Arigi Handbook

Kastelo Inc.

1.1

25/11 2020

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Functionality

Syncthing is an inherently distributed solution lacking a built in form of centralized management. Arigi provides the following centralized functionality on top of Syncthing:

- A dashboard overview of all configured devices and their folders, with indications of online/offline state, in sync/out of date status, and configuration sync status.
- Device name, tag, and folder label filtering and searching for the status dashboard.
- Device configuration management based on tags and templated configuration fragments. Devices are automatically configured based on configuration fragments matching device tags.
- An extensive JSON REST API for integrating Arigi with third party systems or dashboards.
- Aggregation and forwarding of events to external sources like ElasticSearch for further processing and dashboarding.

1.2 Principle of Operation

Arigi simplifies deployment and management of multiple Syncthing devices. Instead of configuring devices one by one, Arigi gives you a single place in which to monitor all devices and apply configuration changes. Device configurations are based on templating and tag matching - you can easily add a new folder to a group of devices, and Arigi will ensure the devices are appropriately reconfigured.

Arigi is a server component that provides a web based interface. It does not require an external web server, although it can be placed behind one for aggregation and authentication purposes.

All communication between Arigi and the managed Syncthing devices' API is initiated from Arigi and carried securely over HTTPS. Syncthing devices are located using standard Syncthing discovery mechanisms or manually configured addresses. For devices behind firewalls or otherwise unreachable via direct connection from Arigi, a tunnel server / reverse HTTPS proxy can be used. This tunnel server connects outward towards Arigi and then allows Arigi to initiate connections towards a given Syncthing device through the tunnel (see *Reverse Tunneling the API*).

Syncthing devices are enrolled with Arigi, using either of the following methods:

- By entering the GUI address (IP and port) plus a valid user ID and password. Arigi will read the API key from the device configuration and use that for further communication.
- By entering the GUI address (IP and port) plus a valid API key.
- By having Arigi act as a private Syncthing Discovery Server and automatically enroll devices registering with the discovery server (see *Automatic Enrollment*).

Once enrolled, a Syncthing device is monitored and controlled by Arigi.

1.3 Getting Help

The latest version of Arigi and the product documentation can be downloaded from https://kastelo.net/arigi/.

When logged into an Arigi instance, the About page also provides access to the matching version of the product documentation, the list of known issues, and contact information to support. Should you be unable to access the GUI, please use the following information to contact support at any time:

- Direct email to support@kastelo.net.
- Web portal at https://support.kastelo.net/

CHAPTER

TWO

INSTALLATION

2.1 Requirements

Please ensure the following requirements prior to commencing installation.

- **A Unixy server.** Arigi runs fine on Debian, Ubuntu, CentOS and RedHat Enterprise Linux. It probably runs fine on most other flavors of Linux too. FreeBSD, macOS ("Darwin") and Solaris are also good choices. The server should have Internet access. The exact resource requirements are dependent on the number of devices and their configuration; however, a baseline of 1 GiB of RAM and one CPU core is sufficient in most cases. A virtual machine is fine.
- **One or more Syncthing devices to manage.** Syncthing version 0.14 and above are supported.

2.2 Procedure

Arigi is distributed as a generic .tar.gz file containing the arigisrv binary and supporting scripts and documentation.

2.2.1 Configuration

While all day to day configuration (managed devices, etc.) is done through the Arigi GUI, some server wide settings must be provided at started.

All options can be set as flags or environment variables. The environment variable corresponding to each flag is named in upper case, with underscores and the `ARIGI_` prefix. For example, the option `--listen-address` can be set by the environment variable `ARIGI_LISTEN_ADDRESS`. For boolean flags, use the environment values `true` or `false`.

The following general options are available:

--check-interval Interval between device status checks. Accepts a number with a unit prefix (`h`, `m`, or `s`) (default: `5m`)

--concurrency Maximum number of parallell HTTPS requests that may be outstanding at any given moment (default: `100`)

--disable-configurator Do not push configuration to devices

--disable-v4-disco Do not use standard global discovery servers over IPv4

--disable-v6-disco Do not use standard global discovery servers over IPv6

--custom-disco=URL Use custom discovery server (can be given multiple times)

--server--external-url The external, visible URL to Arigi (default `https://localhost:2525`)

--home Directory for persistent data (default `/opt/var/arigi`)

--listen-address Address to listen on (default `:2525`)

--listen-insecure Disable HTTPS (see *Authentication & HTTPS*)

--listen-min-tls-version 1.0, 1.1, or 1.2 (default `1.2`)

SMTP options (see *Configuring SMTP*):

--smtp-server SMTP server address

--smtp-user SMTP auth user

--smtp-password SMTP auth password

--smtp-from SMTP from address

ElasticSearch options:

--elastic-url ElasticSearch URL

Automatic enrollment options (see *Automatic Enrollment*):

--enrollment-address Address to listen on for enrollment server

--enrollment-api-key Enrollment server default API key

--enrollment-api-port Enrollment server default API port (default `8384`)

--enrollment-tag Enrollment server tags for new devices (option can be repeated)

LDAP authentication options (see *LDAP Authentication*):

--ldap-server Address to an LDAP server, host:port (option can be repeated)

--ldap-bind-user Bind userID or DN

--ldap-bind-password Bind password

--ldap-search-base Search base DN

--ldap-search-pattern Search pattern, `%s` placeholder (default `(|(sAMAccountName=%s)(mail=%s))`)

--ldap-use-ldaps Use LDAPS for the connection

--ldap-use-starttls Use StartTLS for the connection

Backup and restore:

--backup-to Create backup of the database

--restore-from Restore backup of the database

2.2.2 Installing

Untar the distribution into a suitable directory. Our recommended default is /opt, resulting in the default Arigi installation directory /opt/arigi.

```
# mkdir -p /opt # cd /opt
# tar zxvf ~/arigi-linux-amd64-v1.0.tar.gz
```

You also need a directory for persistent data. The recommended location is the default, /opt/var/arigi. Make sure this directory is owned by the non-root user that will be running Arigi.

```
# mkdir -p /opt/var/arigi
# chown arigi:arigi /opt/var/arigi
```

Any other existing, local user can be used instead of arigi:arigi. Edit the default variables in /opt/arigi/bin/start.sh to set the directory paths in use and the database parameters. To run Arigi you use the start.sh script. It is highly recommended to run Arigi as a separate, non-root user.

```
# su - arigi
$ /opt/arigi/bin/start.sh
```

In summary, you should now have the following files and directories in place:

- `/opt/etc/arigi/`: Directory for read-only configuration.
- `/opt/arigi/`: Directory for the distribution.
- `/opt/arigi/bin/`: Directory for executable files.
- `/opt/arigi/bin/arigi`: A CLI interface to Arigi.
- `/opt/arigi/bin/arigisrv`: The main Arigi binary.
- `/opt/arigi/bin/start.sh`: The startup helper script.
- `/opt/var/arigi/`: Directory for read-write data.

2.2.3 Running

Use your favorite operating system method to keep Arigi running. Possibilities include systemd, runit, daemon-tools, and cron. The arigi binary, and hence the `start.sh` script, will not exit unless Arigi encounters a fatal error.

2.3 Authentication & HTTPS

Arigi by default uses HTTPS and creates a certificate pair on startup if one is missing. It is recommended that this certificate either be replaced with a CA issued certificate, or added

to the local trust store.

It is also possible to run Arigi in HTTP only mode behind a secure reverse proxy, such as Nginx, Apache or Caddy.

Please note that the arigi command line tool requires that a trusted certificate is used, whether CA issued or by local trust policy. As an alternative, plain HTTP can be used with the `--listen-insecure` flag to arigisrv and the `--api-insecure` flag to arigi.

Note: The default credentials on initial login are user ID `admin` and password `arigi`.

2.4 Docker Image

For quick deployment and testing there is a Docker image available.

```
# docker run -d --name arigi --restart=always \
    -p 2525:2525 kastelo/arigi
```

Give the container a few seconds to start, and then visit http://localhost:2525/ (or the address of the host running the container if not localhost). Data in the container is stored in /opt/var/arigi. You can bind a volume in order to get persistent storage independent of the container.

Arigi will generate and use a self signed key pair if one is not already present on the volume.

2.5 Configuring SMTP

TBD

2.6 LDAP Authentication

Arigi can be configured to delegate authentication to an external LDAP source. Typical examples include Microsoft Active Directory and OpenLDAP / OpenDirectory servers. When LDAP authentication is configured the user cannot change their password, email or display name through Arigi. Instead these changes should be done in the LDAP / Active Directory system directly.

To enable LDAP authentication, the following must be set:

- `--ldap-server`: Set to the address of the LDAP server, with hostname and port. For example, `dc1.example.com:389` for standard LDAP, or `dc1.example.com:636` for LDAPS. This option can be repeated to enable multiple servers for failover.
- `--ldap-bind-user` and `--ldap-bind-password`: Set to the user and password for performing directory searches. The user name should be in the format accepted by the LDAP server, typically as a full DN (OpenLDAP) or user@domain (AD).

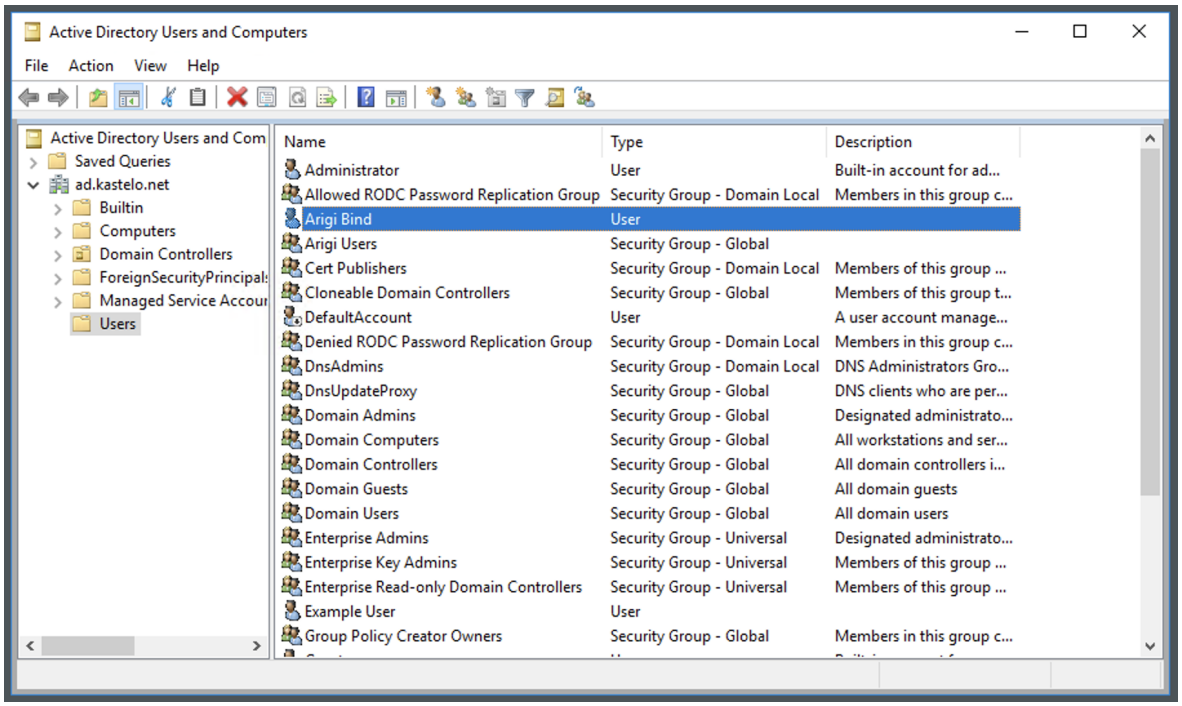
- `--ldap-search-base`: Set to the base DN under which users are found in the directory. Typically, for an AD domain "example.com", something like `CN=Users,DC=example,DC=com`.

To use TLS for connection security, set either `--ldap-use-ldaps` (server port should be 636) or `--ldap-use-starttls` (server port should be 389). In either case the server needs a correct, valid certificate matching the given server hostname.

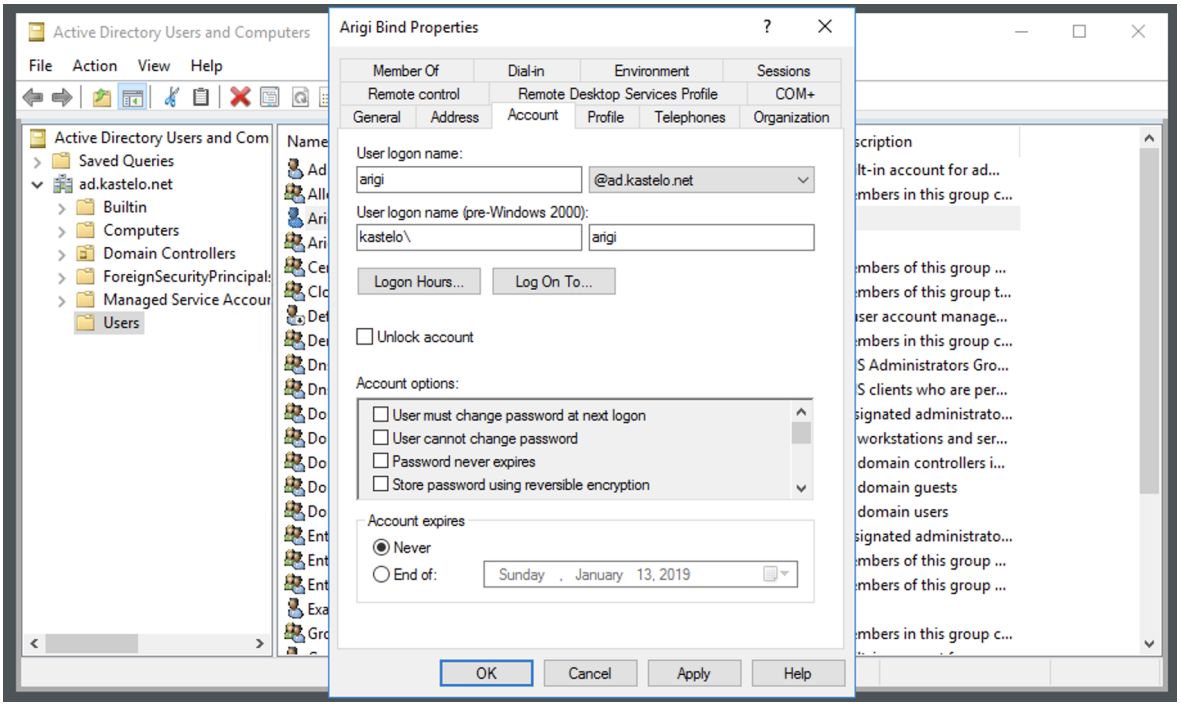
The `--ldap-search-pattern` option is an LDAP search filter to match on the user ID entered by the user in the login form. The default is suitable for AD and matches on either account name or email address.

2.6.1 Active Directory Example Configuration

Consider the following typical AD setup:



We have a domain called `ad.kastelo.net` and a user called "Arigi Bind" that we will use for searches. In the user account details we can see the logon name that we will use:



In this setup we will use the following options:

```
--ldap-server dc1.kastelo.net:389
--ldap-bind-user arigi@ad.kastelo.net
--ldap-bind-password ...
--ldap-search-base CN=Users,DC=ad,DC=kastelo,DC=net
```

The other options are acceptable at their defaults. Users will be able to use their logon name (sAMAccountName) or email to log in.

2.6.2 Enforcing Group Memberhip

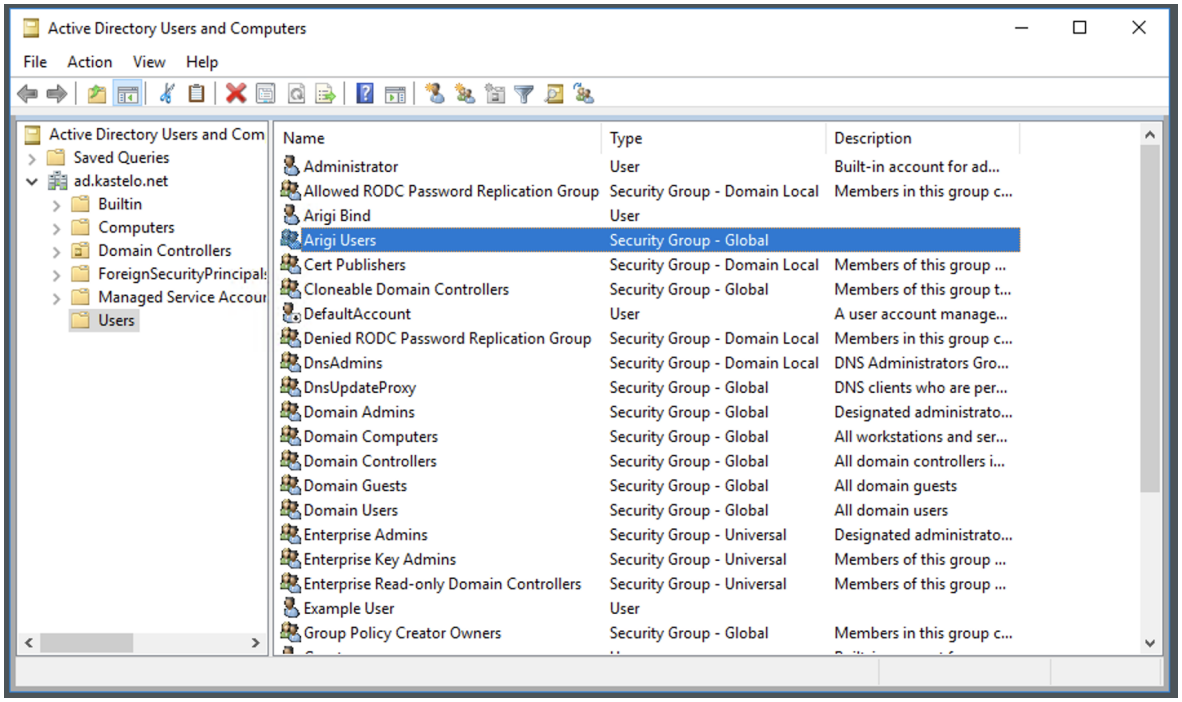
In some cases it may be desirable to restrict login to members of a certain group. We do this by using a custom search filter that matches users who belong to the group only.

The default search filter is:

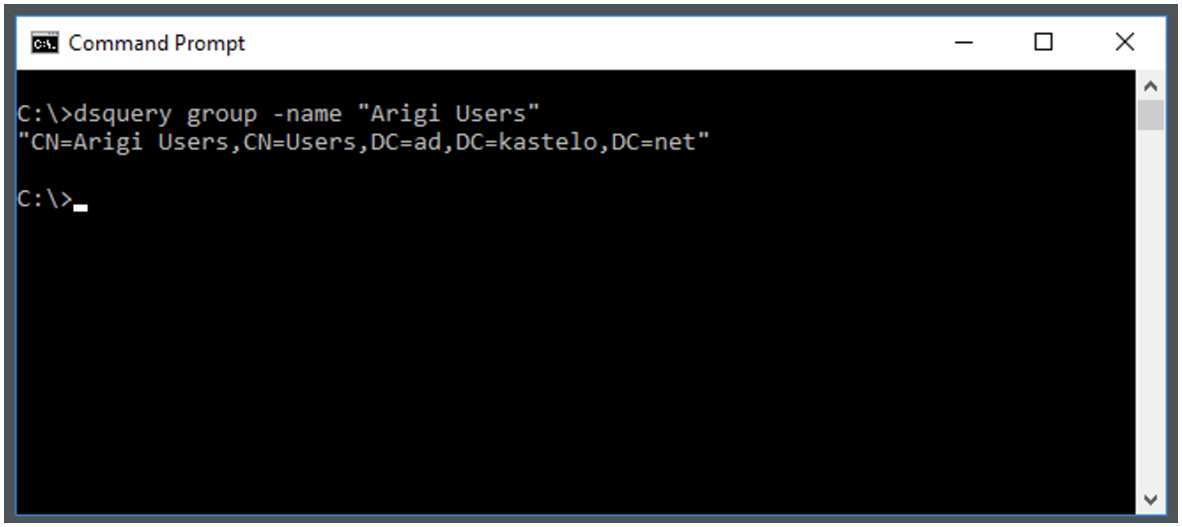
```
(|(sAMAccountName=%s)(mail=%s))
```

When this filter is used, the placeholder `%s` is replaced by the user name entered in the login form. The two terms `(sAMAccountName=...)` and `(mail=...)` then search for users with a matching logon name or email address. The outer vertical pipe `(|...)` is LDAP syntax for "or", meaning we accept matches on either logon name or email address. In order to modify this to match only users belonging to a group, we must first find the group DN.

For the following group:



We can deduce the DN by the domain name and "Users" container to be `CN=Arigi Users,CN=Users,DC=ad,DC=kastelo,DC=net`. We can also use the command line tool `dsquery` to find the same information:



Whichever method we use, we can now add it to the search filter. We need to take the existing, default, search filter and combine it with a memberOf search on the group name using an "and" criteria. By itself, that looks like:

```
(&(memberOf=CN=Arigi Users,CN=Users,DC=ad,DC=kastelo,DC=net)...)
```

where the "..." part is the original search pattern. The total search query option that we use then becomes:

```
--ldap-search-pattern
  "(&(memberOf=CN=Arigi Users,CN=Users,DC=ad,DC=kastelo,DC=net)
```

(continues on next page)

(continued from previous page)

```
  (|(sAMAccountName=%s)(mail=%s)))"
```

(Line breaks for clarity only.)

CHAPTER

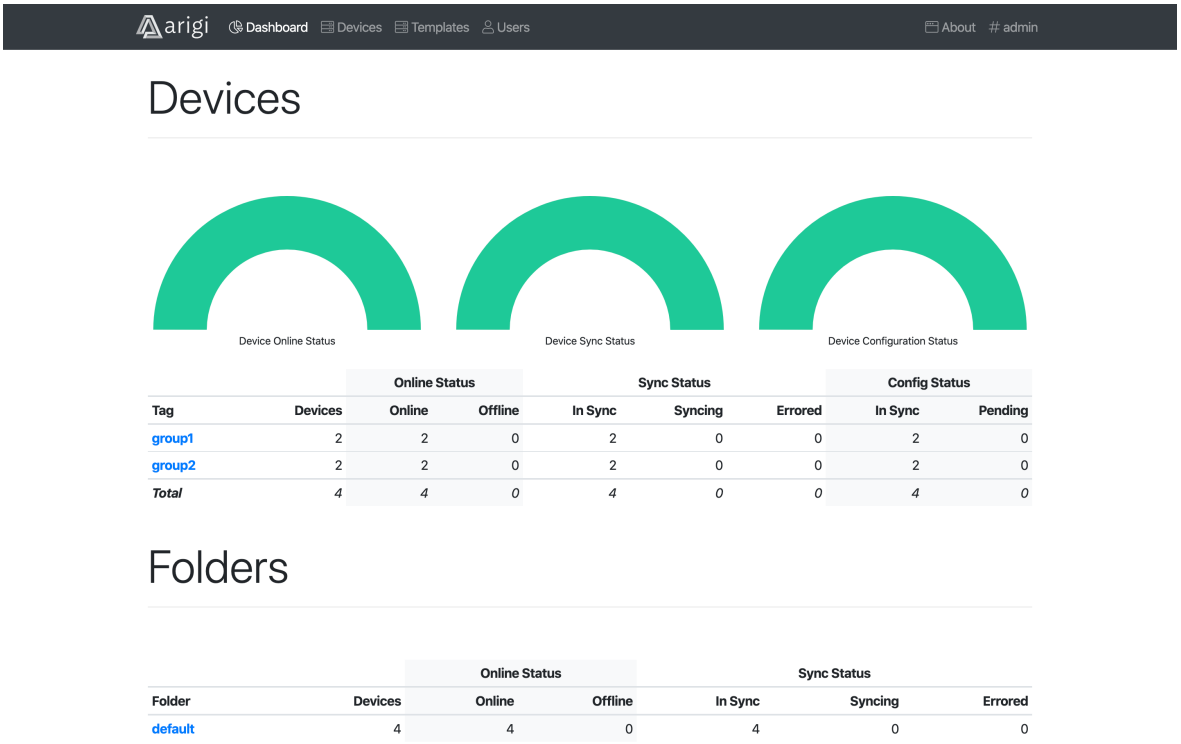
THREE

DASHBOARD

3.1 Overview

The Dashboard is the initial view or front page of Arigi. It shows a summary of the following items:

- Number of devices per tag with online, offline, in sync, syncing, errored, with config up to date, or with config pending update status.
- Number of devices per folder with online, offline, in sync, syncing, or errored status.
- Devices that are experiencing an error or that are offline due to being unreachable by Arigi, with the corresponding error.
- Each view can be filtered by tags, folder ID or folder label. Folder labels can be edited directly in the folder list by clicking on the corresponding label / ID.



3.2 Summary Indicators

At the top of the dashboard there are several graphical indicators that give a quick overview of the current status of all devices.

On the left is the device connection status graph. It indicates the fraction of configured devices that are currently online and reachable by Arigi. In the example, one device out of three is unreachable, resulting in one third of the indicator being red. The colors in the connection status graph are green for online devices, and red for offline devices.

In the middle is the device sync status graph. It indicates the sync status of the configured devices. The colors in this graph are green for online devices that are in sync, blue for online devices that are syncing, yellow for online devices that are experiencing a sync error, and greyfor offline devices (unknown sync status).

To the right is the device configuration sync status graph. It indicates the configuration sync status of the configured devices. The colors in this graph are green for online devices where the configuration is in sync, yellow for online devices that are pending an update to the configuration, and grey for offline devices (unknown configuration sync status).

3.3 Devices Table

The table below the indicators contain the same information in a more detailed manner, broken down per tag. Please note that a given device may be accounted under more than one tag. The coloring in the table is green for positive values (online, in sync), red for values indicating a problem (offline devices), yellow for values indicating a possible but less serious problem (sync error, configuration pending). Note that devices that are not online do not have a determined sync or configuration status and are thus not accounted in those sections.

CHAPTER

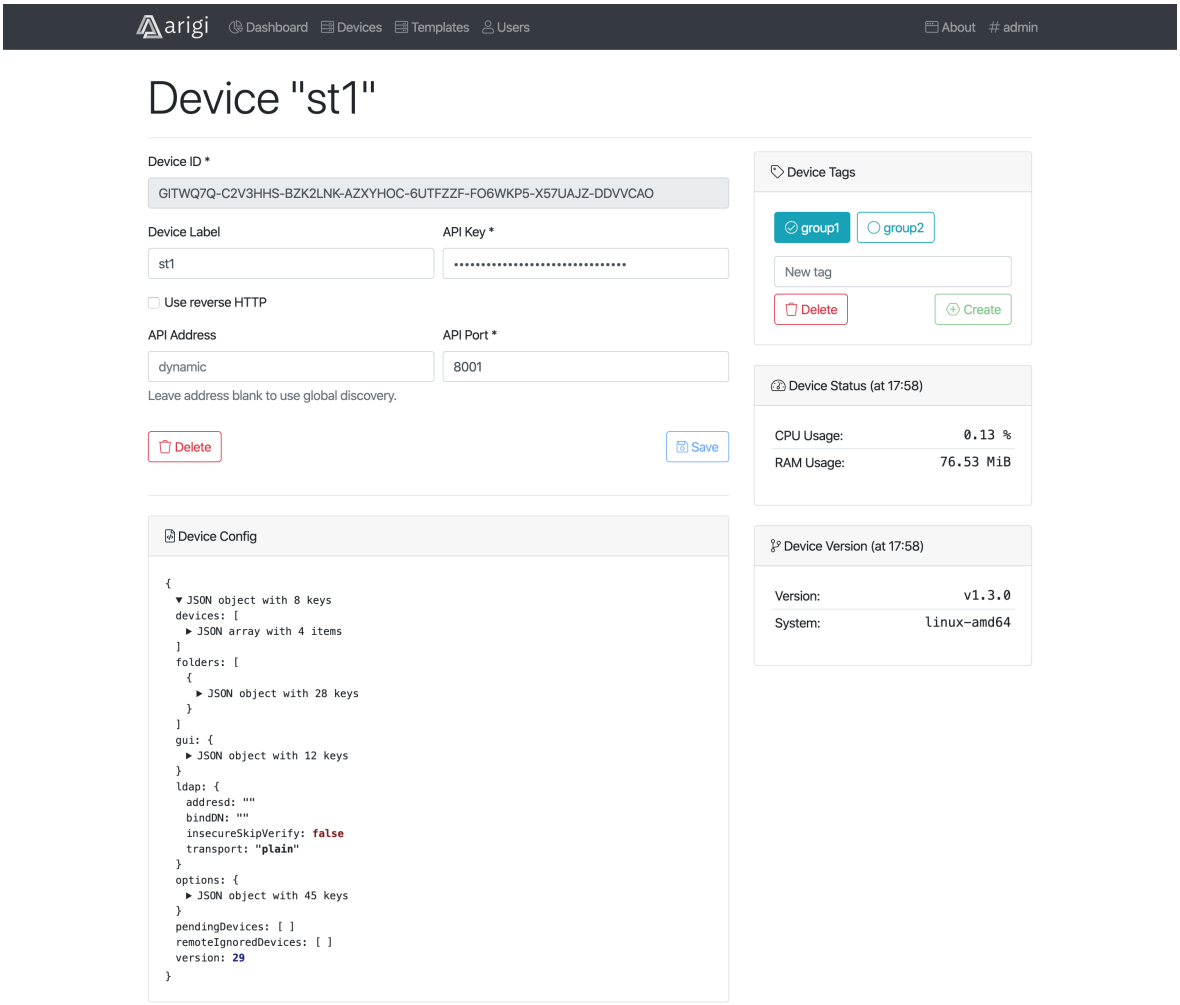
FOUR

DEVICES

4.1 Overview

The Device tab used to enroll (add) and edit devices. For Arigi to be able to manage a device it must know:

- The device ID (Actions > Show ID in Syncthing)
- The device address, which may be an IP address, a domain name, or blank to use the standard Syncthing global discovery infrastructure.
- The device API port. This is the GUI/API listen port in Syncthing and defaults to 8384.
- The API key (also shown in Action > Settings) in Syncthing.



Each device can have a label. The label is strictly for display purposes and doesn't affect the Arigi functionality.

Once a device is saved you can add tags to describe it and link it to templates.

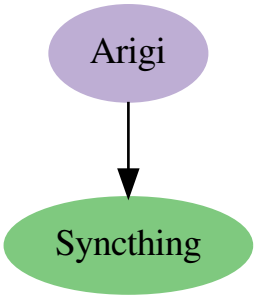
When opening an existing device, the device tab includes some extra information picked up from the device: its version, CPU utilization, etc.

4.2 Automatic Enrollment

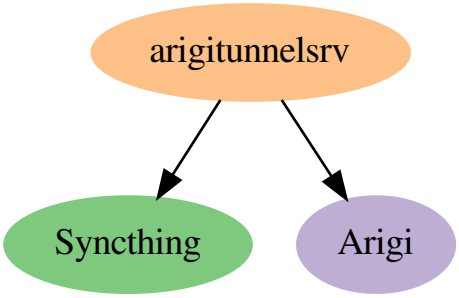
TBD

4.3 Reverse Tunneling the API

The usual flow of API access is for Arigi to make the API connection to the Syncthing device. This uses the configured API port and a configured or discovered IP address.



There are cases where this is not practical, such as when the device is behind a firewall and not reachable from Arigi. In this case a reverse HTTP tunnel can be employed. This uses the tunnel server `arigitunnelsrv` which makes outgoing connections towards both Arigi and the Syncthing API.

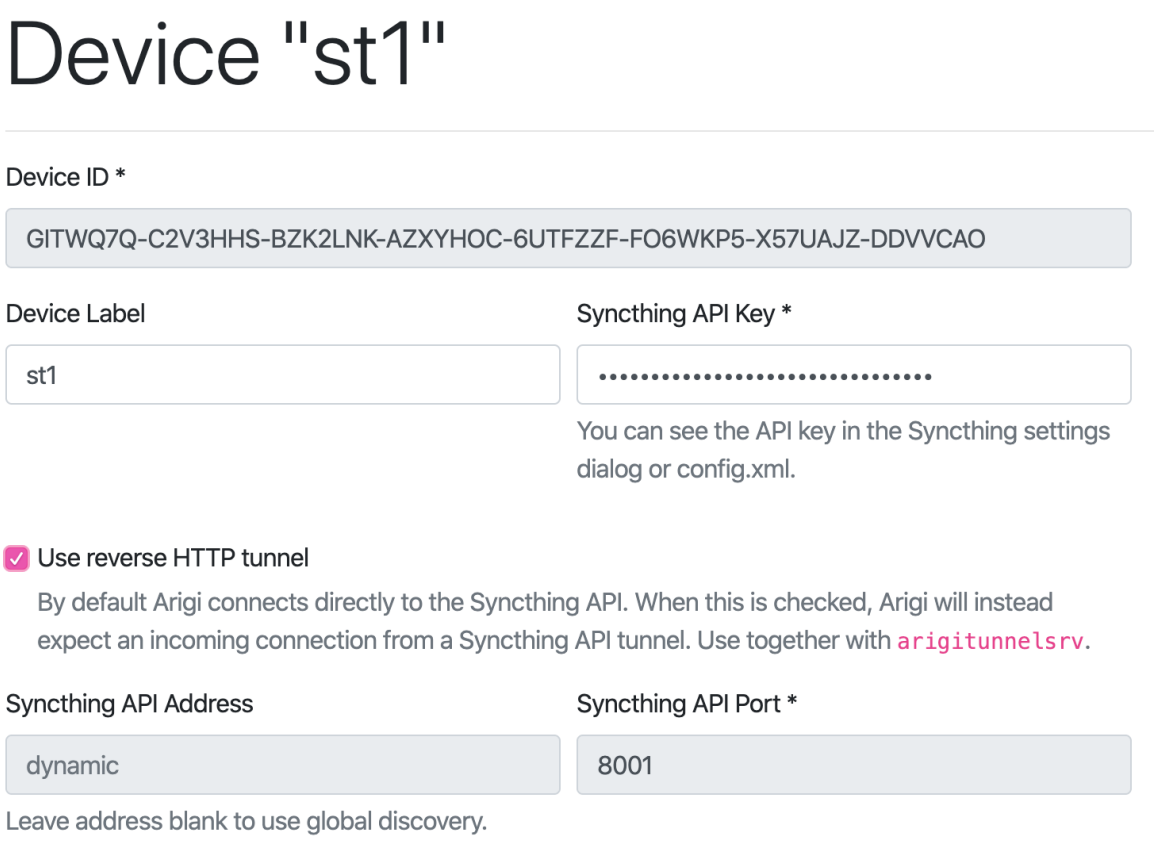


If the tunnel server is placed close to the device, the result is similar to inverting the connection flow: the connection comes *to* Arigi instead of originating at Arigi.

To accomplish this, the tunnel server needs to be started and pointed towards both the Syncthing device to be forwarded and Arigi itself:

```
$ arigitunnelsrv --syncthing-device=GITWQ7Q-...-DDVVCAO \
    --syncthing-addr="192.0.2.42:8384" \
    --arigi-addr="arigi.example.com:80"
```

In addition, Arigi should be configured to expect an incoming tunnel connection:



When checking the "Use reverse HTTP tunnel" checkbox the host and port fields are disabled as they become irrelevant. Further configuration, such as API key, is still available and works in the same way as usual. In particular, the tunnel server has no part in the authentication of the API connection, necessitating Arigi to use the API key as usual. API connections through the tunnel server are end-to-end encrypted between Arigi and the Syncthing device.

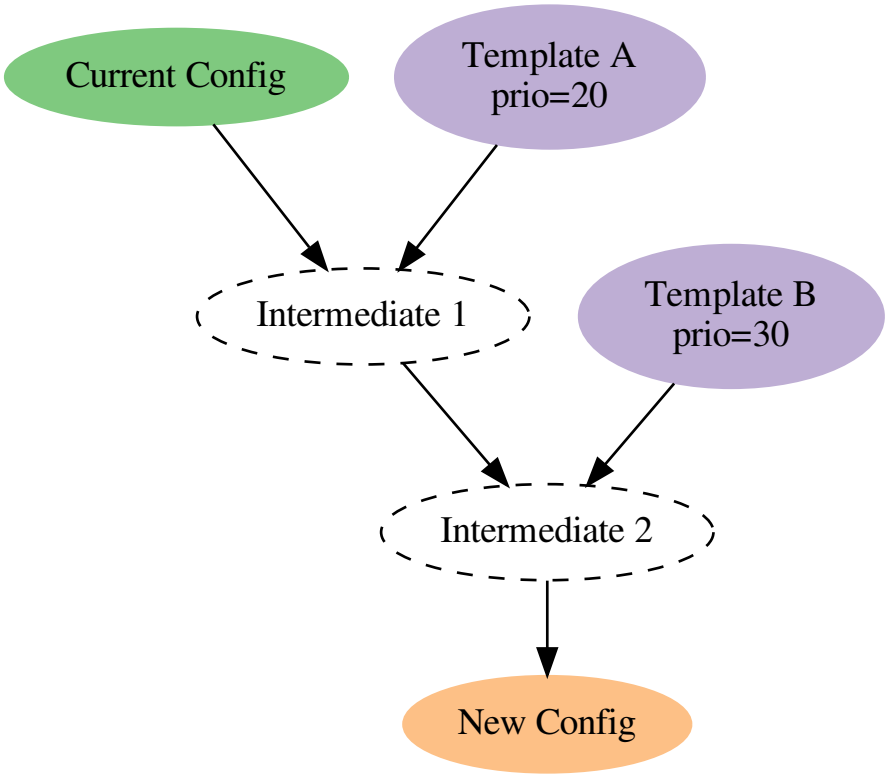
CHAPTER

FIVE

CONFIGURATION TEMPLATING

5.1 Overview

Templating is the mechanism by which we configure devices in Arigi. A device has one or more tags attached to it. These tags in turn match one or more configuration fragments that have the same tags. Arigi evaluates these fragments, assembles them in the order dictated by their priority, and thus creates a complete device configuration. If this configuration differs from what the device in question currently has, Arigi updates the device configuration accordingly. This is a powerful mechanism that allows reuse of configuration fragments between many devices, and automatically ensures that all affected devices are updated when a fragment changes.



Configuration fragments, in turn, can be dynamically built based on the information in Arigi. For example, a single fragment might define a folder shared with all devices having a certain tag. When a new device is added to this tag the fragment will change to accommodate it, and all devices attached to the fragment will be reconfigured to share the folder with the new device.

5.2 Updating by JSON

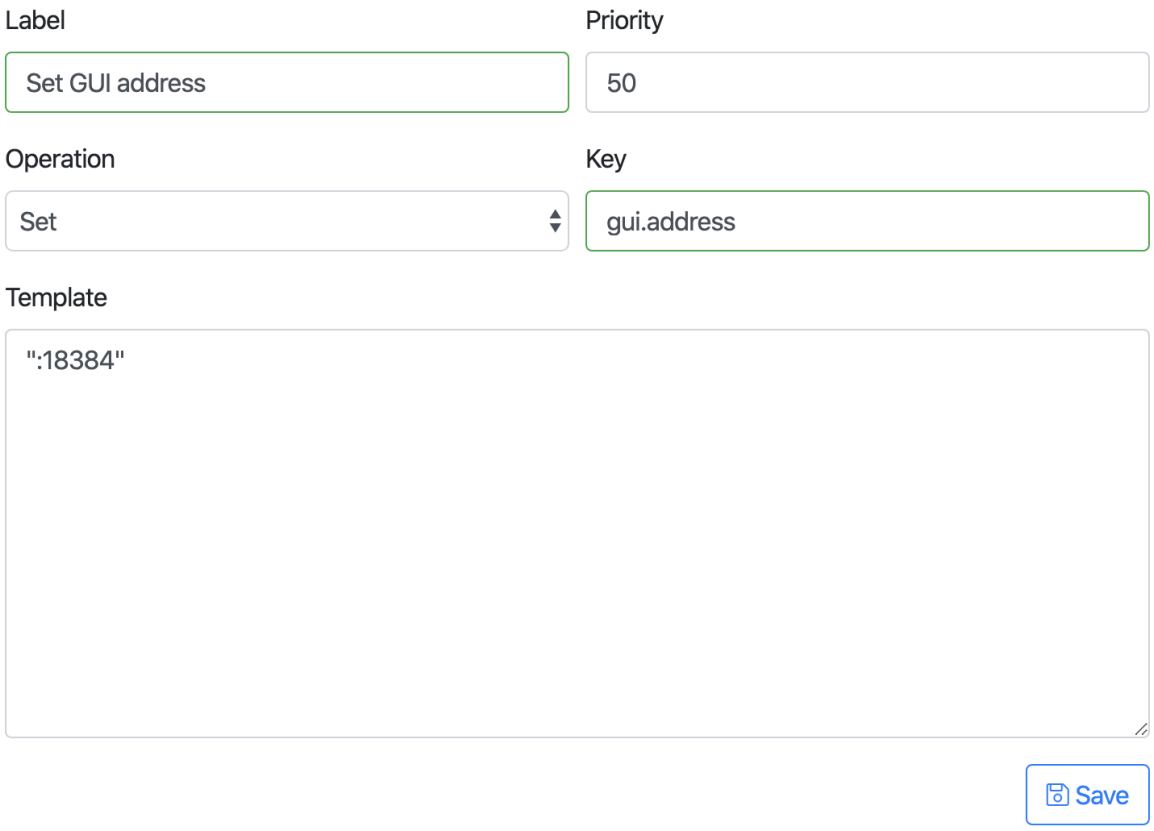
For simple cases, such as updating a config option, the best way is to provide the template in JSON format. For example, if we want to set the GUI port to 18384 instead of the default, a template definition would look like this:

Operation Set

Key `gui.address`

Template `":18384"`

Or, in the GUI:



Note: The value `:18384` is quoted, `":18384"` because the template syntax is JSON and this is a string type value.

The GUI editor also shows the interpretation of the template itself in JSON format, and the resulting configuration after applying it to a device:

Rendered Template (as seen on GITWQ7Q)

```
{
  gui.address: ":18384"
}
```

Resulting Config (as seen on GITWQ7Q)

```
{
  ▼ JSON object with 8 keys
  devices: [
    ▶ JSON array with 4 items
  ]
  folders: [
    {
      ▶ JSON object with 28 keys
    }
  ]
  gui: {
    ▼ JSON object with 12 keys
    address: ":18384"
    apiKey: "VUKuK2giXfHsswuxfANnkPtkR4Xef5A7"
    authMode: "static"
    debugging: false
    enabled: true
```

5.3 Updating by Python (Starlark)

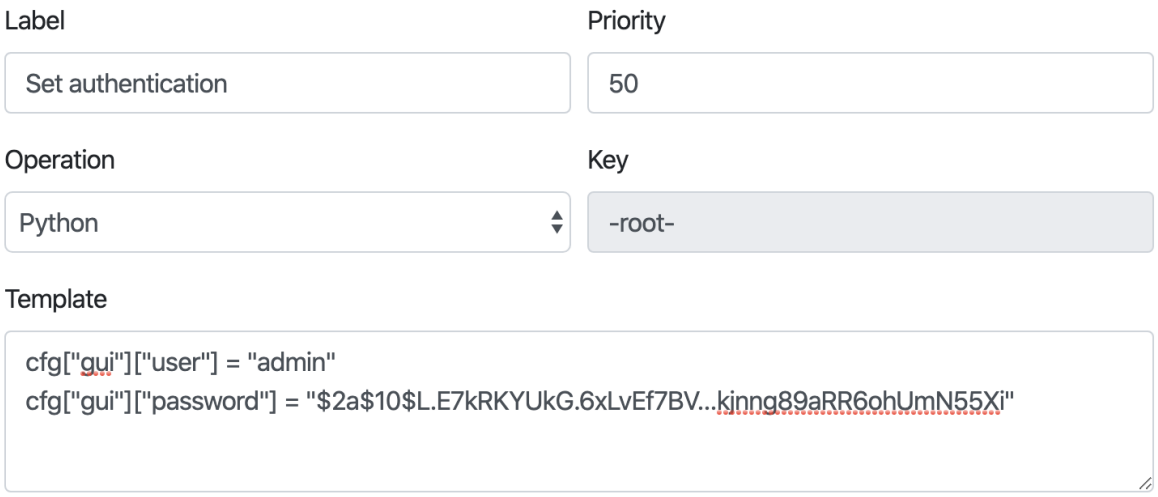
5.3.1 Language Basics

Arigi templates can use the Starlark templating language which is very similar to Python[1]. There are restrictions on the code in these templates – it is for example not possible to perform file I/O or launch external processes.

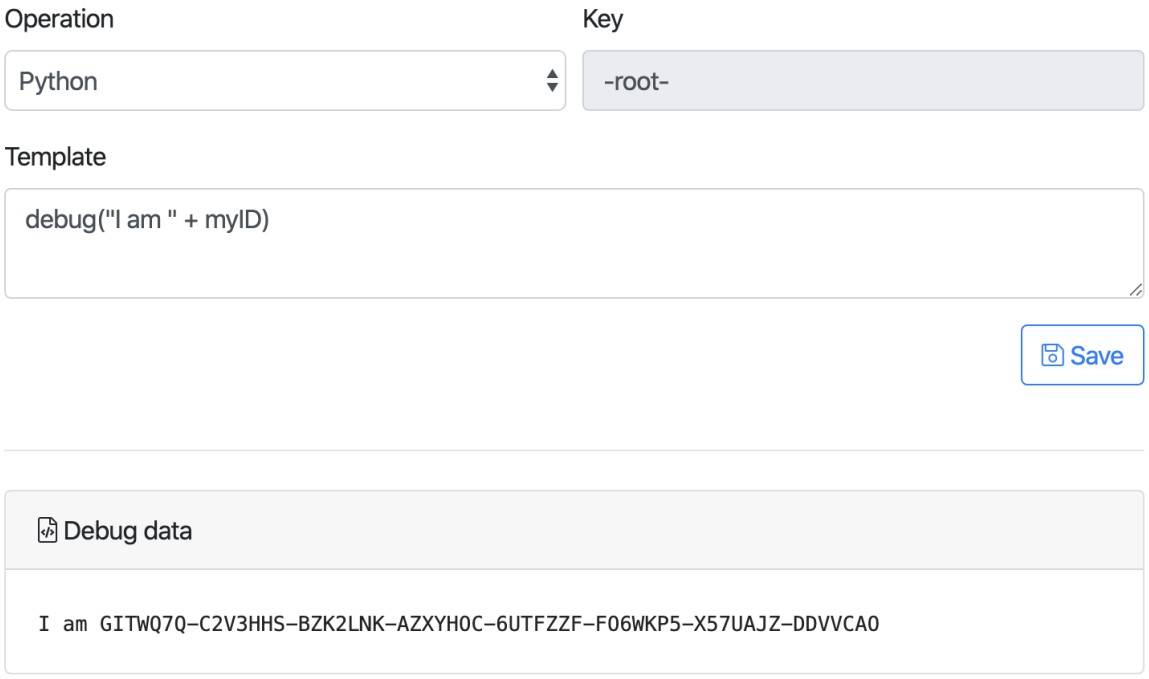
The configuration of the device being templated is available in the `cfg` variable and any changes to this object will be persisted as the result of the template. Python templates do therefore not have a "Key" setting like JSON templates. The ID of the current device is in the `myID` variable.

In the simplest case a Python template can be used to set static values, similar to a JSON template. For example, this template sets the GUI user ID and password hash to predetermined values:

[1] Similar enough that we'll often use the terms "Python" and "Starlark" interchangeably, as Python is more well known.



When developing Python templates it is sometimes useful to inspect values of variables. The `debug` function will output information so that it's visible in the template result. For example:



Functions and standard Python control structures can also be used to inspect and modify the configuration. One difference from standard Python is that loops and conditionals are only permitted inside functions.

Template

```
debug("I am " + myID)

def funcWithLoop():
  debug("Here are my current options:")
  for opt in sorted(cfg["options"].items()):
    debug(opt[0] + " = " + str(opt[1]))

funcWithLoop()
```

Save

Debug data

```
I am GITWQ7Q-C2V3HHS-BZK2LNK-AZXYHOC-6UTFZZF-FO6WKP5-X57UAJZ-DDVVCAO
Here are my current options:
alwaysLocalNets = []
autoUpgradeIntervalH = 12
cacheIgnoredFiles = False
crURL = https://crash.syncthing.net/newcrash
crashReportingEnabled = True
```

5.3.2 Database Access

To perform more advanced configuration templating we often need information about other devices than ourselves. For this purpose we have access to the `db` object, representing a read-only view of the Arigi database.

This allows to, for example, ensure that the default folder is shared will all devices. Here is a more involved example in text form:

```
# List all devices in the database.
all_devs = db.ListDevices()

# Process them into a list of {"devicID": "..."} objects as expected in␣
↪the configuration.
# The string_dict() function forces the key to be of string type, which␣
↪is required in the
# Syncthing configuration but not enforced by default by Starlark.
device_ids = [string_dict({ "deviceID": dev.DeviceID }) for dev in all_
↪devs]

## All devices should share the folder.
```

(continues on next page)

(continued from previous page)

```
# Find the folder in the current list of folders.
default_folder = [f for f in cfg["folders"] if f["id"] == "default"][0]

# Set the label and devices
default_folder["label"] = "Auto generated default folder"
default_folder["devices"] = device_ids

## All devices should be added as peers.

# This is myself. We take it from the existing config in order to␣
 ↪preserve label and such.
my_dev = [d for d in cfg["devices"] if d["deviceID"] == myID]

# This is everyone else.
other_devs = [d for d in device_ids if d["deviceID"] != myID]

# Set the list of devices.
cfg["devices"] = my_dev + other_devs
```

The following methods are available on the database object:

GetDevice(id) Returns the device object for the given device ID.

GetDeviceConfig(id, stage) Returns the current device configuration for the given device ID. Stage is one of `polled` (the latest config returned by the device itself) or `generated` (the latest config we've generated due to template evaluation, but not necessarily pushed to the device yet).

GetDeviceVersion(id) Returns the current device version object for the given device ID.

ListDevices() Returns the list of all device objects.

ListDevicesWithTag(tag) Returns the list of all device objects having the given tag.

ListDeviceTags(id) Returns the list of tags for the given device ID.

ListFolders() Returns the list of currently known folders.

ListTags() Returns the list of all tags.

Using the `debug(...)` and `debug(dir(...))` functions on returned objects can be useful in determining their structure and methods.

CHAPTER

SIX

THE CLI

The installation package (.tar.gz) includes a command line client called simply `arigi`. This utility can be used to manage Arigi installation in most ways the graphical dashboard can. For example, you can:

- Enroll new devices,
- List and inspect status for current devices,
- List and inspect status for synchronized folders,
- Add and modify device tags,
- Add and modify configuration fragments,

… and so on.

To see all available commands and options, please run `arigi --long-help`.

Examples:

```
$ arigi login http://localhost:2525 admin my-admin-password
$ arigi --long-help
$ arigi device add abcdef-123567-... apikeyapikey
$ arigi device list
$ arigi device get abcdef-123567-...
$ arigi device tag abcdef-123567-... mytag
```

CHAPTER

SEVEN

UPGRADING

7.1 Between Patch Versions

Versions with the same major and minor (first and second digit group) and that only differ in the patch version (third digit group) are always compatible, both in terms of configuration and database format. It is always possible to upgrade and downgrade between such versions, for example from `1.0.1` to `1.0.2` and back.

7.2 From 1.0 to 1.1

Arigi version 1.1 introduces backwards incompatible multi user support, and removes support for PostgreSQL databases. The upgrade considerations differ based on your current database usage.

7.2.1 With In Memory Database

No specific action is required to upgrade when using the in memory database (`--use-ram-db`). It is recommended to take a backup of the existing database before upgrading, in case a rollback or downgrade is required.

The `--use-ram-db` option is allowed but ignored as the in memory database is now the only choice.

7.2.2 With PostgreSQL

Arigi version 1.1 uses only the in memory database. When upgrading it is necessary to migrate existing data via a backup:

1. Shut down the existing Arigi 1.0 server.
2. Create a backup of the existing PostgreSQL database using your existing Arigi **1.0.1** or higher **1.0** version:

   ```
   $ arigisrv --backup-to=database.bk --postgres-...
   ```

   When running with the `--backup-to` option, and your usual PostgreSQL configuration options, Arigi will start up, create the named backup file, and then exit.

3. Restore this backup using the new Arigi **1.1** version:

```
$ arigisrv --restore-from=database.bk ...
```

   When running with the `--restore-from` and your usual configuration options, excepting any PostgreSQL options, the database will be initialized and restored from the given backup file. Arigi will then exit.

4. Run `arigisrv` in your usual manner, having removed any PostgreSQL options from the command line or environment.