SCAN

Scanning allows to capture any important details from an already-deployed OSD so that `ceph-volume` can manage it without the need of any other startup workflows or tools (like `udev` or `ceph-disk`). Encryption with LUKS or PLAIN formats is fully supported.

The command has the ability to inspect a running OSD, by inspecting the directory where the OSD data is stored, or by consuming the data partition.

Once scanned, information will (by default) persist the metadata as JSON in a file in `/etc/ceph/osd`. This `JSON` file will use the naming convention of: `{OSD ID}-{OSD FSID}.json`. An OSD with an id of `1`, and an FSID like `86ebd829-1405-43d3-8fd6-4cbc9b6ecf96` the absolute path of the file would be:

```
/etc/ceph/osd/1-86ebd829-1405-43d3-8fd6-4cbc9b6ecf96.json
```

The `scan` subcommand will refuse to write to this file if it already exists. If overwriting the contents is needed, the `--force` flag must be used:

```
ceph-volume simple scan --force {path}
```

If there is no need to persist the `JSON` metadata, there is support to send the contents to `stdout` (no file will be written):

```
ceph-volume simple scan --stdout {path}
```

DIRECTORY SCAN

The directory scan will capture OSD file contents from interesting files. There are a few files that must exist in order to have a successful scan:

- `ceph_fsid`
- `fsid`
- `keyring`
- `ready`
- `type`
- `whoami`

If the OSD is encrypted, it will additionally add the following keys:

- `encrypted`
- `encryption_type`
- `lockbox_keyring`

In the case of any other file, as long as it is not a binary or a directory, it will also get captured and persisted as part of the JSON object.

The convention for the keys in the JSON object is that any file name will be a key, and its contents will be its value. If the contents are a single line (like in the case of the `whoami`) the contents are trimmed, and the newline is dropped. For example with an OSD with an id of `1`, this is how the JSON entry would look like:

```
"whoami": "1",
```

For files that may have more than one line, the contents are left as-is, except for keyrings which are treated specially and parsed to extract the keyring. For example, a `keyring` that gets read as:

```
[osd.1]\n\tkey = AQBBJ/dZp57NIBAAtnuQS9WOS0hnLVe0rZnE6Q==\n
```

Would get stored as:

```
"keyring": "AQBBJ/dZp57NIBAAtnuQS9WOS0hnLVe0rZnE6Q==",
```

For a directory like `/var/lib/ceph/osd/ceph-1`, the command could look like:

```
ceph-volume simple scan /var/lib/ceph/osd/ceph1
```

DEVICE SCAN

When an OSD directory is not available (OSD is not running, or device is not mounted) the `scan` command is able to introspect the device to capture required data. Just like Directory scan, it would still require a few files present. This means that the device to be scanned **must be** the data partition of the OSD.

As long as the data partition of the OSD is being passed in as an argument, the sub-command can scan its contents.

In the case where the device is already mounted, the tool can detect this scenario and capture file contents from that directory.

If the device is not mounted, a temporary directory will be created, and the device will be mounted temporarily just for scanning the contents. Once contents are scanned, the device will be unmounted.

For a device like `/dev/sda1` which **must** be a data partition, the command could look like:

```
ceph-volume simple scan /dev/sda1
```

JSON CONTENTS

The contents of the JSON object is very simple. The scan not only will persist information from the special OSD files and their contents, but will also validate paths and device UUIDs. Unlike what `ceph-disk` would do, by storing them in `{device type}_uuid` files, the tool will persist them as part of the device type key.

For example, a `block.db` device would look something like:

```
"block.db": {
    "path": "/dev/disk/by-partuuid/6cc43680-4f6e-4feb-92ff-9c7ba204120e",
    "uuid": "6cc43680-4f6e-4feb-92ff-9c7ba204120e"
},
```

But it will also persist the `ceph-disk` special file generated, like so:

```
"block.db_uuid": "6cc43680-4f6e-4feb-92ff-9c7ba204120e",
```

This duplication is in place because the tool is trying to ensure the following:

Support OSDs that may not have ceph-disk special files # Check the most up-to-date information on the device, by querying against LVM and `blkid` # Support both logical volumes and GPT devices

This is a sample `JSON` metadata, from an OSD that is using `bluestore`:

```
{
    "active": "ok",
    "block": {
        "path": "/dev/disk/by-partuuid/40fd0a64-caa5-43a3-9717-1836ac661a12",
        "uuid": "40fd0a64-caa5-43a3-9717-1836ac661a12"
    },
    "block.db": {
        "path": "/dev/disk/by-partuuid/6cc43680-4f6e-4feb-92ff-9c7ba204120e",
        "uuid": "6cc43680-4f6e-4feb-92ff-9c7ba204120e"
    },
    "block.db_uuid": "6cc43680-4f6e-4feb-92ff-9c7ba204120e",
    "block_uuid": "40fd0a64-caa5-43a3-9717-1836ac661a12",
    "bluefs": "1",
```

```
    "ceph_fsid": "c92fc9eb-0610-4363-aafc-81ddf70aaf1b",
    "cluster_name": "ceph",
    "data": {
        "path": "/dev/sdr1",
        "uuid": "86ebd829-1405-43d3-8fd6-4cbc9b6ecf96"
    },
    "fsid": "86ebd829-1405-43d3-8fd6-4cbc9b6ecf96",
    "keyring": "AQBBJ/dZp57NIBAAtnuQS9WOS0hnLVe0rZnE6Q==",
    "kv_backend": "rocksdb",
    "magic": "ceph osd volume v026",
    "mkfs_done": "yes",
    "ready": "ready",
    "systemd": "",
    "type": "bluestore",
    "whoami": "3"
}
```