Intro. to Nexoedge Development

September 2023

Helen Chan

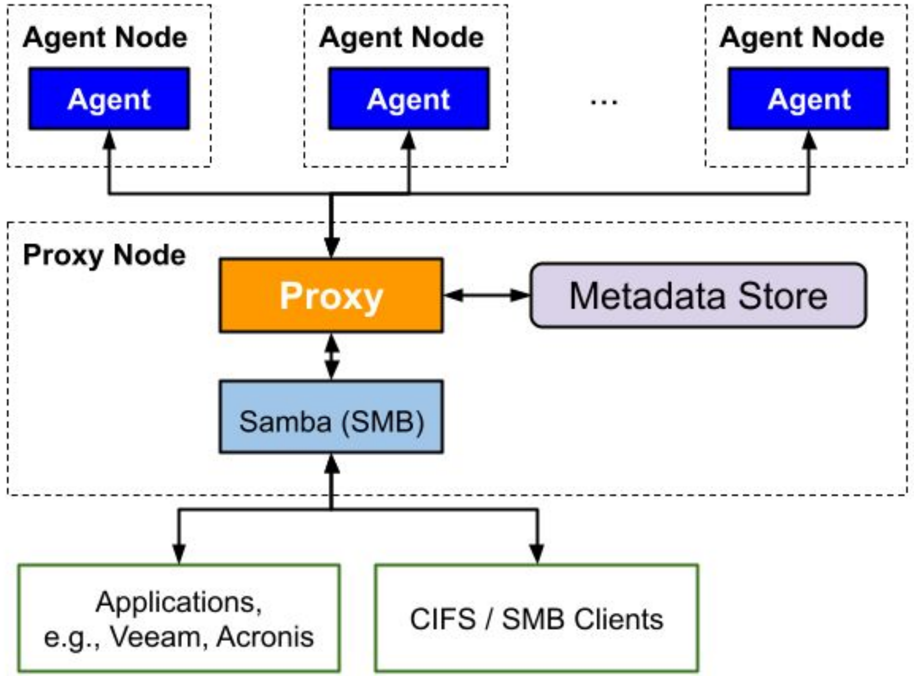
Contents

About Nexoedge

- Nexoedge: Multi-cloud reliable and efficient storage for edge applications, which employs erasure coding for data protection.
    - Previously named nEdge/nCloud (community version).
    - Refer to the slides previously shared by Prof. Lee on system design goals and key features
- Basic understanding of the following concepts
    - Redundant array of independent disks (RAID)
    - Erasure coding, in particular Reed Solomon codes
        - Fault tolerance and storage overhead for the given coding parameters, etc.
    - Client-server model of distributed systems

High-level System Architecture

- Nodes can be physical machines, virtual machines, or containers
- **System Components: Proxy, Agent**
  - Proxy: Data processing (reliability)
  - Agent: Data storage
    - Managed as **storage containers**
- Metadata Store (Key-value store)
- **SMB/CIFS Server**
- Applications
  - A SMB/CIFS client, or any applications that integrate a CIFS/SMB client



Getting Started -- Resources

- **Official GitHub code repository: https://github.com/nexoedge/nexoedge**
    - System components (proxy and agent)
    - Source code license: Apache-2.0.
    - Documentation license: Creative Commons Attribution 4.0 International License.
- **Overview:** https://github.com/nexoedge/nexoedge/blob/master/README.md
- **Installation guide:** https://github.com/nexoedge/nexoedge/blob/master/INSTALL.md
    - Compile the system components from source code.
    - Generate Debian packages for system deployment.
- User documentation: https://github.com/nexoedge/nexoedge/tree/master/docs/user-doc
    - Cover the package installation and other details, e.g., configuration.
    - Mainly for system users/administrators (instead of developers).

Getting Started -- Resources

- **Packages**: In the GitHub Release section
  - **Precompiled SMB/CIFS package**: Samba with Nexoedge VFS extension
  - Precompiled Nexoedge Debian packages (Ubuntu 22.04 only)
  - Web-based portal for system monitoring (source code and manual)
- Docker files: https://github.com/nexoedge/nexoedge/tree/master/docker
  - Nexoedge-specific images for proxy, agent, SMB/CIFS server, and web-based portal frontend and backend.
  - Embedded with scripts that update Nexoedge configurations according to specific environment variables (see the Readme for details)
- Docker compose example: https://github.com/nexoedge/nexoedge/tree/master/docker/examples/cifs/docker-compose
  - Run a local cluster (see the Readme for details)

Getting Started -- System Building and Deployment

Suggested procedures

- **System extension: Source code compilation + SMB/CIFS server setup**
- Application integration (deploying Nexoedge as storage for other applications)
  - Package installation + SMB/CIFS server setup, or
  - Container-based deployment

Remarks

- Before getting familiar with the system deployment, avoid mixing container-based deployment with other setups on the machine.
  - Avoid resource conflicts, e.g., port assignments, which take time to resolve.

Source Code Compilation -- Platform

- Minimal:

Operating System	Ubuntu 22.04 LTS 64-bit
(Virtual) CPU	1
RAM	2GB
Disk storage	16GB

Recommended:

Operating System	Ubuntu 22.04 LTS 64-bit
(Virtual) CPU	4 or above
RAM	8GB or above
Disk storage	32GB or above

(Estimated compilation time for the recommended configuration is 24 minutes, see also the CircleCI auto-build result)

Strongly recommend a *fresh* Ubuntu 22.04 installation for development to avoid troubleshooting overhead.

Source Code Compilation -- Procedures

1. Install the prerequisite libraries

   -
     ```
     $ sudo apt update && sudo apt install -y git cmake g++ libssl-dev libboost-filesystem-dev \
     libboost-system-dev libboost-timer-dev libboost-log-dev libboost-random-dev \
     libboost-locale-dev libboost-regex-dev autoconf libtool nasm pkg-config libevent-dev \
     uuid-dev redis-server redis-tools libxml2-dev libcpprest-dev libaprutil1-dev libapr1-dev \
     libglib2.0-dev libjson-c-dev unzip curl nlohmann-json3-dev libcurl-ocaml-dev
     ```
2. Clone the source code repository

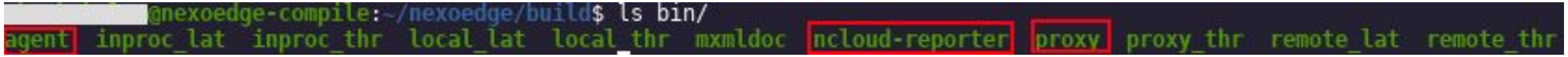
   -
     ```
     $ git clone https://github.com/nexoedge/nexoedge
     ```
3. Create a build directory

   -
     ```
     $ cd nexoedge && mkdir build
     ```
4. Compile the source code

   -
     ```
     $ cd build && cmake .. && make -j2
     ```

Source Code Compilation -- Expected Outcome

- Under the “bin” folder in the build directory
  - Nexoedge component binaries (red): proxy, agent, ncloud-reporter
  - Other programs from third-party libraries

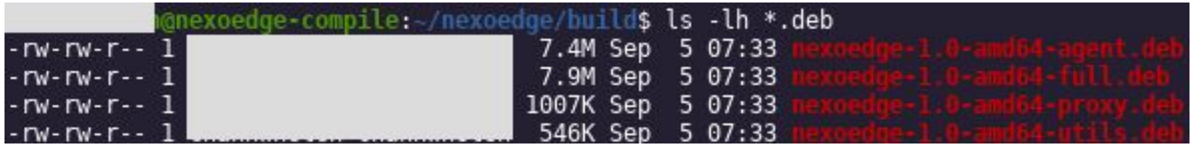


```
[redacted]@nexoedge-compile:~/nexoedge/build$ ls bin/
agent  inproc_lat  inproc_thr  local_lat  local_thr  mxmldoc  ncloud-reporter  proxy  proxy_thr  remote_lat  remote_thr
```

- To run a proxy or an agent manually, e.g., using the sample configuration
  - Proxy: `$ ./bin/proxy ../sample`
  - Agent: `$ ./bin/agent ../sample`
  - Check the system status: `$ ./bin/ncloud-reporter ../sample`

Package Installation -- Package Generation

1. Generate the Debian packages
   a. Reconfigure the source code compilation to Release mode:
      i. `$ cmake -DCMAKE_BUILD_TYPE=Release ..`
   b. Generate the packages
      i. `$ make -j2 package`
   c. Expected Outcome: Debian packages `nexoedge-1.0-amd64-{full,proxy,agent,utils}.deb`
      i. ‘full’: proxy, agent, ncloud-reporter
      ii. ‘proxy’: proxy, ncloud-reporter
      iii. ‘agent’: agent
      iv. ‘utils’: ncloud-reporter



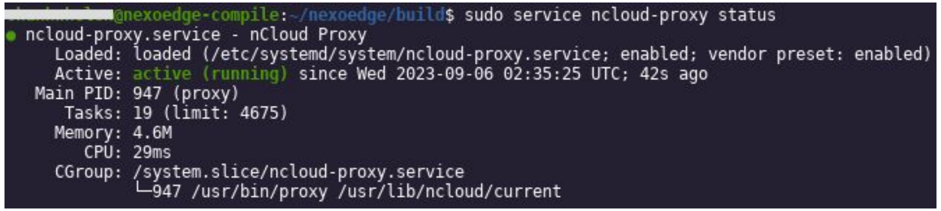
```
[redacted]@nexoedge-compile:~/nexoedge/build$ ls -lh *.deb
-rw-rw-r-- 1 [redacted]  7.4M Sep  5 07:33 nexoedge-1.0-amd64-agent.deb
-rw-rw-r-- 1             7.9M Sep  5 07:33 nexoedge-1.0-amd64-full.deb
-rw-rw-r-- 1            1007K Sep  5 07:33 nexoedge-1.0-amd64-proxy.deb
-rw-rw-r-- 1             546K Sep  5 07:33 nexoedge-1.0-amd64-utils.deb
```

Package Installation -- Installation Procedure

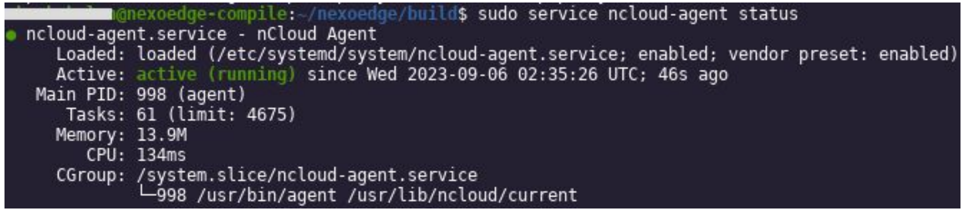
1. Either compile and generate the package from source code or download from the official repository
   a. (refer to the previous slide about the components included in each package)
2. Choose a package to install (execute only one of the following)
   a. Both proxy and agent: `$ sudo dpkg -i nexoedge-1.0-amd64-full.deb`
   b. Proxy only: `$ sudo dpkg -i nexoedge-1.0-amd64-proxy.deb`
   c. Agent only: `$ sudo dpkg -i nexoedge-1.0-amd64-agent.deb`
   d. Reporter only: `$ sudo dpkg -i nexoedge-1.0-amd64-utils.deb`
3. Complete the installation
   a. Install any outstanding prerequisites: `$ sudo apt install -f -y`
   b. Enter “yes” to enable and start the systemd service(s) for the component(s)

Package Installation -- Expected Outcomes

- Component binaries installed under `/usr/bin/`
- Libraries installed under `/usr/lib/`
- Configurations files installed under `/usr/lib/ncloud/`
  - Systemd service configurations for the proxy and/or agent
  - **Default configuration files set for the proxy and agent services: /usr/lib/ncloud/current**
  - Sample configuration files set: /usr/lib/ncloud/sample
- Systemd service(s) running, e.g., after installing `nexoedge-1.0-amd64-full.deb`
  - `$ sudo service ncloud-proxy status`　　`$ sudo service ncloud-agent status`



```
[redacted]@nexoedge-compile:~/nexoedge/build$ sudo service ncloud-proxy status
● ncloud-proxy.service - nCloud Proxy
     Loaded: loaded (/etc/systemd/system/ncloud-proxy.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-09-06 02:35:25 UTC; 42s ago
   Main PID: 947 (proxy)
      Tasks: 19 (limit: 4675)
     Memory: 4.6M
        CPU: 29ms
     CGroup: /system.slice/ncloud-proxy.service
             └─947 /usr/bin/proxy /usr/lib/ncloud/current
```



```
[redacted]@nexoedge-compile:~/nexoedge/build$ sudo service ncloud-agent status
● ncloud-agent.service - nCloud Agent
     Loaded: loaded (/etc/systemd/system/ncloud-agent.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-09-06 02:35:26 UTC; 46s ago
   Main PID: 998 (agent)
      Tasks: 61 (limit: 4675)
     Memory: 13.9M
        CPU: 134ms
     CGroup: /system.slice/ncloud-agent.service
             └─998 /usr/bin/agent /usr/lib/ncloud/current
```

Package Installation -- Other ops. and Uninstallation

- Restart the component (service)
  a. Proxy: `$ sudo service ncloud-proxy restart`
  b. Agent: `$ sudo service ncloud-agent restart`
- Remove a package (choose the one corresponds to the installation step)
  a. Both proxy and agent: $ sudo apt purge -y nexoedge-full
  b. Proxy only: `$ sudo apt purge -y nexoedge-proxy`
  c. Agent only: `$ sudo apt purge -y nexoedge-agent`
  d. Reporter only: `$ sudo apt purge -y nexoedge-utils`

SMB/CIFS Server Setup -- Preparation

1. Download the package
    a.
    ```
    $ curl -L \
    https://github.com/nexoedge/nexoedge/releases/download/v1.0.0/nexoedge-cifs.tar.gz \
    -o nexoedge-cifs.tar.gz
    ```
2. Extract the package
    a.
    ```
    $ tar zxf nexoedge-cifs.tar.gz
    ```
3. Move the precompiled Samba server to `/usr/local`
    a.
    ```
    $ sudo mv samba /usr/local/
    ```
4. Create the (default) SMB share folder
    a.
    ```
    $ sudo mkdir -p /smb/ncloud && sudo chmod 777 /smb/ncloud
    ```

SMB/CIFS Server Setup -- Installation and Setup

5. Set up the server as a systemd service
   a. Run the installation script under the ‘scripts’ folder:
      ```
      $ cd scripts && sudo bash install.sh
      ```
   b. Enter “yes” to enable and start the server systemd service
6. Add a Samba user
   a. Create the user on the operating system:
      ```
      $ sudo useradd -M nexoedge && sudo usermod -L nexoedge
      ```
   b. Create the same user in Samba:
      ```
      $ sudo /usr/local/samba/bin/pdbedit -a nexoedge
      ```
7. [Source code compilation only] Create a link to the proxy-client-communication library in the system directory
   ```
   $ sudo ln -s <absolute path to the nexoedge root directory>/build/lib/libncloud_zmq.so /usr/lib
   ```

SMB/CIFS Server Setup -- Other operations

- Check the server systemd service status
  - `$ sudo service ncloud-cifs status`
- Restart the server
  - `$ sudo service ncloud-cifs restart`
- Remove the server systemd service
  - Run the uninstallation script under the ‘scripts’ folder:

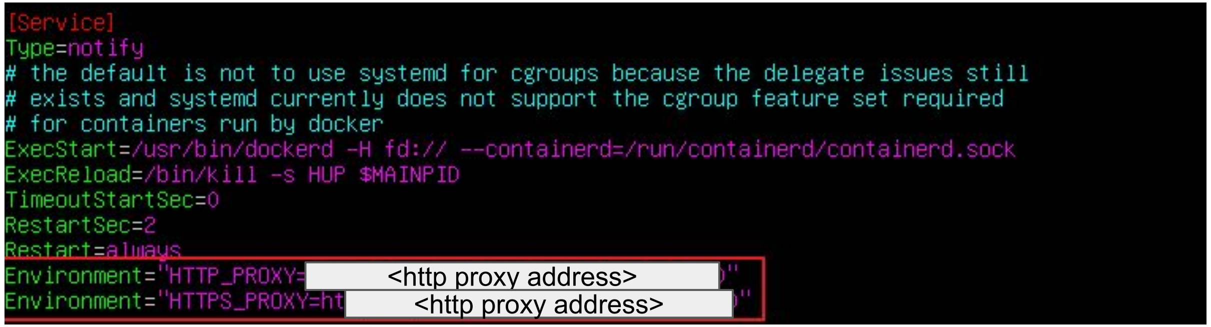
    ```
    $ cd scripts && sudo bash uninstall.sh
    ```
- Default server log path: `/usr/local/samba/var/log.smbd`
- Default server configuration path: `/usr/local/samba/var/etc/smb.conf`

Container-based Deployment -- Image Building

1. Install the latest version of Docker (under the Nexoedge root directory)
   - `$ cd docker && ./install_docker.sh`
2. Copy all the necessary packages to the ‘docker’ folder and rename them
   - All Debian packages: `nexoedge-amd-{proxy,agent,utils,full}.deb`
     - Either download from the GitHub Release section or compile from source code
     - If compiled from source code, remove the version number in the filename.
   - SMB/CIFS package (from Github Release section): `nexoedge-cifs.tar.gz`
   - Web-based portal (from Github Release section): `nexoedge-admin-portal.tar.gz`

Container-based Deployment -- Image Building

3. If the machine is behind a http proxy, set two environment variables "`HTTP_PROXY`" and "`HTTPS_PROXY`" in the service section of the file "`/lib/systemd/system/docker.service`".
   a. Below is the example settings a machine in the CSE department network.



```
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutStartSec=0
RestartSec=2
Restart=always
Environment="HTTP_PROXY=<http proxy address>"
Environment="HTTPS_PROXY=ht<http proxy address>"
```

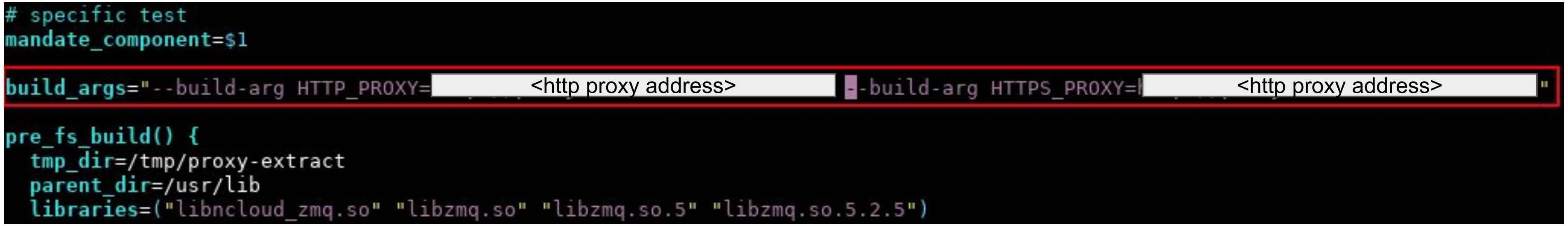
   b. Update and restart the systemd daemon service
      i. `$ sudo systemctl daemon-reload && sudo service docker restart`

Container-based Deployment -- Image Building

4. Build the Docker images
   - `$ ./build_images.sh`

     If the machine is behind an HTTP proxy, add the following to the “build_args” variable in build_images.sh

     ```
     --build-arg HTTP_PROXY=<http proxy address>
     --build-arg HTTPS_PROXY=<http proxy address>
     ```



```
# specific test
mandate_component=$1

build_args="--build-arg HTTP_PROXY=<http proxy address> --build-arg HTTPS_PROXY=<http proxy address>"

pre_fs_build() {
  tmp_dir=/tmp/proxy-extract
  parent_dir=/usr/lib
  libraries=("libncloud_zmq.so" "libzmq.so" "libzmq.so.5" "libzmq.so.5.2.5")
```

Container-based Deployment -- Cluster Run

5. Test-run a Nexoedge cluster
   a. `$ ./example/cifs/start_local_cluster.sh`
6. Run a Nexoedge cluster (locally using Docker compose)
   a. `$ cd examples/cifs/docker-compose`
   b. `$ ./cmd.sh start`
   c. `$ ./cmd.sh show`
7. Test file upload, download, and deletion
   a. `$ sudo apt install -y smbclient`
   b. `$ smbclient -U nexoedge //127.0.0.1/nexoedge nexoedge -c 'put cmd.sh'`
   c. `$ smbclient -U nexoedge //127.0.0.1/nexoedge nexoedge -c 'get cmd.sh download'`
   d. `$ smbclient -U nexoedge //127.0.0.1/nexoedge nexoedge -c 'rm cmd.sh'`

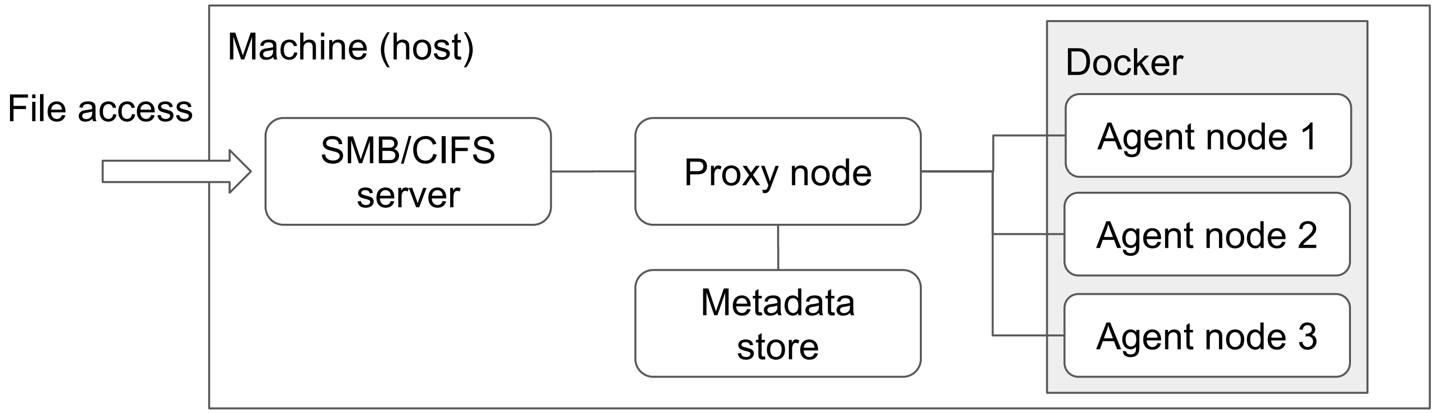
$\texttt{\$ smbclient }\underbrace{\texttt{-U nexoedge}}_{\text{User}}\ \underbrace{\texttt{//127.0.0.1/nexoedge}}_{\text{CIFS share}}\ \underbrace{\texttt{nexoedge}}_{\text{Password}}\ \underbrace{\texttt{-c 'get cmd.sh download'}}_{\text{Command}}$

Container-based Deployment -- Other Operations

- To terminate the Nexoedge cluster
  - `$ ./cmd.sh term`
- All data and states of the Nexoedge cluster is persisted to the path ‘/nexoedge-data’ by default across cluster restart (e.g., `$ ./cmd.sh start`).
- To reset everything, simply remove all content under the default path after terminating the cluster.
  - `$ ./cmd.sh term`
  - `$ sudo rm -r /nexoedge-data/*`

Deployment Example -- Architecture

In this example, we show a way to run the proxy, metadata store, and the SMB/CIFS server on a machine (i.e., a proxy node), while running 3 agent nodes as Docker containers on the same machine.

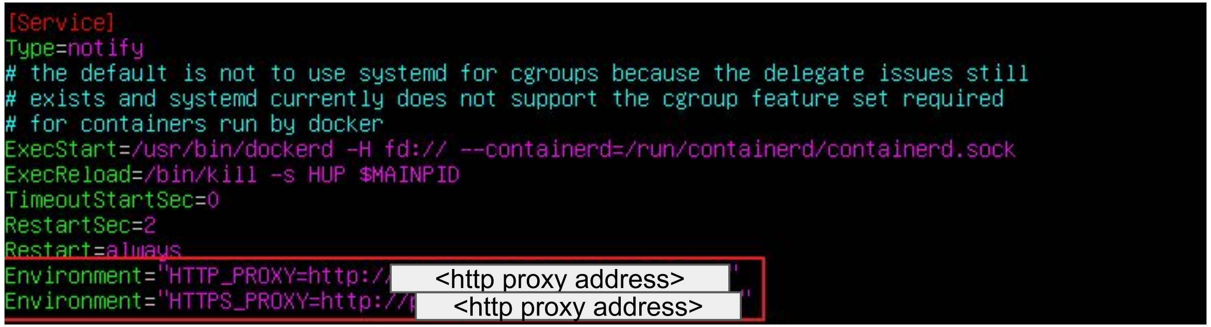


Deployment Example -- Preparation

1. Compile the source code and set up the SMB/CIFS server.
2. Under the build directory (build/), create and copy configuration files to the following directories for the proxy and agents.
   a. Proxy: `$ mkdir proxy && cp ../sample/*.ini proxy/`
   b. Agents: `` $ for i in `seq 1 3`; do mkdir agent${i} && cp ../sample/*.ini agent${i}/; done ``
3. Install Docker: `$ ../docker/install_docker.sh`

Deployment Example -- Preparation

4. If the machine is behind an http proxy, set two environment variables "`HTTP_PROXY`" and "`HTTPS_PROXY`" in the service section of the file "`/lib/systemd/system/docker.service`".
   a. Below is the example settings a machine in the CSE department network.



```
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutStartSec=0
RestartSec=2
Restart=always
Environment="HTTP_PROXY=http://<http proxy address>"
Environment="HTTPS_PROXY=http://<http proxy address>"
```

   b. Update and restart the systemd daemon service
      i. `$ sudo systemctl daemon-reload && sudo service docker restart`

Deployment Example -- Preparation

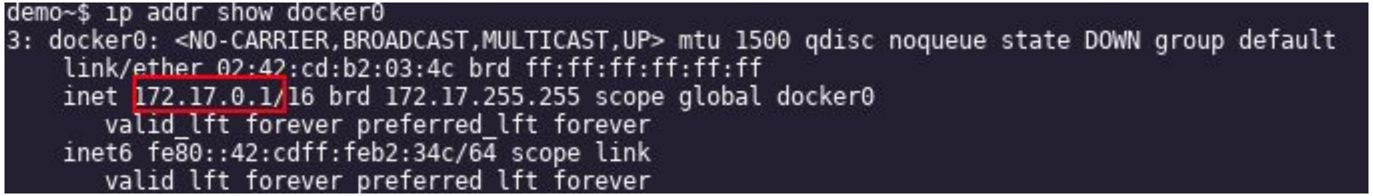
5. Download the Docker file and named it as ‘Dockerfile’.
   a.
   ```
   $ curl -L -k \
   https://gist.githubusercontent.com/helenchw/2e1711672c7ef1fdb1a130fb749bab2a/raw/d3a90b4c4ce54b0f50f1ea5bd
   c2f95a8d18e9d5d/Dockerfile -o Dockerfile
   ```
6. Build and name the image as ‘nexoedge-dev-agent’.
   a.
   ```
   $ sudo docker build -t nexoedge-dev-agent .
   ```

   If the machine is behind an http proxy, set the `HTTP_PROXY` and `HTTPS_PROXY` environment variables using `--build-arg`,

   ```
   $ sudo docker build --build-arg \
   HTTP_PROXY=<http proxy address> \
   --build-arg HTTPS_PROXY=<http proxy address> -t nexoedge-dev-agent .
   ```

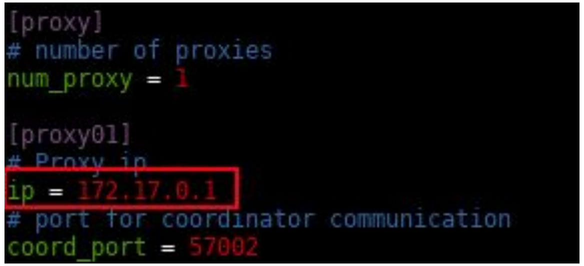
Deployment Example -- Agent Configuration

7. Set the proxy IP address in the configuration files for agents
   a. Find the host IP for the docker interface: `$ ip addr show docker0`



```
demo~$ ip addr show docker0
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:cd:b2:03:4c brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:cdff:feb2:34c/64 scope link
       valid_lft forever preferred_lft forever
```

   b. Update the ‘`ip`’ field under section ‘`proxy01`’ in ‘`agent1/general.ini`’, ‘`agent2/general.ini`’, and ‘`agent3/general.ini`’

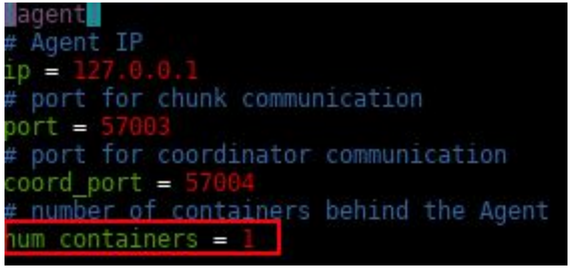


```
[proxy]
# number of proxies
num_proxy = 1

[proxy01]
# Proxy ip
ip = 172.17.0.1
# port for coordinator communication
coord_port = 57002
```

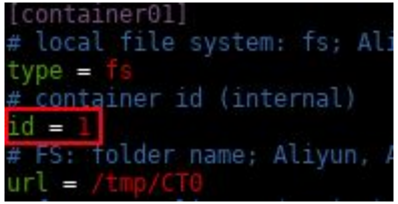
Deployment Example -- Agent Configuration

8. Update the storage container configuration in ‘`agent1/agent.ini`’, ‘`agent2/agent.ini`’, and ‘`agent3/agent.ini`’
   a. Set the ‘num_containers’ field under section ‘agent’ to 1



```
[agent]
# Agent IP
ip = 127.0.0.1
# port for chunk communication
port = 57003
# port for coordinator communication
coord_port = 57004
# number of containers behind the Agent
num_containers = 1
```

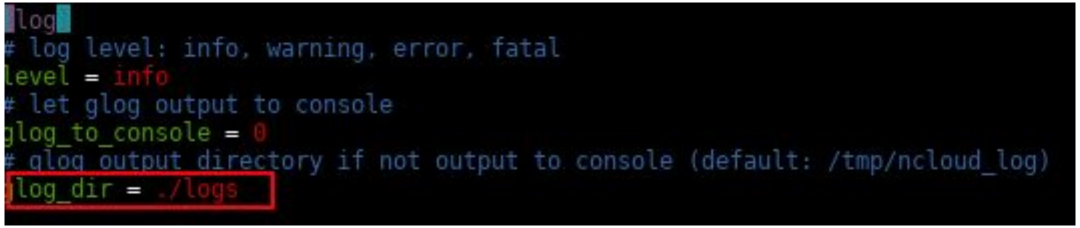
   b. Set the ‘id’ field under section ‘container01’ to unique values, e.g. 1 for agent1, 2 for agent2, 3 for agent3



```
[container01]
# local file system: fs; Ali
type = fs
# container id (internal)
id = 1
# FS: folder name; Aliyun, A
url = /tmp/CT0
```

Deployment Example -- Agent Configuration

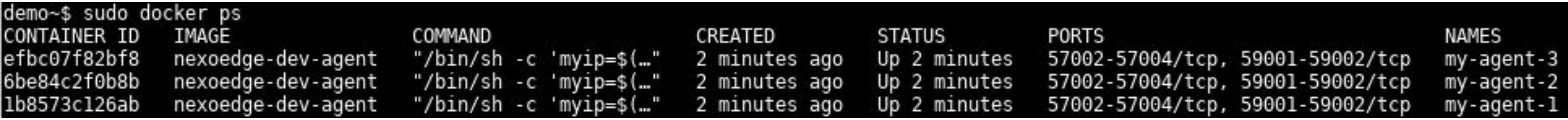
9. Update the log output configuration in '`agent1/general.ini`', '`agent2/general.ini`', and '`agent3/general.ini`'
   a. Set the 'glog_dir' field under section 'general' to '`./logs`'



```
[log]
# log level: info, warning, error, fatal
level = info
# let glog output to console
glog_to_console = 0
# glog output directory if not output to console (default: /tmp/ncloud_log)
glog_dir = ./logs
```

Deployment Example -- Start Agents

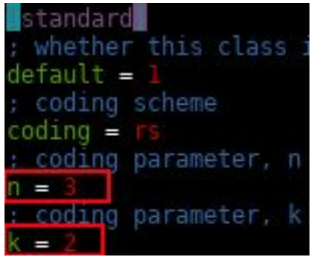
10. Run 3 containers for agents, namely ‘my-agent-1’, ‘my-agent-2’, ‘my-agent-3’, with the build directory mounted under the path ‘`/nexoedge`’ in the containers.
    a.
    ```
    $ for i in {1..3}; do \
    sudo docker run -d -e AGENT_DIR=/nexoedge/agent${i} \
    --name my-agent-${i} \
    -v .:/nexoedge \
    nexoedge-dev-agent; \
    done
    ```
    b. Check the containers are up and running: `$ sudo docker ps`



```
demo~$ sudo docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS         PORTS                                NAMES
efbc07f82bf8   nexoedge-dev-agent   "/bin/sh -c 'myip=$(…"   2 minutes ago   Up 2 minutes   57002-57004/tcp, 59001-59002/tcp     my-agent-3
6be84c2f0b8b   nexoedge-dev-agent   "/bin/sh -c 'myip=$(…"   2 minutes ago   Up 2 minutes   57002-57004/tcp, 59001-59002/tcp     my-agent-2
1b8573c126ab   nexoedge-dev-agent   "/bin/sh -c 'myip=$(…"   2 minutes ago   Up 2 minutes   57002-57004/tcp, 59001-59002/tcp     my-agent-1
```

Deployment Example -- Start Proxy and SMB/CIFS

11. Update the proxy storage configuration in ‘`proxy/storage_class.ini`’
    a. Set the field ‘n’ under the section ‘standard’ to 3
    b. Set the field ‘k’ under the section ‘standard’ to 2

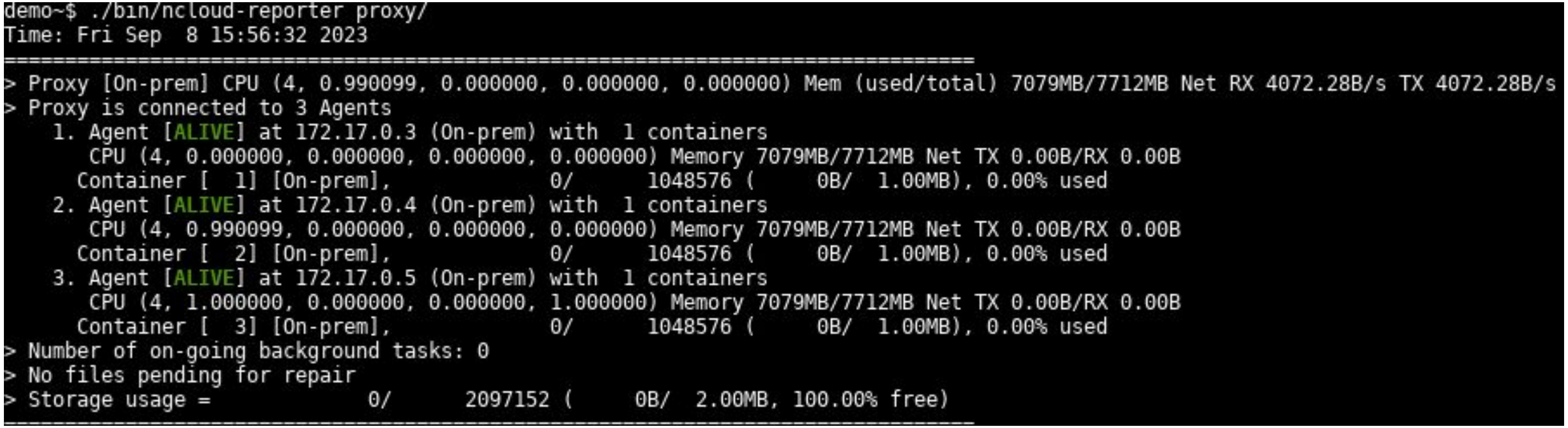


```
[standard]
; whether this class i
default = 1
; coding scheme
coding = rs
; coding parameter, n
n = 3
; coding parameter, k
k = 2
```

12. Update the log output configuration in ‘proxy/general.ini’
    a. Set the ‘glog_dir’ field under section ‘general’ to ‘`./logs`’
13. Run the proxy as a background process
    a. `$ cd proxy && ../bin/proxy &`
14. Run the SMB/CIFS server
    a. `$ sudo service ncloud-cifs restart`

Deployment Example -- Expected Outcome

- Check the cluster deployment
  - `$ ./bin/ncloud-reporter proxy`



```
demo~$ ./bin/ncloud-reporter proxy/
Time: Fri Sep  8 15:56:32 2023
================================================================================
> Proxy [On-prem] CPU (4, 0.990099, 0.000000, 0.000000, 0.000000) Mem (used/total) 7079MB/7712MB Net RX 4072.28B/s TX 4072.28B/s
> Proxy is connected to 3 Agents
    1. Agent [ALIVE] at 172.17.0.3 (On-prem) with  1 containers
       CPU (4, 0.000000, 0.000000, 0.000000, 0.000000) Memory 7079MB/7712MB Net TX 0.00B/RX 0.00B
      Container [  1] [On-prem],             0/      1048576 (     0B/  1.00MB), 0.00% used
    2. Agent [ALIVE] at 172.17.0.4 (On-prem) with  1 containers
       CPU (4, 0.990099, 0.000000, 0.000000, 0.000000) Memory 7079MB/7712MB Net TX 0.00B/RX 0.00B
      Container [  2] [On-prem],             0/      1048576 (     0B/  1.00MB), 0.00% used
    3. Agent [ALIVE] at 172.17.0.5 (On-prem) with  1 containers
       CPU (4, 1.000000, 0.000000, 0.000000, 1.000000) Memory 7079MB/7712MB Net TX 0.00B/RX 0.00B
      Container [  3] [On-prem],             0/      1048576 (     0B/  1.00MB), 0.00% used
> Number of on-going background tasks: 0
> No files pending for repair
> Storage usage =             0/      2097152 (     0B/  2.00MB, 100.00% free)
--------------------------------------------------------------------------------
```

Deployment Example -- File Operation Tests

- Test file upload, download, and deletion

  -
    ```
    $ /usr/local/samba/bin/smbclient -U nexoedge //127.0.0.1/ncloud <password> \
    -c 'put Dockerfile'
    ```

  -
    ```
    $ /usr/local/samba/bin/smbclient -U nexoedge //127.0.0.1/ncloud <password> \
    -c 'get Dockerfile download'
    ```

  -
    ```
    $ /usr/local/samba/bin/smbclient -U nexoedge //127.0.0.1/ncloud <password> \
    -c 'rm Dockerfile'
    ```

Deployment Example -- Misc Operations

- Check the log messages of the components
  - Proxy: see the folder ‘proxy/logs/’
  - Agent: see the folder ‘agent{1,2,3}/logs’
- Restart the agents, e.g., after configuration changes or recompiling the agent
  - `$ sudo docker restart my-agent-{1,2,3}`
- Reset everything
  - Stop the SMB/CIFS server: `$ sudo service ncloud-cifs stop`
  - Stop the proxy: `$ killall -9 proxy`
  - Stop all agents by removing all the containers:

    `$ sudo docker rm -f my-agent-{1,2,3}`
  - Clean the metadata store: `$ redis-cli flushdb`
  - Clean the SMB/CIFS metadata: `$ rm /smb/ncloud/*`

Deployment Considerations -- Network Req.

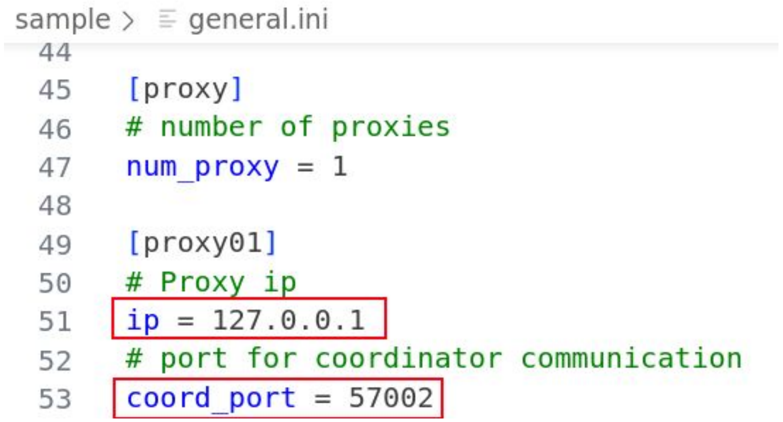
- Networking requirements for SMB/CIFS-proxy-agent communication
- Proxy:
  - 1 listening port for the agents (e.g., 57002)
  - 1 listening port for the SMB/CIFS server (e.g., 59001)
- Agent:
  - 2 listening ports for the proxy (e.g., 57003 and 57004)
  - Unique IP address for every agent

Deployment Considerations -- Data Reliability

- Choice of erasure coding parameters for data reliability
  - Number of data chunks in an erasure coded stripe: *k*
  - Number of data and parity chunks in an erasure coded stripe: *n*
  - Tolerate any (n-k) chunks failures
- Usable storage capacity and raw storage space
  - For example, given that we need to provide 100GB usable storage and there are 3 agents with 50GB raw storage each, one option is n=3 and k=2 with one 50GB storage container per agent such that the usable storage is 50GB*n/k = 50GB*3/2 = 100GB.
    - Another option can be n=6 and k=4 with each agent dividing its storage capacity into two 25GB storage containers. We need two containers per agent for this option as Nexoedge stores at most one chunk in each storage container.

Configurations -- Proxy and Agent

- Proxy configurations (for running proxies and agents):
  - Basic information: In “general.ini”, under section “[proxy01]”
    - Proxy IP address: Field “ip”.
    - Proxy port for agents: Field “coord_port”.

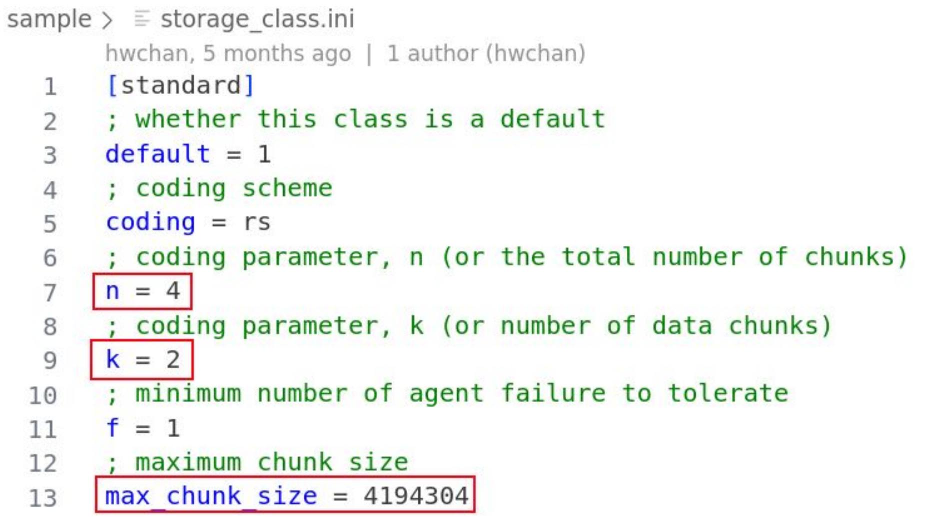

sample > general.ini

```
[proxy]
# number of proxies
num_proxy = 1

[proxy01]
# Proxy ip
ip = 127.0.0.1
# port for coordinator communication
coord_port = 57002
```

Configurations -- Proxy

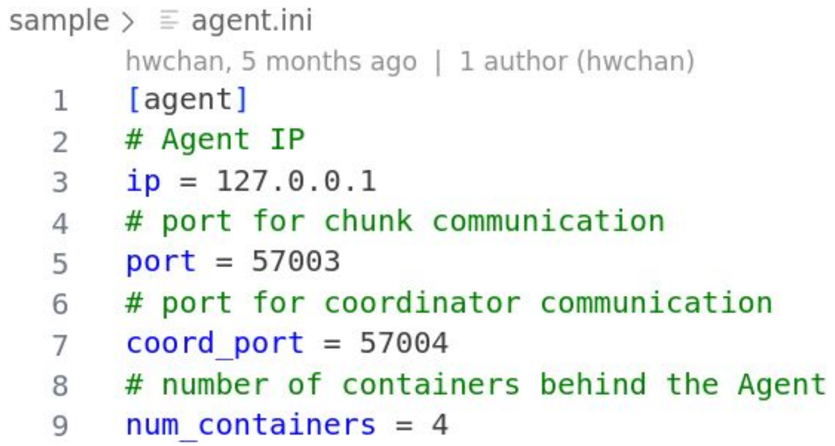
- Proxy configurations (for running proxies):
  - Data reliability: In “storage_class.ini”, under section “[standard]”
    - Number of data chunks in an erasure coded stripe: Field “k”
    - Number of data and parity chunks in an erasure coded stripe: Field “n”
    - Maximum size of a data/parity chunk: Field “max_chunk_size”


sample > storage_class.ini
hwchan, 5 months ago | 1 author (hwchan)

```
[standard]
; whether this class is a default
default = 1
; coding scheme
coding = rs
; coding parameter, n (or the total number of chunks)
n = 4
; coding parameter, k (or number of data chunks)
k = 2
; minimum number of agent failure to tolerate
f = 1
; maximum chunk size
max_chunk_size = 4194304
```

Configurations -- Agent

- Agent configurations (for running agents):
  - Basic information: In “agent.ini”, under section “[agent]”
    - Agent IP address: Field “ip”.
    - Agent port for data transfer: Field “port”.
    - Agent port for status inquiries from a proxy: Field “coord_port”.
    - Number of storage containers managed by an agent: Field “num_containers”

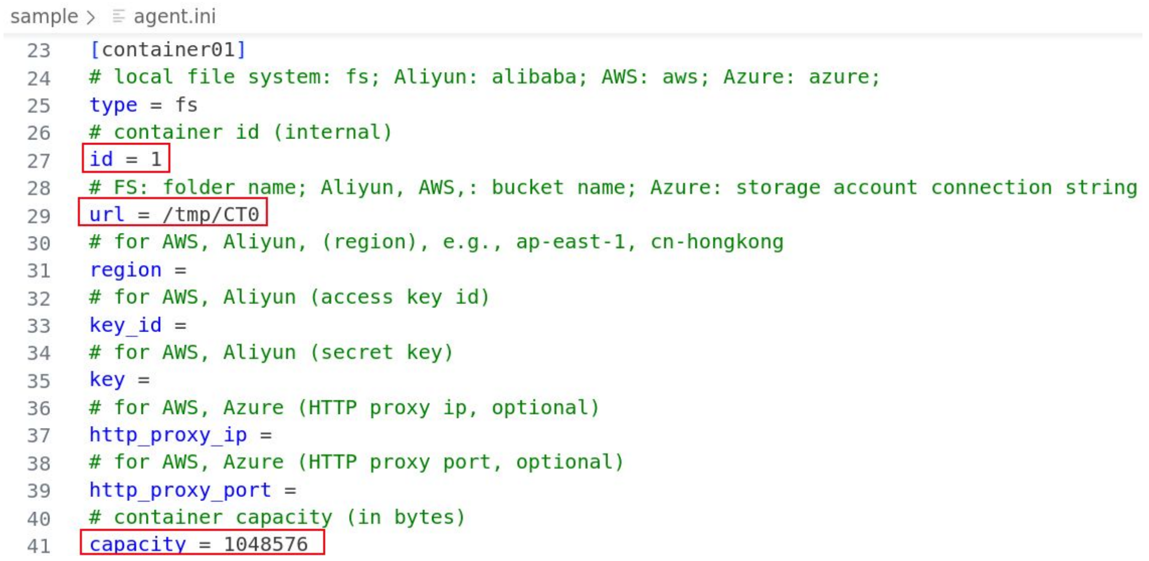

sample > agent.ini

hwchan, 5 months ago | 1 author (hwchan)

```
[agent]
# Agent IP
ip = 127.0.0.1
# port for chunk communication
port = 57003
# port for coordinator communication
coord_port = 57004
# number of containers behind the Agent
num_containers = 4
```

Configurations -- Agent

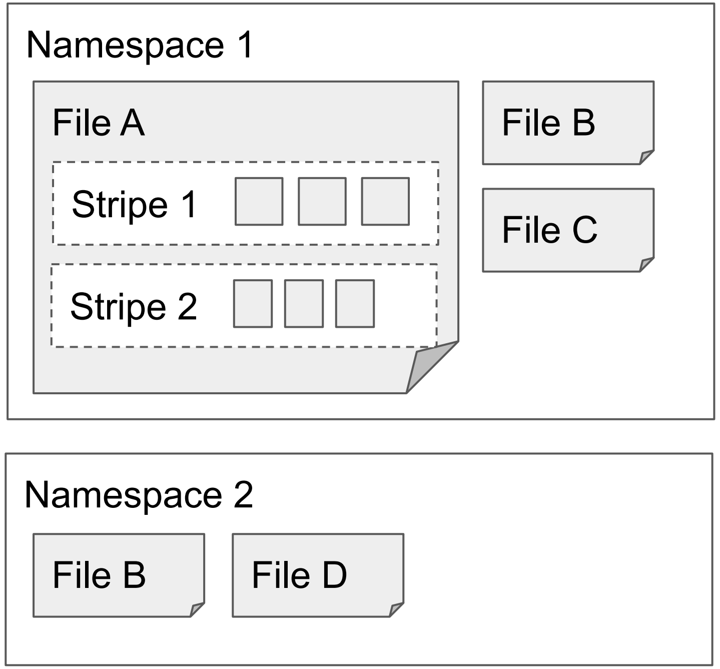
- Agent configurations (for running agents):
  - Storage container information: In “agent.ini”, under section “[containerXX]”, where XX is a left-zero-padded 2-digit monotonic increasing container sequence number.
    - Storage container id: Field “id”
    - Storage capacity: Field “capacity”
    - Storage path: Field “url”


sample > agent.ini

```
[container01]
# local file system: fs; Aliyun: alibaba; AWS: aws; Azure: azure;
type = fs
# container id (internal)
id = 1
# FS: folder name; Aliyun, AWS,: bucket name; Azure: storage account connection string
url = /tmp/CT0
# for AWS, Aliyun, (region), e.g., ap-east-1, cn-hongkong
region =
# for AWS, Aliyun (access key id)
key_id =
# for AWS, Aliyun (secret key)
key =
# for AWS, Azure (HTTP proxy ip, optional)
http_proxy_ip =
# for AWS, Azure (HTTP proxy port, optional)
http_proxy_port =
# container capacity (in bytes)
capacity = 1048576
```

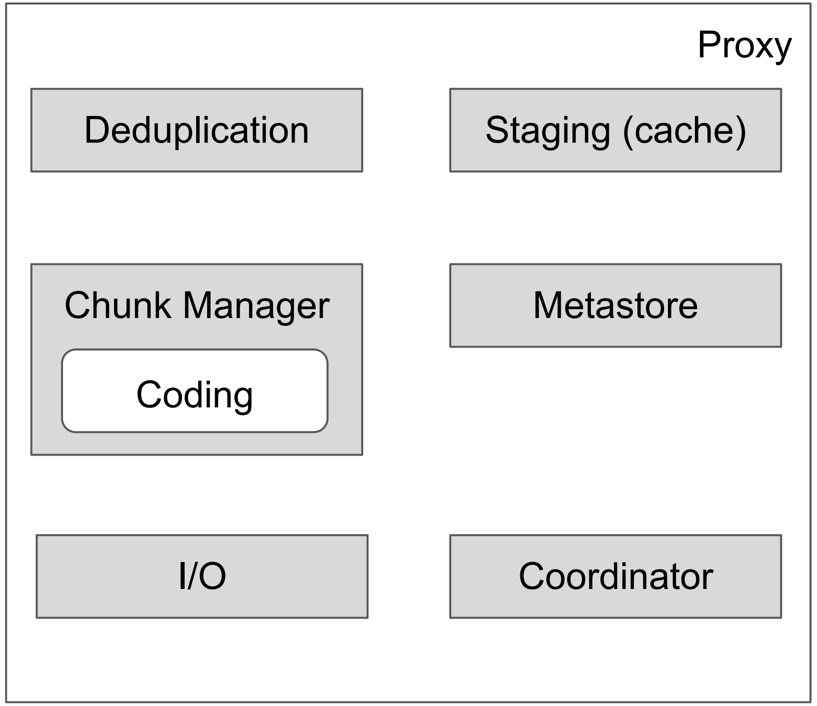
Data Model

- Each file is an independent data object.
- Each file consists of zero or more stripes.
- Each stripe consists of *n* chunks.
    - Chunks has a unique ID within the file.
    - Chunks can be of any non-zero size no larger than the maximum chunk size limit.
- Each file belongs to and is uniquely named in one (and only one) namespace.
    - Files in different namespace can have the same name.
    - Hence, a file is uniquely identified by its namespace and name.



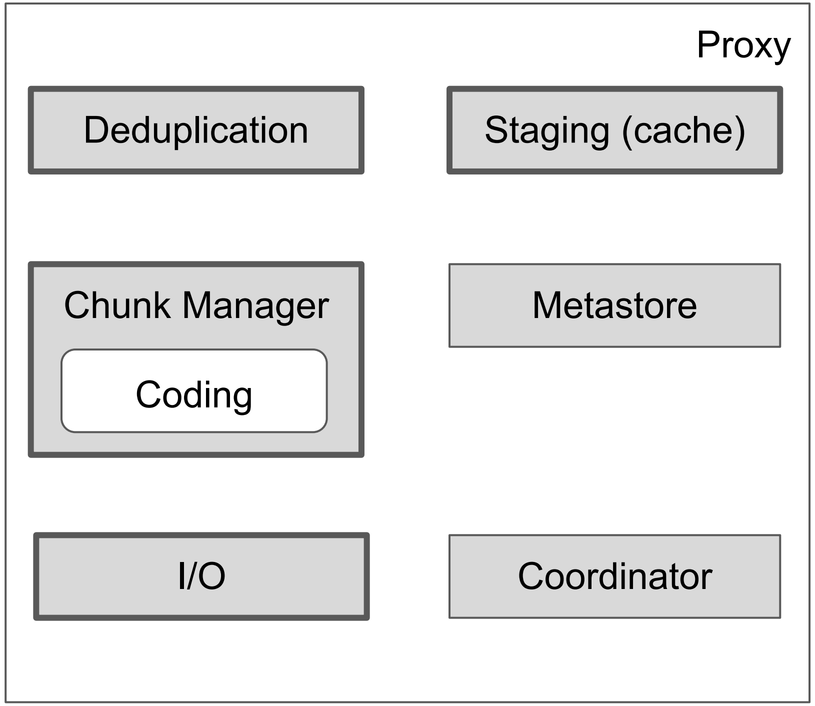
Component Modules -- Proxy

- Data Processing
  - Deduplication: Duplicate data removal
  - Chunk Manager: Data Reliability
  - I/O: Data transfer to/from agents
  - Staging (cache): Local file cache
- Metadata
  - Metastore: Metadata management
- System Status
  - Coordinator: Component connectivity and status, data placement decisions.



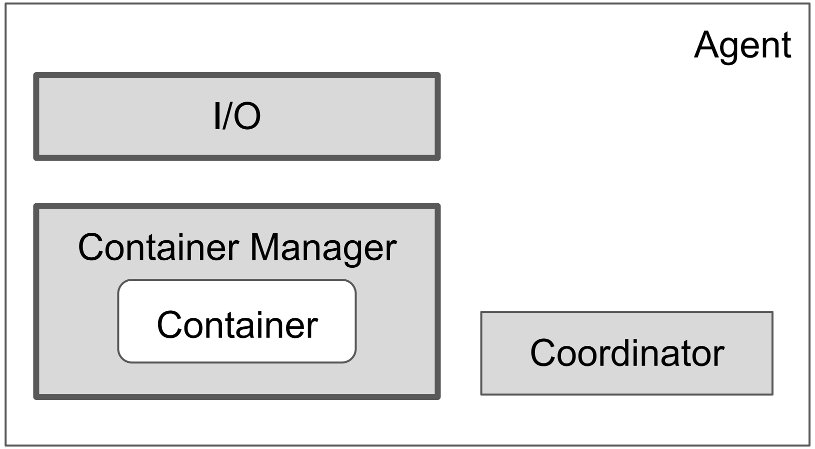
Component Modules -- Proxy Data Flow

- Staging (cache) disabled
  - Write path: Deduplication ⇒ Chunk Manager ⇒ I/O
  - Read path: I/O ⇒ Chunk Manager ⇒ Deduplication
- Staging (cache) enabled
  - Write path: Staging (cache)
  - Background write-back: Staging (cahe) ⇒ Deduplication ⇒ Chunk Manager ⇒ I/O
  - Read path: Staging (cache) or I/O ⇒ Chunk Manager ⇒ Deduplication

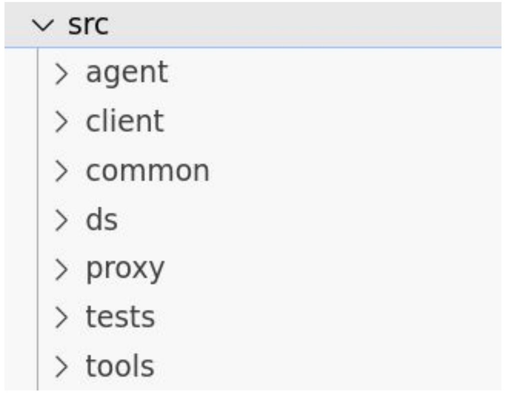


Component Modules -- Agent

- Data Processing
  - I/O: Data transfer to/from proxies
  - Container Manager: Data operations and organization in storage containers
- System status
  - Coordinator: Component status
- Data Flow
  - Write path: I/O ⇒ Container Manager
  - Read path: Container Manager ⇒ I/O

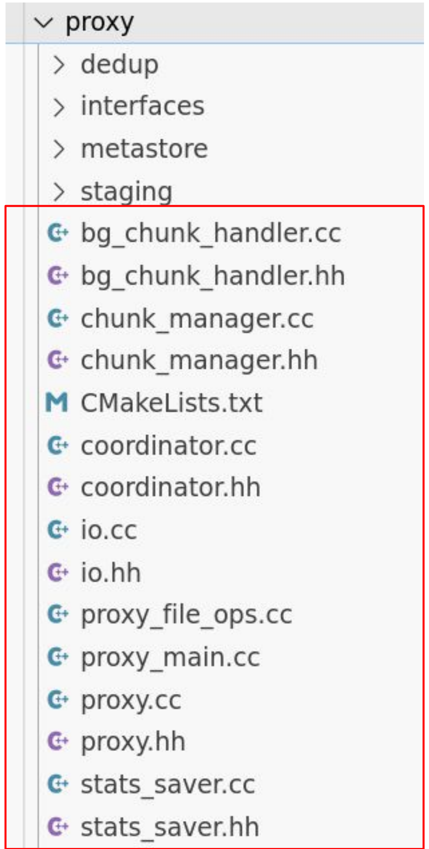


Source Code Organization -- Top Level



- **‘proxy’: Proxy modules**
- **‘agent’: Agent modules**
- **‘common’: Shared modules and definitions between proxy and agent**
- **‘ds’: Definitions for data models, events, and storage access**
- ‘client’: Clients that communicate storage request with a proxy
- ‘tests’: Unit test for modules
- ‘tools’: System tools

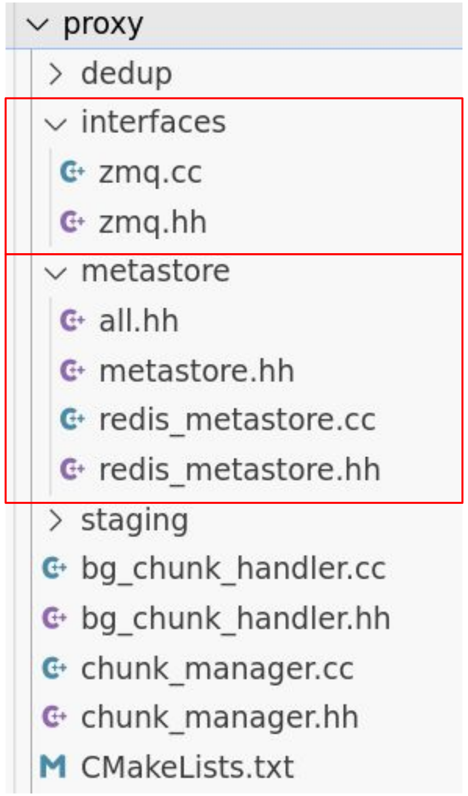
Source Code Organization -- Proxy



```
proxy
  dedup
  interfaces
  metastore
  staging
  bg_chunk_handler.cc
  bg_chunk_handler.hh
  chunk_manager.cc
  chunk_manager.hh
  CMakeLists.txt
  coordinator.cc
  coordinator.hh
  io.cc
  io.hh
  proxy_file_ops.cc
  proxy_main.cc
  proxy.cc
  proxy.hh
  stats_saver.cc
  stats_saver.hh
```

- Main body: Main function and proxy class
  - Main and proxy class: proxy_main.cc, proxy.{cc,hh}
  - **File operations**: proxy_file_ops.cc
- **Chunk manager**: Data chunking and reliability
  - chunk_manager.{cc,hh}
- **Coordinator**: Status exchange with agents
  - coordinator.{cc,hh}
- **I/O**: Data transfer with agents
  - io.{cc,hh}
- Statistics saver: Send statistics to a database
  - stats_saver.{cc,hh}
- Background chunk handler (obsolete)
  - bg_chunk_handler.{cc,hh}

Source Code Organization -- Proxy



```
proxy
  dedup
  interfaces
    zmq.cc
    zmq.hh
  metastore
    all.hh
    metastore.hh
    redis_metastore.cc
    redis_metastore.hh
  staging
  bg_chunk_handler.cc
  bg_chunk_handler.hh
  chunk_manager.cc
  chunk_manager.hh
  CMakeLists.txt
```

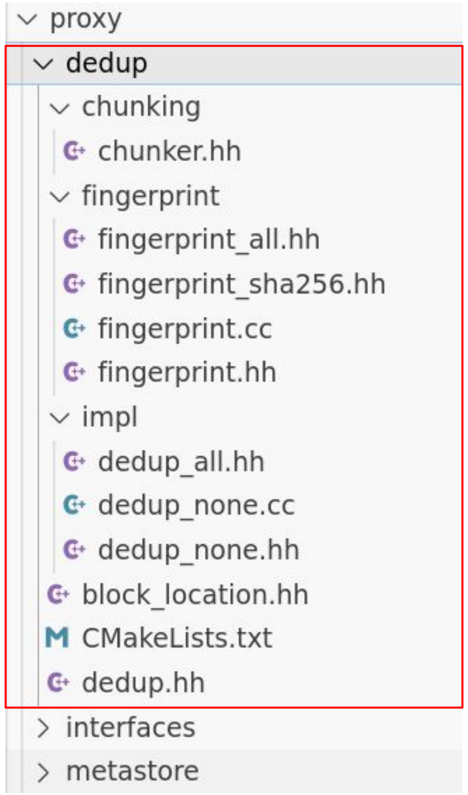
Interfaces for proxy-client communication

- Socket-based interface (using ZeroMQ)
  - interfaces/zmq.{cc.hh}

Metadata storage module

- Module interface: metastore.hh
- **Redis metadata store implementation**
  - redis_metastore.{cc,hh}

Source Code Organization -- Proxy

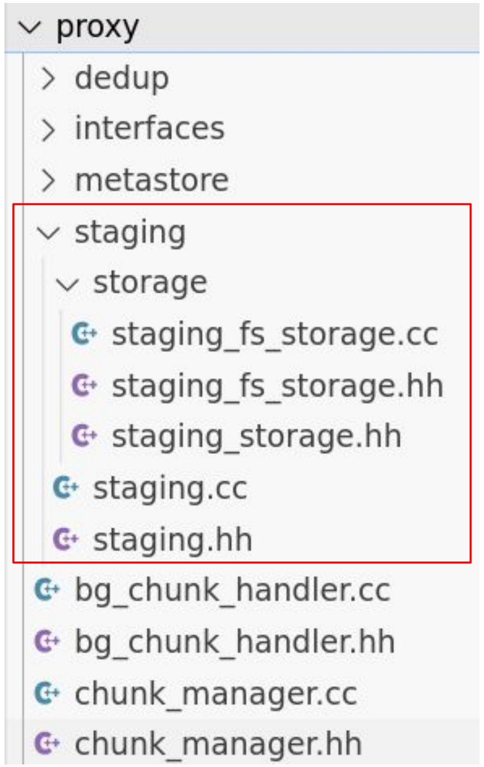

```
proxy
  dedup
    chunking
      chunker.hh
    fingerprint
      fingerprint_all.hh
      fingerprint_sha256.hh
      fingerprint.cc
      fingerprint.hh
    impl
      dedup_all.hh
      dedup_none.cc
      dedup_none.hh
    block_location.hh
    CMakeLists.txt
    dedup.hh
  interfaces
  metastore
```

Deduplication module

- Chunking algorithm: dedup/chunking/*.{hh,cc}
- Fingerprinting algorithms: dedup/fingerprint/*.{hh,cc}
- Deduplication module implementation (deduplication logic): dedup/dedup_*.{hh,cc}
- Block location with a data (stripe) unit from the proxy: dedup/block_location.hh
- Module interface: dedup/dedup.hh

(all of them follow the adaptor design pattern)

Source Code Organization -- Proxy

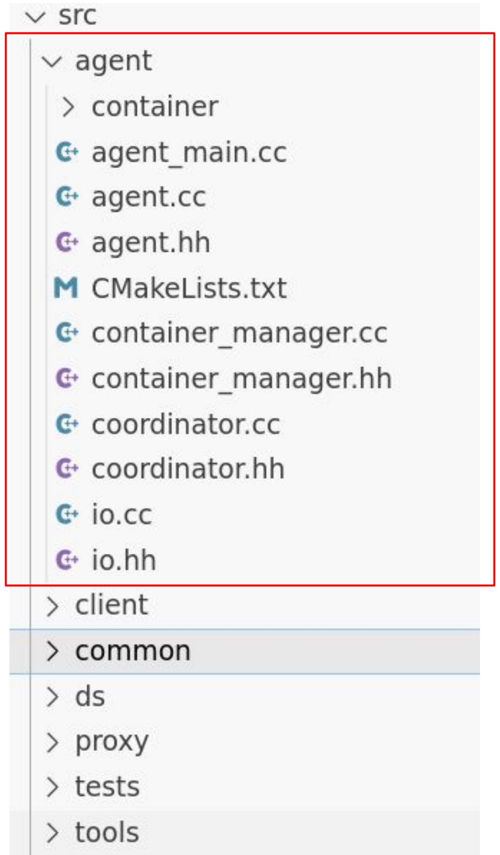


proxy
- dedup
- interfaces
- metastore
- staging
  - storage
    - staging_fs_storage.cc
    - staging_fs_storage.hh
    - staging_storage.hh
  - staging.cc
  - staging.hh
- bg_chunk_handler.cc
- bg_chunk_handler.hh
- chunk_manager.cc
- chunk_manager.hh

Cache/Staging module

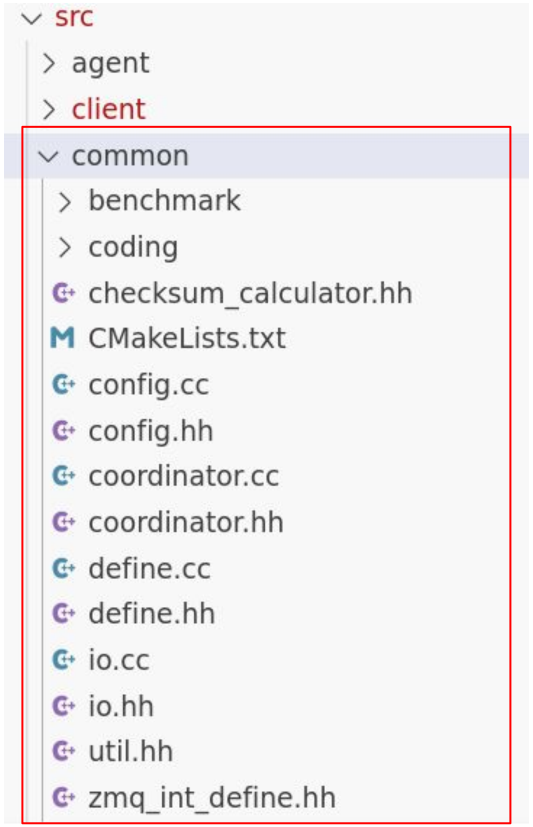
- Cache/staging area interface and management
  - staging.{cc,hh}
- Cache/staging area storage
  - Generic interface: staging_storage.hh
  - Local file system implementation: staging_fs_storage.{cc,hh}

Source Code Organization -- Agent



- Main body: Main function and agent class
  - agent.{cc,hh}, agent_main.cc
- **Storage container manager**
  - container_manager.{cc,hh}
- Storage containers: Wrapper of different storage containers (following an adaptor design pattern)
  - containers/*.{cc,hh}
- **Coordinator**: Status exchange with proxy
  - coordinator.{cc,hh}
- **I/O**: Data transfer with proxy
  - io.{cc,hh}

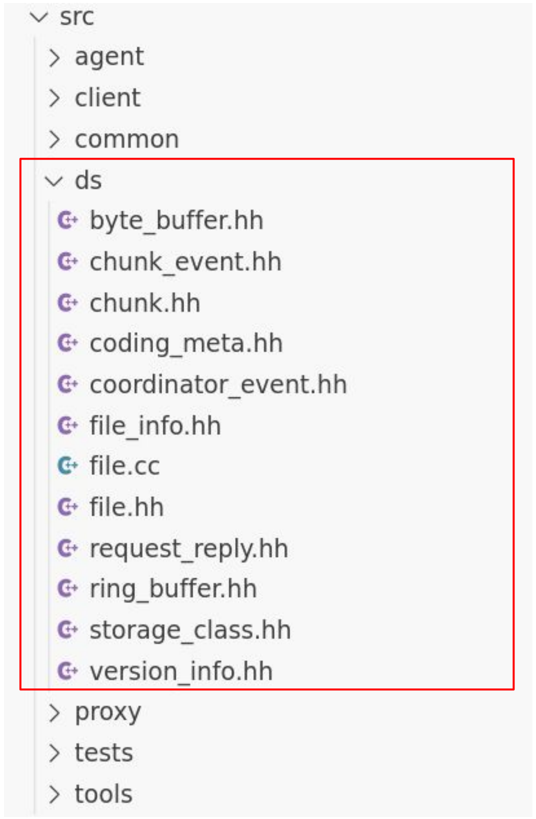
Source Code Organization -- Common



```
src
  agent
  client
  common
    benchmark
    coding
    checksum_calculator.hh
    CMakeLists.txt
    config.cc
    config.hh
    coordinator.cc
    coordinator.hh
    define.cc
    define.hh
    io.cc
    io.hh
    util.hh
    zmq_int_define.hh
```

- For performance benchmark: benchmark/*.{cc,hh}
- **Coding scheme implementation**: coding/*.{cc,hh}
    - (following an adaptor design pattern)
- Checksum computation: checksum_calculator.hh
- **Configurations**: config.{cc,hh}
- **Coordinator base**: coordinator.{cc,hh}
- Global definitions: define.{cc,hh}
- **I/O base for proxy and agent**: io.{cc,hh}
- Helper functions: util.hh
- Definitions for client-proxy communication: zmq_int_define.hh

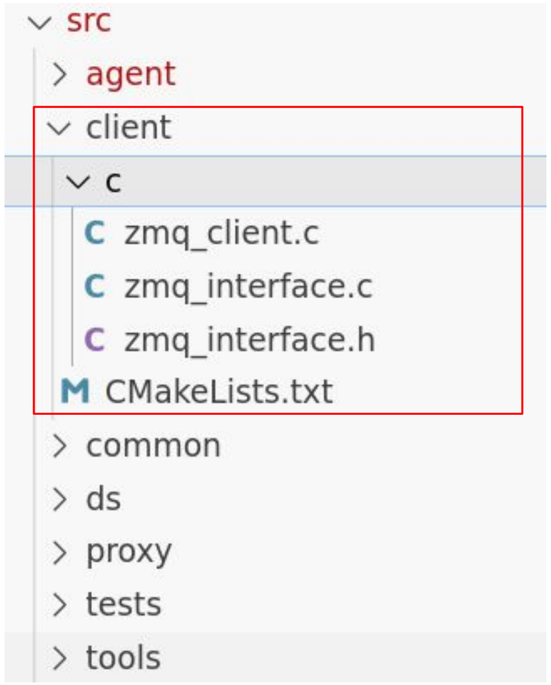
Source Code Organization -- Data Structures



```
src
  > agent
  > client
  > common
  v ds
      byte_buffer.hh
      chunk_event.hh
      chunk.hh
      coding_meta.hh
      coordinator_event.hh
      file_info.hh
      file.cc
      file.hh
      request_reply.hh
      ring_buffer.hh
      storage_class.hh
      version_info.hh
  > proxy
  > tests
  > tools
```

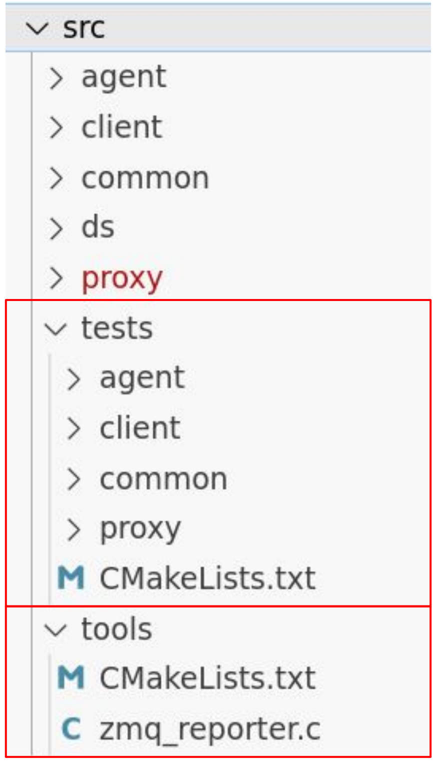
- Data model
  - **File**: file.{cc,hh}, file_info.hh
  - **Chunk**: chunk.hh
  - Storage policy: storage_class.hh
  - Coding parameters: coding_meta.hh
  - Version: version_info.hh
- Proxy-client storage request/reply: request_reply.hh
- **Proxy-agent events**
  - Data transfer: chunk_event.hh
  - System status exchange: coordinator_event.hh

Source Code Organization -- Client



- Socket-based client in C language (using ZeroMQ)
  - Storage request definition: zmq_interface.{c,h}
  - Simple client example: zmq_client.c

Source Code Organization -- Tests and Tools



```
src
  agent
  client
  common
  ds
  proxy
  tests
    agent
    client
    common
    proxy
    CMakeLists.txt
  tools
    CMakeLists.txt
    zmq_reporter.c
```

Unit Tests by components: agent, client, shared code (common), and proxy

Tools

- System status reporter: zmq_reporter.cc

Misc -- System Runtime Troubleshooting

- If the SMB server is unresponsive, check if the proxy is up and running.
- If the SMB denies writes, check if
  - The proxy is connected to sufficient number of agents (and storage containers), e.g., run $ ncloud-reporter <Nexoedge configuration directory>.
  - The mount directory in the SMB configuration is writable by the SMB user.
- If an agent fails to connect to the proxy after a proxy restart, try restarting the affected agent.
- For other issues, try checking the log files for possible causes.