Ciruela Documentation

Release 0.6.12

Paul Colomiets

Oct 24, 2018

User Guide:

Github | Crate | Rust API

CHAPTER 1

Use Cases

This section describles various setups for ciruela for solving specific kinds of problems. You can mix multiple things in single ciruela server, the only current limination is: all the files and directories created by ciruela have single owner (because we don't want to run daemon as root).

In this examples we use all the defaults:

1. User running `ciruela-server` is `ciruela`
2. Configuration dir is `/etc/ciruela`
3. Ciruela port is default `24783`

1.1 Syncing Configs

Let's say we want to configure a daemon called `demonio` (which is spanish for daemon).

About `daemonio`:

1. It stores it's configs in the directory `/etc/daemonio.d`
2. It can detect configuration changes itself (See *Reloading Configs*)
3. This config of daemon is named `my_daemonio`[1]

1.1.1 Server Setup

1. Configure the service to read configs from `/etc/daemonio-configs/current` instead of `/etc/daemonio.d`

[1] If we ever run multiple instances of the daemon on the same machine or accross the same cluster we need to differentiate different config folders

Note: The point here is: ciruela can sync whole directory of configs and replace it atomically. But to do that we need to grant access for user to the parent directory of the actual config folder. We don't want to grant access to the whole `/etc`.

2. Drop the following file into `/etc/ciruela/configs/my-daemonio.yaml`:

```
directory: /etc/daemonio-configs
num-levels: 1
append-only: false
upload-keys: [my_daemonio]
```

3. Drop the **public key** allowed to upload new config into `/etc/ciruela/keys/my_daemonio.key`. (See *Client Setup*)

Note: Multiple keys can be put into the file as well as multiple files can be specified in `upload-keys` in the configuration of the directory. It's useful to create a file per actual user (potentially having multiple keys) or organize keys in any way you want.

4. Set `ciruela` as the owner of all the directories it should sync:

```
chown -R ciruela /etc/daemonio-configs
```

1.1.2 Client Setup

Human Operator Setup

This is mostly useful for testing, because work well mostly for a single user.

1. Generate a key, it's same as ssh but in `ed25519` format:

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ciruela -P ""
```

Note: Ciruela doesn't support password-protected keys yet.

This also means you can use your normal `~/.ssh/id_ed25519` key, if it isn't password-protected, but leaving ssh key plain-text isn't good idea. Ciruela keys are much more lower risk because they allow only uploading a limited set of directories rather than executing any script.

Also you may wish to delete the key and use CI system when you finish testing the setup.

2. Upload the key into `/etc/ciruela/keys/my_daemonio.key`.
3. Run the following every time you need to upload new configs:

```
ciruela sync server.name --replace local/config/path:/my-daemonio/current
```

CI Setup

Usually CI systems allow to put secret variables into environment, so you can use `CIRUELA_KEY` environment variable for storing keys.

1. Generate a key, it's same as ssh but in `id_ed25519` format:

```
ssh-keygen -t ed25519 -f tmp-key -P ""
```

2. Upload the private key into CI for `CIRUELA_KEY`, for example for travis you may use `travis encrypt`:

```
travis encrypt "CIRUELA_KEY=$(cat tmp-key)" --add
```

3. Add upload command to the task:

```
ciruela sync server.name --replace ./cfg:/my-daemonio/current
```

Reloading Configs

All of this works if your service can pick up configuration on the fly without any kind of signals.

Making signals for configuration reload is out of scope of this article but here are some ideas:

1. You can set a script that compares directory timestamp and signals service if that changes. Ciruela replaces directory atomically so reloading is safe at any time (or as quick as the directory is new)
2. You can use some special programs for that (but I'm not sure they are suited for production):
    - nodemon
    - modd
    - watchdog
3. Some supervisors like supervisord (API) and systemd (API) have RPC for the task
4. Maybe you have a UI for the service?

Just to show you that (1) is not as scareful as it sounds, here is an example script for nginx:

```
#!/bin/sh

DIR=/etc/nginx/conf
CMD="nginx -s reload"

last_stat="$(stat --format="%Z/%Y/%d:%i" "$DIR" || "<absent>")"
while sleep 1; do
  new_stat="$(stat --format="%Z/%Y/%d:%i" "$DIR" || "<absent>")"
  if [ "$last_stat" != "$new_stat" ]; then
    $CMD
  endif
done
```

Note: The script doesn't detect most changes done on individual config files, but ciruela always replaces the directory with the new one. And we detect it by checking inode number `"%i"`. Other stat parameters here are just for being more cautious.

Additional Options

- If your service has only one configuration file, you should put it into a directory anyway, as ciruela syncs directories. But it's a good idea since you can add another include file later or just put a README into the dir.

- You may want to check configs before uploading. For example run `daemonio --config=./local_config_dir --check-config` on the CI server before upload.
- You can override keys via `-i`, `-e` (see `ciruela sync --help`)
- You can upload multiple dirs simultaneously via:

  ```
  ciruela sync s1.example.org --replace ./dir1:/dest1 --replace ./dir2:/dest
  ```

- If server name resolves to multiple IP addresses, ciruela will try to upload to at most three of them (random ones if there are more) and will return non-zero exit status if none of them accepts the upload.
- Multiple names on command-line treated as a separate clusters. So ciruela will upload on three servers on each of them:

  ```
  ciruela sync s1.example.org s1.example.org --replace ./dir1:/dest1
  ```

  This will report upload progress for every cluster on it's own.

  If these are individual servers use `-m`:

  ```
  ciruela sync -m s1.example.org s1.example.org --replace ./dir1:/dest1
  ```

  With > 4 servers this makes ciruela upload to at least 75% of them and tolerate few failures. Just like it does for a single cluster name and multiple servers behind.
- Mutliple instances of `daemonio` can be configured with a single upload key you may put multiple configurations into the single directory:
  - /etc/daemonio-configs/my_daemonio
  - /etc/daemonio-configs/other_daemonio
- Or you can group all configured services under single folder (if you don't need to differentiate permissions for them):
  - /etc/syncing-configs/daemonio
  - /etc/syncing-configs/nginx
  - /etc/syncing-configs/my-other-service

1.2 Remote Editing

1.2.1 Setup

Initial setup is the same as for *Syncing Configs*.

The most important thing is this one:

```
directory: /etc/daemonio-configs
num-levels: 1
append-only: false
upload-keys: [my_daemonio]
```

Client setup is also the same. If you can `sync --replace` you can also edit some file:

1.2.2 How it Works

```
ciruela edit server.name --dir /my-daemonio/current --file /config.yaml
```

This will do the following:

1. Download a specified file from a specified directory
2. Launch whatever is specified in `CIRUELA_EDITOR`, `VISUAL` or `EDITOR` environment variables (in that order) or `vim` if none.
2. If editor is exited successfully upload the new file back to the original directory

CHAPTER 2

Configuration

2.1 Daemon Configuration

Default configuration directory is `/etc/ciruela` it's structure looks like this:

```
/etc/ciruela
├── master.key    # optional
├── peers.txt     # optional
├── configs
│   ├── dir1.yaml
│   └── dir2.yaml
└── keys
    ├── key-of-admins.key
    ├── key-of-gitlab.key
    ├── project1.key
    └── project2.key
```

More specifically:

`master.key` a plain-text list of master keys for this server. Master key is that might be used to upload data to any directory. On production deploymejnts master keys are rarely used. Format is similar to `authorized_keys` of SSH daemon (just keys no parameters): one line per key, arbitrary comment a the end.

`keys/*.key` key files that might be used in configs, any key file may contain multiple keys (similarly to `master.key` or `authorized_keys`) and any of them might be used when this name is specified in directory config

`configs/*.yaml` a config per directory. I.e. if there is `dir1.yaml`, this means you can upload to `/dir1/something...`. See *Directory Config* for more information.

`peers.txt` plain list of IP addresses and hostnames to distribute files too, only used/needed if `--cantal` command-line option is not specified.

Note: All configs are reloaded only on restart of the server. Restarting should be seamless if doesn't happen to often (if there is upload in progress, client should reconnect and continue gracefully).

2.2 Directory Config

A config in `/etc/ciruela/configs/NAME.yaml` describe synchronization of a single directory.

See *overview of the configuration* and config syntax for basics.

2.2.1 Example

This is somewhat minimal config:

```
directory: /var/containers
append-only: true
num-levels: 1
upload-keys: [user1, ci2]
```

All properties above are required, except `upload-keys`. We may make more settings optional later, when more patterns appear.

Or a longer example with auto-clean enabled:

```
# /etc/ciruela/configs/project1.yaml
directory: /var/containers/project1
num-levels: 2
append-only: true
auto-clean: true
keep-min-directories: 2
keep-max-directories: 100
keep-recent: 1 day
keep-list-file: /some/external/system/project1-used-containers.txt
```

2.2.2 Options Reference

`directory`
(required) A base path to a directory where these paths will be placed. This directory can be (temporarily) overridden in `overrides.yaml`.

`append-only`
(required) If set to `true` uploads will be rejected if a subdirectory with same name but different contents will be uploaded. It's considered good design to use `append-only: true` if possible.

It only makes sense if `num-levels` is non-zero.

Note: Under some circumstances the contents of the uploaded directory can be changed as a part of reconciliation of the cluster (i.e. if different hosts accepted different contents for the directory).

So if you have strict requirements you have to use some consistent storage to bookkeep contents (ciruela is AP system, meaning it prefers availability over consistency).

`num-levels`

(required) Number of levels of subdirectories to accept. Zero means no subdirectory, meaning the directory has to be atomically uploaded as a whole. Zero is useless with `append-only: true`. Otherwise arbitrary positive integer may be specified although some small value like 1, 2 or maybe 3 make the most sense.

Let's study some use cases:

1. `/var/containers` contain directory for containers. Each container is `/etc/containers/app.v123`. Set `num-levels` to `1` and `append-only` to `true`.
2. `/etc/nginx` contain nginx configuration. Set `num-levels` to `0`, and `append-only` to `false`. In this case you will always upload the whole nginx config and it will switch atomically.
3. `/var/indices` contains multiple indexes of some imaginary replicated DB and each index has multiple versions: `/var/indices/documents/20170101-1653`. Set `num-levels` to `2` and ciruela will automatically create first level directories and will atomically update and move second-level directories.

Note: When `num-levels` is `0` ciruela must be able to write a to the parent directory of the `directory`. For example, if you want to update `/etc/ningx`, the tool is going to write `/etc/.tmp.nginx.cr1d2e3a` then atomically move it to `/etc/nginx`.

`auto-clean`

(default `false`) Enable cleanup of this directory. Every directory up to `num_levels-1` is a separate directory to do cleanup according to `keep-*` rules.

Here is an example of a directory with auto-clean configured:

```
# /etc/ciruela/configs/project1.yaml
directory: /var/containers/project1
num-levels: 2
append-only: true
auto-clean: true
keep-min-directories: 2
keep-max-directories: 100
keep-recent: 1 day
keep-list-file: /some/external/system/project1-used-containers.txt
```

`keep-list-file`

(optional) Read the file for a list of subdirs to keep in this directory. It's needed to keep external system(s) in sync with expections.

The file is a directory name per line. If *num-levels* > 1, then the path of a directory (`dir1/dir`) per line should be specified. Intermediate directories are ignored in this case (empty intermediate directories are cleaned when empty).

Currently, we use the file to skip cleanup of the subdirectories. But we will also download the images in the list if new record appears.

Only used when *auto-clean* is enabled.

`keep-min-directories`

(default `2`) Minimum number of recent subdirectories to keep for this directory.

Only used when *auto-clean* is enabled.

`keep-max-directories`

(default `100`) Maximum number of recent subdirectories to keep for this directory.

Only used when *auto-clean* is enabled.

`keep-recent`

(default `2 days`) Keep directories uploaded within this number of days. Recent directories can be cleaned if there are more than `keep-max-directories` of them. And older directories are left only if there are less than `keep-min-directories` ones which are more recent than `keep-recent` setting.

Note: we track recency of the directory not by upload timestamp on this specific machine, but by timestamp used in signature which is created when upload was first initiated into a cluster.

CHAPTER 3

Command-line Client Usage

3.1 Client Keys

We use the same format as openssh daemon for storing keys. Currently only `ssh-ed25519` (eliptic curve) keys are supported. More key types may be supported in future.

To generate it, run:

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ciruela -P ""
```

3.1.1 Search Paths

If no identity (`-i/--identity`) or environment variables (`-k/--key-from-env`) variables are specified, we the following keys to sign uploads:

- `$HOME/.ssh/id_ed25519`
- `$HOME/.ssh/id_ciruela`
- `$HOME/.ciruela/id_ed25519`
- `$CIRUELA_KEY` environment variable

Note: We only use keys for signing and multiple signatures are okay. So we sign uploads by all the keys found at specified paths. Signing by an extra key does not compromise security.

Warning: We don't support ssh-agent and password-protected keys yet.

3.2 Sync Command

Basic command-line of sync looks like[1]:

```
$ ciruela sync --append=local-dir:/remote/dir cluster.example.org
```

This means:

1. Connect to cluster `cluster.example.org`
2. Upload a local directory `local-dir` to a virtual remote directory `/remote/dir` on whole cluster (whatever machines accept this dir)

Multiple directories can be added simultaneously as well as **multiple clusters**.

There are couple of upload modes:

- `--append` – add directory if not exists, fails if the directory exists *and* its hash doesn't match currently uploaded one
- `--append-weak` – add directory if not exists, but ignore if it exists
- `--replace` – replace a directory on the remote system(s), this only works if directory configured with *append-only* of `false`

Each cluster specified is processed by the same algorithm, which is basically:

1. Find three nodes
2. Subscribe for notifications
3. Start upload
4. Wait for all notifications to complete

More details in *How Sync Works*. All of them are processed at the same time.

If you don't have a common hostname for your cluster you may use `-m` instead:

```
$ ciruela sync --append=local-dir:/remote/dir \
  -m s1.example.org s2.example.org s3.example.org
```

This works the same but tedious to write and hard to maintain.

See `ciruela --help` for more options.

[1] You also need keys for upload. See *Client Keys*

CHAPTER 4

Version Compatibility

We're still in `0.x` series, so we have a bit relaxed version compatibility. Here is how we define it.

Protocol compatibility:

1. Client of `0.x.*` is always compatible with a server of `>= 0.x.0`
2. Servers within the same cluster are expected to be of the same version, so `0.x.y` is always compatible with `0.x.y` if versions differ there is no guarantee.

Rust API obeys Semantic Versioning.

Note: Points above are guaranteed, but because we have a single version number between client, server and rust API we sometimes break only single one, for example 0.3 and 0.4 was release only to fix API issues, client was not broken. 0.5, 0.6 were released to test major feature better so didn't break anything at all.

Precedentally, we have only broken protocol once in 0.2. But we expect it to happen again before 1.0.

CHAPTER 5

Ciruela Changes by Version

5.1 Ciruela 0.6.12

- Timeout for "index download" changed from 30 - 90 seconds
- Bugfix: if there is a file at destination location of the download, remove it (previously failed on `NotADirectory` error)

5.2 Ciruela 0.6.11

- bugfix: without auto-clean enabled ciruela was considering keep-max-directories anyway when reconciling

5.3 Ciruela 0.6.10

- bugfix: without auto-clean enabled ciruela was considering keep-max-directories anyway when scanning list of dirs at start

5.4 Ciruela 0.6.9

- Bugfix: fix check for upload finish when there is only one destination node, and cluster is bigger than one

5.5 Ciruela 0.6.8

- Feature: command-line prints public keys used for signature to stderr before upload. This makes it easier to debug keys mismatch.

5.6 Ciruela 0.6.7

- Feature: add `ciruela put-file` command that adds/replaces a single file in the target directory.
- Feature: add old image identifier support in `ciruela sync --replace` (and rust API) which means we can do (limited version of) atomic updates to the directory.
- Feature: `ciruela edit` now fails if directory was changed a remote system while you were editing a file (same failure applies for `put-file` too)
- Bugfix: when all discovered hosts have no config ciruela finishes with rejection instead of waiting indefinitely

CHAPTER 6

Network Protocols

There are three protocols used by ciruela:

1. HTTP just to show you some status pages
2. A protocol to send file data on top of websockets
3. A gossip protocol on top of UDP

In future we will probably expose some JSON API over HTTP just to allow easier interoperability with the daemon.

6.1 Websockets Protocol

We use standard websockets handshake with `Sec-WebSocket-Protocol:  ciruela.v1` and no extensions.

6.1.1 Serialization

Payload is serialized using CBOR. There are three kinds of messages:

1. Request
2. Response
3. Notification

All three types of messages can be sent at any time into any direction. Each request includes a numeric identifier that is used in corresponding response. Each side of the connection can create request identifiers independently. Each request has exactly one response. If more than one response is provided it's built by some higher level construct.

Every message is contiguous, messages can't interleaved. Protocol has no flow control besides what TCP provides. If more concurrency desired than multiple connections might be used.

We will use CDDL for describing message format. Here is the basic structure of a message:

```
message = $message .within message-structure

message-structure = [message-kind, message-type, *any] .and typed-message
message-kind = &( notification: 0, request: 1, response: 2 )
message-type = $notification-type / $request-type

typed-message = notification / request / response
notification = [0, $notification-type, *any]
request = [1, $request-type, request-id, *any]
response = [2, $request-type, request-id, *any]
request-id = uint
```

6.1.2 Signing Uploads

Signature of the upload consists of the following fields packed as the CBOR length-prefixed array in this specific order:

```
signature-data = [
    path: text,      ; destination path
    image: bytes,    ; binary hashsum of the image (bottom line of the
                     ; index file but in binary form)
    timestamp: uint, ; milliseconds since unix epoch when image was signed
]
```

Ciruela currently only supports ed25519 algorithm for signatures, but more alorithms (RSA in particular) can be used in future.

The `signature` itself is an array of at least two arguments with type as the first element and rest depends on the signature algorithm:

```
signature = ["ssh-ed25519", bytes .size 64]
```

Note: the ed25519 signature includes public key as a part of the signature as per standard. Other signatures might require different structure.

6.1.3 Commands

AppendDir

Schedule a an adding the new directory. This sends only a signed hash of the directory index and marks this directory as incoming.

Note: If different images have been scheduled for upload by different peers in the cluster cluster may end up with different images on different nodes

If upload for this path and image already exists at node another signature is added.

If there is no such index on the peer it asks this peer or any other available connection for the index data itself and subsequently asks for missing chunks (some chunks may be reused from different image).

Content of the message is a dictionary (CBOR object):

```
$message /= [1, "AppendDir", request-id, append-dir-params]
$message /= [2, "AppendDir", request-id, append-dir-response]
append-dir-params = {
    path: text,                 ; path to put image to
    image: bytes,               ; binary hashsum of the image (bottom line
                                ; of the index file but in binary form
    timestamp: uint,            ; milliseconds since the epoch
    signatures: [+ signature],  ; one or more signatures
}
append-dir-response = {
    accepted: bool,             ; whether directory accepted or not
    ? reject_reason: text,      ; a machine-parseable reason for rejection
    ? hosts: {* bytes => text}, ; hosts that will probably accept the
                                ; directory
}
```

Note: *accepted* response here doesn't mean that this is new directory (i.e. same directory might already be in place or might still be downloaded). Also it doesn't mean that download is already complete. Most probably it isn't, and you should wait for a completion notification.

The `hosts` field may or may be not sent both in case of `accepted` is true or not. In the latter case, it might be useful to reconnect to one of these hosts. In the former case, we can track `ReceiveImage` messages from all these hosts. Note: we transmit machine ids (key in mapping) and host names. Client should track notifications by machine_id, but may use name for human-readable output. Note2: while in most cases `hosts` will be exhaustive list for all clusters it may be not so, if not is just restarted and has not picked up all the data in gossip subsystem.

ReplaceDir

Schedule a replacing the directory with the new image. This sends only a signed hash of the directory index and marks this directory as incoming.

Note: If different images have been scheduled for upload by different peers in the cluster the one with latest accross the cluster timestamp in the signature will win

If there is no such index on the peer it asks this peer or any other available connection for the index data itself and subsequently asks for missing chunks (some chunks may be reused from different image).

```
$message /= [1, "ReplaceDir", request-id, replace-dir-params]
$message /= [2, "ReplaceDir", request-id, replace-dir-response]
replace-dir-params = {
    path: text,                 ; path to put image to
    image: bytes,               ; binary hashsum of the image (bottom line
                                ; of the index file but in binary form)
    ? old_image: bytes,         ; hash olf the previous image
    timestamp: uint,            ; milliseconds since the epoch
    signatures: [+ signature],  ; one or more signatures
}
replace-dir-response = {
    accepted: bool,             ; whether directory accepted or not
    ? reject_reason: text,      ; a machine-parseable reason for rejection
    ? hosts: {* bytes => text}, ; hosts that will probably accept the
                                ; directory
}
```

Note: if no `old_image` is specified the destination directory is not checked. Use `AppendDir` to atomically update first image.

See *AppendDir* for the explanation of `hosts` usage.

PublishImage

Notifies peer that this host has data for the specified index. This is usually executed before `AppendDir`, so that when receiving latter command server is already aware where to fetch data from.

```
$message /= [0, "PublishImage", publish-index-params]
publish-image-params = {
    id: bytes,                ; binary hashsum of the image (bottom line
                              ; of the index file but in binary form)
}
```

This notification basically means that peer can issue `GetIndex` in backwards direction.

ReceivedImage

Notifies peer that some host (maybe this one, or other peer) received and commited this image. The notification is usually sent after `PublishImage` for the specified id.

The notification can be used by cicuela command-line client to determine that at least one host (or at least N hosts) received the image and it's safe to disconnect from the network and also to display progress.

```
$message /= [0, "ReceivedImage", received-image-params]
received-image-params = {
    id: bytes,                ; binary hashsum of the image (bottom line
                              ; of the index file but in binary form)
    path: text,               ; path where image was stored
    machine_id: bytes,        ; machine-id of the receiver
    hostname: text,           ; hostname of the receiver
    forwarded: bool,          ; whether message originated from this host
                              ; or forwarded
}
```

The `forwarded` field might be used to skip check on `hostname` field.

AbortedImage

Notifies peer that some host (maybe this one, or other peer) have aborted receiving this image. The notification is usually sent after `PublishImage` for the specified id.

The notification can be used by cicuela command-line client to notify that image can't be written for some reason, or to determine when it's find to retry upload in case of `already_uploading_different_version` (`-x` flag of CLI).

```
$message /= [0, "AbortedImage", aborted-image-params]
aborted-image-params = {
    id: bytes,                ; binary hashsum of the image (bottom line
                              ; of the index file but in binary form)
    path: text,               ; path where image was stored
    machine_id: bytes,        ; machine-id of the receiver
    hostname: text,           ; hostname of the receiver
```

(continues on next page)

(continued from previous page)

```
    forwarded: bool,          ; whether message originated from this host
                              ; or forwarded
    reason: text,             ; reason of why image was aborted
}
```

The `forwarded` field might be used to skip check on `hostname` field.

GetIndex

Fetch an index data by it's hash. This method is usually called by server after *AppendDir* and *ReplaceDir* has been received. And it is sent to the original client (in backwards direction). But the call only takes place if no index already exists on this host or on one of the peers.

```
$message /= [1, "GetIndex", request-id, get-index-params]
$message /= [2, "GetIndex", request-id, get-index-response]
get-index-params = {
    id: bytes,                ; binary hashsum of the image (bottom line
                              ; of the index file but in binary form)
    ? hint: text              ; virtual_path where index can be found
}
get-index-response = {
    ? data: bytes,            ; full original index file
}
```

Note: index file can potentially be in different formats, but in any case:

- Consistency of index file is verified by original *id* which is also a checksum
- Kind of index can be detected by inspecting data itself (i.e. first bytes of index file should contain a signature of some kind)

Note 2: server implementation can ignore or can use `hint` value, client implementation can supply or can skip `hint`. Current state is: `ciruela upload` does not use hint, while `ciruela-server` always sends but never uses a hint value (still, the virtual path where index resides is used internally, so it may become useful in future if we will ever forward the `GetIndex` requests)

GetIndexAt

Fetch an index data by it's path. It's usually used to download image by a client (perhaps to execute modify and update cycle).

Note: image id is a part of index data so is not provided separately.

```
$message /= [1, "GetIndexAt", request-id, get-index-at-params]
$message /= [2, "GetIndexAt", request-id, get-index-at-response]
get-index-at-params = {
    path: text                    ; virtual_path to check image at
}
get-index-at-response = {
    ? data: bytes,                ; full original index file
    ? hosts: {* bytes => text}, ; hosts that contain a directory
}
```

The index file returned is a similar way to `GetIndex`. If there is no such config response may include a list of hosts to search for a directory at. Similarly to how it's done in `AppendDir` and `ReplaceDir`.

GetBlock

Fetch a block with specified hash.

```
$message /= [1, "GetBlock", request-id, get-block-params]
$message /= [2, "GetBlock", request-id, get-block-response]
get-block-params = {
    hash: bytes,                ; binary hashsum of the block
    ? hint: [text, text, uint], ; virtual_path, path, and position where
                                ; the blocks can be found found
}
get-block-response = {
    ? data: bytes,           ; full original index file
}
```

Note: server implementation can ignore or can use `hint` value, client implementation can supply or can skip `hint`. Current state is: `ciruela upload` does not use hint, while `ciruela-server` always sends and uses a hint value.

6.2 How Sync Works

This documents tries to describe what happens after you run:

```
$ ciruela sync --append=local-dir:/remote/dir cluster.example.org
```

This might look easy: just upload all the files to all the machines, but it's not so simple. Here is a non-comprehensive list of complexities:

1. We don't want to upload to all machines, because it's inefficient
2. But we want status of all uploads on all servers to be delivered to the client running `sync` anyway
3. Not all directories are accepted on all nodes, we want single entrypoint hostname for all of them anyway
4. Connections might be interrupted and some nodes are down
5. Download acks can be lost (see point above) and dir might already be there even when starting sync
6. Nodes don't have persistent connections between each other too, for efficiency

6.2.1 (0) Indexing

Some offline preparation is done: scan specified directory and make an "index" of it. Index is a file that contains list of paths and hashsums of the file contents.

6.2.2 (1) Direct Connections

First ciruela resolves `cluster.example.org` and chooses a random sample of up to **three**[1] individual IP addresses to connect to.

On each direct connection client firstly sends *PublishImage* then either *AppendDir* or *ReplaceDir* request. Upon receiving request ciruela (daemon) does the following things:

1. Checks if this directory is configured on this server

[1] Can be configured in configured. In future, we might add a command-line parameter too

2. Checks whether path exists and it's id matches request, if both are false rejects AppendDir command
3. Checks whether signature matches any of accepted keys for that directory
4. Registers that this image should be downloaded to this path

On the failure path of (1) server returns a list of hosts where to connect to. Client establishes new connections and repeats a cycle of *PublishImage* and *AppendDir* to few of the specified hosts so that number of connections to hosts which accepted directory are three.

The *PublishImage* call does two things:

1. Registers client as a "source" of the image, so that server knows where to fetch this image from[2]
2. Also server marks that this client is "watching" the download progress for this image (so that completion notifications are delivered here later)

6.2.3 (2) Download Process

The initial *AppendDir* / *ReplaceDir* kicks off the whole cluster synchronization process.

1. Right after registration initial node sends "download progress" message to few random nodes (with 0 progress at this point)[3]
2. Then ciruela computes hash of the parent directory of the uploaded path and sends that hash to few random nodes[3]
3. Each node (including first ones) starts the download from fetching index (if not already cached here)
4. Then server looks in several folders in the same dir of whether there are files which are exactly like ones being downloaded, if there are, it hardlinks all such files in the new directory.
5. Then it starts to download blocks (the actual file data)

Each index and block download works approximately by the following agorithm:

1. TBD

(TBD)

[2] We don't send actual image in AppendDir/ReplaceDir call because it's expected that either image's index or some blocks of the actual data can exist on the destination host

[3] These two messages serve different purpose. The "download progress" message is to find out where blocks of the image are already avaialable, so we can fetch them from that host. And hash of the parent directory is used to initiate downloads.

CHAPTER 7

Storage Bookkeeping and Management

The directory used for bookeeping is `/var/lib/ciruela` by default and is used for bookkeeping of what chunks are requested to be uploaded and/or downloaded or removed due to retention policy.

7.1 Signatures

Signatures are stored in `/var/lib/ciruela/signatures`. The structure of this directory somewhat replicates the structure of destination directories.

I.e. if you have a *Directory Config* `/etc/ciruela/configs/images.yaml`, which configured as *num-levels: 1*, and you have uploaded an image `hello.123`, you will have the following files:

- `/var/lib/ciruela/signatures/images/hello.123.log`
- `/var/lib/ciruela/signatures/images/hello.123.state`

First file contains just a log of signatures as they were uploaded or fetched from other hosts. The second file contains state of the destination directory.

7.1.1 State File

State file contains image hash and signatures:

```
signature_entry = [
  timestamp: uint,            ; milliseconds since the epoch when signing
  signature: signature,
]

state_file = {
  image: bytes,                         ; binary hashsum of the image
  signatures: [+ signature_entry],
}
```

See *Signing Uploads* for description of signature format

7.2 Indexes

Index is basically a list of files and directories and their checksums that will be reconstructed when image is downloaded. The indexes are stored in `/var/lib/ciruela/indexes`.

Supported index formats:

- `.ds1` – dir-signature v1

Indexes are named by it's own hash (not the hash of the index file itself but the hash that is stored in the bottom line of the index file). The first two characters of a hash file are the directory name, so the full path of the index file is:

> /var/lib/ciruela/indexes/e8/e8082d95318cb704297975988aca7b95770a3d6bb3023687dae68dcfff644d84.ds1

Note: we store all indexes here regardless of which user requested the upload and what directory it should be put into. Retention policy of index files is very much different to the retention of the files themselves. In the first implementation we keep all indexes that are used anywhere in the cluster on every node. We're considering to tighten the scope in future.

CHAPTER 8

Indices and tables

- genindex
- search

Index