exception was thrown. Usually, this only helpful with content related problems.

Identifier Encoding Tests

For the methods that take an Identifier in the URL of the method call, identifier encoding tests are performed to make sure that the Member Node can handle the encoded identifiers properly. About 30 identifiers using characters from different ranges of the Unicode standard are tested against each of these methods.

If one example fails, the entire test fails, and you will need to do some detective work to find out the problem. Comparing suspicious characters in the failing identifier to characters in the passing examples can help to narrow the problem. Also, in the name of the identifier itself is a terse indication of the types of characters being tested, and in one case, the word 'tomcat' is used as it was first found to be problematic with certain tomcat web server configurations.

Authentication and Authorization Tests

Tier 2 and higher MNs will be subject to Authorization and Authentication testings. In this case, it is mostly the same API method being tested under different circumstances, and to ensure that various Access Policies are being interpreted correctly vs. a set of users (as represented by calls made by the testing service using different X509 client certificates).

As with the Identifier encoding tests, each test contains several related examples. and so comparing failing examples to passing examples gives a sense of where the problem may be. Often times, one root problem in the MN's authorization algorithm can cause dozens of failures.

For consistency with other DataONE nodes, DataONE strongly recommends using the reference authorization algorithms in d1_common_python or d1_common_java if at all possible. For those not using these packages, note that not only the algorithm will need to be transcribed, but you will need to also need to do Subject comparisons on canonical serializations of the client subject(s) and Subjects in the object's AccessPolicy.

Alternate Testing Approaches

If the web tester facility is not flexible enough, the d1_integration package is available for download, so that it can be run from a local machine. Testing Tier2 and higher nodes will require some setup and acquisition of testing certificates to work, however.

1.2.9 DataONE infrastructure environments

In addition to the production environment that the end user sees when interacting with DataONE services, DataONE maintains several other independent environments for development and testing. These environments emulate the production environment and allow developing and testing of DataONE components without affecting the production environment.

Each environment is a set of Coordinating Nodes (CNs) along with a set of Member Nodes (MNs) that are registered to those CNs. Each environment maintain sets of content, formats, and DataONE identities independent of each other. By registering a MN to an environment, you enable content to synchronize with the CNs, and be replicated to other nodes in the same environment.

Since there are no connections between the environments, information registered in one environment - certificates, DataONE identities, and formats - cannot be carried over into another environment.

The environments are:

Environment	URL	Description
Production	https://cn.dataone.org	Stable production environment for use by the public.
Staging	https://cn-stage.test.dataone.org	Testing of release candidates.
Sandbox	https://cn-sandbox.test.dataone.org	Like Production, but open to test instances of MNs. May contain both test and real science objects.
Development	https://cn-dev.test.dataone.org	Unstable components under active development.

The Production environment is only used by completed MNs that hold production quality data.

If a MN is under development or if it is experimental in nature, for instance, if the purpose is to learn more about the DataONE infrastructure or if the MN will be populated with objects that may not be of production quality, one of the test environments should be used.

To register a Member Node into an environment, follow the steps in *Node Registration*.

1.2.10 Appendix: Deploying the Generic Member Node

The *GMN* Server supports the **'DataONE Member Node API'_** and can be used as a general data and metadata repository. The GMN is used both as a Python reference implementation of the DataONE MemberNode API a working command-line based member node implementation.

For current information on downloading and deploying the GMN stack, see the GMN Documentation on Python-Hosted.org.

1.2.11 Plan the implementation

Status: Draft

Each institution that wishes to join DataONE as a MN will have unique decisions to make with regard to their MN implementation. The end goal is to be able to serve content (data and metadata) as part of the larger network. There may be many possible paths to this goal, each with different demands on available resources and with different amounts of technical effort involved.

Since there are a number of groups with expertise in the DataONE API and other software, MN developers should communicate with fellow developers and the DataONE development team in order to capitalize on previous experience before selecting an implementation strategy and during implementation. Based on the requirements of the organization, input from fellow developers and the DataONE development team, MN developers should create an implementation plan that will guide the development, testing, deployment and maintenance of the service.

Operators should understand the DataONE MN Tiers, and should consider different software approaches when developing this plan.

Choosing an implementation approach

When planning the MN implementation, developers should consider the following possibilities, listed in order of increasing effort.

(A) Use an existing repository that already has a DataONE API

(B) Modify existing software that implements the DataONE API to work with an existing repository

(C) Use existing DataONE libraries to add DataONE APIs to an existing repository

(D) Write custom software from scratch that implements the DataONE API and communicates with an existing repository

(E) Write a custom repository with a DataONE API from scratch.

For (A) and (B), DataONE provides two complete MN implementations that can be used for exposing data to DataONE. These are *Metacat* and *GMN*. Both support all MN tiers. Other institutions have also added DataONE interfaces to their existing repositories and these may provide good starting points.

For (C), DataONE provides the DataONE Common Library and DataONE Client Library for *Java* and *Python*. These libraries provide language specific abstractions for interacting with the DataONE infrastructure. In particular, they make it easy to perform and respond to DataONE API calls and produce and consume the DataONE types used in those calls. As there are many subtle implementation details that relate to issues such as URL-encoding, Unicode, REST messaging formats and working with the DataONE types, it is highly recommended to use these libraries if possible.

For (D) and (E), DataONE provides extensive documentation of the DataONE API and types. The documentation covers details such as URL-encoding, REST messaging formats and usage for all APIs. In addition, developers should work with the community and the DataONE development team to ensure that their implementation is correct. All the existing implementations are open source, and can be consulted if anything is unclear in the documentation. The source is available in the DataONE Subversion Repository.

Even if a MN will eventually require a custom implementation, it may be possible to use an existing MN implementation to quickly get an MN online, so that the organization can have a presense on DataONE while a fully functional MN is being implemented.

TODO Create a page for existing MN implementations

TODO Verify that the Java Client Library page is up to date

TODO Point to releases.dataone.org for common and libclient downloads

1.2.12 Member Node Approval Process

After reviewing DataONE literature and deciding that a partnership with DataONE would be beneficial to all parties, a Member Node will, working with project personnel, prepare a "proposal" including information about data holdings, scope of interaction with DataONE, etc. This includes completion of a Member Node Description Document.

The DataONE Member Nodes team will evaluate the proposal using several of the criteria below:

Selection criteria

- First Principle: Select MNs for joining DataONE based on number of instantiations.
- Second Principle: Select MNs based on quantity and quality of data.
- Third Principle: Select high-profile MNs.
- Fourth Principle: Select pragmatically - knowing that there are resources and enthusiasm to complete the tasks.
- Fifth Principle: Select MNs based on their ability to contribute to long-term sustainability.
- Sixth Principle: Select MNs that are diverse and geographically distributed so that we cover a range of MN types and globally distributed nodes.
- Seventh Principle: Select based on showing exemplars for what we want DataONE to be.

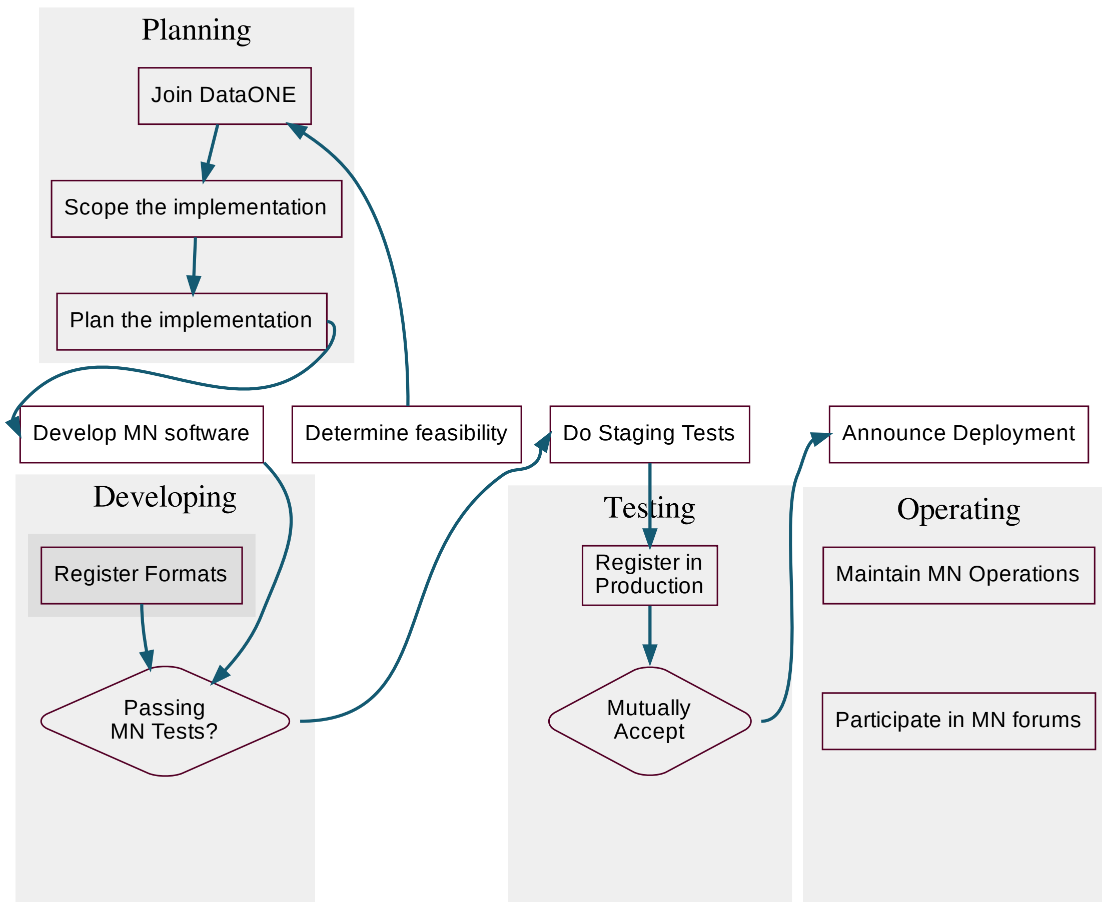
Criteria for Standing Up a DataONE Member Node

- Size and visibility of the community represented by the candidate MN.
- The collections are significant or enable new science or both.
- Fills significant gaps in the content available through DataONE.
- The data are unique data in the broader community.
- Collections are strong in breadth, depth or both
- The candidate brings significant contributions to the DataONE resource base, including: #. strategic partnerships #. professional expertise in managing data, computing, administration, etc. #. synergistic services #. new funding streams or other sustainability enhancing resources #. technical resources such as storage capacity, bandwidth, and processing power #. compatibility with DataONE services, minimizing cost of deployment
- The candidate adds diversity to the collective membership of DataONE such as through: #. geographic diversity: a new state or region, new country, new continent #. under-represented group #. linguistic and cultural diversity #. different type of institution #. diversity of funding sources
- The candidate offers compliance with best practices and standards, including: #. quality assurance #. data sharing policies #. metadata creation #. security

1.2.13 Member Node Deployment Checklist WORKING DRAFT

Overview

Four distinct phases of activity need to be undertaken to successfully become a Member Node (MN) in the *DataONE* federation of repositories. While these phases are presented linearly for clarity and planning, some steps will often be done in parallel with others.



Planning

Plan the new Member Node.

- **Determine feasibility**

  Member Node representatives review the *Member Node Documentation*, in particular the DataONE Partnership Guidelines and determine if a partnership with DataONE makes sense, and if the organization has the resources required for successfully implementing and operating a MN. Member Nodes can ask DataONE for information or help via the Contact Us page on the DataONE website.

- **Join the DataONE federation**
  - Member Node representatives, with assistance from DataONE personnel, collate MN information (such as high-level descriptions of the data the MN will provide).
  - The MN creates a Proposal. This includes completion of the Member Node Description Document. [we need a detailed process below this]
  - Submits MN Proposal to DataONE for review. See *Member Node Approval Process*.
  - After an agreement has been reached to proceed, the Member Node requests a DataONE identity which grants access to https://docs.dataone.org/ and Redmine (for monitoring/tracking purposes). MN personnel

will also be added to distribution lists (such as the DataONE developers list) and meetings (bi-weekly Member Node Forum, etc.).

- **Scope the implementation**

  The decisions made during this step will drive the details of planning the implementation below.

  – Data: First, the MN should decide how much of and what data they wish to make discoverable via DataONE. Some MNs choose to expose all their data, others only some, and still others expose all their data to a limited audience.

    The MN should also consider the mutability of their data; i.e. is their data static or continuously updated, or a combination of these characteristics.

  – DataONE Functionality: In conjunction with defining the scope of their holdings made visible via DataONE, the MNs also must select-tier

    Member Nodes choose to expose various services, which we have organized into four tiers, starting with the simple read only access (Tier 1) and progressing through more complex services including authentication (Tier 2), write access (Tier 3), and replication (Tier 4). Select the level of functionality that the MN will provide as a partner in the DataONE infrastructure.

  – Member Node Software Stack: Decide if the MN will be fully or partially based on an existing software stack, such as Metacat or GMN, or if a completely custom implementation is required, or if a hybrid approach will be used to adapt an existing DataONE compatible software system to interact with an existing repository system.

- implementation-planning

  After determining the scope of data holdings to be exposed via DataONE and the related questions above, the MN will determine the best approach for the MN implementation.

  1. The MN will need to plan for any needed infrastructure changes at their site.
  2. Data: if not all data holdings will be made discoverable via DataONE, the MN will need to plan/develop a mechanism to identify what data is to be harvested or create a subset of data for DataONE use.

     In any case, each data object will need to be assigned a DOI if not already assigned one "locally".

  1. Functionality: Based on the desired Tier of operations, the MN may need to implement additional [security measures - this isn't the right way to say this].
  2. Software Stack/other development: Depending on resource requirements for any software development (i.e. new/modified software stack), the MN should plan to allocate appropriate (human) resources to the effort.

     Determine if there will be new data formats or new metadata formats which need to be registered.
     An example of this might be [put an example here]. If there is no software stack development or no
     new data/metadata formats to be registered, the Developing phase will be less costly in terms of time
     and resources.

  1. Define a data management plan. If the MN already has an institutional DMP in place, this may be used or modified to reflect interactions with DataONE.
  2. Consider the question of persistent identifiers (related to the mutability of data issue). See Identifiers in DataONE.

Developing

The scope of the developing phase is to build and test a working member node that passes the basic tests in the web-based Member Node Tester. The main things to put in place are the member node itself and any formats that would be new to DataONE.

- **Develop MN Software**

  Unless you are fortunate to already be using Metacat, or don't have an existing data collection, developing the Member Node usually involves writing at least some integration code, and for some organizations, implementing the API methods themselves. At this point in the process you will be simply following your development plan.

  You can iteratively use the web-based Member Node testing service throughout your development process to measure incremental progress.

  – development-testing

- **Register Formats**

  If you are working with a format new to DataONE, it will need to be registered before D1 can successfully synchronize content registered with that format. This is a distinct process that is also set up to run outside of Member Node deployment. If you are registering a new *metadata* format, DataONE developers will need to build, test, and deploy an indexing parser and html renderer to the CNs. Testing these elements is best done in DEV, with the content of the new format originating either from the new member node or by submitting sample content to an existing node in the DEV environment. This decision should be discussed with coredev.

- **Passing MN Tests?**

  DataONE provides a Member Node Testing service to test that the Member Node API is implemented correctly. When all required tests are passing, you are ready to enter the Testing phase, where more thorough testing is done.

  – development-testing

Testing

Once all data formats are registered and your software is fully developed, whether by yourself or by utilizing an existing MN software stack, you can then deploy and configure your node and register it to our Stage environment to allow us to conduct a real-world test in an environment that is identical to the Production environment. The end-point of this phase is a fully functional and integrated Member Node "in production".

- **Test in STAGE**

  STAGE testing allows DataONE to conduct a real-world tests in an environment that is identical to the Production environment. It is the first time that the entire Member Node's content is synchronized, so this is the place where non-systematic content issues are usually revealed. Configuration issues are also identified here, especially related to certificates and user subjects.

  STAGE testing involves the following steps:

  1. Member Node team registers the live Member Node into STAGE environment, using

  – STAGE as the target environment

  – which MN client certs???

  – which CN cert???

  – which CILogon CA??

  2. Member Node Service tests are run against the node to uncover any configuration or content issues.

  3. DataONE operations support approves the node registration and the node begins synchronizing content. DataONE reports back any problems that might arise.

  4. The Member Node team and DataONE jointly reviews the presentation of content in ONEMercury.

- **Deploy in Production Environment**

  After successful testing in the Stage environment, the MN can be deployed and registered in the Production environment (see register-in-production). Registering the MN in the Production environment is the final technical step required for DataONE to approve the node and for it to enter into operational status.

- **Mutual Acceptance**

  After the node is registered in the Production environment, both the node operators and DataONE will do a final review on the node to determine that it is operating as expected. This includes checks for content disparities and other issues that may not be detected by the automated tests. The node description and other metadata are checked for consistency and clarity. When the review is complete, both DataONE and the node operators mutually approve the registration and move the MN into an operational state.

  mutual-acceptance

Operating

Operate the MN in production.

- **Announcement**

  The MN organization announces the new MN and DataONE showcases the MN through channels such as the DataONE newsletter and press releases.

- **Ongoing Production operations**

  The MN is operational and delivers services to the broader research community. Coordinating nodes monitor the MN to ensure that it operates as intended. The node's data and metadata are made available via the various MN and Coordinating MN services. Logs are kept on all services provided, and the Coordinating nodes provide automated access to aggregated statistics back to the MN operators.

- **Participate in MN forums**

  The MN organization participates in MN forums to help monitor and evolve the DataONE federation to meet the research data needs of the community.

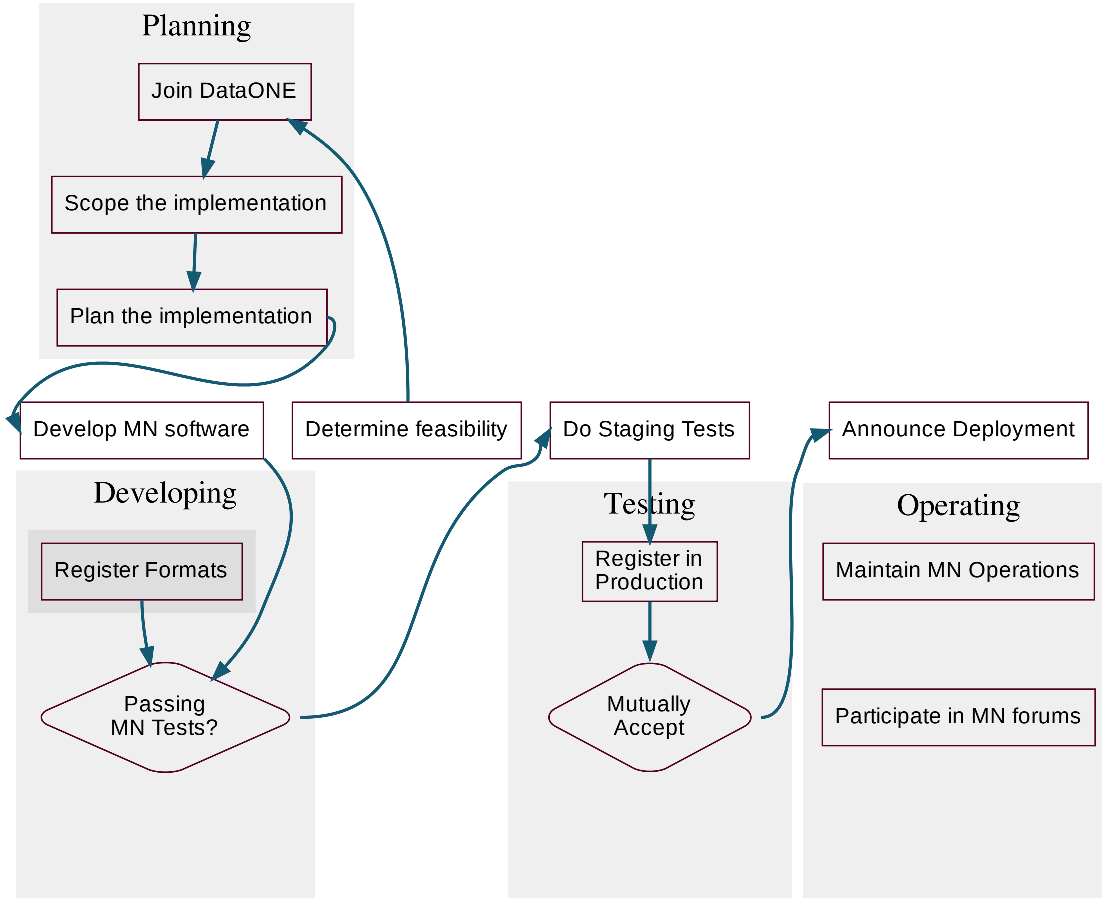
1.2.14 Member Node Deployment Checklist

status: under development

Overview

This document provides details about the process of becoming a DataONE member node. For an overview of the process, see Member Node Deployment Process on the main website.

The process of becoming a DataONE Member Node can be viewed as four phases: *Planning*, *Developing*, *Testing*, and *Operating*. While we present these as a linear process, for clarity and planning, steps are often done in parallel and a given organization may be doing some tasks in a later phase while the bulk of effort is in an earlier phase.



Planning

Plan the new Member Node.

- **Determine feasibility**

  Member Node representatives review the *Member Node Documentation*, in particular the DataONE Partnership Guidelines and determine if a partnership with DataONE makes sense, and if the organization has the resources required for successfully implementing and operating a MN. Member Nodes can ask DataONE for information or help via the Contact Us page on the DataONE website.

- **Join the DataONE federation**
  - Member Node representatives, with assistance from DataONE personnel, collate MN information (such as high-level descriptions of the data the MN will provide).
  - The MN creates a Proposal. This includes completion of the Member Node Description Document. [we need a detailed process below this]
  - Submits MN Proposal to DataONE for review. See *Member Node Approval Process*.
  - After an agreement has been reached to proceed, the Member Node requests a DataONE identity which grants access to https://docs.dataone.org/ and Redmine (for monitoring/tracking purposes). MN personnel

will also be added to distribution lists (such as the DataONE developers list) and meetings (bi-weekly Member Node Forum, etc.).

- **Scope the implementation**

  The decisions made during this step will drive the details of planning the implementation below.

  - Data: First, the MN should decide how much of and what data they wish to make discoverable via DataONE. Some MNs choose to expose all their data, others only some, and still others expose all their data to a limited audience.

    The MN should also consider the mutability of their data; i.e. is their data static or continuously updated, or a combination of these characteristics.

  - DataONE Functionality: In conjunction with defining the scope of their holdings made visible via DataONE, the MNs also must select-tier

    Member Nodes choose to expose various services, which we have organized into four tiers, starting with the simple read only access (Tier 1) and progressing through more complex services including authentication (Tier 2), write access (Tier 3), and replication (Tier 4). Select the level of functionality that the MN will provide as a partner in the DataONE infrastructure.

  - Member Node Software Stack: Decide if the MN will be fully or partially based on an existing software stack, such as Metacat or GMN, or if a completely custom implementation is required, or if a hybrid approach will be used to adapt an existing DataONE compatible software system to interact with an existing repository system.

- implementation-planning

  After determining the scope of data holdings to be exposed via DataONE and the related questions above, the MN will determine the best approach for the MN implementation.

  1. The MN will need to plan for any needed infrastructure changes at their site.
  2. Data: if not all data holdings will be made discoverable via DataONE, the MN will need to plan/develop a mechanism to identify what data is to be harvested or create a subset of data for DataONE use.

     In any case, each data object will need to be assigned a DOI if not already assigned one "locally".

  1. Functionality: Based on the desired Tier of operations, the MN may need to implement additional [security measures - this isn't the right way to say this].
  2. Software Stack/other development: Depending on resource requirements for any software development (i.e. new/modified software stack), the MN should plan to allocate appropriate (human) resources to the effort.

     Determine if there will be new data formats or new metadata formats which need to be registered. An example of this might be [put an example here]. If there is no software stack development or no new data/metadata formats to be registered, the Developing phase will be less costly in terms of time and resources.

  1. Define a data management plan. If the MN already has an institutional DMP in place, this may be used or modified to reflect interactions with DataONE.
  2. Consider the question of persistent identifiers (related to the mutability of data issue). See Identifiers in DataONE.

Developing

The scope of the developing phase is to build and test a working member node that passes the basic tests in the web-based Member Node Tester. The main things to put in place are the member node itself and any formats that would be new to DataONE.

- **Develop MN Software**

  Unless you are fortunate to already be using Metacat, or don't have an existing data collection, developing the Member Node usually involves writing at least some integration code, and for some organizations, implementing the API methods themselves. At this point in the process you will be simply following your development plan.

  You can iteratively use the web-based Member Node testing service throughout your development process to measure incremental progress.

  – development-iteration

  – development-testing

  – test-system

- **Register Formats**

  If you are working with a format new to DataONE, it will need to be registered before D1 can successfully synchronize content registered with that format. This is a distinct process that is also set up to run outside of Member Node deployment. If you are registering a new *metadata* format, DataONE developers will need to build, test, and deploy an indexing parser and html renderer to the CNs. Testing these elements is best done in DEV, with the content of the new format originating either from the new member node or by submitting sample content to an existing node in the DEV environment. This decision should be discussed with coredev.

- **Passing MN Tests?**

  Once the required tests of the Member Node testing service are passing, (see development-testing) the prospective Member Node is ready to enter the Testing phase, where more thorough testing is done.

Testing

Once all data formats are registered and your software is fully developed, whether by yourself or by utilizing an existing MN software stack, you can then deploy and configure your node and register it to our Stage environment to allow us to conduct a real-world test in an environment that is identical to the Production environment. The end-point of this phase is a fully functional and integrated Member Node "in production".

- **Test in STAGE**

  STAGE testing allows DataONE to conduct a real-world tests in an environment that is identical to the Production environment. It is the first time that the entire Member Node's content is synchronized, so this is the place where non-systematic content issues are usually revealed. Configuration issues are also identified here, especially related to certificates and user subjects.

  STAGE testing involves the following steps:

  1. Member Node team registers the live Member Node into STAGE environment (see *Node Registration*)
  2. Member Node Service tests are run against the node to uncover any configuration or content issues.
  3. DataONE operations support approves the node registration and the node begins synchronizing content. DataONE reports back any problems that might arise.
  4. The Member Node team and DataONE jointly reviews the presentation of content in ONEMercury.

- **Deploy in Production Environment**

  After successful testing in the Stage environment, the MN can be deployed and registered in the Production environment (see register-in-production). Registering the MN in the Production environment is the final technical step required for DataONE to approve the node and for it to enter into operational status.

- **Mutual Acceptance**

After the node is registered in the Production environment, both the node operators and DataONE will do a final review on the node to determine that it is operating as expected. This includes checks for content disparities and other issues that may not be detected by the automated tests. The node description and other metadata are checked for consistency and clarity. When the review is complete, both DataONE and the node operators mutually approve the registration and move the MN into an operational state.

mutual-acceptance

Operating

Operate the MN in production.

- **Announcement**

  The MN organization announces the new MN and DataONE showcases the MN through channels such as the DataONE newsletter and press releases.

- **Ongoing Production operations**

  The MN is operational and delivers services to the broader research community. Coordinating nodes monitor the MN to ensure that it operates as intended. The node's data and metadata are made available via the various MN and Coordinating MN services. Logs are kept on all services provided, and the Coordinating nodes provide automated access to aggregated statistics back to the MN operators.

- **Participate in MN forums**

  The MN organization participates in MN forums to help monitor and evolve the DataONE federation to meet the research data needs of the community.

1.2.15 Member Node Documentation

The DataONE website has links to several documents which describe DataONE and questions for data entities to address when they are considering becoming a Member Node. A potential MN should review the following information to help decide if a partnership with DataONE is appropriate.

- The Member Nodes page at DataONE.
- Member Node Fact Sheet
- Member Node Partnership Guidelines
- The Member Node Description Document, which is submitted as part of the proposal process.

Still have questions? Send an email to mninfo.dataone@gmail.com or submit your question via the Contact Us form.

1.2.16 Mutable Content Member Node Development

Document Status

Status	Comment
DRAFT	(Nahf) Initial draft

Purpose

This document is meant as an introduction to the DataONE infrastructure for operators of repositories that manage persistent mutable entities and wish to join DataONE as a Member Node. Member Nodes are required to provide access to identified, immutable entities, so the challenge for these repositories is to meet that requirement. This document will layout the concepts needed to determine the best implementation approach, whether it be internalizing version-level persistence and implementing the DataONE Member Node API, going the Slender Node route, or deploying a pre-build Member Node and publishing selected versions to it.

Terminology

Because DataONE has a different storage model from repositories of mutable content, great care has been taken in this document to avoid ambiguous or overloaded terminology. To that end, some terms are defined up front to avoid confusion.

entity the uniquely identified, persisted electronic item being stored, whether mutable or not, in the MN repository

object a uniquely identified, persisted and immutable representation of the entity, with the semantics of "version".

version synonymous with object, also implying immutability.

immutable functionally, an entity is immutable if a byte-for-byte equivalent representation is retrievable indefinitely, verifiable by checksum comparison.

series the ordered set of objects that arise from a changing entity.

series head the most current object of a series

system metadata (*a.k.a.* sysmeta) the DataONE metadata document that contains all of the administrative properties of the object. There is one sysmeta per object.

Synopsis

DataONE manages the synchronization and replication of persisted, immutable versions of entities it calls objects. For repositories that manage mutable entities, the main challenge is how to reflect these mutable entities as immutable objects. As of version 2, DataONE supports this through recognition of an additional identifier field for the mutable entity, as long as a unique identifier for the version is also still provided.

With these two identifiers, DataONE can see Member Nodes that manage mutable content as Member Nodes that host only the latest version of an entity. Over time, it will synchronize and replicate a significant portion of those versions as immutable objects, and be able to resolve the entity identifier to the object representing the most current version of that series.

This approach can lead to complexities that increase with the volatility of the entity and removal of past-version system metadata records. In the case of highly volatile entities, the main issue is the reliability of the current object getting persisted as an object in DataONE, and being useful for data consumers. In the case of removal of past-version system metadata, the main issue is object orphaning (the inability to further maintain the synchronized objects).

MN API implementers will also need to be able to reflect their data management actions into actions understood by DataONE. The final section will be devoted to recipes for a wide range of potential data management actions.

Important Concepts

Determining Content Immutability

The MNRead API is primarily an API for versions, with layered semantics that allow the retrieval of the head of a series. When presented an object identifier, MN Read API methods MUST return the byte array originally persisted

as that object. DataONE ensures this by periodically validating the checksum of the object, and flagging any copy of an object it finds invalid.

Even if persisted faithfully, repository owners should be aware of any post-processing that might occur during object retrieval, either within the application, or the web server container, and take steps to eliminate them, as they are potential sources of mutability, especially with software component upgrades.

Synchronization and Replication

Through synchronization, DataONE registers new objects to its collection, and for objects of type METADATA and RESOURCE_MAP, replicates the objects to the CN for downstream index processing. Synchronization also triggers Member Node replication of DATA objects, which is a process that sends requests to volunteering Member Nodes to preserve the object bytes. Under typical operating conditions, this process can take from 3 to 20 minutes. Under periods of routine maintenance, these services can be down for 3 - 24 hours. Other variables that can impact this time is the amount of backlog synchronization has to work through, the Member Node's synchronization schedule, and network problems.

For Member Nodes of mutable content, the main concern is whether the version of the entity remains available between the time it is queued for synchronization, and when it is retrieved.

Update Location

The update location for each object is the authoritative Member Node listed in an object's system metadata. The update location is the only Node where changes to an entity (reflected as creation of a related object) or an object's system metadata can take place. This is done both to avoid race conditions (changes coming from two locations on the system at the same time), and to prevent other users from accidentally or intentionally registering objects on other Member Nodes and effectively hijacking control of an entity.

Updated entities and system metadata from the authoritative Member Node get synchronized to the DataONE CNs, and replicated out to MNs listed as replica nodes for that object. Attempted updates originating from any other node are rejected.

If the listed authoritative Member Node of an object no longer has that system metadata, it is said to have been orphaned, making it difficult to update system metadata properties without administrative intervention.

DataONE update logic

Since DataONE persists only immutable objects, an update to an entity is accomplished by creating a new object that references its predecessor through the 'obsoletes' property in the system metadata, as well as associating the entity identifier. The obsoletes reference is vital for MNs that only expose immutable objects, and the association of the entity identifier vital for MNs that manage mutable entities and don't maintain system metadata for previous versions exposed as objects.

Preferred Implementation

The two main problems to be overcome by Member Nodes of mutable content are decreased retrieval reliability and object orphaning.

Of the two, the least burdensome of the two to address is object orphaning. The idea is to keep system metadata for all exposed versions, regardless of whether or not the object corresponding to the version is still retrievable. Doing so ensures Member Node local control over all synchronized versions, and avoids orphaning the object. It also provides DataONE with more reliable provenance to allow better resolution of the entity identifier to the latest object available.

This can be accomplished either by following the slender node design (GMN configured to use the Remote URL mode), or building system metadata storage into the custom implementation.

The second problem, decreased retrieval reliability, is directly related to the ability of DataONE to successfully synchronize and replicate each version as an object. The unavoidable synchronization lag is the main factor to this problem, as is the volatility of the entity. Highly volatile or regularly changing entities will frustrate synchronization attempts, as will DATA entities that disallow replication.

A worst case scenario is an end-user calling cn.resolve using the entity identifier which points back to the latest known version on the Member Node, but that returns a NotFound because the newer version is still waiting to be synchronized, and the older version is no longer hosted. In this way, a published object can temporarily be NotFound on the DataONE systems.

Ideally, a local cache of previous versions is maintained to compensate for the worst of synchronization lags, on the order of 24 hours to 3 days.

Identifying versions as immutable objects

Assuming that the entities to be exposed to DataONE can reasonably be expressed as a series of immutable objects, the task is to properly identify and describe each version. The rules for this are straightforward:

1. Each version MUST be assigned an object identifier that is globally unique and IS NOT the entity identifier. This identifier is mapped to the *identifier* property.
2. Each version MUST have the entity identifier associated to it via the *seriesId* property.
3. The object identifier MUST be able to retrieve the same byte array as was determined when the version was identified, for as long as that object is available on the Member Node.

The object identifier is the identifier used within DataONE to facilitate synchronization, and indexing. The entity identifier as *seriesId* will be used to logically relate all objects of the series, and allow resolution to the latest version via the DataONE search interface.

Any time a change is made to the mutable entity that would result in a different byte array being returned from the MNRead.get(id) call, the Member Node MUST identify another version and create a new system metadata document for it.

Complete requirements for handling changes are found in the update section below.

identifier <versionIdentifier>

seriesId <entityIdentifier>

It is worth mentioning that version identifiers cannot be reused for other content even after they are no longer in use on the Member Node (see later sections on dealing with multiple versions).

Identifier Resolution in READ APIs

DataONE relies on Member Nodes to provide faithful access to the specific versions of entities it hosts. Specifically:

- MNRead.get(persistent identifier) MUST return either the specified version, or NotFound if it no longer has it.
- MNRead.get(persistent identifier) MUST NOT return any other version!
- MNRead.get(series identifier) MUST return the object representing the latest hosted version, or NotFound if not recognized.
- ======> are there any other situations (deletes) to consider?

- MNRead.getSystemMetadata(persistent identifier) MUST return the system metadata of the specified version, or NotFound if it not longer has the system metadata for that version. It SHOULD return the system metadata even if it doesn't have the object bytes anymore.

CNRead.resolve logic

CNRead.resolve is a centralized method to direct users to the location of the object requested, and is the primary method for data consumers to access discovered objects. It does this in two steps:

1. if given an identifier that is a SID, determine the PID that represents the most current version.
2. return the locations of that current PID and provide an HTTP redirect of the request to the MNRead.get(currentPID) to one of those locations.

- ==== note to self =====> in the redirect URL, resolve should prefer replica nodes to the authoritativeMN, at least for Mutable Member Nodes, because it lessens the chance of NotFounds.
- ==== note to self =====> a good reason to track the data storage model of MNs in the (CN controlled section of) Node Properties.

Determining the current PID

- DataONE uses the obsoletes , obsoletedBy, and dateUploaded properties to determine the current version of an entity.
- As explicit indications of ordering, obsoletes and obsoletedBy information takes priority
- dateUploaded is used to if there are two or more versions that are not obsoleted
- mutable member nodes are more prone to missing versions and obsoletedBy information, so rely heavily on the accuracy of the dateUploaded field.

Entity Creation

When a new entity is created in a Mutable Member Node, a corresponding system metadata document must also be created containing all of the required administrative properties of that object. This includes:

System Metadata:

identifier a version Identifier

seriesId the entity Identifier

checksum the checksum of the version

size byte-length of the version

dateUploaded the version creation date

obsoletedBy null

obsoletes null, typically

Renaming Entities

Once registered, identifiers cannot be disassociated from the object originally assigned. Therefore, renaming an entity is achieved by creating a new PID (version identifier) and SID (the new entity identifier). While this causes a duplicate

object, the duplication is acceptable. Entity renaming follows the semantics of updates, except the checksum and size will be inherited from the previous object.

Previous System Metadata for A1:

identifier A1

seriesId S1

dateUploaded t1

checksum C1

size Z1

New System Metadata for A2:

identifier A2

seriesId S2

dateUploaded t2 (where t2 > t1) <====== question: can t2 = t1?) ========

checksum C1

size Z1

obsoletes A1

New System Metadata for A1 [if system metadata for back versions is maintained]:

identifier A1

seriesId S1

dateUploaded t1

checksum C1

size Z1

obsoletedBy A2

Entity Updates

An update to an entity constitutes the creation of a new version (object) related to the previous version it replaces. If the new version overwrites the previous version without preserving the old object, the previous version is de facto orphaned, and DataONE consumers will need to rely on any replicas created within the DatAONE network.

Previous System Metadata for A1:

identifier A1

seriesId S1

dateUploaded t1

checksum C1

size Z1

dateSystemMetadataModified t2

New System Metadata for A2:

identifier A2

seriesId S2

dateUploaded t3 (where t3 > t1)

checksum C2

size Z2

obsoletes A1

dateSystemMetadataModified t4

New System Metadata for A1 [if system metdata for back versions is maintained]:

identifier A1

seriesId S1

dateUploaded t1

checksum C1

size Z1

obsoletedBy A2

dateSystemMetadataModified t4

Entity Archiving

If the repository supports archiving, it can be reflected in the entity's system metadata, by setting the archived property to true. The system metadata MUST remain available.

System Metadata:

archived true

dateSystemMetadataModified current date-time

This will be processed by DataONE as a system metadata update. Archiving an object in DataONE removes it from the DataONE solr search index, but leaves it available for retrieval through READ APIs. The main use case is to limit discovery of outdated content, without making the object unavailable to users already relying on the content for ongoing analyses, or retrieval by identifier found in a publication.

Entity Unarchiving

Once the archived property is set to true, it cannot be set to false or null. <======== Question: why? =========

Therefore, to unarchive an entity, you must create a duplicate version with the same seriesId.

System Metadata:

identifier a new version identifier

seriesId same seriesId

dateUploaded the version creation date

dateSystemMetadataModified the version creation date.

Entity Deletion

Simple removal of an entity is mapped to a DataONE archive action, that is, setting the archived flag to true (see entity archiving). Member Nodes MUST at a minimum keep the system metadata of the latest verison available for retrieval.

Entity Deletion for Legal Reasons

DataONE supports deletion of content (removal of replicas across all Member and Coordinating Nodes) for legal take-downs of inappropriate content or over-replication.

All of these deletions are coordinated centrally by DataONE, and all Member Nodes should contact DataONE administrators to plan such events.

Entity Reversion

When an entity reverts back to the previous version, DataONE provides no mechanism to rearrange versions, and Member Node administrators SHOULD NOT try to retro-fix system metadata properties to otherwise fool resolution services. Instead a new version identifier should be generated for the version along with a new dateUploaded later than the widthdrawn version.

This will result in possible duplication of content in the DataONE system, but this is an acceptable outcome.

System Metadata for Av1:

identifier A1

seriesId S1

dateUploaded t1

checksum C1

size Z1

System Metadata for Av2:

identifier A2

seriesId S1

dateUploaded t2

obsoletes A1

checksum C2

size Z2

System Metadata for Av3:

identifier A3

seriesId S1

dateUploaded t3

obsoletes A2

checksum C1

size Z1

Question What is the impact of duplicate objects on MN reporting?

Cruft

Due to the distributed nature of responsibility, and infrastructure, we require an explicit hand off of this role through an update to the system metadata.

(immutability allows asynchrony and eventual consistency)

reflecting mutable entities as immutable objects

- read
- resolve
- create
- update
- delete
- archive
- collection management / inventory

DataONE manages the synchronization and replication of persisted, immutable versions of entities it calls objects. Repositories that can identify and expose specific versions of their stored content as objects through DataONE Member Node APIs can with modest effort participate as Member Nodes of DataONE - even those whose entities are mutable.

The purpose of this article is to provide a concise guide for owners of repositories of mutable content who wish to implement the Member Node API in their application server. Before embarking on development iterations, hopeful developers should carefully consider the cost-benefit of this approach versus other deployment options using tested DataONE Member Node packages (described elsewhere).

By implementing the Member Node APIs, developers will need to handle more complexity, including:

1. multi-reader locking
2. implementing transactions so changes to entities are always accompanied by change to the system metadata.
3. implementing any needed change controls to satisfy object immutability requirement.
4. minimizing post-processing steps in the retrieval methods. Even standard libraries change behavior over time, or with reconfiguration. (For example, XML formatters)
5. maintaining a storage space for DataONE SystemMetadata
6. maintaining a storage space for Resource Maps.

For systems not built for it, byte-level consistency for retrievals is a difficult thing to add after the fact. Whether it be the need to assemble the object or apply post-processing upon retrieval, . . .

1.2.17 Node Registration / Update Script

Document Status

Status	Comment
DRAFT	(CSJ) Initial draft
REVIEWED	(RN) clarified the dependency on xmlstarlet

Node Registration Script

Member Node operators will often use the CN API calls within languages such as Java and Python to register a node and update it's capabilities. However, this can also be done using a simple bash script that depends on curl and optionally xmlstarlet (for friendlier display of responses) being installed on your workstation.

DataONE maintains a number of bash scripts in the d1_client_bash project, and the d1noderegister script is one of those scripts.

To use this script to register a Member Node in a DataONE environment, copy the script into a file called *d1noderegister*, and ensure that the file is executable. On Mac and Linux (Windows via Cygwin), this can be done with the following commands:

```
$ curl -k -o d1noderegister \
 "https://repository.dataone.org/software/cicore/trunk/itk/d1_client_bash/
↪d1noderegister"
$ chmod +x ./d1noderegister
```

To see the options for the command, use the *-h flag*:

```
$ ./d1noderegister -h
```

An example of simple usage would be:

```
$ ./d1noderegister -f node.xml -b https://cn-dev.test.dataone.org/cn -E client.pem
```

The above command would register the MN described by the node.xml document with the DataONE development environment, using the concatenated client SSL certificate and key issued to the MN by DataONE. See the example node document that shows some typical Tier 1 Member Node values, using the USGSCSAS node values.

Update Node Capabilities Script

Once a node has been registered in an environment, there are times during ongoing operations that the Node information needs to be updated. For instance, for scheduled system maintenance, a Member Node may be unavailable for a short time, and the Node.state should be set to *down* to reflect the temporary outage.

To update a node using bash simple script, use the d1nodeupdate utility.

Copy the script into a file called *d1nodeupdate*, and ensure that the file is executable. On Mac and Linux (Windows via Cygwin), this can be done with the following commands:

```
$ curl -k -o d1nodeupdate \
 "https://repository.dataone.org/software/cicore/trunk/itk/d1_client_bash/d1nodeupdate
↪"
$ chmod +x ./d1nodeupdate
```

To see the options for the command, use the *-h flag*:

```
$ ./d1nodeupdate -h
```

An example of simple usage would be:

```
$ ./d1nodeupdate -f node.xml -b https://cn-dev.test.dataone.org/cn -E client.pem
```

The above command would update the MN described by the node.xml document with the DataONE development environment, using the concatenated client SSL certificate and key issued to the MN by DataONE.

1.2.18 Operational Acceptance

Todo: Need to define what is required for operational acceptance.

1.2.19 Production Registration

Document Status

Status	Comment
DRAFT	(MBJ) Initial draft

Production registration involves deploying and registering the production Member Node that is to be operated as part of DataONE. Registering the production node is the final technical step required for DataONE to approve the node and for it to enter into operational status.

To register the node, you need to have a DataONE issued client certificate which identifies the Member Node to the Coordinating Node, and then you use that certificate to register the node with DataONE. Specifically:

a. Obtain a production certificate for the Member Node
    - Request from 'support@dataone.org'
b. Install the production certificate on the production implementation of the node
c. Call the CN.register() service to register the node
d. Send an email to 'support@dataone.org' to request node approval and transition into operational status

1.2.20 Node Registration

Contents

- *Node Registration*
    - *Register the Administrative Contact identity*
    - *Create the Node document*
    - *Obtain a client side certificate*
    - *Register the MN*
    - *DataONE evaluates the submission*

A Member Node (MN) becomes part of DataONE through Node Registration. Registering the MN allows the Coordinating Nodes (CNs) to synchronize content, index metadata documents and resource maps, and replicate its content to other MNs. Before registering the MN into the production DataONE environment, it is registered into the Staging environment for testing.

Follow the following steps for each environment you register your node to:

1. Register the DataONE identity of the administrative contact for the new MN.
2. Create a Node document that includes the administrative contact.
3. Obtain a client certificate from DataONE for the MN.

4. Submit the Node document to DataONE using the client certificate.
5. Notify your primary DataONE contact of the registration request

DataONE then evaluates the submission. Upon approval, the registration is complete, and the MN is part of the environment in which it was registered.

Register the Administrative Contact identity

This step must be performed by the person who will be the contact for the new MN. The contact person is often also the administrator for the MN.

Each DataONE environment has a web-based Identity Manager where DataONE identities are created and maintained. To create a DataONE identity, you will use the Identity Manager to authenticate with a *CILogon*-recognized identity, and then attach your name and contact email. At this point, DataONE will validate this information manually.

To register the administrative contact's DataONE identity in the target environment, perform the following steps:

1. Navigate to the Identity Manager of the target environment:

Environment	Identity Manager URL
Production	https://cn.dataone.org/portal
Staging	https://cn-stage.test.dataone.org/portal
Sandbox	https://cn-sandbox.test.dataone.org/portal
Development	https://cn-dev.test.dataone.org/portal

2. Follow the prompts to authenticate against your *Identity Provider*. If your institution is not listed, you can use a Google or ProtectNetwork account.
3. Once authenticated and back at the DataONE portal, supply your name and email, and then click on the **Register** or **Update** button (only one will be present).
4. Record (copy to clipboard) the identity string shown in the 'Logged in as' field. This value is taken from the CILogon certificate issued when you authenticated against your chosen *Identity Provider*, and is also a DataONE subject.
5. Paste this value into the contactSubject field of the Node document you plan to submit in the next step.
6. DataONE requires that DataONE subjects that are to be used as contacts for MNs be verified. To verify the account, send an email to support@dataone.org (launch). In the email, include the identity string obtained in the step above and request that the account be verified. You do not need to wait for a reply to continue to the next step.

Create the Node document

The Node document is a set of values that describe a MN or CN, its internet location, and the services it supports.

The values in the Node document are described in the Node document section in the architecture documentation.

What follows is an overview of only the required values in the Node document. Also see the example node document.

- **identifier**: A unique identifier for the node of the form urn:node:NODEID where NODEID is the node specific identifier. This value MUST NOT change for future implementations of the same node, whereas the baseURL may change in the future.

  NODEID is typically a short name or acronym. As the identifier must be unique, coordinate with your DataONE developer contact to establish your test and production identifiers. The conventions for these are `urn:node:mnTestNODEID` for the development, sandbox and staging environments and `urn:node:NODEID` for the production environment. For reference, see the list of current DataONE Nodes.

E.g.: urn:node:USGSCSAS (for production) and urn:node:TestUSGSCSAS (for testing).

- **name**: A human readable name of the Node. This name can be used as a label in many systems to represent the node, and thus should be short, but understandable.

  E.g.: USGS Core Sciences Clearinghouse

- **description**: Description of a Node, explaining the community it serves and other relevant information about the node, such as what content is maintained by this node and any other free style notes.

  E.g.: US Geological Survey Core Science Metadata Clearinghouse archives metadata records describing datasets largely focused on wildlife biology, ecology, environmental science, temperature, geospatial data layers documenting land cover and stewardship (ownership and management), and more.

- **baseURL**: The base URL of the node, indicating the protocol, fully qualified domain name, and path to the implementing service, excluding the version of the API.

  E.g.: https://server.example.edu/app/d1/mn

- **contactSubject**: The appropriate person or group to contact regarding the disposition, management, and status of this Member Node. The contactSubject is an X.509 Distinguished Name for a person or group that can be used to look up current contact details (e.g., name, email address) for the contact in the DataONE Identity service. DataONE uses the contactSubject to provide notices of interest to DataONE nodes, including information such as policy changes, maintenance updates, node outage notifications, among other information useful for administering a node. Each node that is registered with DataONE must provide at least one contactSubject that has been verified with DataONE.

  The contactSubject must be the subject of the DataONE identity that was created in the *previous step*.

  E.g.: 'CN=My Name,O=Google,C=US,DC=cilogon,DC=org'

- **replicate**: Set to true if the node is willing to be a *replication target*, otherwise false.

  This setting is ignored if the Tier of the node is lower than 4. It is intended for temporarily disabling replication. For permanently disabling replication, set TIER lower than 4 as well as this setting to False.

- **synchronize**: Set to true if the node should be synchronized by a CN, otherwise false.

- **type**: The type of the node. For MNs, set to `mn`.

- **state**: The state of the node.

  Set to 'up' when the Node is running correctly. Set to 'down' when it is down for maintenance or an unscheduled outage. See the documentation on the CNRegister.updateNodeCapabilities() API call for details.

Obtain a client side certificate

DataONE will create and issue your node an *X.509* certificate issued by the DataONE *CA*. This *client side certificate* is to be used when the MN initiates REST API calls to CNs and other MNs. Certificates issued by DataONE are long-lasting *X.509* certificates linked to a specific MN via its *DN*.

Tier 1 MNs using http for MN API calls will likely only need this certificate when administering their node using the CNRegister API calls, which may be done from any client machine. Nevertheless, it is advisable to store this certificate on the Member Node server.

To obtain a client side certificate:

1. create an account on the DataONE Registration page,

2. notify DataONE by sending an email to support@dataone.org. In the email, state that you are requesting a client side certificate for a new MN and include the MN identifier, in the form `urn:node:NODEID`, *selected previously*.

DataONE will create the certificate for you and notify you of its creation with reply to your email. At this point:

1. follow the link provided in the email, and sign in using the account created or used in the first step, above.

You will initially receive a certificate that is valid for any and all of the test environments. When the new MN is ready to go into production, you will receive a production certificate.

> **Warning:** Anyone who has the private key can act as your Node in the DataONE infrastructure. Keep the private key safe. If your private key becomes compromised, please inform DataONE so that the certificate can be revoked and a new one generated.

Register the MN

The MN registration procedure is designed to be automatable in MN implementations and is automated in *Metacat* and *GMN*. For MN implementations that do not automate this step, DataONE also provides a simple script to perform this step. Registering the MN is done by calling the CNRegister.register() API. The call essentially submits the previously created Node document to DataONE in a HTTP POST request over a TLS/SSL connection that has been authenticated with the previously obtained certificate.

After running the script or running an automated registration, the Member Node should email support@dataone.org to notify of the registration request, and notify their primary technical contact at DataONE.

DataONE evaluates the submission

DataONE evaluates the submitted Node document and contacts the person listed as *contactSubject* in the Node document by email with the outcome of the approval process. After the node has been approved, the MN is part of the infrastructure environment in which it has been registered, and the CNs in that environment will start processing the information on the node.

1.2.21 Select the DataONE Tier

While Member Nodes share the same baseline-implementation-requirements of managing their object collections, selecting a Tier level for implementation is a matter of deciding how much of the DataONE Member Node API your organization is ready to implement. DataONE has defined several tiers, each of which designates a certain level of functionality exposed through the MN API. The tiers enable MN developers to implement only the methods for the level at which they wish to participate in the DataONE infrastructure.

Each tier implicitly includes all lower numbered tiers. For instance, a Tier 3 MN must implement tiers 1, 2 and 3 methods.

The tiers range from a simple, public, read only MN, to a fully implemented, writable MN that enables replica storage for other MNs.

The DataONE MN APIs are defined in groups that correspond to the tiers.

The tiers are as follows:

- **Tier 1**: Read, public objects (MNCore and MNRead APIs)

  Provides read-only access to publicly available objects (Science Data, science metadata, and Resource Maps), along with core system API calls for monitoring and logging.

- **Tier 2**: Access control (MNAuthentication API)

  Allows the access to objects to be controlled via access control list (ACL) based authorization and certificate-based authentication.

- **Tier 3**: Write (MNStorage API)

  Provides write access (create, update and archive objects).

  Allows using DataONE interfaces to create and maintain objects on the MN.

- **Tier 4**: Replication target (MNReplication API)

  Allows the DataONE infrastructure to use available storage space on the MN for storing copies of objects that originate on other MNs, based on the Node Replication Policy.

Note: Support for the Node Replication Policy used by Tier 4 MNs is currently under development on the Coordinating Nodes and a Tier 4 MN may currently receive requests to replicate content which should be blocked by its Node Replication Policy.

Note: MN developers may choose to implement the tiers in separate phases. For instance, a MN can initially be implemented and deployed as a Tier 1 MN and operated as such while higher tiers are implemented.

Despite the Tier level, content maintained (created, updated, archived) outside of the DataONE API (but represented through it) is expected to conform to DataONE requirements.

1.2.22 Establish Testing Environment

Document Status

Status	Comment
DRAFT	(MBJ) Initial draft

A testing environment consists of a test Member Node (MN) installation on either a physical or virtual machine. This MN software can optionally be registered in a DataONE testing environment in order to emulate the real-world production scenario. **In the early phases of development, registering a Member Node in a dataONE environment isn't required**. Rather, the MN operator can use the Member Node Check service to test their implementation. This testing service will make calls to all of the expected API endpoints on the MN for each Tier, and will report on success, failures, or warnings. This service can check against multiple versions of the API, so operators should choose tests for the version they are targeting (likely the most recent).

DataONE provides multiple deployment environments that emulate the production environment. An 'environment', per se, is a set of Coordinating Nodes (CNs) communicating with each other, and a set of Member Nodes (MNs) that are registered with those Coordinating Nodes. They communicate with the CNs, and potentially with other MNs if replication is enabled.

Once a strategy has been chosen for a technical implementation, Member Node operators should establish a test server environment (either physical or virtual machines), preferably separate from the server or servers to be used in the production environment. For those choosing existing software stacks, this means installing the operating system, all of the prerequisite software, and then installing the DataONE-enabled repository software itself. Our reference implementations have been built using Ubuntu Linux 10.04 LTS, and we will be migrating to Ubuntu 12.04 LTS. With a test server established, operators can make use of one of three development environments used for registering and testing the Member Node functionality. These are:

1) The Dev environment: Contains up to 3 Coordinating Nodes and multiple internal testing Mebmer Node deployments. This is the most unstable of all of the environments, and is good for early Member Node development work.

2) The Sandbox environment: Contains up to 3 Coordinating Nodes and multiple Member Node deployments. It is used for testing new features in a more stable environment than Dev.

3) The Staging environment: Contains up to 3 Coordinating Nodes and a multiple Member Node deployments from partner institutions. The deployed software is identical to the production environment, but still contains test content.

There are a couple prerequisites to register your test Member Node with one of these environments:

1) **Sign into the environment as the Member Node operator**. Only 'verified' accounts are able to register nodes with the Coordinating Nodes, and the first step is to sign in and register your name and email. This can be done in each environment (including production), by visiting the /portal endpoint for the environment. For instance, in the staging environment, visit https://cn-stage.test.dataone.org/portal and begin the login form. This will redirect you to the CILogon service, which prompts you to authenticate against your 'Identity Provider'. If your institution isn't listed, you can use a Google or ProtectNetwork account. Once authenticated, you will be returned to the DataONE portal to fill in your name and email. Once registered, you will see the Distinguished Name assigned to you from the CILogon service. This DN is used in the contactSubject field of your Member Node document used to register your Node. At this point you can contact the DataONE developer you're working with, or support@dataone.org, to have the verified flag set on your account.

2) **Obtain DataONE client SSL certificates**. Cient-side SSL certificates are used to identify users and nodes. Nodes are issued a long-lasting X.509 certificate with the Node.subject embedded in it. Send an email to your DataONE developer contact, or to support@dataone.org, to request your certificates. Tier 1 Member Nodes will only need the certificate when calling the CN during MNRegister calls, and other select calls like CNReplication.isNodeAuthorized(). Tier 2 and higher nodes will use this certificate identity during all of the service calls.

3) **Construct a Node document to be registered**. To register your node in a test environment, you POST an XML Node document to the /node REST endpoint. The field values in this document affect both programmatic Node communication, as well as graphical interfaces used by scientists to understand what your Member Node represents. Some details are:

- *identifier*: The node identifier matches the pattern `urn:node:XXXXX` where `XXXXX` is a short string representing your repository. Coordinate with your DataONE developer contact on establishing your test identifier and your production identifier. As an example, the Dryad Data Repository uses a test identifier of `urn:node:mnTestDRYAD`, and a production identifier of `urn:node:DRYAD`.
- *name*: The node name is visible in graphical interfaces, so should be short, but should represent the repository. A good example is the `USGS Core Sciences Clearinghouse`. It corresponds with an identifier of `urn:node:USGSCSAS`, and a description of `US Geological Survey Core Science Metadata Clearinghouse archives metadata records describing datasets largely focused on wildlife biology, ecology, environmental science, temperature, geospatial data layers documenting land cover and stewardship (ownership and management), and more.`
- *description*: The node description gives more details than the name. See above.
- *subject*: The node subject is the Distinguished Name that represents the Node. This subject must match the Subject listed in the DataONE client SSL certificate used for authentication and authorization of all Node-level operations. This DN follows the pattern 'CN=urn:node:XXXXX,DC=dataone,DC=org', where the common name portion of the DN matches the identifier of the node.
- *state*: Member Nodes should keep their registered Node document up to date in the production environment. For example, if the Node is running correctly, the state is 'up', but when it is down for maintenance or an unscheduled outage, the state should be set to 'down'. See the documentation on the CNregister.updateNodeCapabilities() API call for details.

There are other required and optional fields in the Node document, but the list above mentions fields that are particularly notable.

4) **Register the Node**: Finally, POST the XML Node document to the /node REST endpoint for the given environment. See the details in the CNRegister.register() API call. You must send your DataONE client SSL certificate with the request for authentication.

1.2.23 Establish a test system

Document Status

Status	Comment
DRAFT	(MBJ) Initial draft
DRAFT	(Dahl)

Once a strategy has been chosen for a technical implementation, Member Node developers should establish a test server. A test server is a physical or virtual machine that runs the MN software that is under development and that can be reached from the Internet.

For those choosing existing software stacks, this means installing the operating system, all of the prerequisite software, and then installing the DataONE-enabled repository software itself. Our reference implementations have been built using Ubuntu Linux 10.04 LTS, and we will be migrating to Ubuntu 12.04 LTS.

In the early phases of development, when the new MN is not yet ready for general testing, MN developers will typically need to perform basic compliance testing of specific REST APIs that they are currently implementing. For this purpose, DataONE provides the Member Node Check service. The service performs a series of calls to the REST API endpoints that a MN of a given tier must support. It then analyses the responses from the MN and provides a compliance report for each of the APIs. The Member Node Check service can run tests against any server that can be reached from the Internet.

1.2.24 Other Documents and Resources

- Member Node issue tracker
- Redmine Wiki
- GitHub MN info tools
- GitHub MN ticket tools
- Past MN deployment docs

1.3 Coordinating Nodes

Figure 1. Typical setup of Coordinating Nodes in an environment. The primary CN is the CN with the environment DNS entry pointing to it. The primary CN must have `d1-processing` running on it. The indexing CN is the CN that has both `d1-index-task-generator` and `d1-index-task-processor` running on it. The indexing CN may also be the primary CN. All CNs in an environment have `ldap`, `postgres`, `apache`, `zookeeper`, `solr`, and `tomcat7` running on them.

1.3.1 Enter / Exit Read Only Mode

Putting a DataONE envrionment in read only mode is achieved by turning off the `d1-processing` service which should be running only on the primary CN.

Synopsis

To shutdown `d1-processing`:

1. Set the service control properties to `FALSE` on the primary CN
2. Check that batch processing has completed
3. Shutdown the `da-processing` service

To startup `d1-processing`:

1. Set the service control properties to `TRUE`
2. Start the `d1-processing` service on the primary CN

Entering read only mode

`d1-processing` is responsible for synchronization, replication, and log aggregation. Each of these processes involves long-running batches of work where uncontrolled disruption of the batch may leave work in an incomplete state. `d1-processing` monitors service control properties and will initiate orderly shutdown of services when the value of the property changes to `FALSE`. Hence, `d1-processing` should always be shut down in a controlled manner by toggling the value of the service control properties.

`d1-processing` monitors three property files, one for each of synchronization, replication, and log aggregation. The property files are located under `/etc/dataone/process`.

Process	Property File	Service Control Property
Synchronization	`synchronization.properties`	`Synchronization.active`
Replication	`replication.properties`	`Replication.active`
Log Aggregation	`logAggregation.properties`	`LogAggregator.active`

In each case, the valid values for the service control property are `TRUE` or `FALSE`, with `FALSE` indicating the service should shut itself down if running and not start when the `d1-processing` service starts up.

The value of the property can be set by directly editing the properties file or through a utility that can toggle the values. On the CNs, the script `/usr/local/bin/d1processingstate` will report and set the value of the service control property for each of the three services:

```
$ d1processingstate
Synchronization.active=TRUE
Replication.active=TRUE
LogAggregator.active=TRUE

$ sudo d1processingstate FALSE
Previous:  Synchronization.active=TRUE
New:       Synchronization.active=FALSE
Previous:  Replication.active=TRUE
New:       Replication.active=FALSE
Previous:  LogAggregator.active=TRUE
New:       LogAggregator.active=FALSE
```

A fabric script to toggle service values is also available for remotely setting the service control property values:

```
d1cnprocessingstate -S FALSE -H cn-orc-1.dataone.org
```

After setting the service control properties to FALSE, it may take some time for services to shutdown before the `d1-processing` service can be shutdown.

The state of the services can be determined by watching the respective service logs.

Process	Log File
Synchronization	/var/log/dataone/synchronize/cn-synchronization.log
Replication	/var/log/dataone/replicate/cn-replication.log
Log Aggregation	/var/log/dataone/logAggregate/cn-aggregation.log

The cn-synchronization.log will emit a message like:

```
[ WARN] 2018-07-02 11:51:10,000 [SynchronizationQuartzScheduler_Worker-21] 
↪(MemberNodeHarvestJob:execute:75) null-  ObjectListHarvestTask Disabled
[ WARN] 2018-07-02 11:51:47,982 [SynchronizationQuartzScheduler_Worker-20] 
↪(SyncMetricLogJob:execute:50) SyncMetricLogJob Disabled
```

When satisfied that d1-processing activity has completed, the service may be stopped:

```
sudo service d1-processing stop
```

Exiting read only mode

Exiting read only mode requires ensuring that the service control properties are set to TRUE then starting the d1-processing service:

```
$ sudo d1processingstate TRUE
Previous:  Synchronization.active=FALSE
New:       Synchronization.active=TRUE
Previous:  Replication.active=FALSE
New:       Replication.active=TRUE
Previous:  LogAggregator.active=FALSE
New:       LogAggregator.active=TRUE

$ sudo service d1-processing start
```

1.3.2 Restarting Services

d1-processing

d1-index-task-generator

d1-index-task-processor

tomcat7

solr and zookeeper

apache

postgres

slapd

1.3.3 System Restart

A kernel upgrade on a Coordinating Node requires a system restart. Restarting a CN requires that the environment is placed in *read-only mode* to help avoid content inconsistency between the CNs operating within an *Environment*. Restarting the *Primary CN* in an Environment requires that the *Environment DNS* be switched to another CN so the read only service (resolution, retrieval, and search) remain available to clients. The DNS switch is required in the production environment and is optional in the various test environments.

Figure 1. Typical setup of Coordinating Nodes in an environment. When restarting *CN-1* it is necessary to change the Environment DNS to point to one of the other nodes so that operations such resolve and external applications (e.g. Search UI) continue to function.

Procedure

1. Broadcast notification
2. Set *read-only mode*
3. Update non-primary nodes (*Indexing CN* and *Third CN*), avoiding an update of DataONE packages:

```
#optional hold on DataONE packages
sudo apt-mark hold dataone*
sudo apt-get update
sudo apt-get dist-upgrade
#when ready, restart the server
sudo reboot
```

```
#undo hold on DataONE packages
sudo apt-mark unhold dataone*
# verify new kernel running
uname -r
```

4. Switch DNS to a non-primary node. For example, switch the environment DNS entry to point to the *Indexing CN*.
5. Update the remaining node. As for #4.
6. Switch DNS back to the original primary node.
7. Leave *read-only mode*
8. Broadcast notification

1.3.4 Reindexing Content

```
$ cd /usr/share/dataone-cn-index/
$ sudo java -jar d1_index_build_tool.jar -h
DataONE solr index build tool help:

This tool indexes objects the CN's system metadata map.
   Nothing is removed from the solr index, just added/updated.

Please stop the d1-index-task-processor while this tool runs:
       /etc/init.d/d1-index-task-processor stop
```

(continues on next page)

(continued from previous page)

```
And restart when the tool finishes:
       /etc/init.d/d1-index-task-processor start

-d     System data modified date to begin index build/refresh from.
       Data objects modified/added after this date will be indexed.
       Date format: mm/dd/yyyy.

-a     Build/refresh all data objects regardless of modified date.

-c     Build/refresh a number data objects, the number configured by this option.
        This option is primarily intended for testing purposes.

-pidFile   Refresh index document for pids contained in the file path
           supplied with this option.  File should contain one pid per line.

-migrate   Build/refresh data object into the next search index
             version's core - as configured in:
              /etc/dataone/solr-next.properties
Exactly one option amoung -d or -a or -pidFile must be specified.

So, I usually create a pid list using MN.listObjects(), save it to, say, /tmp/pids.
↪txt, and then use:

sudo java -jar d1_index_build_tool.jar -pidFile/tmp/pids.txt

(notice the no space between the pidFile option and the path argument).
```

1.3.5 Approving a Member Node

Approving a Member Node involves setting the `d1NodeApproved` property for the node entry in LDAP and triggering a refresh of the node list in `d1-processing` so that the new node is subsequently processed for synchronization, log-aggregation and replication. Updating LDAP alone will not trigger a refresh of the node list. The `dataone-approve-node` tool will update the entry and trigger the node list refresh.

Procedure

The approval process is performed after the MN has registered in the environment.

To approve a MN:

1. `ssh` to a Coordinating Node in the environment.
2. run `sudo /usr/local/bin/dataone-approve-node`
3. Answer the questions, e.g.:

```
Choose the number of the Certificate to use
0)  urn:node:cnStageUCSB1.pem
1)  urn_node_cnStageUCSB1.pem
0

Pending Nodes to Approve
0) urn:node:mnStageORC1 1) urn:node:mnStageLTER 2) urn:node:mnStageCDL  3) urn:
↪node:USGSCSAS
4) urn:node:ORNLDAAC  5) urn:node:mnTestTFRI  6) urn:node:mnTestUSANPN  7) urn:
↪node:TestKUBI
```

(continues on next page)

(continued from previous page)

```
8) urn:node:EDACGSTORE  9) urn:node:mnTestDRYAD 10) urn:node:DRYAD  11) urn:node:
↪mnTestGLEON
12) urn:node:mnDemo11 13) urn:node:mnTestEDORA  14) urn:node:mnTestRGD  15) urn:
↪node:mnTestIOE
16) urn:node:mnTestNRDC 17) urn:node:mnTestNRDC1  18) urn:node:mnTestPPBIO  19)␣
↪urn:node:mnTestUIC
20) urn:node:mnTestFEMC 21) urn:node:IEDA_MGDL
Type the number of the Node to verify and press enter (return):
21

Do you wish to approve urn:node:IEDA_MGDL (Y=yes,N=no,C=cancel)
Y

Node Approved in LDAP
Hazelcast Node Topic published to. Approval Complete
```

There may be an ERROR message complaining of “No certificate installed in expected location: /tmp/x509up_u0”. This can be safely ignored.

Sources

See: https://repository.dataone.org/software/cicore/trunk/cn/d1_cn_tools/d1_cn_approve_node/

1.3.6 Upgrade and Configuration

Metacat Upgrade

After upgrading Metacat, it is necessary to visit:

> https://node name/metacat/admin

and configure metacat for operation. The defaults should be preserved from previous setup.

1.3.7 Upgrading the Search UI

UI screenshot

Updating the Search UI

The production Search UI, `search.dataone.org`, is hosted on two machines, and all need to be upgraded individually:

- `search-ucsb-1.dataone.org`
- `search-orc-1.dataone.org`

For the DataONE test environments, all Search UIs are hosted on a single machine:

- `search.test.dataone.org`

This machine is configured to host the following virtual hosts:

- `search-dev.test.dataone.org`
- `search-dev-2.test.dataone.org`
- `search-sandbox.test.dataone.org`
- `search-sandbox-2.test.dataone.org`
- `search-stage.test.dataone.org`
- `search-stage-2.test.dataone.org`

When upgrading the SearchUI software for each environment, it will be installed in the appropriate `DocumentRoot` configured for the virtual host.

Upgrade steps - Scripted upgrade

1. Clone the `metacatui-config` repository to your home directory on the Search UI machine

```
git clone https://github.nceas.ucsb.edu/walker/metacatui-config.git
```

2. In the `scripts` directory, run the upgrade script

   For production: Run the `upgrade.sh` script and send the MetacatUI version via the `-v` flag and the deployment name (`dataone`) via the `-d` flag.

```
cd metacatui-config/scripts/
bash upgrade.sh -v 2.12.0 -d dataone
```

   For development environments: Run the `upgrade-test-dataone.sh` script and send the MetacatUI version via the `-v` flag. This script upgrades all of the test DataONE search UIs.

```
cd metacatui-config/scripts
bash upgrade-test-dataone.sh -v 2.12.0
```

Upgrade steps - Manual upgrade

1. Download the latest version of MetacatUI from https://github.com/NCEAS/metacatui/releases

```
 wget https://github.com/NCEAS/metacatui/archive/metacatui-2.12.0.zip
```

2. Unpack the `.zip` or `.tar.gz` file

```
unzip metacatui-2.12.0.zip
```

3. Backup the currently-installed Search UI files

```
cp -rf /var/www/search.dataone.org .
```

4. Open the `metacatui-2.12.0/src/index.html` file in a text editor and change the `appConfigPath` variable towards the beginning of the HTML file to match the location where the DataONE MetacatUI theme config file will be deployed.

Production Config

For production, this will be in the `themes` directory included in MetacatUI:

````
```
...
<script type="text/javascript">
  // The path to your configuration file for MetacatUI. This can be any web-
↪accessible location.
  var appConfigPath = "/js/themes/dataone/config.js";
</script>
...
```
````

Development Config

Config files for development environments are not released in the MetacatUI repository, but instead are maintained in a separate `metacatui-config` repository here: https://github.nceas.ucsb.edu/walker/metacatui-config

Each CN development environment will have a config file in that repository. Download the corresponding config file and deploy to the `DocumentRoot` configured for the virtual host. Set the `appConfigPath` to that root path:

````
```
...
<script type="text/javascript">
  // The path to your configuration file for MetacatUI. This can be any web-
↪accessible location.
  var appConfigPath = "/config.js";
</script>
...
```
````

See the MetacatUI documentation for more detailed instructions and for customization options: https://nceas.github.io/metacatui/

Move the new search UI files to the root directory where web files are served.

```
cp -rf metacatui-2.12.0/src/* /var/www/search.dataone.org/
```

Installing a new Search UI

1. Set up a VM and configure Apache to serve web files from a directory in `/var/www/`. Follow steps 1-5 above to install MetacatUI in the `/var/www/` directory.

2. Add the FallbackResource Apache directive and allow encoded slashes:

```
...
# Serve index.html instead of a 404 error in the MetacatUI directory
<Directory "/var/www/search.dataone.org">
  FallbackResource /index.html
</Directory>

# Allow encoded slashes in URLs so encoded identifiers can be sent in MetacatUI␣
↪URLs
AllowEncodedSlashes On
...
```

See the MetacatUI Apache configuration documentation for further details: https://nceas.github.io/metacatui/install/apache.html

1. If having issues with CORS requests to the CN, configure Apache to proxy requests to DataONE CN API calls. Add the following Apache directives:

```
SSLProxyEngine on
ProxyPass "/cn/v2/" "https://cn.dataone.org/cn/v2/"
ProxyPassReverse "/cn/v2/" "https://cn.dataone.org/cn/v2/"
```

Enable these Apache mods:

```
a2enmod proxy_http
a2enmod proxy
```

1.3.8 Certificates

Todo: add general certificate information

Client Certificate

Todo: add client cert info

Server Certificate

See also [[Connectivity and Certificates|CN_connectivity]]

Install Certbot

Install the `cerbot` tool to generate LetsEncrypt certificates.

See [[LetsEncrypt]] for installation details.

Adjust `node.properties`

Two additional properites need to be added to `/etc/dataone/node.properties`:

Key	Description
environment.hosts	Space delimited list of host names for CNs participating in the environment
cn.rsyncuser	Username of account to use when syncing content across CNs.

For example, on cn-stage-ucsb-1.test.dataone.org:

```
environment.hosts=cn-stage-ucsb-1.test.dataone.org cn-stage-unm-1.test.dataone.org cn-
↪stage-orc-1.test.dataone.org
cn.rsyncuser=rsync_user
```

Create Account for rsyncuser

```
sudo adduser ${RUSER} --disabled-password
sudo su - ${RUSER}
mkdir .ssh
chmod 0700 .ssh
cd .ssh
ssh-keygen -N "" -f id_rsa
cp id_rsa.pub authorized_keys
chmod 0600 *
cd ~
mkdir bin
nano bin/rsync-wrapper.sh
  ...
chmod u+x bin/rsync-wrapper.sh
```

rsync-wrapper.sh:

```
#!/bin/bash

LOG="/home/rsync_user/actions.log"
echo "$(date) " $@ >> ${LOG}
/usr/bin/sudo /usr/bin/rsync $@
```

Prepare for Verification

Before running the certificate generation command it is necessary to create the working folder that will be used for the verifications. Do the following on each CN:

```
PROPERTIES="/etc/dataone/node.properties"
RSUSER=$(grep "^cn.rsyncuser=" ${PROPERTIES} | cut -d'=' -f2)
sudo mkdir -p /var/www/.well-known/acme-challenge
sudo chown -R ${RSUSER}:${RSUSER} /var/www/.well-known/acme-challenge
sudo setfacl -Rdm g:${RSUSER}:rw /var/www/.well-known/acme-challenge/
sudo chmod g+s /var/www/.well-known/acme-challenge/
```

Apache must be configured to not redirect the verification address in the .well-known folder. The following example is for cn-stage-ucsb-1.test.dataone.org. Adjust ServerName, ServerAlias, and RedirectMatch with appropriate values for the respective environment and host:

```
<VirtualHost *:80>
  ###
  # This config only comes into play when DNS for cn-stage.test.dataone.org
```

(continues on next page)

(continued from previous page)

```
  # is pointing to this server
  ###
  ServerName cn.dataone.org
  ServerAlias cn-stage-ucsb-1.dataone.org
  ServerAdmin administrator@dataone.org
  DocumentRoot /var/www/
  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

  # Redirect all traffic except certbot to HTTPS
  RedirectMatch ^/(?!.well-known)(.*) https://cn-stage.test.dataone.org/$1
</VirtualHost>
```

Supporting Scripts

The certificate generation process relies on the following authentication and cleanup hooks to copy verification information to other nodes participating in the environment and to cleanup afterwards.

`/etc/letsencrypt/renewal/manual-auth-hook.sh`:

```
#!/bin/bash
PROPERTIES="/etc/dataone/node.properties"
RSUSER=$(grep "^cn.rsyncuser=" ${PROPERTIES} | cut -d'=' -f2)
HOSTS=$(grep "^environment.hosts=" ${PROPERTIES} | cut -d'=' -f2)
THIS_HOST=$(hostname -f)
FNVALID="/var/www/.well-known/acme-challenge/$CERTBOT_TOKEN"
CREDS="/home/${RSUSER}/.ssh/id_rsa"
echo $CERTBOT_VALIDATION > ${FNVALID}
for TARGET_HOST in ${HOSTS}; do
  if [ "${TARGET_HOST}" != "${THIS_HOST}" ]; then
    echo "Copying verification to ${TARGET_HOST}"
    scp -i ${CREDS} ${FNVALID} ${RSUSER}@${TARGET_HOST}:${FNVALID}
  fi
done
```

`/etc/letsencrypt/renewal/manual-cleanup-hook.sh`:

```
#!/bin/bash
PROPERTIES="/etc/dataone/node.properties"
RSUSER=$(grep "^cn.rsyncuser=" ${PROPERTIES} | cut -d'=' -f2)
HOSTS=$(grep "^environment.hosts=" ${PROPERTIES} | cut -d'=' -f2)
THIS_HOST=$(hostname -f)
FNVALID="/var/www/.well-known/acme-challenge/$CERTBOT_TOKEN"
CREDS="/home/${RSUSER}/.ssh/id_rsa"
rm ${FNVALID}
for TARGET_HOST in ${HOSTS}; do
  if [ "${TARGET_HOST}" != "${THIS_HOST}" ]; then
    echo "Removing verification from ${TARGET_HOST}"
    ssh -i ${CREDS} ${RSUSER}@${TARGET_HOST} "rm ${FNVALID}"
  fi
done
```

After a certificate is renewed, it is necessary to notify administrators that some action is required. Place the following *notify-administrators.sh* in the *renew-hook.d* folder. Any scripts in that folder will be called on a successful certificate renewal.

```
#!/bin/bash
PROPERTIES="/etc/dataone/node.properties"
THIS_HOST=$(hostname -f)
THIS_ENVIRONMENT=$(grep "^cn.router.hostname=" ${PROPERTIES} | cut -d'=' -f2)
ADMIN="administrator@dataone.org"

cat <<EOF | mail -s "Certificate Renewal on ${THIS_ENVIRONMENT}" ${ADMIN}
Hi!
certbot running on ${THIS_HOST} has generated a new server certificate for the
${THIS_ENVIRONMENT} environment.

Some manual steps must be taken to complete the installation of the new
certificate. The process for this is documented at:

  https://github.com/DataONEorg/DataONE_Operations/wiki/LetsEncrypt-CNs

but basically entails running:

  /etc/letsencrypt/renewal/post-cn-cert-renew.sh

then restarting services on each CN in the ${THIS_ENVIRONMENT} environment.

cheers
EOF
```

Account for Synchronization

- Create account, disable password
- Create ssh keys
- Distribute ssh public keys
- Verify ssh to other hosts
- Enable rsync for account

Certificate Generation

The server certificate must have a primary subject of the primary CN name and must also include as subject alternative names the host names of each CN participating in the environment. For example, the stage environment would include: `cn-stage.test.dataone.org`, `cn-stage-ucsb-1.test.dataone.org`, `cn-stage-orc-1.test.dataone.org`, and `cn-stage-unm-1.test.dataone.org`.

Certificate generation is performed by `certbot` with the following command run on the primary host only (remove the `--dry-run` parameter to do an actual request):

```
PROPERTIES="/etc/dataone/node.properties"
HOSTS=$(grep "^environment.hosts=" ${PROPERTIES} | cut -d'=' -f2)
THIS_ENVIRONMENT=$(grep "^cn.router.hostname=" ${PROPERTIES} | cut -d'=' -f2)
DOMAINS="-d ${THIS_ENVIRONMENT}"
for DHOST in ${HOSTS}; do DOMAINS="${DOMAINS} -d ${DHOST}"; done

sudo certbot certonly --dry-run --manual \
  --preferred-challenges=http \
  --manual-auth-hook=/etc/letsencrypt/renewal/manual-auth-hook.sh \
```

(continues on next page)

(continued from previous page)

```
  --manual-cleanup-hook=/etc/letsencrypt/renewal/manual-cleanup-hook.sh \
  --cert-name ${THIS_ENVIRONMENT} ${DOMAINS}
```

After a successful first time certificate generation, is is necessary to configure various services to use the new certificates. This procedure should only need to be done once.

Adjust Apache Configuration

Apache HTTPS configuration is straight forward:

```
<VirtualHost *:443>
  ServerName cn.dataone.org
  # Change the following for the respective host
  ServerAlias cn-ucsb-1.dataone.org
  ...

  SSLCACertificateFile /etc/ssl/certs/DataONECAChain.crt

  SSLCertificateKeyFile  /etc/letsencrypt/live/cn.dataone.org/privkey.pem
  SSLCertificateFile  /etc/letsencrypt/live/cn.dataone.org/fullchain.pem
  SSLCertificateChainFile /etc/letsencrypt/lets-encrypt-x3-cross-signed.pem
</VirtualHost>
```

Adjust Postgres Certificate References

`Postgres` is configured to use the server certificate and expects the certificate and key to be located in `/var/lib/postgresql/9.3/main/` (Note that "9.3" is the current version of postgres installed. The actual location may change in the future).

Symbolic links may be used to refer to the actual certificate location. Replace the existing `server.crt` and `server.key` for postgress with:

```
PROPERTIES="/etc/dataone/node.properties"
THIS_ENVIRONMENT=$(grep "^cn.router.hostname=" ${PROPERTIES} | cut -d'=' -f2)
CERTS="/etc/letsencrypt/live/${THIS_ENVIRONMENT}"
sudo mv /var/lib/postgresql/9.3/server.crt "/var/lib/postgresql/9.3/server.crt.$(date
↪+%Y%m%d)"
sudo mv /var/lib/postgresql/9.3/server.key "/var/lib/postgresql/9.3/server.key.$(date
↪+%Y%m%d)"
sudo ln -s "${CERTS}/cert.pem" /var/lib/postgresql/9.3/server.crt
sudo ln -s "${CERTS}/privkey.pem" /var/lib/postgresql/9.3/server.key
```

The linked files will survive a refresh of the certificates, so this only needs to be done once.

cn.server.publiccert.filename=/etc/letsencrypt/live/cn-dev-2.test.dataone.org/cert.pem cn.rsyncuser=rsync_user environment.hosts=cn-stage-ucsb-2.test.dataone.org cn-stage-unm-2.test.dataone.org

Configure the DataONE Portal Application

- portal.properties
- set permissions
- restart tomcat

Certificate Renewal

LetsEncrypt certificates are relatively short lived (three months), so an automated mechanism to check and update the certificates is needed. Since restarting services on the DataONE Coordinating Nodes requires some coordination across the servers, this process is not yet entirely automated, though all that should be necessary is for an administrator to execute a script to distribute the certificate and then manually restart services on each CN. Basically:

1. `certbot` generates a new certificate from a `cron` job
2. DataONE administrators are notified of the need for action
3. An administrator distributes the certificate to each CN
4. An administrator restarts services as necessary

The certificate renewal process is performed by `cron` using the task `/etc/cron.weekly/certbot-renew` listed below:

```
#!/bin/bash
set -e
logger "Checking for LetsEncrypt certificate renewal"
/usr/bin/certbot renew -n --quiet \
  --renew-hook "/bin/run-parts /etc/letsencrypt/renew-hook.d/"
```

The tasks in `/etc/letsencrypt/renew-hook.d/` are executed when certificates are successfully renewed. For the CNs, a successful renewal results in a notification being sent to administrators requesting that the next steps of the certificate renewal are followed.

The following script will ensure the certificates have the correct permissions and synchronize the certificates to other servers using rsync.

`/etc/letsencrypt/renewal/post-cn-cert-renew.sh`:

```
#!/bin/bash
PROPERTIES="/etc/dataone/node.properties"
RSUSER=$(grep "^cn.rsyncuser=" ${PROPERTIES} | cut -d'=' -f2)
HOSTS=$(grep "^environment.hosts=" ${PROPERTIES} | cut -d'=' -f2)
THIS_HOST=$(hostname -f)
THIS_ENVIRONMENT=$(grep "^cn.router.hostname=" ${PROPERTIES} | cut -d'=' -f2)

function synchronize_certs() {
  logger "INFO: Synchronizing letsencrypt certificates to other CNs..."
  #Set permissions for ssl-cert group access
  echo "Setting permissions on certificates..."
  chgrp -R ssl-cert /etc/letsencrypt/archive
  chmod g+rx /etc/letsencrypt/archive
  chgrp -R ssl-cert /etc/letsencrypt/live
  chmod g+rx /etc/letsencrypt/live
  #This is needed for Postgres to start:
  chmod 0640 /etc/letsencrypt/archive/${THIS_ENVIRONMENT}/privkey*

  #Synchronize with other servers
  for TARGET_HOST in ${HOSTS}; do
    if [ "${TARGET_HOST}" != "${THIS_HOST}" ]; then
      echo "Syncing certificate info to ${TARGET_HOST}"
      rsync -avu --rsync-path="/home/${RSUSER}/bin/rsync-wrapper.sh" \
        -e "ssh -i /home/${RSUSER}/.ssh/id_rsa -l ${RSUSER}" \
        /etc/letsencrypt/*  \
        ${RSUSER}@${TARGET_HOST}:/etc/letsencrypt/
    fi
```

(continues on next page)

(continued from previous page)

```
  done
}

echo "Using variables:"
echo "RSUSER = ${RSUSER}"
echo "HOSTS = ${HOSTS}"
echo "THIS_HOST = ${THIS_HOST}"
echo "THIS_ENVIRONMENT = ${THIS_ENVIRONMENT}"
echo
read -p "Does this look OK (y/N)?" -n 1 -r
echo
if [[ $REPLY =~ ^[Yy]$ ]]; then
  synchronize_certs
  exit 0
fi
echo "Aborted."
```

Service Restarts

After a new certificate has been distributed it is necessary to restart `apache2`, `postgresql`, and `tomcat7` to pick up the change:

```
# Verify apache configuration is OK
sudo apache2ctl -t
sudo service apache2 restart
sudo service postgres restart
```

TODO: refer to procedure for tomcat restart on CNs

Verification

Verification that the new certificate basically comes down to three checks:

1. Check service is running

- Is the service running?

```
sudo service apache2 status
sudo service postgres status
sudo service tomcat7 status
```

- Is a listener on the expected port?

```
sudo netstat -tulpn
```

2. Verify the new certificate is being used

   The following command run from the command line will show the certificate being used by the server in its plain text form:

```
TARGET="cn-ucsb-1.dataone.org:443"
echo "Q" | openssl s_client -connect ${TARGET} | openssl x509 -text -noout
```

3. Verify that a client can connect as expected

   Use a web browser to check the server responds as expected. Use a DataONE client to interact with the server.

1.4 Development

1.4.1 Setting Hostname on Ubuntu

For server `my-server.dataone.org` with IP `12.34.56.78`:

`/etc/hostname`:

```
my-server
```

`/etc/hosts`:

```
127.0.0.1 localhost
12.34.56.78 my-server.dataone.org my-server

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Then reboot or:

```
sudo hostname my-server
```

1.4.2 Debian Packages

Debian packages are built by the *continuous build system* which are then used to install on Ubuntu systems such as the DataONE Coordinating Nodes.

Packages sources are pulled from subversion, the trunk versions are located at: https://repository.dataone.org/software/cicore/trunk/cn-buildout/

1.4.3 Continuous Build System

The continuous build system is a Jekins instance that builds and tests Java components used by DataONE.

Location: http://jenkins-1.dataone.org/jenkins/

Maven Repository

URL https://maven.dataone.org

Artifacts are deployed to the maven repository on successful project builds.

The maven repository is indexed using the Maven::Indexer CLI

```
sudo /etc/cron.daily/maven-index
```

Backup Jenkins Configurations

```
HUDSON_HOME="/var/lib/jenkins/"
B_DEST="/var/lib/jenkins-backup"
rsync -r  --delete --include "jobs/" --include "users/" --include "*.xml" \
--include "jobs/*/config.xml" --include "users/*/config.xml" \
--include "userContent/*" \
--exclude "jobs/*/builds" --exclude "jobs/*/last*" --exclude "jobs/*/next*" \
--exclude "*.log" --exclude "jobs/*/workspace*" --exclude "jobs/*/cobertura" \
--exclude "jobs/*/javadoc" --exclude "jobs/*/htmlreports" --exclude "jobs/*/ncover" \
--exclude "jobs/*/modules"\
--exclude "users/*/*" --exclude "/*" --exclude ".svn" --exclude "svnexternals.txt" \
${HUDSON_HOME} ${B_DEST}/backup/

rsync -r  --delete \
--include="jobs/" \
--include="*.xml" \
--include="jobs/*/config.xml" \
--include="users/*/config.xml" \
--include="userContent" \
--exclude-from="excludes.txt" \
${HUDSON_HOME} ${B_DEST}/backup/

--exclude="*.java*" \
--exclude=".*" \
--exclude=".*/" \
--exclude="fingerprints/" \
--exclude="secrets/" \
--exclude="*secret*" \
--exclude="identity.*" \
--exclude="jobs/*/builds" \
--exclude="jobs/*/last*" \
--exclude="jobs/*/next*" \
--exclude="*.log" \
--exclude="jobs/*/workspace*" \
--exclude="jobs/*/cobertura" \
--exclude="jobs/*/javadoc" \
--exclude="jobs/*/htmlreports" \
--exclude="jobs/*/ncover" \
--exclude="jobs/*/modules"  \
--exclude="*.tar" \
--exclude=".svn" \
--exclude="svnexternals.txt" \
${HUDSON_HOME} ${B_DEST}/backup/

jobs/*/cobertura
jobs/*/javadoc
jobs/*/htmlreports
jobs/*/ncover
jobs/*/modules
```

1.4.4 Profiling Java Applications

There are a number of tools available for profiling Java applications.

JProfiler

The commercial JProfiler is able to remotely connect to a running instance of a JVM and provide useful statistics on activity.

Generating Flamegraphs

Flamegraphs show the relative time spent in different operations and the hierarchy of calls needed to service an operation. Hierarchy is shown in the vertical stack. Horizontal placement has no significance as output is aplhabetically ordered.

The async_profiler is much easier to use than honest_profiler, however it doesn't work on the UCSB systems because of the shared kernel virtualization there.

Insead, used honest_profiler as follows.

(note the following process was hacked together one Friday afternoon - there's lots of improvements to the process that can be made, but this is the basic process followed.)

The following steps were followed to generate flamegraphs for CN operations using cn-dev-ucsb-2.test.dataone.org as the example.

Build the libagent.so c-lib (can be shared across systems using the same version of gcc)

```
sudo apt-get install cmake
sudo apt-get install build-essential
mkdir profiling
cd profiling
git clone https://github.com/jvm-profiling-tools/honest-profiler.git
cd honest-profiler/
export JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
cmake CMakeLists.txt
export LC_ALL=C
make
```

Setup the profiler application:

```
cd ~/profiling
mkdir profiler
cd profiler
wget http://insightfullogic.com/honest-profiler.zip
unzip honest-profiler.zip
# copy the previously built libagent.so
cp ../honest-profiler/build/liblagent.so .
```

Get tool to convert profiler output for flamegraph generation:

```
cd ~/profiling
git clone https://github.com/cykl/hprof2flamegraph.git
```

Setup output folder and permissions (tomcat will be run as tomcat7 user):

```
sudo chgrp tomcat7 ~/profiling
sudo chmod g+w ~/profiling
# also enable write to a destination folder
sudo mkdir /var/www/profiling
sudo chgrp sudo /var/www/profiling
sudo chmod -R g+w /var/www/profiling
```

Generated flamegraphs will be at https://host.name/profiling/

Script to start tomcat7 for profiling (cn-dev-2 configuration), `start_tomcat`:

```
#!/bin/bash
PDIR=/home/vieglais/profiling
if [ "$#" -ne 2 ]; then
  echo "Must provide log name (no extension) and titles as parameters."
  exit 1
fi

LOG_FILE=${PDIR}/${1}
FG_PARAMS="--width=2000 --title='${2}'"

sudo -u tomcat7 /usr/lib/jvm/java-8-openjdk-amd64/bin/java \
-agentpath:${PDIR}/profiler/liblagent.so=interval=1,logPath=${LOG_FILE}.hpl,start=0,
↪host=127.0.0.1,port=9999 \
-Djava.util.logging.config.file=/var/lib/tomcat7/conf/logging.properties \
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager \
-Djava.awt.headless=true \
-XX:+PreserveFramePointer \
-Xmx8192m \
-XX:+UseParallelGC \
-Xms1024M \
-XX:MaxPermSize=512M \
-Djava.endorsed.dirs=/usr/share/tomcat7/endorsed \
-classpath ${PDIR}/profiler/honest-profiler.jar:/usr/share/tomcat7/bin/bootstrap.jar:/
↪usr/share/tomcat7/bin/tomcat-juli.jar \
-Dcatalina.base=/var/lib/tomcat7 \
-Dcatalina.home=/usr/share/tomcat7 \
-Djava.io.tmpdir=/tmp/tomcat7-tomcat7-tmp \
org.apache.catalina.startup.Bootstrap start

# Process log into svg
python hprof2flamegraph/stackcollapse_hpl.py ${LOG_FILE}.hpl > ${LOG_FILE}.txt
hprof2flamegraph/flamegraph.pl ${FG_PARAMS} ${LOG_FILE}.txt > ${LOG_FILE}.svg
cp ${LOG_FILE}.svg /var/www/profiling/
```

`p_start` script to start profiling data collection after service has started:

```
#/bin/bash
echo start | nc 127.0.0.1 9999
```

`p_stop` script to stop profiling data collection:

```
#/bin/bash
echo stop | nc 127.0.0.1 9999
```

Script to warm up tomcat a bit, start data collection, execute a call and stop data collection, e.g. `test_viewservice`:

```
#!/bin/bash

SVC_URL="https://cn-dev-ucsb-2.test.dataone.org/cn/v2/views/metacatui/"
PIDS="ajpelu.6.8 ajpelu.6.9 Akasha.16.1 Akasha.16.2 Akasha.16.3 Akasha.16.4 Akasha.16.
↪5 Akasha.16.6 Akasha.16.7 Akasha.16.8"
#Warm up tomcat a little
for PID in ${PIDS}; do
```

(continues on next page)

(continued from previous page)

```
  curl "${SVC_URL}${PID}" > /dev/null
done

./p_start
curl "${SVC_URL}doi%3A10.5063%2FF1R49NQB" > /dev/null
./p_stop
```

The process to generate a profile is then:

1. Open two terminals and cd into `~/profiling`
2. Put the environment into read-only mode, on the primary CN:

   ```
   sudo d1processingstate FALSE
   sudo service d1-processing stop
   ```

3. In one terminal, shutdown the tomcat7 service and startup the script to run tomcat7 (script will ask for sudo):

   ```
   sudo service tomcat7 stop
   ./start_tomcat view_service "cn/v2/views/metacatui"
   ```

4. Wait for tomcat to fire up. This takes about 100 seconds or so. . .
5. In the other terminal, run `test_viewservice`
6. After `test_viewservice` is done, shutdown tomcat7 with a ctrl-c in the first terminal.
7. View the resulting flamegraph in your web browser by visiting:

   https://cn-dev-ucsb-2.test.dataone.org/profiling/view_service.svg

1.5 DataONE Environments

An "environment" in the context of these documents refers to a complete installation of the components necessary to provide a functional DataONE federation. Each environment is completely independent of others which facilitates testing of new software releases within an environment without impacting others.

Each environment is completely disconnected from other environments - there is no communication between systems participating in different environments.

End users will typically only interact with the Production environment. All other environments are used for development and testing purposes.

Table 1: Table 1. Environments used by DataONE infrastructure.

Environment	Role	Description
Production	Production	The *production* environment operates the DataONE production infrastructure. Uptime of this environment is of highest priority. Maintenance should be scheduled and users must be informed via the *operations and cicore email lists*.
Stage	Production testing	The *stage* environments are used for testing Member Nodes, updates to Coordinating Nodes, and any other changes before deployment or participation in the production environment. Except when testing CN components, the stage environment should be operating the same version software as the production environment.
Stage-2	Production testing	As for the *stage* environment.
Sandbox	Testing	The *sandbox* environment is used for evaluating new features and other potentially disruptive changes. The sandbox environment may be running different (typically newer) versions of software components compared to the stage and production environments.
Sandbox-2	Testing	As for the *sandbox* environment.
Dev	New features	The *dev*, or *development* environment is the most volatile environment and is used for developing new features or other disruptive changes. The development environment may or may not be available at any point in time. Changes to the development environments is coordinated within the developers via slack or email.
Dev-2	New features	As for the *development* environment.

1.5.1 Realtime Environment Status

Realtime status of the various environments is available at: https://monitor.dataone.org/status

1.6 Glossary

CA certificate A certificate that belongs to a *CA* and serves as the root certificate in a term:*chain of trust*.

CA

Certificate Authority A certificate authority is an entity that issues digital *certificate* s. The digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others (relying parties) to rely upon signatures or assertions made by the private key that corresponds to the public key that is certified. In this model of trust relationships, a CA is a trusted third party that is trusted by both the subject (owner) of the certificate and the party relying upon the certificate. CAs are characteristic of many public key infrastructure (PKI) schemes.

http://en.wikipedia.org/wiki/Certificate_authority

CA signing key The private key which the *CA* uses for signing *CSR*s.

Certificate A public key certificate (also known as a digital certificate or identity certificate) is an electronic document which uses a digital signature to bind a public key with an identity – information such as the name of a person or an organization, their address, and so forth. The certificate can be used to verify that a public key belongs to an individual.

http://en.wikipedia.org/wiki/Public_key_certificate

Chain of trust The Chain of Trust of a Certificate Chain is an ordered list of certificates, containing an end-user subscriber certificate and intermediate certificates (that represents the Intermediate CA), that enables the receiver

to verify that the sender and all intermediates certificates are trustworthy.

http://en.wikipedia.org/wiki/Chain_of_trust

CILogon The CILogon project facilitates secure access to CyberInfrastructure (CI).

http://www.cilogon.org/

client An application that accesses the DataONE infrastructure on behalf of a user.

Client Library Part of the DataONE *Investigator Toolkit (ITK)*. Provides programmatic access to the DataONE infrastructure and may be used to form the basis of larger applications or to extend existing applications to utilize the services of DataONE.

Available for Java and Python.

Java Client Library documentation

Java Client Library source

Python Client Library documentation

Python Client Library source

Client side authentication *SSL* Client side authentication is part of the *SSL handshake*, where the client proves its identity to the web server by providing a *certificate* to the server. The certificate provided by the client must be signed by a *CA* that is trusted by the server. Client Side Authentication is not a required part of the handshake. The server can be set up to not allow Client side authentication, to require it or to let it be optional.

Client side certificate *Certificate* that is provided by the client during *client side authentication*.

cn

CN

Coordinating Node A server that implements the DataONE Coordinating Node API.

Common Library Part of the DataONE *Investigator Toolkit (ITK)*. Provides functionality commonly needed by projects that interact with the *DataONE* infrastructure, such as serialization and deserialization of the DataONE types to and from types native to the programming language.

It is a dependency of DataONE *Client Library*.

Available for Java and Python.

TODO We need to point to releases.dataone.org for the Common Libraries. For now, see https://repository.dataone.org/software/cicore/trunk/

Coordinating Node API The Application Programming Interfaces that Coordinating Nodes implement to facilite interactions with *MN* and DataONE clients.

http://mule1.dataone.org/ArchitectureDocs-current/apis/CN_APIs.html

CSR Certificate Signing Request

A message sent from an applicant to a *CA* in order to apply for a *certificate*.

http://en.wikipedia.org/wiki/Certificate_signing_request

DataONE Data Observation Network for Earth

https://dataone.org

Data Packaging Data, in the context of DataONE, is a discrete unit of digital content that is expected to represent information obtained from some experiment or scientific study.

http://mule1.dataone.org/ArchitectureDocs-current/design/DataPackage.html

DN Distinguished Name

environment

Environment The collection of Coordinating Nodes, Member Nodes, and applications (e.g. search interface) that work together as a federation. There is a single *Production Environment* and several test environments.

environment dns

Environment DNS The DNS entry that all systems interacting with CNs in and Environment should use. During maintenance, the Environment DNS entry will be adjusted to point to another CN in the same Environment, thus helping to ensure ongoing availability of services while other CNs are offline. For example, the DataONE Production Environment has three CNS, `cn-ucsb-1.dataone.org`, `cn-unm-1.dataone.org`, and `cn-orc-1.dataone.org`. The Environment DNS is `cn.dataone.org` and points to one of the three CNs. The Environment DNS entry has a relatively short TTL, and its associated IPAddress should not be cached for more than a few seconds.

GMN DataONE Generic Member Node

GMN is a complete implementation of a *MN*, written in Python. It provides an implementation of all MN APIs and can be used by organizations to expose their Science Data to DataONE if they do not wish to create their own, native MN.

GMN can be used as a standalone MN or it can be used for exposing data that is already available on the web, to DataONE. When used in this way, GMN provides a DataONE compatible interface to existing data and does not store the data.

GMN can also be used as a workbone or reference for a 3rd party MN implementation. If an organization wishes to donate storage space to DataONE, GMN can be set up as a *replication target*.

Identity Provider A service that authenticates users and issues security tokens.

In the context of DataONE, an Identity Provider is a 3rd party institution where the user has an account. *CILogon* acts as an intermediary between DataONE and the institution by creating *X.509* certificates based on identity assertions made by the institutions.

Investigator Toolkit (ITK) The Investigator Toolkit provides a suite of software tools that are useful for the various audiences that DataONE serves. The tools fall in a number of categories, which are further developed here, with examples of potential applications that would fit into each category.

http://mule1.dataone.org/ArchitectureDocs-current/design/itk-overview.html

Java A statically typed programming language.

http://java.com

LOA Levels of Assurance

CILogon operates three Certification Authorities (CAs) with consistent operational and technical security controls. The CAs differ only in their procedures for subscriber authentication, identity validation, and naming. These differing procedures result in different Levels of Assurance (LOA) regarding the strength of the identity contained in the certificate. For this reason, relying parties may decide to accept certificates from only a subset of the CILogon CAs.

http://ca.cilogon.org/loa

Member Node API The Application Programming Interfaces that a repository must implement in order to join DataONE as a Member Node.

http://mule1.dataone.org/ArchitectureDocs-current/apis/MN_APIs.html

Metacat Metacat is a repository for data and metadata (documentation about data) that helps scientists find, understand and effectively use data sets they manage or that have been created by others. Thousands of data sets are currently documented in a standardized way and stored in Metacat systems, providing the scientific community

with a broad range of Science Data thatâ€“because the data are well and consistently describedâ€“can be easily searched, compared, merged, or used in other ways.

Metacat is implemented in Java.

http://knb.ecoinformatics.org/knb/docs/

MN

Member Node A server that implements the DataONE Member Node API.

MNCore API A set of *MN* APIs that implement core functionality.

http://mule1.dataone.org/ArchitectureDocs-current/apis/MN_APIs.html#module-MNCore

MNRead API A set of *MN* APIs that implement Read functionality.

http://mule1.dataone.org/ArchitectureDocs-current/apis/MN_APIs.html#module-MNRead

OAI-ORE Open Archives Initiative's Object Resource and Exchange

http://www.openarchives.org/ore/

OpenSSL Toolkit implementing the *SSL* v2/v3 and *TLS* v1 protocols as well as a full-strength general purpose cryptography library.

primary cn

Primary CN

Primary Coordinating Node The CN on which the `d1-processing` daemon is running. The Primary CN must always have the *Environment DNS* pointing to it.

Python A dynamic programming language.

http://www.python.org

RDF Resource Description Framework

The Resource Description Framework (RDF) [1] is a family of World Wide Web Consortium (W3C) specifications [2] originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats.

[1] http://www.w3.org/RDF/ [2] http://en.wikipedia.org/wiki/Resource_Description_Framework

Read Only

read-only mode The state of an environment when updates to content throught the DataONE service interfaces is disabled. Services including `resolve`, `get`, `getSystemMetadata`, `getLogRecords`, and `search` continue to function enabling user access to the content without disruption.

Replication target A *MN* that accepts replicas (copies) of Science Data from other MNs and thereby helps ensuring that Science Data remains available.

Resource Map An object (file) that describes one or more aggregations of Web resources. In the context of DataONE, the web resources are DataONE objects such as *Science Data* and *Science Metadata*.

http://www.openarchives.org/ore/1.0/toc

REST Representational State Transfer

A style of software architecture for distributed hypermedia systems such as the World Wide Web.

http://en.wikipedia.org/wiki/Representational_State_Transfer

SciData

Science Data An object (file) that contains scienctific observational data.

SciMeta

Science Metadata An object (file) that contains information about a *Science Data* object.

subject In DataONE, a subject is a unique identity, represented as a string. A user or Node that wishes to act as a given subject in the DataONE infrastructure must hold an *X.509* certificate for that subject.

DataONE defines a serialization method in which a subject is derived from the *DN* in a X.509 certificate.

Self signed certificate A *certificate* that is signed by its own creator. A self signed certificate is not a part of a *chain of trust* and so, it is not possible to validate the information stored in the certificate. Because of this, self signed certificates are useful mostly for testing in an implicitly trusted environment.

http://en.wikipedia.org/wiki/Self-signed_certificate

Server key The private key that Apache will use for proving that it is the owner of the *certificate* that it provides to the client during the SSL handshake.

Server Side Authentication *SSL* Server Side Authentication is part of the *SSL handshake*, where the server proves its identity to the client by providing a *certificate* to the client. The certificate provided by the server must be signed by a *CA* that is trusted by the client. Server Side Authentication is a required part of the handshake.

Server side certificate *Certificate* that is provided by the server during *server side authentication*.

SSL Secure Sockets Layer

A protocol for transmitting private information via the Internet. SSL uses a cryptographic system that uses two keys to encrypt data â^' a public key known to everyone and a private or secret key known only to the recipient of the message.

SSL handshake The initial negotiation between two machines that communicate over SSL.

http://developer.connectopensource.org/display/CONNECTWIKI/SSL+Handshake

http://developer.connectopensource.org/download/attachments/34210577/Ssl_handshake_with_two_way_authentication_with_certificates.png

SysMeta

System Metadata An object (file) that contains system level information about a *Science Data*-, *Science Metadata*- or other DataONE object.

Overview of System Metadata <http://mule1.dataone.org/ArchitectureDocs-current/design/SystemMetadata.html>

Description of the System Metadata type <http://mule1.dataone.org/ArchitectureDocs-current/apis/Types.html#Types.SystemMetadata>

Tier A tier designates a certain level of functionality exposed by a *MN*.

DataONE Member Node Tiers.

TLS Transport Layer Security

Successor of *SSL*.

X.509 An ITU-T standard for a public key infrastructure (PKI) for single sign-on (SSO) and Privilege Management Infrastructure (PMI). X.509 specifies, amongst other things, standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm.

http://en.wikipedia.org/wiki/X509

CHAPTER 2

Indices and Tables

- genindex

See also:

- https://github.com/DataONEorg/DataONE_Operations/wiki
- https://purl.dataone.org/index.html

Index

C

CA, 82
CA certificate, 82
CA signing key, 82
Certificate, 82
Certificate Authority, 82
Chain of trust, 82
CILogon, 83
client, 83
Client Library, 83
Client side authentication, 83
Client side certificate, 83
CN, 83
cn, 83
Common Library, 83
Coordinating Node, 83
Coordinating Node API, 83
CSR, 83

D

Data Packaging, 83
DataONE, 83
DN, 84

E

Environment, 84
environment, 84
Environment DNS, 84
environment dns, 84

G

GMN, 84

I

Identity Provider, 84
Investigator Toolkit (*ITK*), 84

J

Java, 84

L

LOA, 84

M

Member Node, 85
Member Node API, 84
Metacat, 84
MN, 85
MNCore API, 85
MNRead API, 85

O

OAI-ORE, 85
OpenSSL, 85

P

Primary CN, 85
primary cn, 85
Primary Coordinating Node, 85
Python, 85

R

RDF, 85
Read Only, 85
read-only mode, 85
Replication target, 85
Resource Map, 85
REST, 85

S

SciData, 85
Science Data, 86
Science Metadata, 86
SciMeta, 86
Self signed certificate, 86
Server key, 86
Server Side Authentication, 86
Server side certificate, 86
SSL, 86

SSL handshake, **86**
subject, **86**
SysMeta, **86**
System Metadata, **86**

T

Tier, **86**
TLS, **86**

X

X.509, **86**