GO.DATA
IT ADMINISTRATOR'S GUIDE

1 September 2021

V3.1

"I need to plan, maintain, tune or debug my instance of Go.Data …

… what do I need to do to configure Go.Data to work correctly for my users or to connect it to other systems?"

Table of Contents

Welcome

Purpose

Welcome to the Go.Data IT Administration Manual.

This manual provides detailed information for IT administrators that need to maintain, tune or debug their instance(s) of the Go.Data software and is a compliment to the user and implementation guides which can be found on the Go.Data Community website along with all other documentation at: -

https://community-godata.who.int/

The guide deals with tasks such as configuring the SMTP email server for the dispatch of Go.Data's forgotten password message, understanding the configuration and log files for the tool, scaling a solution and the security features of Go.Data etc.

As such, the document is aimed at **IT administrators** or others that are managing a deployment of the Go.Data tool, typically on a server for a community of users. The document therefore assumes familiarity with the tool and will not cover the functionality present within the solution (this is found in the user guide).

If you are installing a stand-alone copy of Go.Data on your local machine, then this manual is likely to be of less relevance but potentially still useful for debugging.

This document is not therefore intended to be read cover to cover but to act as a reference guide when seeking to address a particular IT issue.

Getting more help

If you do not find the information you seek in this document, then further support for Go.Data is available by: -

Emailing the support team at:	godata@who.int
Joining the online community of users at:	https://community-godata.who.int

Basics

Go.Data can be installed on Windows and Linux operating systems and comprises of MongoDB, NodeJS solution and the Electron browser for delivering its web interface in a “form” environment. The electron browser can be bypassed however, and the solution can equally be accessed using the browser of the user’s choice at the following address on the machine on which it is installed: -

http://localhost:8000

Port 8000 is the default and can be modified during the installation of the tool – for instance if something is already running on that port.

The tool exposes an Application Programming Interface (API) whose methods are self-documented using the Loopback API tool. This documentation can be viewed at: -

http://localhost:8000/explorer

If you are accessing Go.Data somewhere other than a local machine you can still add *“/explorer”* to the end of the URL to see the API methods.

Go.Data also provides a smartphone application for Android and iOS which communicates with this API for the exchange of data.

In fact, all communication from both web front end and smartphones goes via the same API.

Note: It is recommended to update windows OS to version 8.1 or newer versions due to enhancements in the upcoming Go.Data versions 39 and upwards but if you can’t upgrade your windows OS due to some circumstances and still want to use windows 7 or earlier versions you need to set the following environment variable “NODE_SKIP_PLATFORM_CHECK“ to 1 on your system, otherwise the GoData server won't start anymore due to security constrains on windows 7 and previous versions this is because Microsoft has officially ended support for these operating systems and no security updates are provided any longer.

Installing Go.Data

For full details on installing Go.Data, please consult the Deployment Manual. This section deals with IT Administration questions and troubleshooting around deploying a new instance of Go.Data.

Minimum requirements

The minimum requirements for installing Go.Data are given: -

- **Android phones:**
  - Hardware: quad-core CPU (2 GHz or higher), 2GB RAM (Lowest end device tested inside Clarisoft is Samsung Galaxy S5: https://www.gsmarena.com/samsung_galaxy_s5-6033.php) Software: Android 5.0 or higher (preferably newer versions due to the updated security patches)
  - Please make sure you have a minimum 1 GB or more storage space available due to the increase in the database size
- **iOS iPhone:**
  - Hardware: iPhone 5s or higher(preferably iPhone 6 or newer due to bigger screen size) Software: iOS 9.3 or higher(preferably 10 or higher)
  - Please make sure you have a minimum 1 GB or more storage space available due to the increase in the database size
- **Windows (64bits)**
  - OS: 8.1
  - free space: 8gb (might need more, if we want to store a huge amount of data)
  - CPU: 2GHz (the application might be really slow on processors with lower frequency)
  - ram: 8GB
- **Linux (64bits)**
  - OS: Ubuntu 12.04, Fedora 21, Debian 8…
  - free space: 8gb (might need more, if you want to store a huge amount of data)
  - CPU: 2GHz (the application might be really slow on processors with lower frequency)
  - ram: 8GB

32-bit support

Although earlier versions of Go.Data were available in both 64 bit and 32 bit, more recent versions of Go.Data (Version 39 onwards) are 64 bits only. This decision was made when there was very little initial uptake/demand for the 32-bit installer. Therefore starting with Go.Data build 39, There is no more support for 32bit installers.

Installation permissions

When installing Go.Data on MS Windows, there are options to install as a stand-alone or server instance. If the "standalone" option is selected then Go.Data installs without Windows services, meaning that the components of the solution (API and Database) will run only as long as the Go.Data app itself is running.

Alternatively, when installing as a "server" option, two Windows services are run in the background so that even when the Go.Data interface is closed, the solution is still alive.

Running Windows services usually requires the installing user to have elevated rights however and so after installing Go.Data as a "server", it should be started by right-clicking and selecting "Run as Admin".

For Linux users, try running the installer with the "***sudo***" command.

In general, elevated privileges (*sudo*) is needed whenever you are modifying system level services/functions. Example: modifying firewall, starting a service, etc.

To prevent potential security pop-ups blocking/failing your install, you need to run *sudo*.

However, if for any reason you do not trust the application you are installing, do not run *sudo*.

Installation on non-English machines

Some users have experienced issues when installing Go.Data on a machine that does not have "English" as a default language. This will cause errors either during the actual installation or during the database start-up, when the process simply hangs indefinitely on the database start-up message.

As a workaround, change the system language to English and re-install Go.Data. The system language can be changed back after installation has completed and should not affect further upgrades of the tool.

Warning: Please be careful to change System Language, **NOT** Keyboard/Interface Language, as this will not suffice. This issue will be addressed in a future release.

Installation on cloud platforms

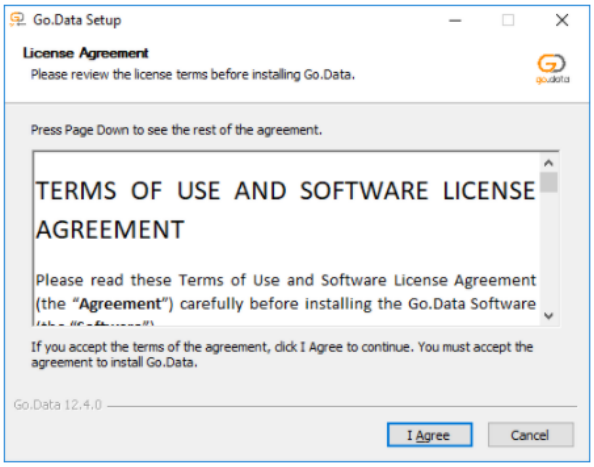
Go.Data can be installed on cloud platforms mindful of all other advice in this document for hosting the solution.

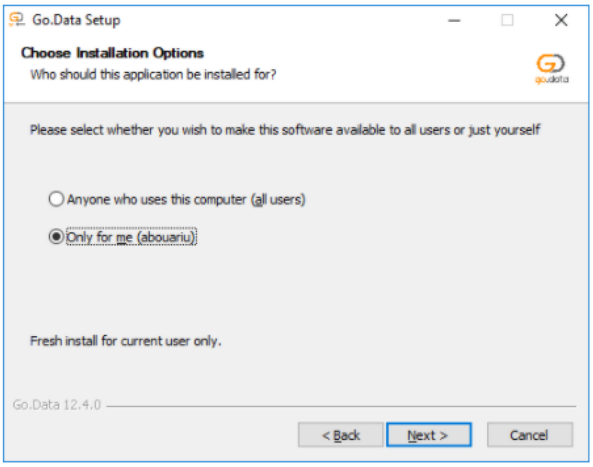
For installations on Microsoft Azure, previous installations have run into issues with the Application Gateway service on Azure. Under configuration, the Application Gateway's capacity type has to be changed from "autoscale" to "manual" and then a restart of the service is needed.

Windows installation

To install Go.Data on Windows, run the installer .exe file. The installer will start with the Go.Data terms of use and software license agreement.

After the license agreement is accepted, the installer will ask whether to install Go.Data for the current user or all users





Installing Go.Data for the current user does not require elevated permissions and the installation is only accessible to the current logged in user. The application data will only be available to the current user.

Installing Go.Data for all users requires elevated permissions and will re-launch the installer after acquiring permissions. This installation type allows any user to run Go.Data. The application data will be accessible to all users

Default installation folder

For the current user, the default installation path is:

[Windows Drive:]\Users\[Current User]\AppData\Local\Programs\Go.Data

For all users, the default installation path is:

[Windows Drive:]\GoData\bi

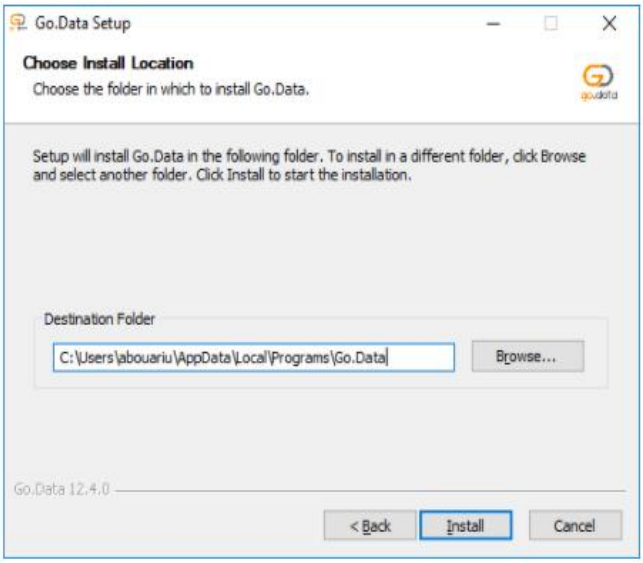
Installation settings and data folder

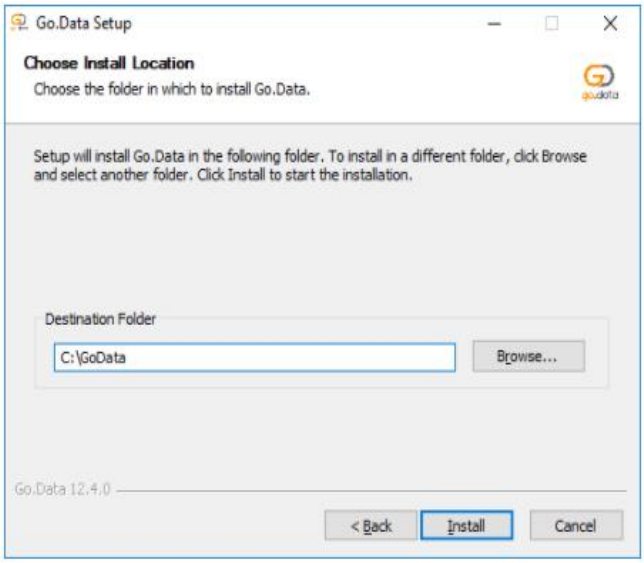
For the current user, the application data path is:

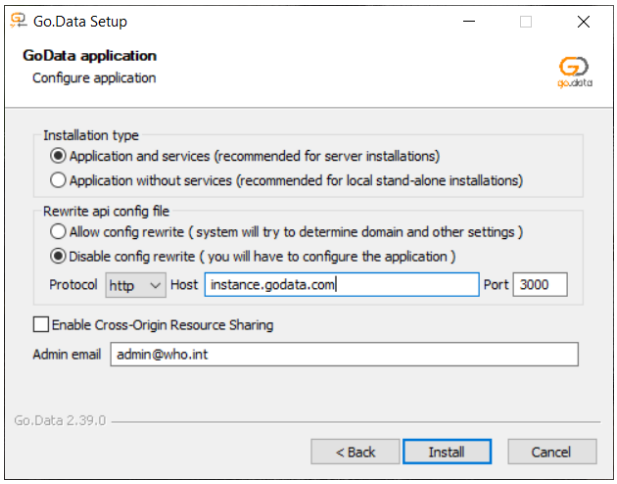
[Windows Drive:]\Users\[Current User]\AppData\GoData

For all users, the application data path is the same as the installation path and defaults to:

[Windows Drive:]\GoData\data







Go.data configuration options:

Select any of the suitable option below for how GoData startup during installation:

- **Application with services**: GoData website will work even if GoData desktop application is not running, since the services will be running in background
- **Application without services**: GoData website will work only while GoData desktop application is running, if you close it, GoData website will stop working.

Rewrite api config file:

- **Allow config rewrite** option will try to determine domain the first time you access the website through its domain
- **Disable config rewrite** will require for you to enter/specify the GoData URL & port number

Enable Cross-Origin Resource Sharing option will enforce (CORS) whitelist to be applied

Admin email allows you to change super admin email in case you don't want to use the default one.

After installation, Go.Data can be launched as the last step of the installer, from the installation folder or searched in the start menu.

Uninstalling

To uninstall Go.Data, run the uninstaller that is located at the installation path or uninstall from *Add or Remove Programs* in *Windows Control Panel*.

Additionally, remove the Go.Data application data folder from its location mentioned above.

Linux installation

To install Go.Data on a Linux server, unzip the .tar.gz file to any location that has write permissions to the current user. The application data folder will be created in the same path, unless otherwise configured.

```
tar -xvzf go-data-linux-x64.tar.gz
```

Upgrading

To update Go.Data, download the newest version and install it to a different location from the previous installed version. When launching Go.Data from the script, set the --dbpath parameter to point to the database path used by the previous version.

Notes: Before updating Go.Data, perform a database back-up by creating a copy of the database folder.
Backup folder location: C:\Go.Data\bin\resources\go-data\build\backups

Updated Linux steps:

- download Go.Data Linux archive

- stop Linux server (api + db)

  - one way of stopping server is to run the following commands

    - open a terminal inside the Go.Data application directory (e.g. "go-data-linux-x64 2.34.7")

    - platforms/linux/x64/default/node/bin/node app-management/bin/pm2 stop server

    - kill -9 $(lsof -t -i:27017)

    - kill -9 $(lsof -t -i:8000)

Note: (27017 is the default port for Mongo, while for GoData Mongo server the default port is 27000. But this port can be changed, therefore in case you want to replace the default port for Mongo the command should be “kill -9 $(lsof -t -i:27017) where 27017 should be replaced with the new port number for your Mongo server) The same process applies to replacing the default GoData mongo port 8000.

    - platforms/linux/x64/default/node/bin/node app-management/bin/pm2 ls

      - you should see a server in the list - the old one

    - platforms/linux/x64/default/node/bin/node app-management/bin/pm2 delete server

    - platforms/linux/x64/default/node/bin/node app-management/bin/pm2 ls

      - the list should be empty now

  - as a precaution, to make sure all services are stopped, you could restart the Linux machine

- unzip the new Go.Data version in a different location than the old one

- copy the following folders from old version of Go.Data to the new one (make sure you include hidden files too - e.g. **.appVersion file** from db folder, **otherwise the system will replace database with a new one, instead of upgrading the existing database**)

  - database (db folder)

  - backups (go-data/build/backups)

  - storage (go-data/build/server/storage) folders

- update config.json (go-data/build/server/config.json)

  - do not copy the old file over the new one since new settings might've been added with the new version

- start the new Go.Data Linux server

  - You see a message similar to "Migrating database from version 'old version' to 'new version'" and not "Populating database..."

  - wait until the server starts

- open the browser and make sure everything works :)

- you can delete old Go.Data directory if you don't need it anymore

Uninstalling

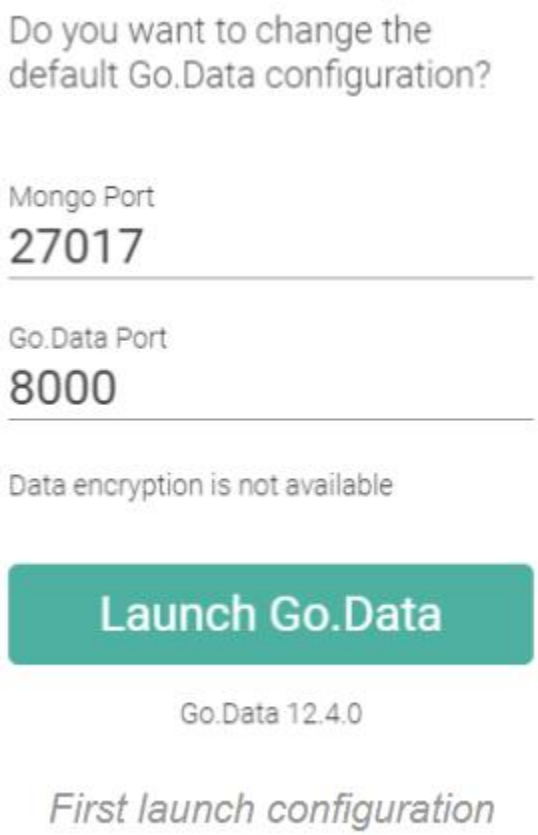
To uninstall Go.Data, delete the installation folder.

Startup configuration

When Go.Data is launched for the first time, the following options can be configured:

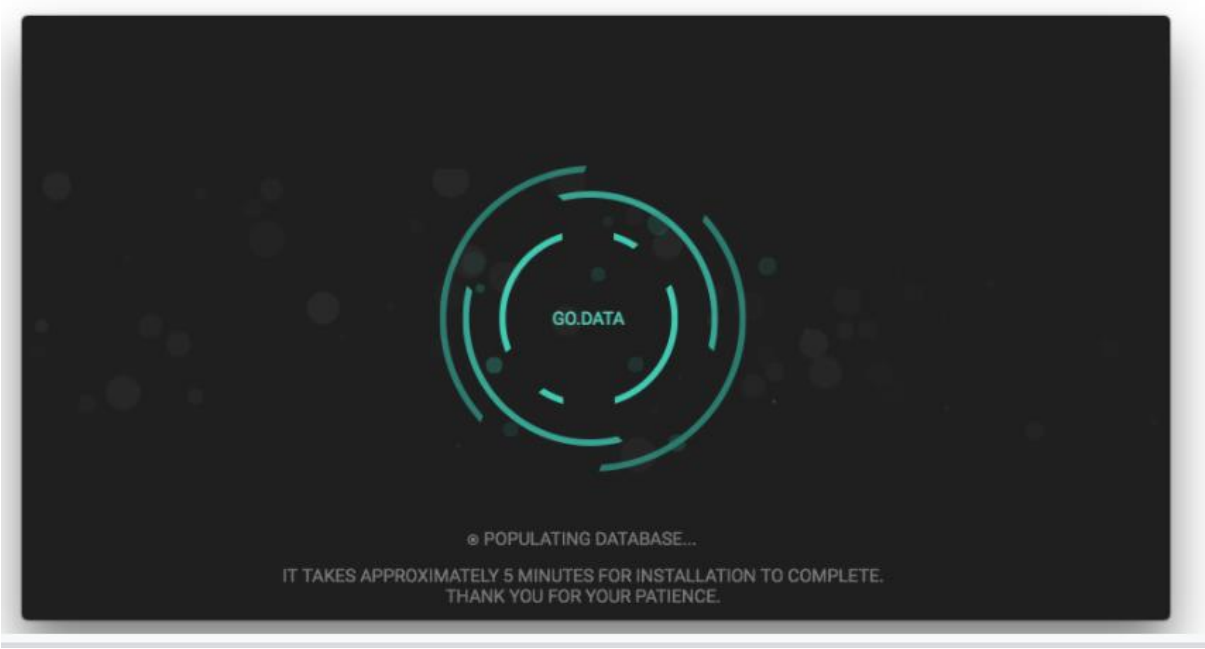
- Mongo Port -The default port number used by the database engine is updated to 27000
- Web App Port - The port used by the web app, defaults to 8000
- Encryption - If the device supports encryption, there will be an option to enable it.
    - Note: Enabling encryption on Windows will launch a wizard to export the encryption key. We advise to save the encryption key so it can be used to decrypt the Go.Data application data.

For more information on the ports and on encryption, please see the relevant sections of this document.



First launch configuration

The first time that Go.Data application starts, it will take 5 to 10 minutes to configure the data and populate the database with default data. The next launches are faster and usually start the app in about 30 seconds.
Upgrades usually migrate data from previous version to a new one therefore the Go.Data application doesn't take long to start after an upgrade.



Linux startup options

On Linux, Go.Data is launched from the go-data-x64.sh. Optionally, the following arguments can be passed:

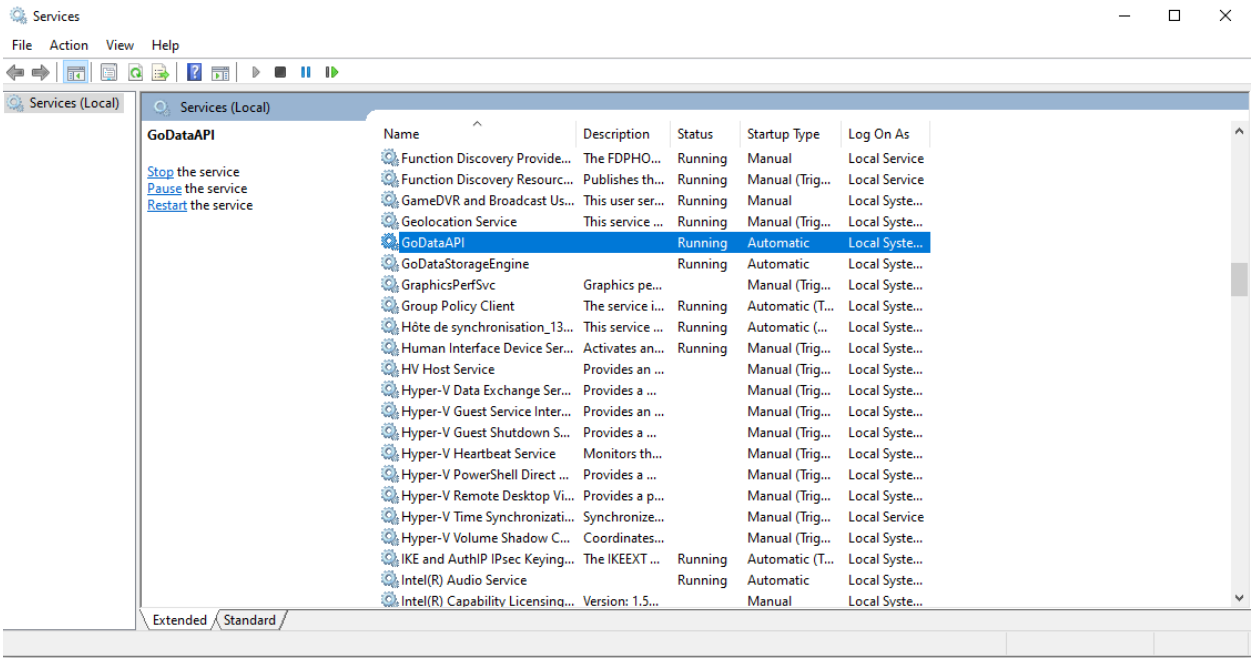
- --dbport
  - specifies the port for Mongo
  - between 1025 and 65535
  - must be different from port
  - defaults to 27000
- --dbpath
  - specifies the path for Mongo files
  - defaults to db
- --port
  - specifies the port for Go.Data
  - between 1025 and 65535
  - must be different from dbport
  - defaults to 8000

./go-data-x64.sh --dbport=3001 --dbpath=~/Desktop/db --port=3000

Automatically launching on boot

Windows Launching

On Windows for a “server”-type of install (as opposed to a “stand-alone” install), Go.Data launches its processes as Windows services which can be automatically started on boot of the machine. You can set this automatic start up by going to the Windows menu, typing “Services” and opening the Services window as shown below.



The two Go.Data services are named: -

- **GoDataAPI**
- **GoDataStorageEngine**

And should be set to “Automatic” for “Startup Type”. You change the startup type and start, stop, pause and restart services from this window.

Linux Launching

On Linux, there are various ways to start Go.Data on boot, depending on the Linux distribution and its version. Check the Linux distribution manual for more information on how to set up the launch script as a service.

A general way to creating services on Linux is by using systemctl.

Create a file called godata.service in /etc/systemd/system.

```
cd /etc/systemd/system
```

```
touch godata.service
```

Write the following service configuration in the godata.service file:

```
[Unit]
Description=GoData service
StartLimitIntervalSec=0
[Service]
Type=simple
Restart=always
RestartSec=1

# Fill here user account
User=user

# Fill here the path to Go.Data launch script
ExecStart=/home/user/Desktop/Go.Data/go-data-x64.sh
[Install]
WantedBy=multi-user.target
```

The configuration above can vary between different Linux distributions.

Once the file is saved, start the service:

```
systemctl start godata
```

Once the service launches successfully, configure it to be automatically launched on boot:

```
systemctl enable godata
```

Setting up installers / update server

Although Go.Data checks at start-up to see if a new version of the software is available and prompts the user to optionally upgrade, administrators can also request for the link to the **Go.Data** installation files for each new version themselves by sending an email to godata@who.int

The release notes of new Go.Data versions will be announced on the Go.Data community portal at https://community-godata.who.int and a link provided from there. Is very important to make it a good habbit of checking release notes on the Go.Data community portal before upgrading your current version in order to prepare in advance in case you need to make adjustments to your scripts.

Mobile phone installer

The latest versions of the Go.Data smartphone application for Android and iOS are found in the Play Store and App Store respectively and are listed under the WHO channel.

Upgrade to the latest version of the Go.Data smartphone proceeds as per any other app update with the user prompted for an optional upgrade.

It is recommended to keep your copy of Go.Data server and smartphones up to date to the latest version always to benefit from ongoing fixes and enhancements to functionality.

If you require the source .apk file for installing, Go.Data on Android devices which do not have access to the Play Store then please contact godata@who.int with this request.

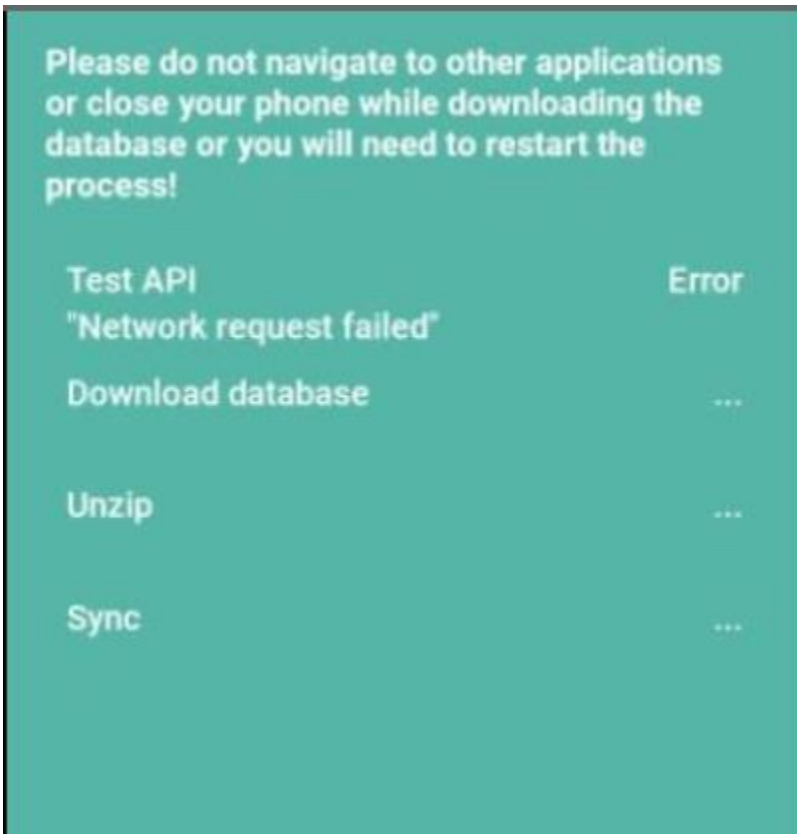
Connecting and synchronizing Go.Data on (Android & IOS) mobile phones to Go.Data application on Laptops offline. (No internet connection required)

There are 3 ways of establishing an offline network connection between your mobile phone device and your Laptop
You can connect using the following devices:
1. **Connection with Mobile phone personal hotspot** (Computer/laptop needs to connect to the mobile phones hotspot to a establish a network connection with the phone without data or internet)
2. **Connection with TP-Link router** (Both the computer/Laptop and the mobile phone device needs to connect to the TP-Link router network so that IP addresses on the laptop and phone are straight away the same except for the last digit)
3. **Connection on any WiFi network with an IP of (196.168.0.1)** (Both the computer/Laptop and the mobile phone device needs to connect to the same WiFi network so that the IP addresses on the laptop and the phone are straight away the same except for the last digit for a connection to be established)

Go.Data should be installed on the personal computer or on the Laptop as a server with the default options.
After a successful installation you need to open Go.Data on the web browser with the computer's IP address in place of the localhost for example (http://localhost:8000/auth/login then change to http://192.168.0.33:8000/auth/login) the IP that replaces the localhost should be same as the computer's IP address.
So if the IP address of the computer changes, it needs to be updated in the URL when opening Go.Data in the web browser and in the URL on the hub configuration on the phone. Also in case the computer IP address changes is best to update or generate a new the QR code for the mobile phones to scan to synchronize data.
After the above connections have been established, follow the steps in the (**Syncing the Go.Data Mobile app**) guide on the Go.Data community site to generate the QR code and scan with the phone to configure the hub and synchronize data between the mobile phone and the laptop.

Resolving Go.Data synchronization issues on mobile Devices.

For most mobile phone synchronization issues with the Go.Data server like the once shown in the screenshot below, it would be best to restart the Go.Data application, find a better network connection or reinstall or update the mobile application and try again furthermore make sure the Mobile phone and Go.Data server are on the same network. Also use the correct hub configuration settings with /api at end like this link: https://vhf.h....go.u../api and make sure the Go.data server's URL is accessible on the mobile phones web browser.



Backup & restore

Go.Data contains its own functionality for performing backup and restore as detailed in the user guide. At backup, Go.Data creates a ZIP file containing json descriptions of its data and this can be used for disaster recovery/to reset the system to a point in time.

Backup is a full database backup and if you require only a subset of the data held in the system then you should use sync packages also described in the user guide.

The location of backup is not configurable, it is set to a defined path below the Go.Data installation directory. For full disaster recovery, you should therefore configure your own scheduled job to copy the backup files created by Go.Data to a different machine or configure the directory to participate in cloud backup with a tool such as OneDrive.

Backup files are secured with a password which is present in the *config.json* as described later in this document: -

```
"backUp": {
  "password": "x$^56fwm137s1a#2@oalxbnk0z541lsd"
},
```

Administrators may also like to take a copy of the *config.json* and *datasources.json* files when backing up.

Windows backup

There is no special consideration for Windows backup, you can use the built-in features of Go.Data as described in the previous section and in the user guide.

Methods in the Go.Data API can also be used if the backup of Go.Data must be controlled from an external tool/trigger.

Linux backup

To manually backup the Go.Data data on a Linux installation: -

- Login to the Go.Data Web Portal
- Click on Menu->System Configuration->Backups
- Click on Quick actions->Create Backup
- Accept the default parameters or change as necessary
- Click on Create Backup

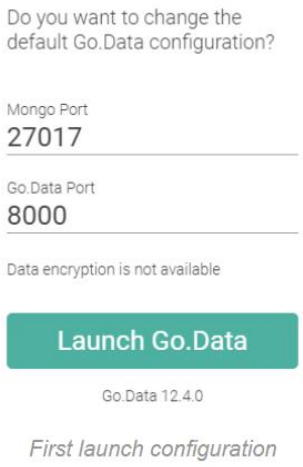
Note: As a precaution always copy backup files to a different offline location before an upgrade.

To restore the Go.Data data on a Linux installation: -

- Login to Linux
- Spawn a terminal console shell
- Change directory to Go.Data installation directory
- Execute the command <installdir>\go-data-restore-backup-x64.sh --file <backupfile>

Application ports

The ports used by Go.Data are chosen at installation and can also be changed in the *config.json* configuration file.



First launch configuration

Mongo port

The Mongo port is the port that is used by Go.Data to launch the database service. Go.Data uses Mongo as database storage and launching the Mongo service is one of the preliminary steps in launching Go.Data.

The default port has been upgraded from 27017 to 27000 and can be changed on the first launch or from the Go.Data settings when the app is running.

Go.Data port

The Go.Data port is the port where the Go.Data web app will be available. The default port is 8000 and can be changed on the first launch or from the Go.Data settings when the app is running.

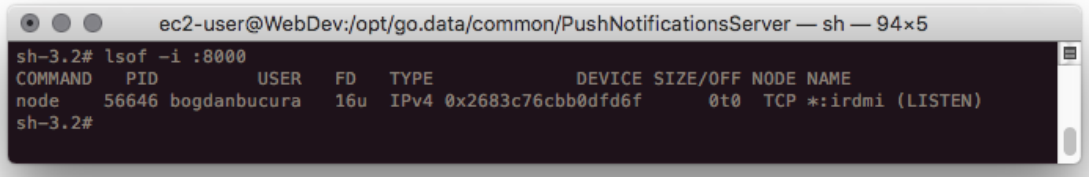
Additional information on ports

Valid ports that can be used for Mongo and Go.Data range from 1025 to 65535. Do not use the same port for Mongo and Go.Data.

Other applications may run on the available ports specified above, including another Mongo service on 27017. It is advised to use a Mongo port that is not used by other apps. Go.Data will attempt to start Mongo by terminating any app that is running on the specified port.

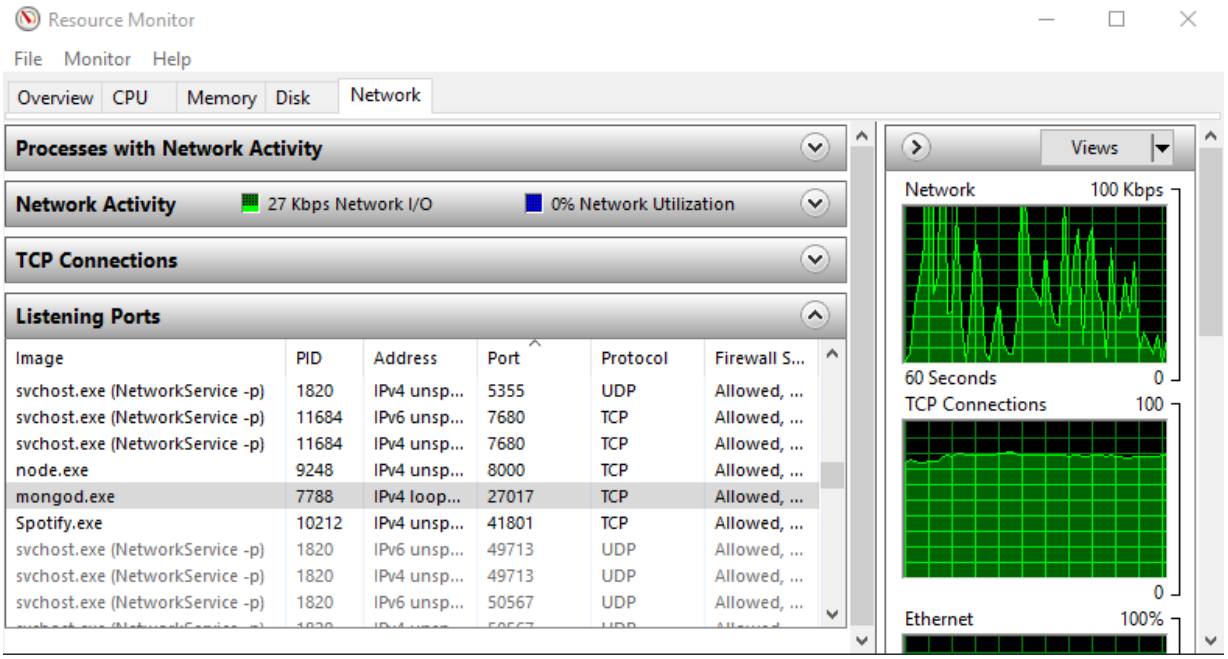
To check if a port is currently used by another app:

OSX and Linux - Run in terminal *lsof -i :8000* (for port 8000). Results that end with *(LISTEN)* are used ports.



Checking ports in use on OSX and Linux

On Windows - See the listening ports in the Resource Monitor application.



Checking ports in use on Windows

Encryption

Encryption on Windows

For Windows, Go.Data encryption is only available on Windows versions that support EFS (Encryption File System). These versions usually include all versions except from Windows Home.

The list of Windows versions that support EFS in December 2018 are the following:

- Windows 2000 Professional, Server, Advanced Server and Datacenter editions
- Windows XP Professional, also in Tablet PC Edition, Media Center Edition and x64 Edition
- Windows Server 2003 and Windows Server 2003 R2, in both x86 and x64 editions
- Windows Vista Business, Enterprise and Ultimate editions[9]
- Windows 7 Professional, Enterprise and Ultimate editions
- Windows Server 2008 and Windows Server 2008 R2
- Windows 8 and 8.1 Pro and Enterprise editions
- Windows Server 2012 and Windows Server 2012 R2
- Windows 10 Pro, Enterprise, and Education editions.
- Windows Server 2016

Using EFS, Go.Data only encrypts the application data folder.

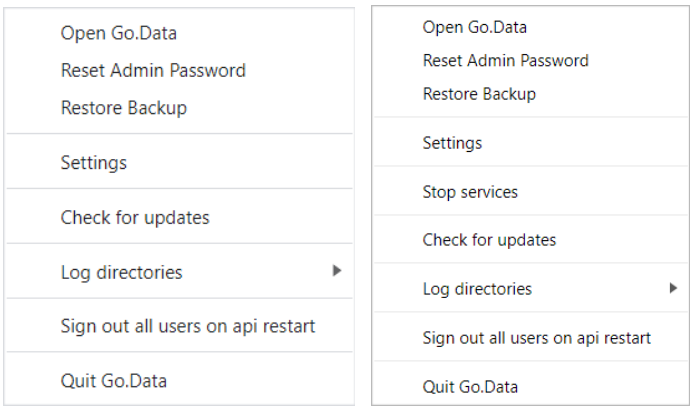
Encryption on Linux

Data encryption is not available on Go.Data for Linux.

Windows system tray icon

As soon as Go.Data starts in Windows, a web page will open with the application and a system tray icon will appear. The system tray icon has the following application options:

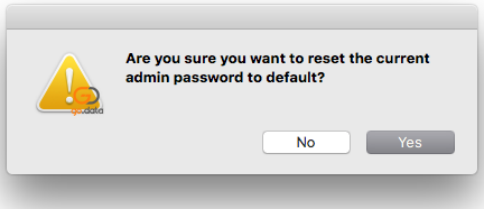
- Open Go.Data - Opens Go.Data in a new web page
- Reset Admin Password - Resets the password to default
- Restore Backup - User can select a zip file with a backup and restore it
- Settings - Opens the settings to change the database port, the web app port or the encryption settings
- Stop / Start services – This option is visible only if you installed GoData with services during the initial installation. It will allow you to either stop the services if you want to disable users access the GoData website, or if you want to restart the application itself after an update in the configuration files in case they were changed therefore restarting the computer itself won't be required. This options works differently depending on which startup option was selected during the initial installation that is GoData application with or without services. The difference here is that, the Go.Data "with" services option give you the option to "Stop services", followed by "Start services" while the option for Go.Data "without" services only allows to restart GoData application itself (Quit & Start) for any changes to the application or config file to take effect.
- Check for updates - Checks and installs newer Go.Data versions
- Log directories – Can be used to access the database, api or GoData logs files
- Sign out all users on api restart – Once you choose this option it restarts the api and all users currently logged into Go.Data through the web interface will be logged out and they will have to sign in again before being able to continue what they were doing.
- Quit Go.Data – Terminates the Go.Data desktop application but can still be accessed through the website if Go.Data "with" services option was selected during the initial installation if not once the desktop application is terminated it also affects the web interface.



Go.Data system tray options (without / with services) (Windows)

Reset the administrator password

The Go.Data system tray allows to reset the Administrator to default. When resetting the password, a confirmation popup will appear and the password will be reset after confirmation.



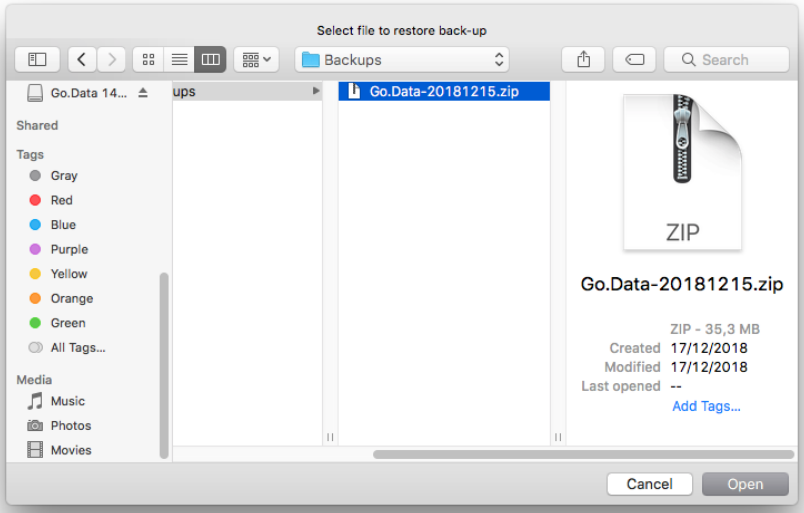
Reset password confirmation

Reset password completed

Instructions for resetting the password on Linux can be found in a later part of this document under the Configuring Go.Data section.

Restore backup

With this option, the Go.Data database can be restored to a previous version. Go.Data will ask for a zip file containing the back-up files to import in the database.



Selecting a backup to restore

Configuring Go.Data

Go.Data installs with a set of configuration files which are found in the installation directory and can be used to modify different aspects of the system's set-up.

Note that Go.Data must be restarted after a change to a configuration file for its changes to take effect.

Configuration files

The main configuration files used for Go.Data are: -

{your\installation\directory}\bin\resources\go-data\build\server\config.json
{your\installation\directory}\bin\resources\go-data\build\server\datasources.json.

So, for a default installation on Windows, for example, the location of the configuration files will be: -

C:\Go.Data\bin\resources\go-data\build\server

These two files contain a mix of properties which are used by the Loopback component included in Go.Data to document the API and custom properties specific to Go.Data itself.

In Linux, the same files can be found in {your\installation\directory}\go-data\build\server\

NOTE THAT: Go.Data will make a number of start-up checks and will overwrite certain information related to its URL if it believes that the configuration files needs to be corrected. So even having made a change to a file, it may be overwritten at start-up. To disable this behavior, change the *enableConfigRewrite* setting in the config.json file as described in the next section.

Options to change *enableConfigRewrite* parameters

Please check if "enableConfigRewrite" from config.json file is set to false, if it isn't then do these steps:
- stop api (service)
- change config.json file => enableConfigRewrite to false
- change config.json file => public to proper values
- start api

enableConfigRewrite affects only config.json file and only rewrites the following properties below.

- public.protocol
- public.host
- public.port

Config.json file configuration

The *config.json* file has properties required by Loopback and a set of custom ones specific to Go.Data.

In this document we will only include details about custom properties, as the Loopback documentation already covers the basic configuration here: https://loopback.io/doc/en/lb3/config.json.html

JSON Fragment	Explanation
"restApiRoot": "/api",	Although restApiRoot, host and port are Loopback defined properties, they should not be changed as they are required for other services as well (e.g. restApiRoot, which is the Loopback API path, is required for the web client).
"public": { "host": "localhost", "protocol": "http", "port": "8000" },	Specifies the end-point at which Go.Data expects its client application (web) to be running. This information is used for a check at start/up that Go.Data is indeed live at this URL and is also used to create the link in the password reset email. • public.host - Web domain / IP • public.protocol - Web protocol • public.port - Web port
"passwordReset": { "ttl": 900, "path": "/auth/reset-password", "from": "no-reply@who.int" },	passwordReset properties are used for configuring the password reset email. For this email to be sent, Go.Data must have access to an AMTP server (it does not provide its own SMTP functionality) and details of this server must be specified in the datasources.json file. • passwordReset.ttl - reset password token time to live (how long until reset password associated with the reset password email expires) - default 15 minutes • passwordReset.path - API endpoint path to reset password. When you reset your password, an email with a link is used to reset your password. This link is constructed as follows: `${config.public.protocol}://${config.public.host}:${config.public.port}${config.passwordReset.path}` (so this field should stay as it is) • passwordReset.subject - reset password email subject • passwordReset.from - reset password 'from' email address

"logging": {
"level": "info",
"maxSize": 10000000,
"maxFiles": 10,
"requestResponse": {
"trim": true,
"maxLength": 1024
},
"trim": true,
"maxLength": 1024
},
	This fragment is used configure the API logger.
	• logging.level - defines what type of errors should be logged (valid types are error / info / debug)
	• logging.maxSize - max log file size in bytes
	• logging.maxFiles - max number of log files until round robin kicks in
	• logging.requestResponse.trim - determines whether we should trim the API response which will be included in error details
	• logging.requestResponse.maxLength - max length of the response before trimming
	• logging.trim - determines whether we should trim the error message
	• logging.maxLength - max length of the error message before trimming (e.g. exceptions might contain stack trace info which could be really long)

"sync": {
"asyncActionsSettings": {
"intervalTimeout": 10000,
"actionTimeout": 3600000
},
"actionCleanupInterval": 24,
"encrypt": false,
"debug": false
},
	Settings used to syncronise data between Go.Data servers.

"pushNotifications": { "serverURL": "http://whocd.clarisoft.com:1337/api", "appId": "b61f5946-1af3-4e07-9986-9ffd1e36ae93", "masterKey": "KlYddh2OdVycHuVBhXv2" },	Configuration of the parse server used to remotely wipe data from mobile devices (in case a device gets stolen etc.)
"backUp": { "password": "x$^56fwm173s1a#2@123456790....." "disabled": false },	Used to encrypt sync and backup files (can be removed if you don't need to encrypt files). config.json backups are enabled by default, but it can be disabled by changing "backup ‘’disabled" to true on a multi instance setup where only one instance is required to take the backups. (available from Go.data version "2.36.2" onwards)

"defaultArcGisServers": [
{
"name": "WHO Background",
"url": "MAP1",
"type": "type1",
"styleUrl": "url1",
"styleUrlSource": "esri"
},
{
"name": "WHO Reference",
"url": "MAP2",
"type": "type1",
"styleUrl": "url2",
"styleUrlSource": "esri"
}
],
	An array of map layers used as default for each outbreak e.g. if you leave the second tab of the “Outbreak” screen empty when creating a new outbreak then these are mapservers used.
	
	This information is a default only and can always be overwritten when we create / update an outbreak within the tool.
	
	These servers are used as the background layers to display maps (e.g. case / contact movement page, count cases page etc.)

"cors": { "enabled": false, "whitelist": [] },	Go.Data contains features to block cross origin resource sharing to protect it from potential malicious exploitation. If CORS is enabled then only whitelisted domains will be accepted in the "Origin" header and requests without Origin header (server-to-server, mobile). This property differs from the cors property from remoting (which should always be false). • cors.enabled: enable/disable CORS • cors.whitelist: list of whitelisted domains
"signoutUsersOnRestart": false,	This property dictates the behavior regarding authentication tokens and whether these are invalidated by a restart of the Go.Data service. When set to false, a Go.Data server restart does not log out any active sessions and their tokens remain valid. When set to true, a restart of the Go.Data server destroys tokens so that users will need to log back in. By default, after you update the application this flag resets to default value, so, in case it was checked, you will have to check it again after finishing the update process.
"authToken": { "ttl": 600 },	Authentication Token “Time To Live” in seconds. Indicates the duration until the token will expire, if no other requests were issued. The recommended value is at least 300 seconds.
"session": { "appSId": "GoData",	The property used to configure session variables for the API> /

"secret": "5d854dbc-5d5b-11ea-bc55-0242ac130003" },	• session.appSId : "GoData" - cookie id used by session variables, • session.secret : "5asdasdbc-5ddf-11ea-bc55-0242ac130003" => secret key used to generate session variables

"login": {
"maxRetries": 10,
"resetTime": 30,
"resetTimeUnit": "minutes",
"twoFactorAuthentication": {
"defaultLogin": {
"enabled": false
},
"oauthLogin": {
"enabled": false
},
"code": {
"length": 6,
"charset": "0123456789",
"ttlMinutes": 30
},
"emailFrom": "no-reply@who.int"
}
},
	Restrictions for the login and oauth token endpoints to allow only a configured number of login retries within a given period of time. (Note: If the user successfully resets its password the ban is lifted.)
	
	• login.maxRetries: 10, (number of login retries, after which the user is banned for a specific time)
	• resetTime: 1, (number of hours/minutes/seconds is banned)
	• resetTimeUnit: "minutes" (time unit for "resetTime" property) (allowed values: years, months, weeks, days, hours, minutes, seconds, milliseconds)
	Two Factor Authentication allows you to enable two step login process. If it is enabled, after you enter the email & password user, if they are correct, user will receive a code on his email. He has to enter this code in the ui to finish the authentication process. The code content can be changed though two Factor Authentication. Code object. By default, the code is a 6 digit number, but you can change the alphabet to contain letter and special chars if you want to.

"removeSyncSnapshotsAfter": 24, "removeTmpUploadedFilesAfter": 24, "removeTmpUploadedImportFilesAfter": 24,	removeSyncSnapshotsAfter: number (default: 24, number of hours before a snapshot file is removed - this zip file is when a sync between mobile and API is done) removeTmpUploadedFilesAfter: number (default: 24, number of hours before a temporary file or directory is removed - these temporary files are created by API to perform different actions) removeTmpUploadedImportFilesAfter: number (default: 24, number of hours before an imported file is removed - these temporary files are created when you upload a file to import it into the system)
"removeAuditLogsOlderThanNDays": 180,	- Introduced in Go.data version 40.0 - This option is enabled by default and configured to '180' days. - removeAuditLogsOlderThanNDays, If a number bigger than 0 then it will remove audit logs older than n days (e.g."removeAuditLogsOlderThanNDays": 30 will remove all audit logs older than 30 days). This option allows you to reduce the size and time needed to create backups since audit log table can become very large in time.

"captcha": {
"login": false,
"forgotPassword": false,
"resetPasswordQuestions": false
},
	Settings regarding the CAPTCHA behavior (CAPTCHA is available from Go.Data version 34.5 onwards):
	
	• captcha.login => boolean (default - false)
	  ○ true - enable captcha for login screen
	  ○ false - disable captcha for login screen
	• captcha.forgotPassword => boolean (default - false)
	  ○ true - enable captcha for forgot password screen
	  ○ false - disable captcha for forgot password screen
	• captcha.resetPasswordQuestions => boolean (default - false)
	  ○ true - enable captcha for reset password using questions screen
	  ○ false - disable captcha for reset password using questions screen

"bruteForce": { **"resetPassword": {** **"enabled": false,** **"maxRetries": 10,** **"resetTime": 30,** **"resetTimeUnit": "minutes"** **}** **}**	Used to disable reset password with question & answers for a time for a specific user if he tries to many times (available from Go.Data version 37.0 onwards)
"skipOldPasswordForUserModify": true	True if the system should request the old password when you try to change it from the UI (Default set to true, available from Go.Data version 34.4 onwards)

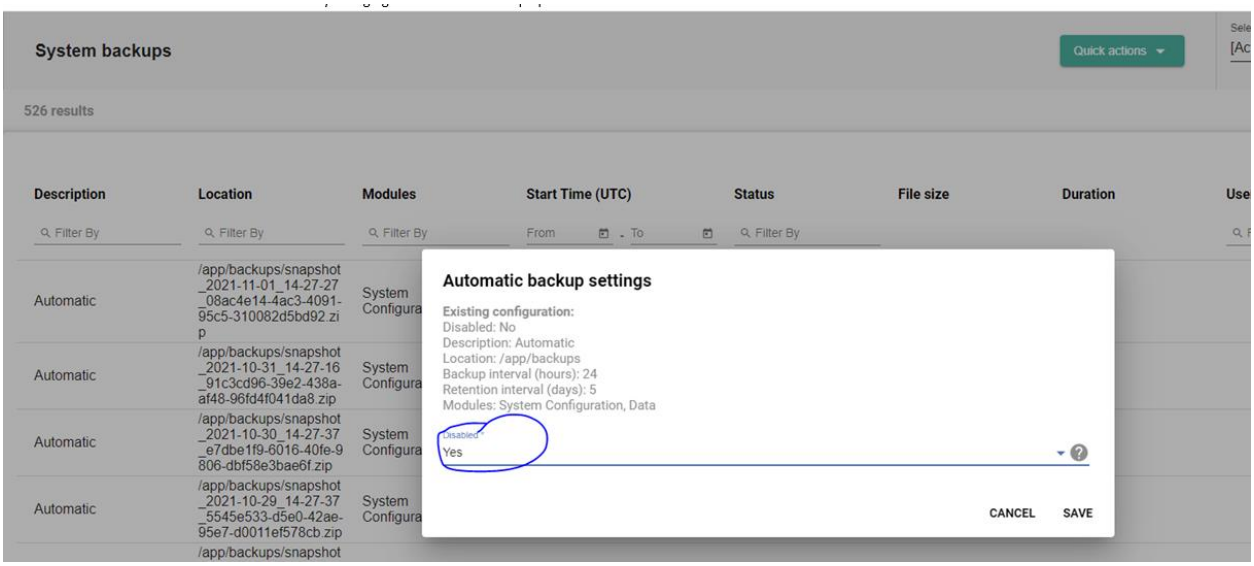
"jobSettings": { **"generateFollowups": {** **"batchSize": 500** **},** **"bulkModifyFollowUps": {** **"batchSize": 1000** **},** **"setRelationshipInformationOnPerson": {** **"batchSize": 10000** **},** **"setUsualPlaceOfResidenceLocationIdOnPerson": {** **"batchSize": 10000** **},** **"setUsualPlaceOfResidenceLocationIdOnFollowUp": {** **"batchSize": 10000** **},** **"caseCountMap": {** **"batchSize": 10000** **},** **"importResources": {** **"batchSize": 100,** **"maxFileSize": 4096** **},** **"updateMissingDuplicateKeys": {** **"batchSize": 10000,** **"updateBatchSize": 50** **}** **}**	Configurations for jobs that need to process data in batches. If we don't process data in batches system will fail on larger database due to not having enough memory (available from Go.Data version 34.4 and made improvements to it in builds 34.7, 34.8, 35.0, 36.0 and 38.0). Added with build 2.40.0, "maxFileSize": 4096, default value is 4 GB. Used by import system to limit the maximum size of a file that is imported, simply said if a file is bigger then this value then the system won't import it until config.json maxFileSize is changed.
"caching": { **"location": {** **"enabled": true** **},** **"user": {** **"enabled": true** **}** **}**	Used to cache locations for different query to speed up the retrieval process. caching. Location is used to enable caching for all endpoints that retrieve location data, while caching. User is used to enable caching for user team locations (available from Go.Data version 35.0 onwards)
"cot": { **"containerPath": "./storage/files"** **}**	Where to store chains of transmission snapshots data used on chains of transmission graph page (available from Go.Data version 36.0 onwards)
"cluster": { **"enabled": true,** **"processesNo": "max"** **}**	Allows vertical scaling. Simply said GoData has a main thread that handles request from users, which means that a user needs to wait until previous request was resolved (even if this process is most of the time fast, there might be cases where it takes longer). If you enable clustering then the system can take advantage of multi-processor machine and spawn multiple GoData threads that handle request from users. Enabled by default (available from Go.Data version 36.0 onwards).
"demoInstance": { **"enabled": false,** **"label": "DEMO",** **"style": {** **"opacity": 0.5,** **"backgroundColor": "gray",** **"color": "black",** **"fontWeight": "bold",** **"fontSize": "40px"** **}** **}**	Allows you to display a placeholder on your GoData website on top right side corner in case you want to tell users that this is a demo instance or for any other reason. Disabled by default (available from Go.Data version 36.4 onwards).
"adminEmail": "admin@who.int"	Admin email (available from Go.Data version 37.0 onwards).
"duplicate": { **"disableCaseDuplicateCheck": false,** **"disableContactDuplicateCheck": false,** **"disableContactOfContactDuplicateCheck": false,** **"executeCheckOnlyOnDuplicateDataChange": false**	Used to check for duplicates when you create or modify a case, contact or contact of contact. By default it is enabled which means that the system will always prompt user with a dialog when it detects duplicates so user can tell the system how to handle the duplicate (cancel create / update, merge, mark as not duplicate). (available from Go.Data version 38.0 onwards)

}	
"count": { **"limit": 10000** **}**	Maximum number of records that are counted on list pages that have many records (cases, contact …). This will help when using different filters that take time to count when it comes to millions of records. You still have the option to see the total number by clicking a link that will appear on the right side of the maximum number of records (if that number is reached) (available from Go.Data version 38.0 onwards).
"export": { **"batchSize": 7000,** **"locationFindBatchSize": 1000,** **"noLookupIfPrefilterTotalCountLessThen": 20000,** **"saveFilter": false,** **"saveAggregateFilter": false,** **"xlsx": {** **"maxColumnsPerSheet": 16000,** **"maxRowsPerFile": 1000000** **},** **"xls": {** **"maxColumnsPerSheet": 250,** **"maxRowsPerFile": 12000** **},** **"ods": {** **"maxColumnsPerSheet": 250,** **"maxRowsPerFile": 12000** **}** **},**	Settings to configure export data for all endpoints. Most limitations aren't GoData related but file type related (e.g. xls file doesn't allow more than 255 columns per spreadsheet, or more than ~65000 rows) (available from Go.Data version 38.0 onwards).
"alternateUniqueIdentifierQueryOnImport": { **"case": false,** **"contact": false,** **"contactOfContact": false,** **"labResult": false** **}**	- Introduced in Go.Data version 40.0 - All options are disabled by default - alternateUniqueIdentifierQueryOnImport.**case** if true, enables updating a case on import by searching it either by the UUID (if provided) or by "Case ID" and "Outbreak ID" if provided - alternateUniqueIdentifierQueryOnImport.**contact** if true, enables updating a contact on import by searching it either by the UUID (if provided) or by "Contact ID" and "Outbreak ID" if provided - alternateUniqueIdentifierQueryOnImport.**contactOfContact** if true, enables updating a contact of a contact on import by searching it either by the UUID (if provided) or by "Contact of Contact ID" and "Outbreak ID" if provided - alternateUniqueIdentifierQueryOnImport.**labResult** if true, enables updating a lab result on import by searching it either by the UUID (if provided) or by "Lab sample ID" and "Outbreak ID" if provided
"allowCustomIDsOnCreate": false	- Introduced in Go.Data version 40.0 - This option is disabled by default - allowCustomIDsOnCreate: if true, enables creating records through API using custom ids. You don't have to always provide an id (if custom id isn't provided then API will generate one). This works almost for all resources (cases, contacts, locations...)

Enabling / disabling backups in config.json

By default the config.json backups are enabled, but you can disable them for the instance where backup is not required by changing **"backUp.disabled"** from false to true. Also on a multi go.data instance architecture, you should delegate only one instance for backups otherwise multiple backups will be started. You can do that by changing the "backUp.disabled" option to false for all instances except the one delegated to do the backup.

Furthermore, There are two ways of enabling / disabling backups:
- The setup per instance includes (changing the config.json file properties for ‘’backUp’’ as described in the above screenshot for the required instance)
- The setup for all instances connected to the same database is by changing the automatic backup option to true from UI as shown in the screenshot below: This option disables backups for all instances.



The automatic backup disable option from the (UI) which disables backups for all instances that connects to the same database was introduced in Go.data version "2.36.2" onwards whiles the automatic backup disable for a specific instance which is configured in the (config.json file) was introduced in Go.data version "2.36.4" onwards

Note: Is always advisable to upgrade your version of Go.data to the latest version since each newer version brings many improvements and fixes. (Current version is 2.39.0) Before upgrading always check "Release notes" to see if there are breaking changes for which you might need to prepare in advanced (like scripts that used an endpoint that was changed)

Go.data "Release notes" can be found using the link below:

https://community-godata.who.int/topics/release-notes/5fd8eca8f5c77e114e6c7de2

Datasources.json file configuration

As the name implies, the file is used to define various configurations regarding data source access (e.g. mysql connector, mongo connector etc.) The default information about this file can be found here: https://loopback.io/doc/en/lb3/datasources.json.html

JSON Fragment	Explanation

"email": {
"name": "email",
"connector": "mail",
"transports": [
{
"type": "SMTP",
"host": "smtp.example.com",
"secure": true,
"port": 465,
"auth": {
"user": "user",
"pass": "password"
}
}
]
},
	Configure the SMTP server used to dispatch Go.Data's forgotten password email. – which is the only email generated by the system.
	
	The transport section describes the server address and port for communication with the SMTP and whether this is secure or not.
	
	If secure is true then the auth section must be included, otherwise it should be removed.
	
	After upgrade of Go.Data, this section should be rechecked as it may get restored by the update and unwanted JSON fragments re-inserted.

For more details on the configuration of SMTP, see the appropriate section later in this document

Datasource.json properties for filters

- mongoDb (prohibitHiddenPropertiesInQuery) should always be configured to false, otherwise some query filters won't work this new property comes with version 39.0 and it will be added automatically to all instances that will upgrade to 39.0 (linux & windows)
- This new property should be added right after AuthSource as shown in the screenshot below however API will add this properties automatically once Go.Data is started with version 39.0 therefore users wouldn't be required to do anything.

component-config.json file

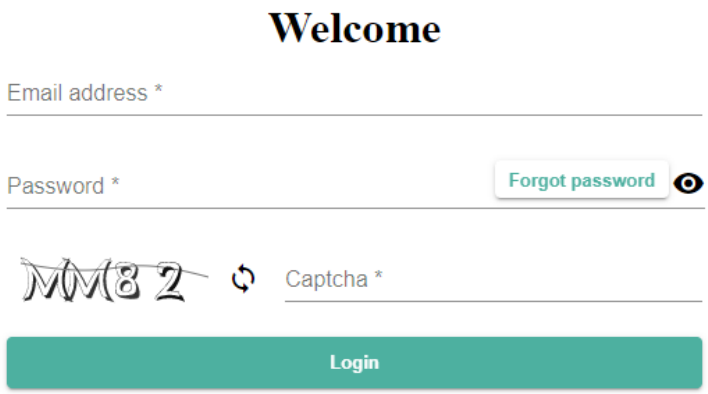
A file specific to the use of loopback, included with Go.Data and as such, should not need to be modified for any reason.

middleware.json file

A file specific to the internal set-up of Go.Data and as such, should not need to be modified for any reason.

Configuring captcha

Captcha was included in V34.5 of Go.Data onwards and can optionally be activated to challenge the user at login, password reset email request and the resetting of the password.



Captcha is configured using the following fragment in the *config.json* file as described in an earlier section dealing with that file.

```
"captcha": {
  "login": false,
  "forgotPassword": false,
  "resetPasswordQuestions": false

},
```

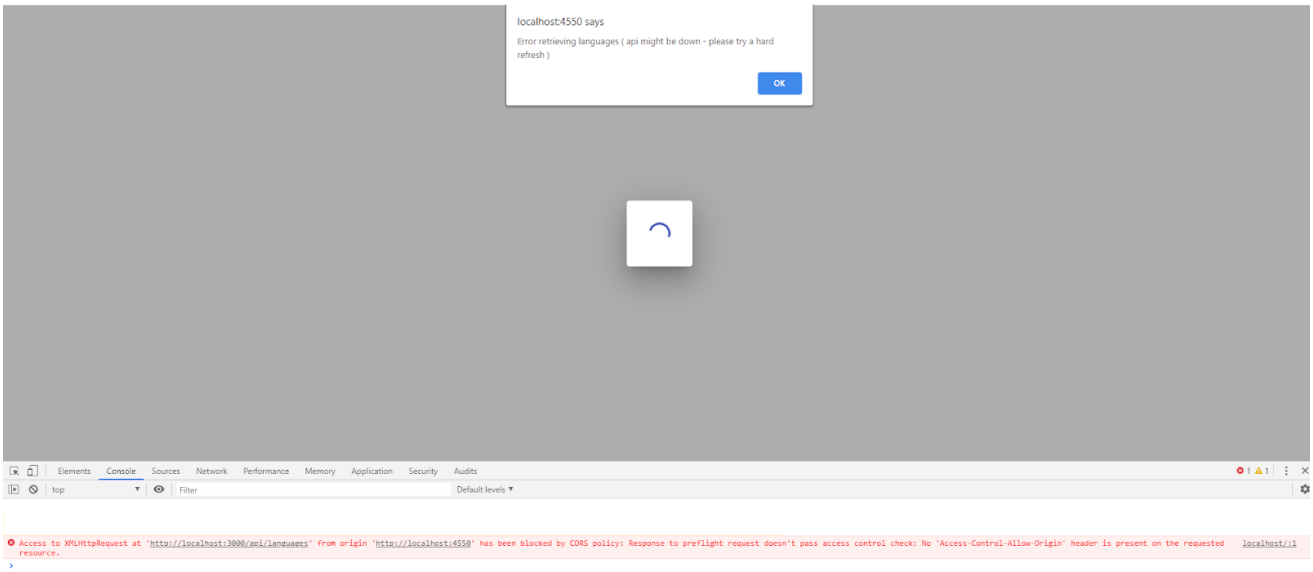
Configuring CORS

Cross original resource sharing is configured in the *config.json* file using the following json fragment: -

```
"cors": {
  "enabled": false,
  "whitelist": []
},
```

This can cause issues, particularly if your Go.Data server is behind a reverse proxy in which Go.Data may perceive an inbound request as being cross-origin. If this is the case then, on attempting to browse Go.Data you will likely see an error with the very first call made by the tool to its API – the call to retrieve interface languages for the login page.

If you see an error similar to the one below and the login page refuses to load, then you need to examine the *config.json* file and set CORS to false or whitelist the proxy server and restart Go.Data.



If you arrive on the Go.Data login page and see the red console error: "Access to … from origin has been blocked by CORS policy...", it means that CORS is enabled and it wasn't configured properly, or it accessed from a website that isn't whitelisted.

Starting with Go.Data version 2.34.4 CORS is disabled by default to avoid confusions. Previously CORS policy was enabled by default.

In order to configure the service correctly, additional settings must be modified in the ***config.json*** file located in {your\installation\directory}\bin\resources\go-data\build\server

Here is a list of possible configuration options:

Disable CORS from ***config.json*** file, followed by a restart of the API.
Very important: you need to update the enabled property **under "cors" object** and **not** cors property under remoting.

```
"cors": {
  "enabled": false,
  "whitelist": []
}
```

Keep CORS active, followed by disabling config rewrite and configure public properties, to match the way the website is accessed (same as before, requires API restart after changing config)

```
"public": {
  "host": "test.host.com",
  "protocol": "http",
  "port": 80
},
"enableConfigRewrite": false,
```

Keep CORS active, followed by configuring CORS whitelist (same as before, requires API restart)

```
"cors": {
  "enabled": true,
  "whitelist": [
            "http://www.test.com"
  ]}
```

As a rule, make sure that CORS isn't enabled or that the domains from which the requests are made are properly whitelisted (if requests are made from a browser that sends headers used by CORS).

Configuring SMTP

Locate the *datasources.json* file. On Windows, for a default install this is under C:\Go.Data\bin\resources\go-data\build\server

Update this json fragment with the details of the SMTP server to be used: -

```
"type": "SMTP",
"host": "smtp.example.com",
"secure": true,
"port": 465,
"auth": {
  "user": "user",
  "pass": "password"
}
```

If the SMTP server does not need authentication, then the whole "auth" block can be removed.

Change the *datasource.json* file from API source files (server directory).

The library used to send emails (nodemailer) allows unauthenticated SMTP requests by removing the 'auth' section altogether from the 'email' configuration.

See: https://nodemailer.com/smtp/#authentication

General options

- port – is the port to connect to (defaults to 587 if is *secure* is *false* or 465 if *true*)
- host – is the hostname or IP address to connect to (defaults to *'localhost'*)
- auth – defines authentication data (see authentication section below)
- authMethod – defines preferred authentication method, defaults to 'PLAIN'

Hostnames for the **host** field are resolved using `dns.resolve()`. If you are using a non-resolvable hostname (e.g. something listed in */etc/hosts* or you are using different resolver for you Node apps) then provide the IP address of the SMTP server as **host** and for the actual hostname user **tls.servername** parameter. This way not hostname resolving is attempted but TLS validation still works.

TLS options

- secure – if *true* the connection will use TLS when connecting to server. If *false* (the default) then TLS is used if server supports the STARTTLS extension. In most cases set this value to *true* if you are connecting to port 465. For port 587 or 25 keep it *false*
- tls – defines additional node.js TLSSocket options to be passed to the socket constructor, e.g. *{rejectUnauthorized: true}*.
- tls.servername - is optional hostname for TLS validation if host was set to an IP address
- ignoreTLS – if this is *true* and *secure* is false then TLS is not used even if the server supports STARTTLS extension
- requireTLS – if this is *true* and *secure* is false then Nodemailer tries to use STARTTLS even if the server does not advertise support for it. If the connection cannot be encrypted, then message is not sent

Setting **secure** to **false** does not mean that you would not use an encrypted connection. Most SMTP servers allow connection upgrade via [STARTTLS](https://tools.ietf.org/html/rfc3207#section-2) command but to use this you have to connect using plaintext first

Connection options

- name – optional hostname of the client, used for identifying to the server, defaults to hostname of the machine
- localAddress – is the local interface to bind to for network connections
- connectionTimeout – how many milliseconds to wait for the connection to establish (default is 2 minutes)

- greetingTimeout – how many milliseconds to wait for the greeting after connection is established (default is 30 seconds)
- socketTimeout – how many milliseconds of inactivity to allow (default is 10 minutes)

Debug options

- logger – optional bunyan compatible logger instance. If set to true then logs to console. If value is not set or is false then nothing is logged
- debug – if set to true, then logs SMTP traffic, otherwise logs only transaction events

Security options

- disableFileAccess – if true, then does not allow to use files as content. Use it when you want to use JSON data from untrusted source as the email. If an attachment or message node tries to fetch something from a file, the sending returns an error
- disableUrlAccess – if true, then does not allow to use URLs as content

If authentication data is not present, the connection is considered authenticated from the start. Otherwise you would need to provide the authentication options object.

auth is the authentication object:

- type indicates the authentication type, defaults to 'login', other option is 'oauth2'
- user is the username
- pass is the password for the user if normal login is used

Configuring base maps

The Go.Data "web" server has several reports which provide a spatial visualization of the data; the case count map and the case/contact movement map. Go.Data is not intended as a full geographical information service (GIS) solution and only provides the basic tools to support the key operation of contact tracing.

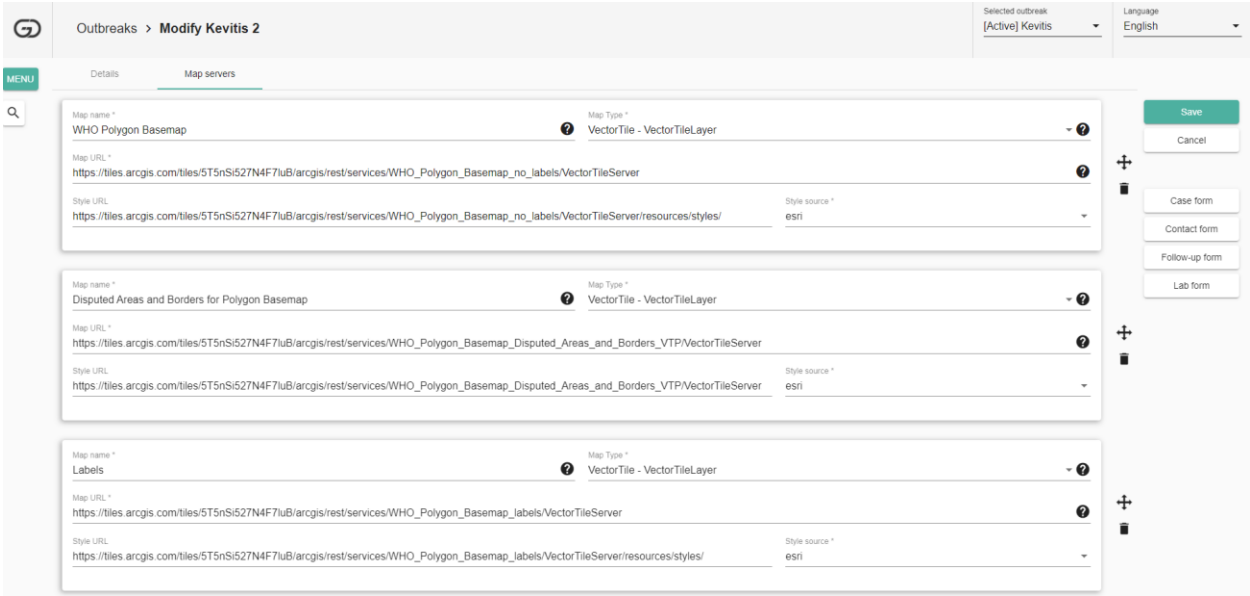
In Go.Data's mobile phone application, the phone's own native mapping software is used for any spatial visualization.

Maps in the Go.Data server are produced using two elements: -

1) All maps are point maps (there are no choropleth) and use the latitude and longitude either entered directly as part of the address of a case/contact or read from the Go.Data reference data of locations.

2) These points are overlaid on one or more “base map(s)” the background canvas on which the data is shown.

For the “base maps”, these can be set on an outbreak by outbreak basis, by giving Go.Data the URLs of the map servers in the second tab when creating/editing an Outbreak.



The URLs given must resolve to a map server so that Go.Data can read in the needed information and create the map background. For this reason, the map visualization only works in Go.Data when online, there is no caching of this information for offline use.

If multiple map servers are given in the screenshot above, then Go.Data draws them in the order in which they are given i.e. the first one on the page is first and the second will overlay the first etc.

When first creating an outbreak, the “Map servers” tab (shown in the screenshot above) will be blank and if left blank, then Go.Data assigns the default map servers as specified in the *config.json* file in json fragment “defaultArcGisServers”: -


```
  "defaultArcGisServers": [
    {
      "name": "WHO Polygon Basemap",
      "url":
"https://tiles.arcgis.com/tiles/5T5nSi527N4F7luB/arcgis/rest/services/WHO_Polygon_Basemap_no_labels/VectorTile
Server",
      "type":
"LNG_REFERENCE_DATA_OUTBREAK_MAP_SERVER_TYPE_VECTOR_TILE_VECTOR_TILE_LAYER",
      "styleUrl":
"https://tiles.arcgis.com/tiles/5T5nSi527N4F7luB/arcgis/rest/services/WHO_Polygon_Basemap_no_labels/VectorTile
Server/resources/styles/",
      "styleUrlSource": "esri"
```

```
    },
    {
      "name": "Disputed Areas and Borders for Polygon Basemap",
      "url":
"https://tiles.arcgis.com/tiles/5T5nSi527N4F7luB/arcgis/rest/services/WHO_Polygon_Basemap_Disputed_Areas_and
_Borders_VTP/VectorTileServer",
      "type":
"LNG_REFERENCE_DATA_OUTBREAK_MAP_SERVER_TYPE_VECTOR_TILE_VECTOR_TILE_LAYER",
      "styleUrl":
"https://tiles.arcgis.com/tiles/5T5nSi527N4F7luB/arcgis/rest/services/WHO_Polygon_Basemap_Disputed_Areas_and
_Borders_VTP/VectorTileServer/resources/styles/",
      "styleUrlSource": "esri"
    },
    {
      "name": "Labels",
      "url":
"https://tiles.arcgis.com/tiles/5T5nSi527N4F7luB/arcgis/rest/services/WHO_Polygon_Basemap_labels/VectorTileServ
er",
      "type":
"LNG_REFERENCE_DATA_OUTBREAK_MAP_SERVER_TYPE_VECTOR_TILE_VECTOR_TILE_LAYER",
      "styleUrl":
"https://tiles.arcgis.com/tiles/5T5nSi527N4F7luB/arcgis/rest/services/WHO_Polygon_Basemap_labels/VectorTileServ
er/resources/styles/",
      "styleUrlSource": "esri"
    }
  ],
```

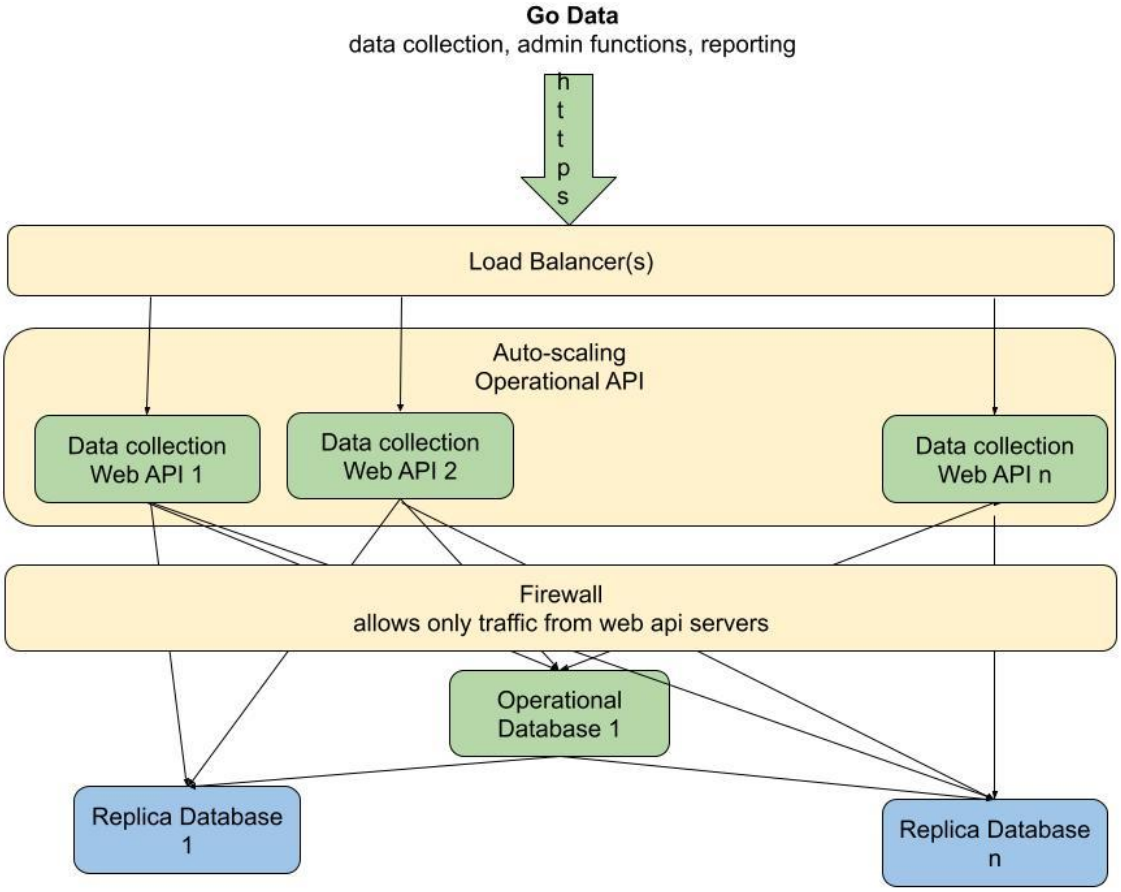

After installation, Go.Data provides three WHO-hosted map servers as the default setting, these are the WHO base map, disputed borders and territories and a layer of reference data for place names.

A useful list of other map servers that work with Go.Data is found at: -

https://community.microstrategy.com/s/article/KB47086-List-of-some-available-basemap-examples-to-be-used-with?language=en_US

Configuring load balancer

Go.Data can be deployed with a load balancer to create a network of machines for higher performance and hot fail-over. A typical network deployment is shown in the diagram on the next page.



The elements of this set-up are: -

A) Traffic is received by the load balancer.
B) HTTPS can be enabled at the level of the load balancer.
C) The load balancer is responsible for routing traffic to 2 or more servers on which Go.Data is installed. This approach does not need to use sticky sessions.
D) The Go.Data database must have been separated from the web servers as described in the previous section to ensure that all copies are using the same, centralized MongoDB.
E) MongoDB replica should be created accordingly to MongoDB documentation. https://docs.mongodb.com/manual/replication/

F) The connection parameters to database should be changed to

"mongodb://<USER>:<PASSWORD>@<PRIMARY_IP>,<SECONDARY_IP>?replicaSet=<REPLICASET_NAME>&readPreference=secondaryPreferred"

G) A firewall or other mechanism should be used, as described in the previous section, to ensure that only authorized machines can communicate with the Go.Data database.

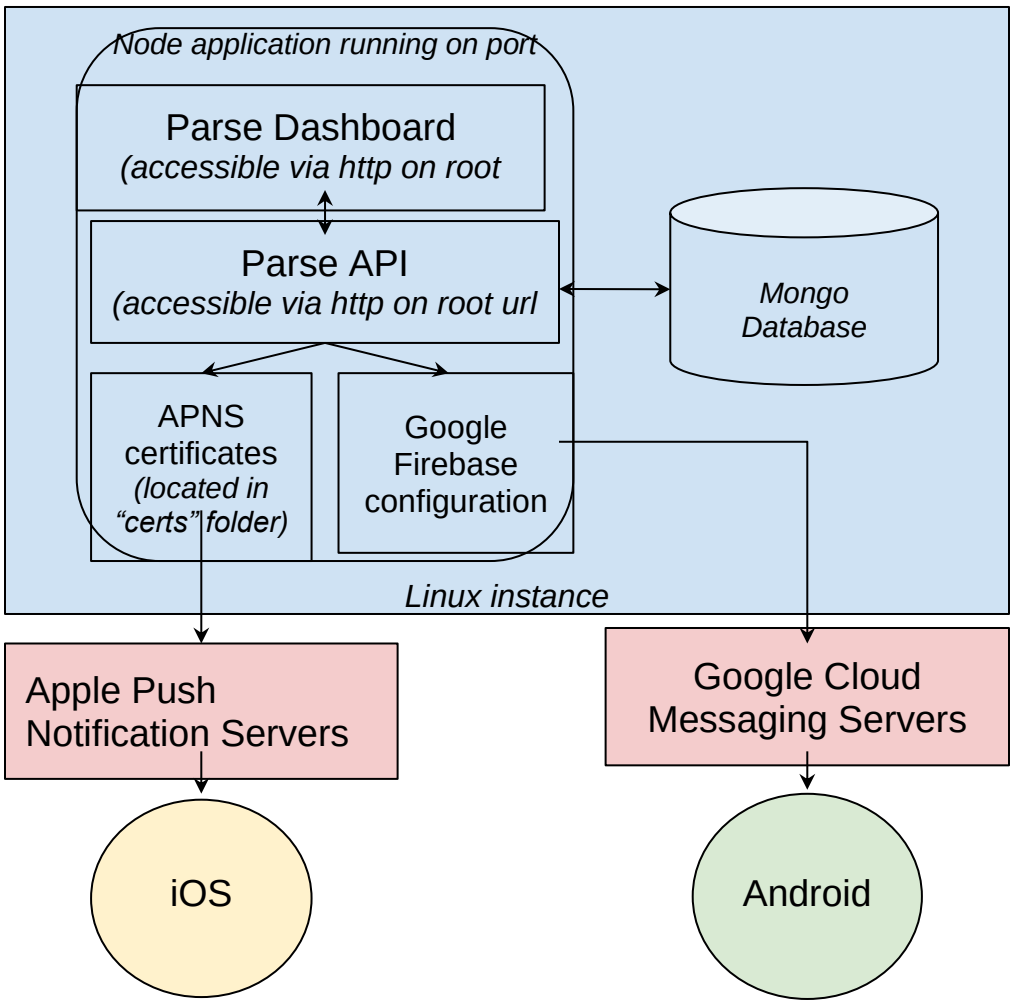
If your hosting environment supports auto-scaling, then the API servers can be set as part of the same scaling group so when CPU or Memory usages are high, new instances are created automatically, and when the CPU or Memory usage are low the newly created instances will be destroyed. This will ensure higher availability, better response time and effective costs. All major cloud hosting providers (Azure, Google etc.) offer some form of autoscaling.

Configuring parse server

The Parse server is a dedicated server used to send push notifications to the devices that installed the Go.Data mobile app. The push notifications are used to remotely wipe the devices.

Push notifications server

The application has the following design:



Let's assume that the IP of this Linux instance is linked to the subdomain http://parse.who.int. This means that the Parse dashboard will be accessible at http://parse.who.int:1337 and the Parse API will be accessible at http://parse.who.int:1337/api.
The dashboard authentication is performed via the users from the configuration file, and the API authentication is performed via the "appId" and "masterKey" values.

A firewall rule on the application port may be required to allow access to the application from the Internet.

Configuration file

The configuration file for the parse server is as follows: -

```
{
  "port": 1337,
  "database": {
    "uri": "mongodb://172.31.47.246:27017/go-data-push-notifications"
  },
  "app": {
    "serverURL": "http://whocd.clarisoft.com:1337/api",
    "appId": "b61f5946-1af3-4e07-9986-9ffd1e36ae93",
    "masterKey": "KlYddh2OdVycHuVBhXv2",
    "appName": "Go.Data"
  },
  "users": [
    {
      "user": "admin@who.int",
      "pass": "admin"
    }
  ],
  "allowHttpAccess": true,
  "push": {
    "android": {
      "senderId": "674826924884",
      "apiKey": "AIzaSyBk3ARREFLke2Srem4OGT6lGtUuZm5xSQ4"
    },
    "ios": {
      "pfx": "certs/WHO-APNS.p12",
      "passphrase": "a",
      "topic": "com.clarisoft.who",
      "bundleId": "",
      "production": false
    }
  }
}
```

- port - this is the port where the application is running. It defaults to 1337, it can be changed to anything between 1025 and 65535.
- database.uri - this is the URL for the database. Since the mongo database runs on the same instance as the application, it can be set to “mongodb://127.0.0.1:port/databaseName”. The Mongo port can be configured when starting the Mongo service (see Configuring Mongo) and you can choose any database name, the application will create it automatically.
- app.serverURL. This is the application API where the application dashboard will connect to display data in the Parse portal.
- app.appId - This is a regular string that is used with the “app.masterKey” value to authenticate the API requests. Do not share the application id.
- app.masterKey - This is a regular string that is used with the “app.appId” value to authorize the API requests. Do not share the master key.
- app.appName - This value is only used to display the application name in the Parse Dashboard.
- users - a list of usernames and passwords that can access the Parse Dashboard. Please change to more secure credentials than the current default ones.
- allowHttpAccess - setting this value to “false” will only allow https access.
- push.android.senderId - this value can be found on Firebase Console settings (see Configuring the Firebase Cloud messaging settings).
- push.android.apiKey - this value can be found on Firebase Console settings (see Configuring the Firebase Cloud messaging settings).
- push.iOS.pfx - path to the APNS certificates. These certificates expire annually therefore they must be renewed every year.

- push.iOS.passphrase - once created, APNS certificates can be exported in the p12 format and can optionally have a password. If they do have a password, this value should contain it.
- push.iOS.topic - the bundle id of the iOS app.
- push.iOS.bundleId - leave blank.
- push.iOS.production - set to “true”.

MongoDB

If Mongo is not installed on the instance or you want to move the Parse server on to a new instance, install Mongo 3.2 using the official documentation. By default, Mongo will be configured to run on port 27000. You can change this port to any free port between 1025 and 65535. Mongo for Go.Data is scheduled to be upgraded to a stable version of Mongo (5.x) in upcoming Go.Data version 40.

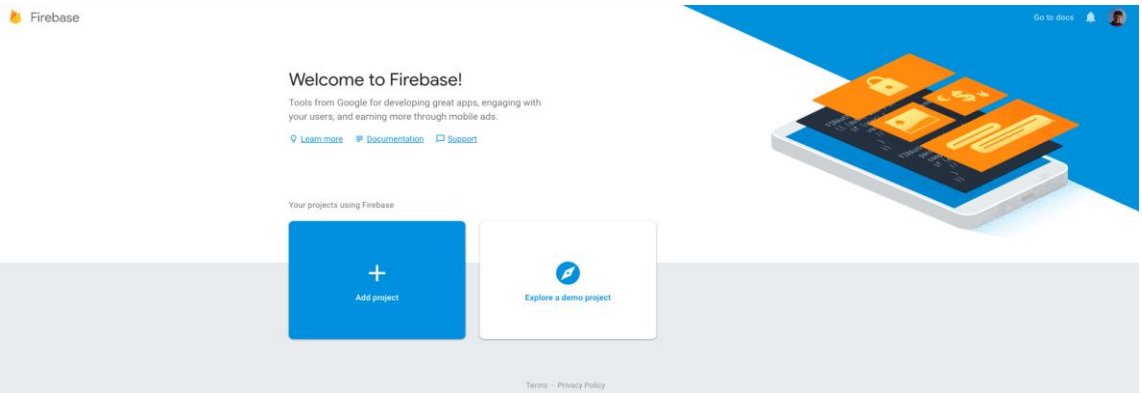
By default, the database engine will not have authentication and the Parse Server connects to the database without performing any authentication.

With this design, it is absolutely required to prevent any outside access on the database port using firewall rules (block outside access on port 27000 or the custom configured port) and with the Mongo “bindIp” setting.

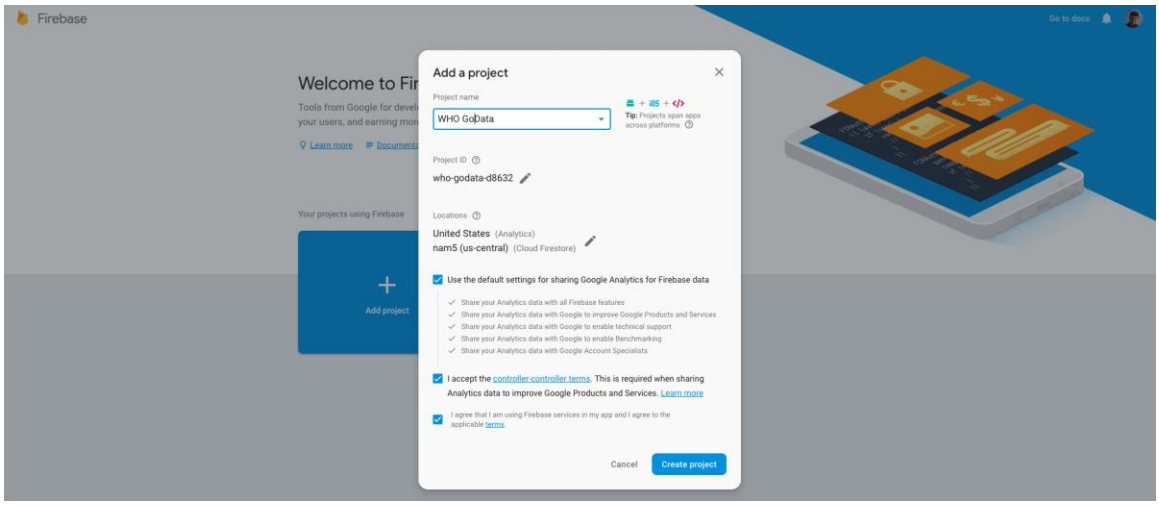
By default,, starting with Mongo 3.2, the database engine allows connections only from the local host, which is the desired behavior to prevent unauthorized access. Please make sure you check this configuration nevertheless.

Configuring firebase cloud

Google Firebase is one component of the parse server set up and settings can be configured by accessing Google Firebase Console. If the Go.Data project hasn’t been configured or you want to move to another Google account, you will have to set up a new project.

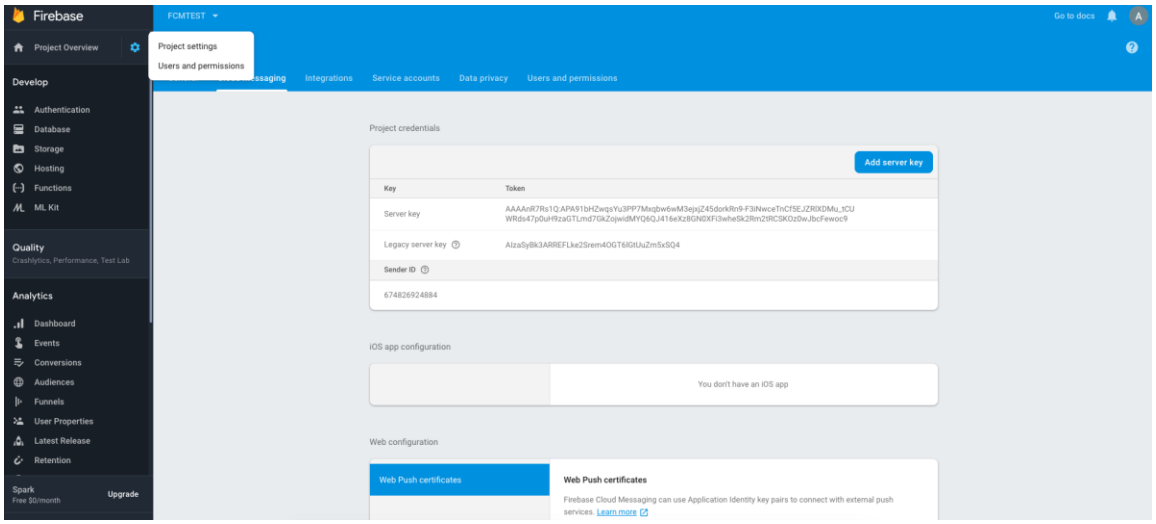


Create a new project with default settings and accept the terms and conditions.



API Key

Go to Project Overview - Project Settings, and select the “Cloud Messaging” tab.



Here, you will find the API key (“Legacy server key”) and Sender ID that must be set in the Parse configuration file.

Log files

Application and database logs are always available in the application data folder, default to:

OSX - *~/Library/Application Support/Go.Data/logs*

Windows current user - *[Windows Drive:]\Users\[Current User]\AppData\GoData\logs*

Windows all users - *[Windows Drive:]\GoData\data\logs*

Windows logs

For Windows, Go.Data's log files can be found in two locations depending on the type of the installation:

1. If you chose to install Go.Data for your user only:
    1.1. Application logs: {your\installation\folder}\Go.Data\logs\app
        E.g.: C:\Users\YourUser\AppData\Roaming\Go.Data\logs\app
    1.2. API/Service logs: {your\installation\folder}\Go.Data\resources\go-data\build\logs
        E.g.: C:\Users\YourUser\AppData\Local\Programs\Go.Data\resources\go-data\build\logs
    1.3. Database logs: {your\installation\folder}\Go.Data\db
        E.g.: C:\Users\YourUser\AppData\Roaming\Go.Data\db

2. If you choose to install Go.Data for everyone:
    2.1. Application logs: {your\installation\folder}\Go.Data\data\logs\app
        E.g.: C:\Go.Data\data\logs\app
    2.2. API/Service logs: {your\installation\folder}\Go.Data\bin\resources\go-data\build\logs
        E.g.: C:\Go.Data\bin\resources\go-data\build\logs
    2.3. Database logs: {your\installation\folder}\Go.Data\data\logs\db
        E.g. : C:\Go.Data\data\logs\db

Logging level can be configured by changing the config file (*config.json* from server directory).

Go.Data can write down to log files any action that has the same or a higher level as that configured in the *config.json*.

Can be one of the following values:

- *debug => log everything*
- *info*
- *error*

Linux logs

On Linux: -

- Application logs (only for windows): {your\installation\folder}\Go.Data\data\logs\app
  e.g.: C:\Go.Data\data\logs\app
- API logs: {your\installation\folder}\Go.Data\bin\resources\go-data\build\logs
  e.g.: C:\Go.Data\bin\resources\go-data\build\logs
- Database logs: {your\installation\folder}\Go.Data\data\logs\db
  e.g. : C:\Go.Data\data\logs\db

Language files

With regards to missing language labels that do not appear while preserving any custom ones that might be there already.
If this is a custom language, then you need to download again one of the default languages, change it and upload it on the custom language. With Go.Data upgrades from version 38.1 onwards default language translations won't be overwritten anymore.
If the issue is caused by having a custom language in which some new tokens were added in a version after that language file was created this will be addressed in version 39.0 which will copy tokens translations to all custom languages if they don't have translations (default translation is in English)
For older versions of Go.Data the workaround will be to take a backup of the language file, update and then re-upload the original file in the system again.
For older versions you need to download one of the default languages again, translate tokens, import it over the custom language (this might take a long time to do)

Note: When upgrading to 39.0, system fills any missing language tokens for all languages (including custom ones), so no need to import, change and upload again, but yes that would fix any missing tokens too

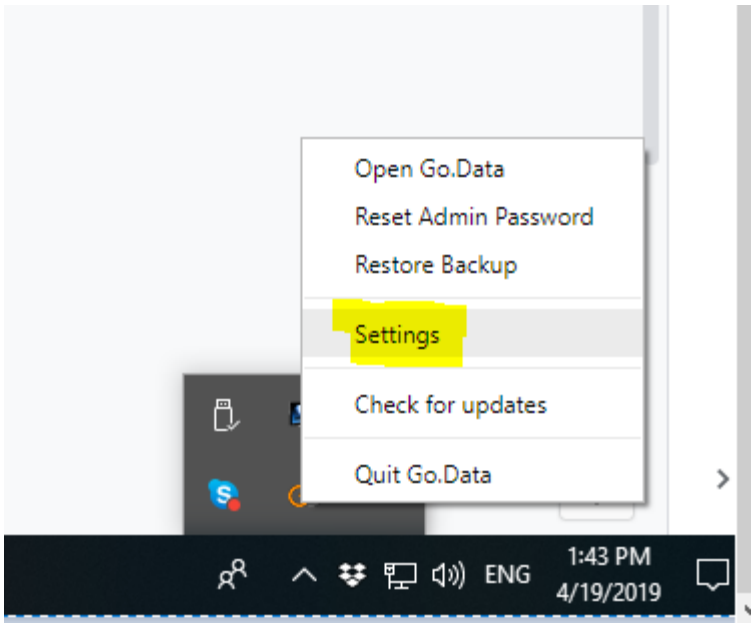
Working with MongoDB

In normal operation, users should not need access the MongoDB and interaction directly with the source database is discouraged due to the high likelihood of corrupting data.

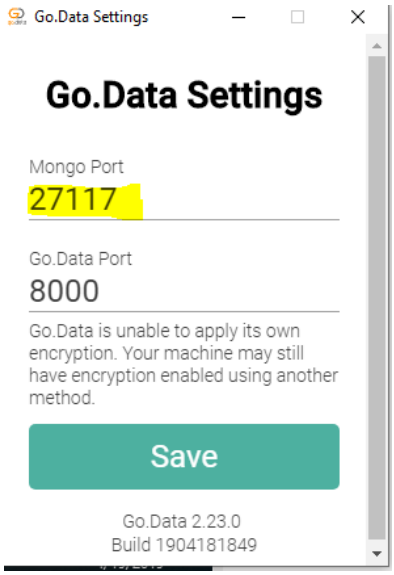
If however, for debugging or low/level intervention a user does need access to MongoDB then you can proceed as follows.

For Windows applications:

1. Install Robo 3T (Robo 3T).
2. Start Go.Data application
3. Determine on which port runs Mongo
    a. Open settings popup

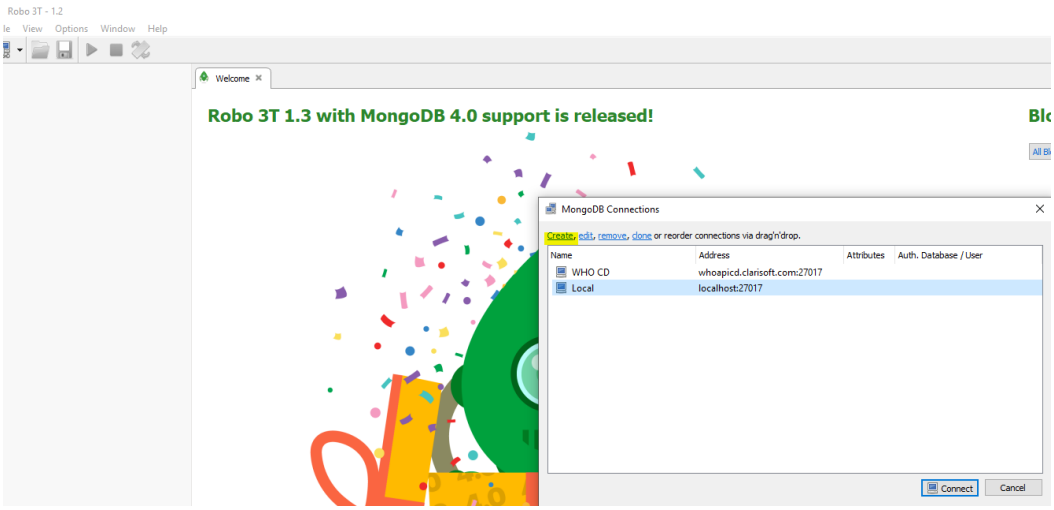


b. Get mongo port (in case you didn't set a custom value, most of the time the port should be the one you set up in steps before)

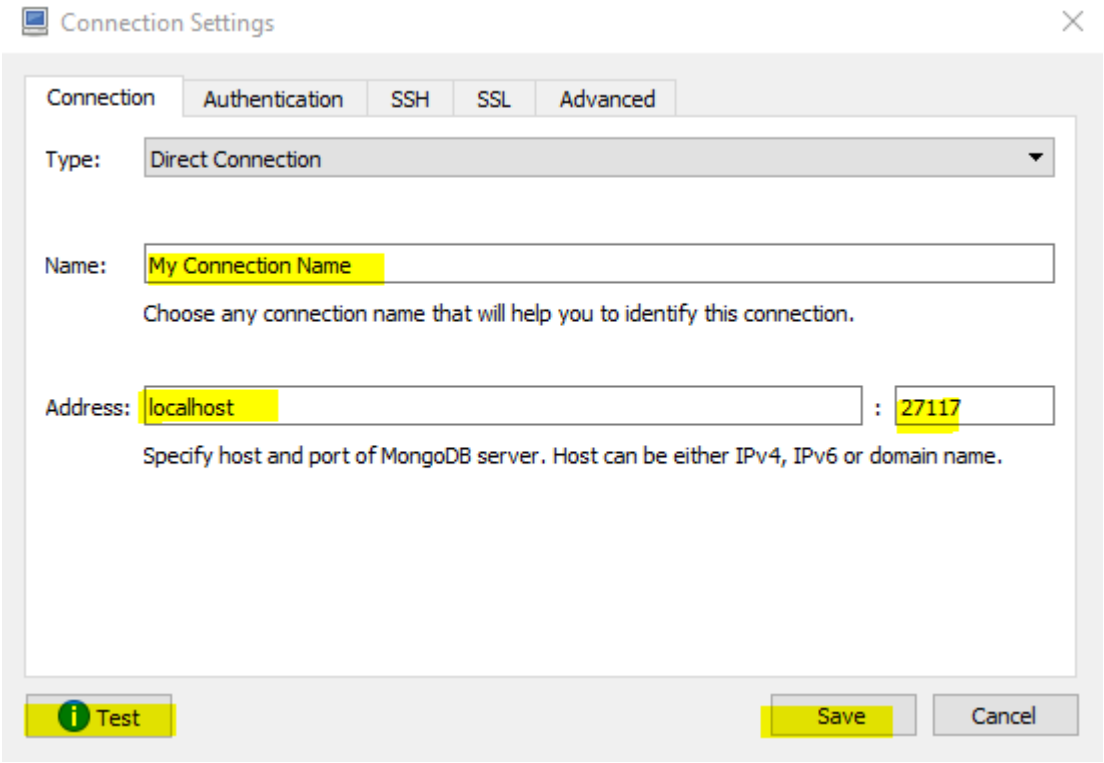


4. Connect to database:
   IMPORTANT: Any changes you make to the database structure / records might break the application if they break the expected behaviour. A backup before working in the database is recommended.

   a. Open Robo 3T application



b. Click “Create” connection if you didn’t do this already (this step needs to be done only once, since after that we will use the saved connection that was previously created)

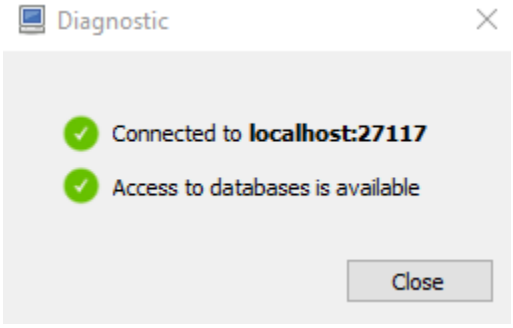


i. Make sure that the following fields are filled
   - Name: Connection Name - you can enter anything here
   - Address
     - if the Go.Data instance was installed on this machine you need to leave **localhost** here

- Port: you need to write down the port you've seen at step 3b (most of the time should be the one used in previous steps, if you didn't change it)

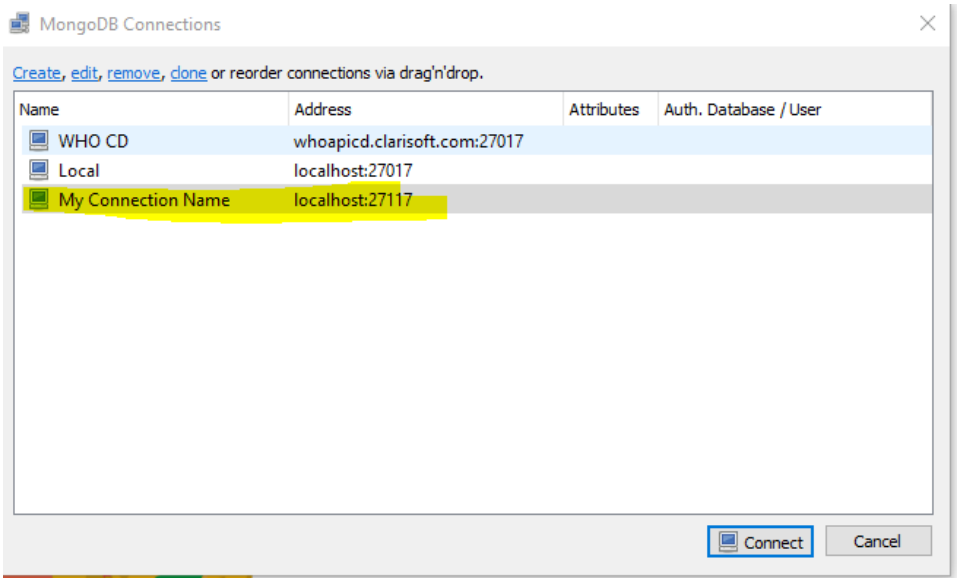
ii. Click "Test"

- It should say that the connection was established



- Otherwise, please make sure that:
  - Go. Data application isn't closed
  - The connection port is the proper one (see steps 3a and 3b)

iii. If connection was established, then you can click the "Save". You should see the connection in the connections dialog that is opened every time you start Robo 3T application.

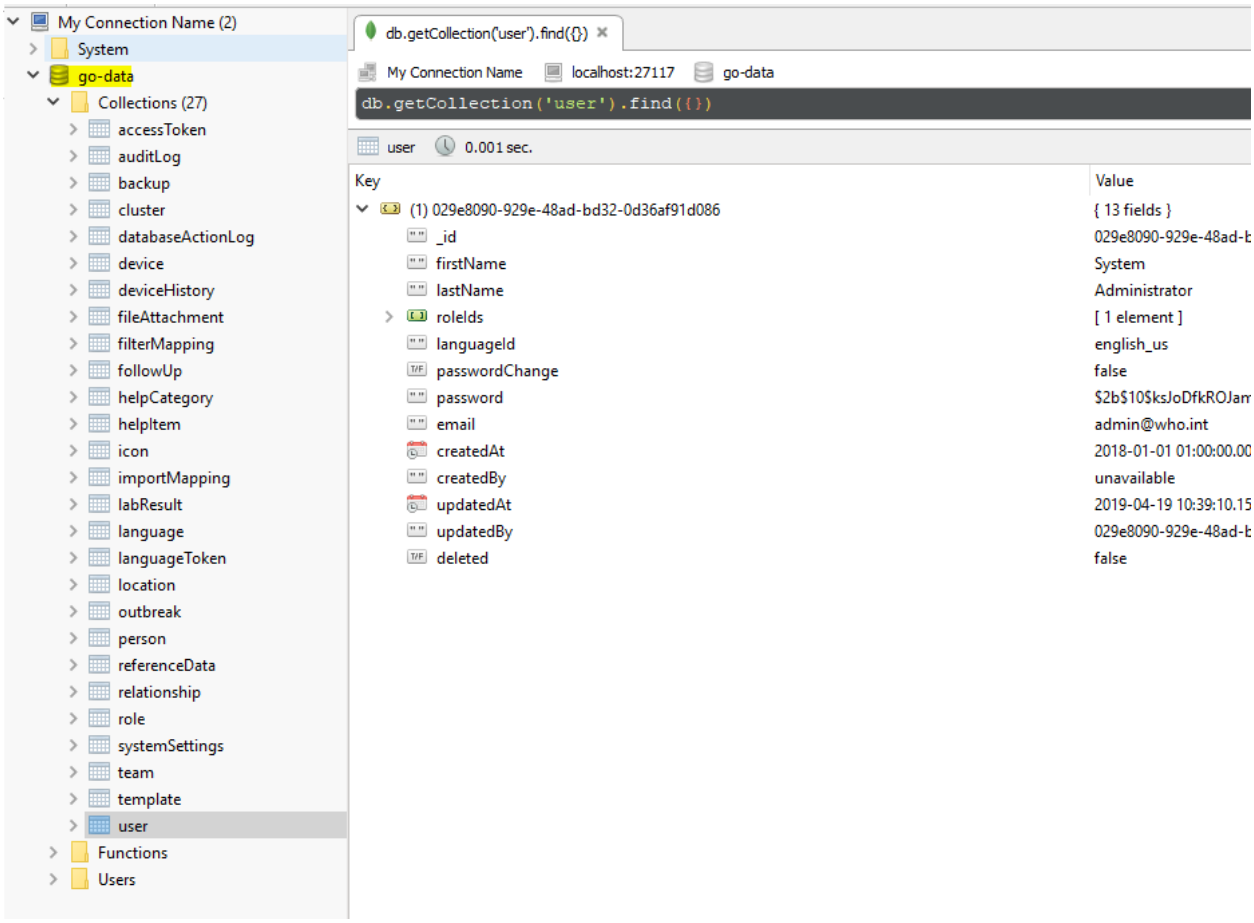


c. Select your connection and click the "Connect" button

i. If the connection was created previously and you can't connect anymore, please make sure that:

- Go. Data application isn't closed
- The connection port is the proper one (see steps 3a and 3b)

d. You should see the Go.Data database now. From here you can view or modify records (**IMPORTANT: Any changes you make to the database structure / records might break the application if they break the expected behaviour**)



Separating web and database elements

By default, Go.Data installs its runtime source-code and MongoDB data store on the same machine. This behavior is by design and appropriate for the majority of situations including stand-alone machines or local networks/servers without performance issues.

In some scenarios however it may be desirable to split these to create machines dedicated to each component of the Go.Data solution.

This separation is desirable for: -

1) Working with Go.Data when larger amounts of data are loaded for performance reasons. For example, if Go.Data is loaded with more than 10,000 cases on a minimum

specification server then the users may notice a degradation in performance. In this scenario, one option might be to split web/database parts to share the load between two servers.

2) For security and disaster recovery, some network administrators may feel that a web/database hosted on the same machine are an elevated risk.

The process for deploying Go.Data on two separate machines is as follows: -

A. Install Go.Data on Server A.
B. Install Go.Data (at the same version as Server A) on Server B.
C. Note the port on which MongoDB is working on Server B and the IP address or URL of the Server B machine. This must of course be visible from Server A.
D. If you're working with a Windows copy of Go.Data then go to the following directory on Server A C:\Go.Data\bin\resources\go-data\build\server and find file *datasources.json*.
E. Edit the *datasources.json* with WordPad or similar and find the following fragment in which you can update the host and port and URL (if needed) to tell Go.Data on Server A where to find its database on server B.
F. You will need to stop and restart the Go.Data service on Server A for this change to take effect.

```
"mongoDb": {
  "host": "127.0.0.1",
  "port": 27000,
  "url": "",
  "database": "go-data",
  "password": "",
  "name": "mongoDb",
  "user": "root",
  "connector": "mongodb",
  "allowExtendedOperators": true,
  "enableGeoIndexing": true,
  "ignoreUndefined": true,
  "maxDepthOfQuery": 24,
  "lazyConnect": true,
  "authSource": "admin"
},
```

G. Server B should be configured to only accept incoming traffic on the database port from Server A's address/IP address – specifically the web API and port to ensure that there is no danger of a user accidentally browsing to Server B and attempting to use the Go.Data web application there. This is also a best practice security approach for limiting access to the database.

The above instructions deal with installing a new copy of Go.Data on two machines. If you're looking to split the database for an existing deployment of Go.Data then the process will be similar but with one additional step; after than installing a new, empty copy of Go.Data at Step B you should perform a backup of Go.Data from Server A and restore that to Server B to ensure that the database is copied across.

Performance tuning

Go.Data has been load tested up to around 100,000 cases and contacts (total) on an average specification machine (4 core CPU, 8GB of RAM) to ensure that performance of the server component remained stable.

Data loads to the mobile app should always be kept lower since the purpose is contact follow-up and there should be a limited number of contacts that an individual can see in one day i.e. there is business justification for setting up Go.Data to only send a sub-set of data to the mobile phone.

Both horizontal and vertical scaling are possible with Go.Data's architecture and the Nodejs worker process which is the main service providing Go.Data processing can also be tuned to allow it to consume more memory. Optimising the business process for using Go.Data can also assist with achieving high performance.

Horizontal scaling

The components of Go.Data can be split between multiple servers as described in the previous sections on splitting web and database components and using a load balancer.

Vertical scaling

For machines with multiple CPUs, Go.Data can be configured to spawn multiple processes and spread load across the available processing power.

This vertical scaling is can be configured through the *config.json* file and is available from V36.2 of Go.Data. In the *config.json* file there is the following settings:

```
"cluster": {
  "enabled": true,
  "processesNo": "max"
}
```

By default, when Go.Data is upgraded to 36.2 or to a newer version this option is enabled by default. If it is not enabled however, then this can be changed by setting the *cluster.enabled* option to *true* (e.g. see above example) and setting *cluster.processesNo* to *max* (by default it should already be set to max).

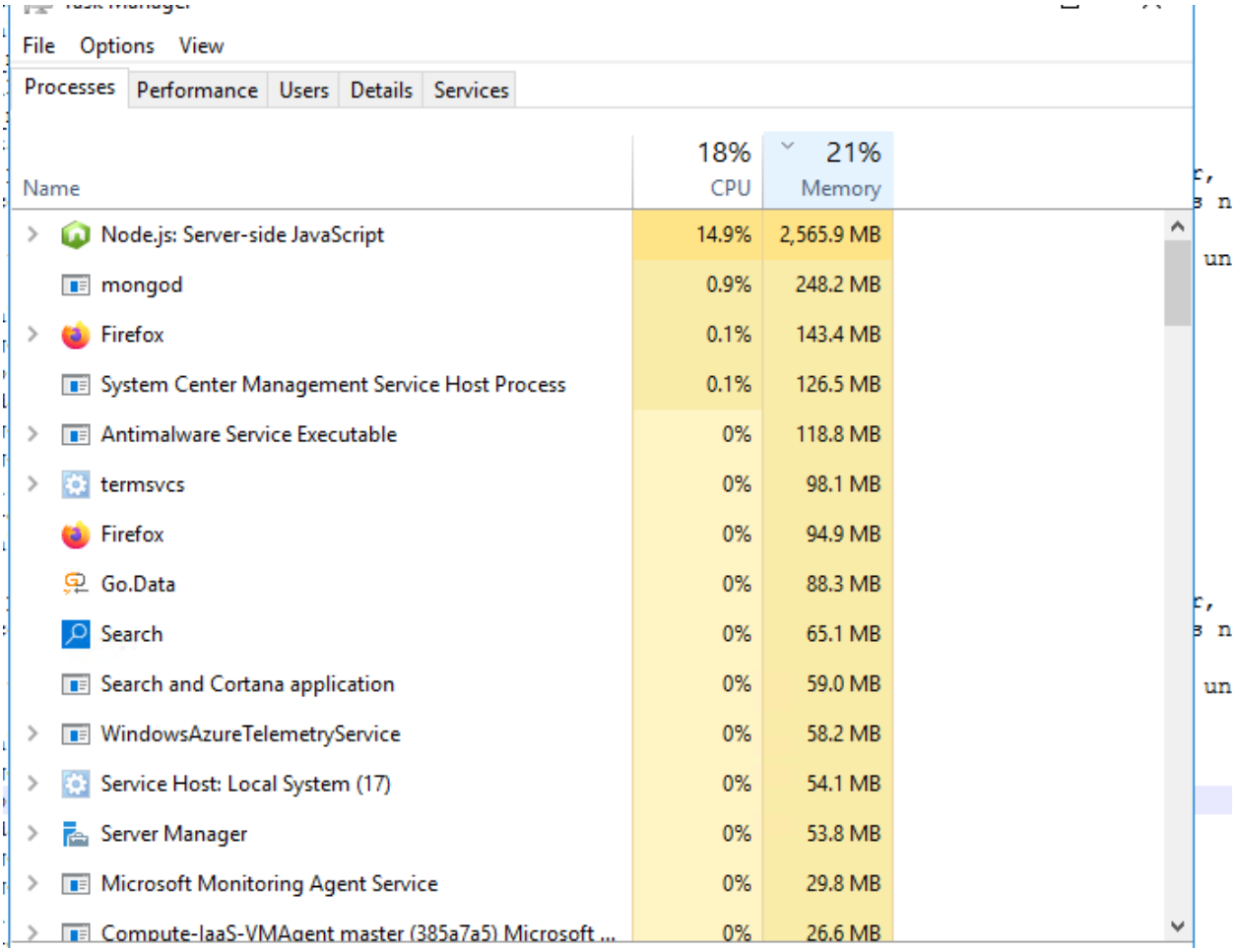
- **cluster.enable**: enables / disables the vertical scaling of a Go.Data server.
- **cluster.processesNo**: defines how many Go.Data processes can be started on the same machine. If this is set to "max" then the system will use the maximum number of processes that can be optimally used (the number of CPUs). The number can be reduced if you don't want to use all CPUs.

Note that before V36.2 when Go.Data uses a single core and couldn't take advantage of multiple physical or virtual cores then you are better off provided a single high-performance core rather than multiple cores.

Nodejs process tuning

If you experience a degradation of performance at higher data loads in Go.Data, then tuning of the worker process itself can also help.

Go.Data's main worker process is the *Nodejs* process as shown below – you should check if there is a memory limit on the RAM that can be given to this process and increase it if so. On a Windows 10 server, this process sometimes failed when it reached 1.4Gb of memory and so increasing to 3GB or up to 8GB if needed will help performance.



To configure the memory allocation for all *nodejs* processes running on a machine, you need to set an environment variable (in both Windows and Linux) and configure it so that it persists after machine restart and does not lose its value.

The environment variable that is needed is **NODE_OPTIONS**. More about it can be found here:

https://nodejs.org/docs/latest-v8.x/api/cli.html#cli_node_options_options

The property **max-old-space-size** is the maximum memory that is allocated.
The value of NODE_OPTIONS environment variable should be something like: -

'--max-old-space-size=4096'

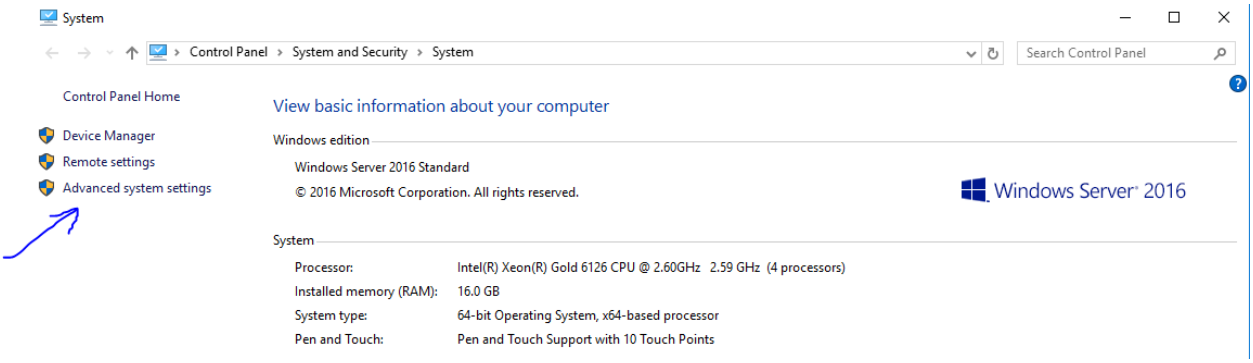
This translates to "maximum memory allocation for each one of the *nodejs* processes is 4096 MB".

The maximum memory value is in megabytes and can be any value as long as the host machine has enough memory to accommodate that value.

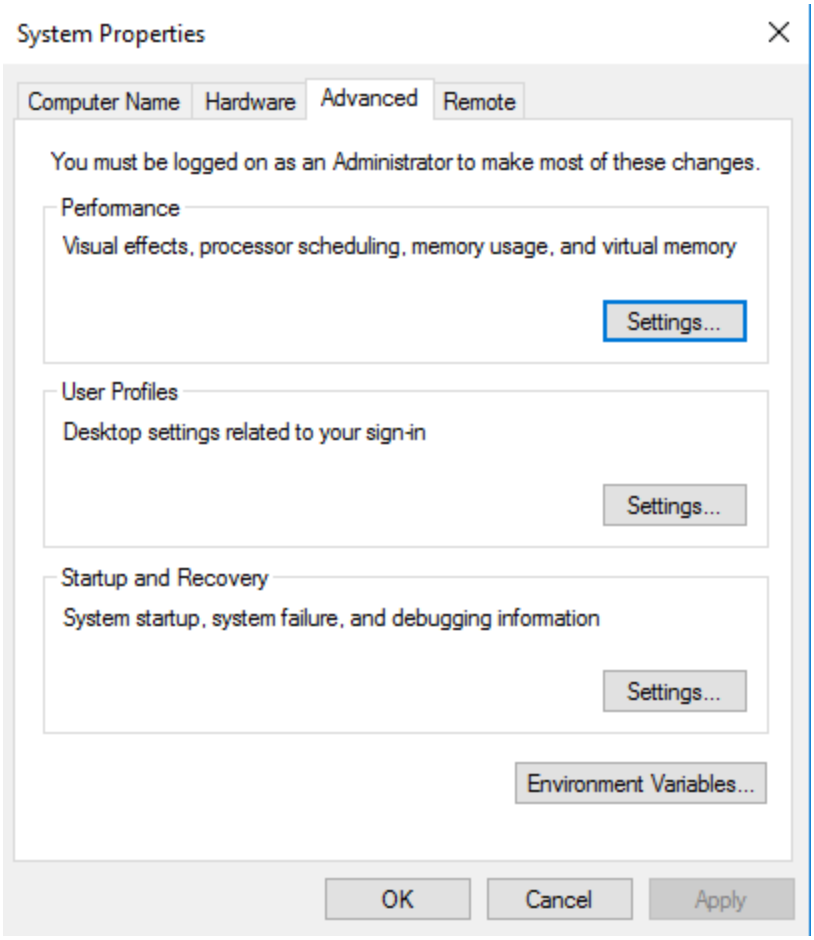
Since multiple *nodejs* processes might run at the same time this means that host machine needs to accommodate the maximum memory allocation across all active processes i.e. number of processes * 4096 in the example above.

Memory will be used on demand so most *nodejs* processes won't reach the maximum memory allocation permitted to them. It is difficult to predict how much memory the host needs to have since it is difficult to predict how many *nodejs* processes will run at the same time and how much memory will be required by each one of them (this depends of things like number of concurrent users and so on).

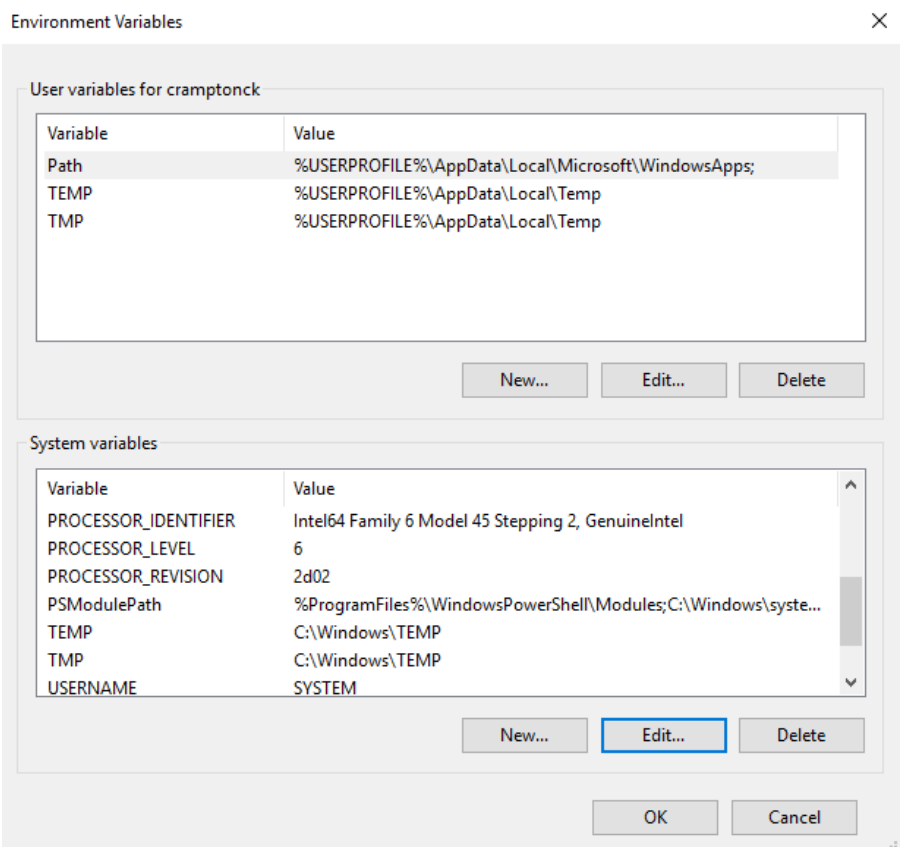
On Windows, the process to increase the memory available to NodeJS is to open the “System” window by right clicking on the Window icon at the bottom left of the screen and choosing “System”. The following Windows (taken from Server 2016) is shown.



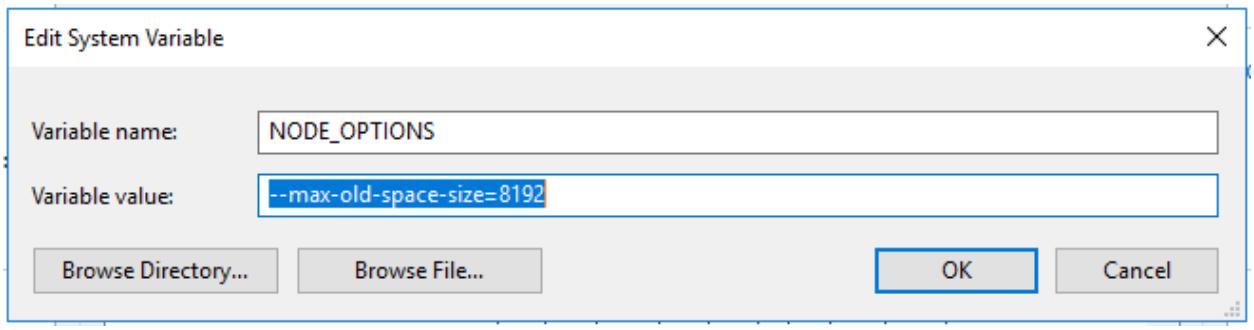
Click on “Advanced System Settings” to open the System Properties window as shown in the next screenshot.



Click on “Environment Variables…”



Click on the “New” button and enter the NODE_OPTIONS information as follows – the screenshot sets 8GB as an example.



Optimizing outbreaks

Go.Data works with the dataset for a single outbreak at a time and so if you have an outbreak that is likely to grow to large amounts of data then you can consider setting this up as multiple outbreaks. A typical scenario is to create individual outbreaks for a country's regions rather than a single national outbreak. Such a setup does make it more difficult to bring data together for global reporting but will help performance.

Optimizing team set-up

In order for Go.Data web-app and mobile app to work well in a complementary fashion, and in a way that enables performance supervision of field-based contact tracers, we advise strategically setting up your teams and supervision in such a way that cases/contacts are partitioned to smaller geographic areas, and certain staff are responsible for these smaller areas.

Designing your teams in this way enhances the performance of supervision, creates more streamlined communication for support and ensures that mobile devices are not overloaded with too much data and only have the cases/contacts that are really needed for follow-up i.e. those that are within the limited geographical area in which a team has been assigned to work.

It is not advised, for example, that the a mobile-app be used at the central level with access to all cases/contacts in an outbreak. Even doing this at the division-level could introduce problems (potentially still very large number of cases/contacts for one device to handle).

In summary, the Go.Data mobile app is intended to be very tailored to a team that covers one area.

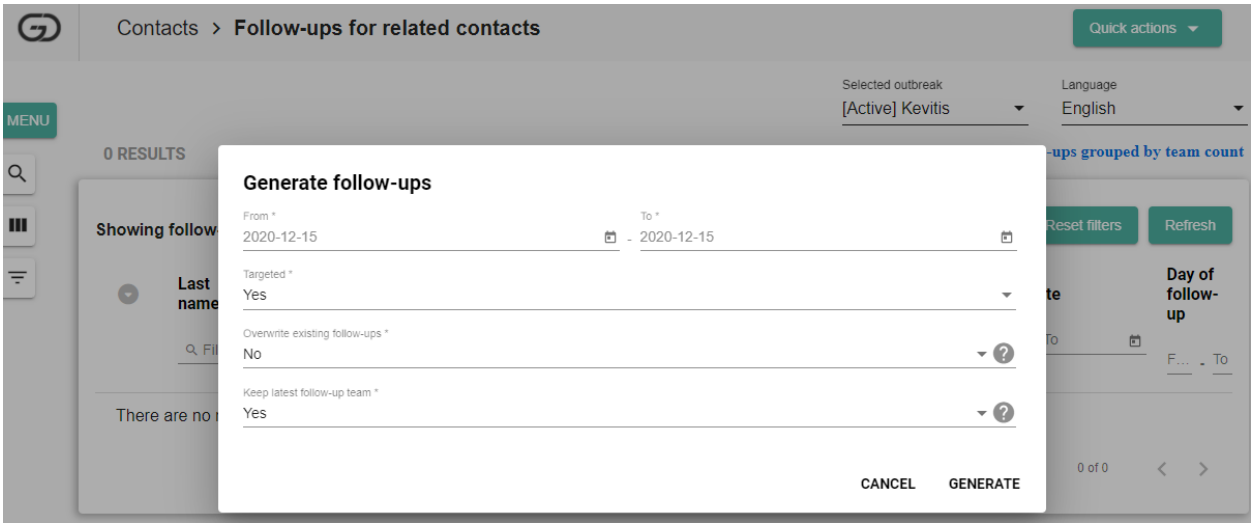
Limiting imported data

The main functionality of Go.Data is to enable case investigation and contact follow up. Although it can be used as a centralized database repository, utilizing it more as a database repository might take away from its ability to be a nimble field-based tool.

Administrators should always avoid importing data into the system that will not be used – for example creating the geographic structure of an entire country in the "Locations" section of Go.Data when, in fact, an outbreak occurs in just one province and only that province's location hierarchy is needed for operations.

Duplicate follow-up records

There is a small risk that the operation to "Generate Follow-Up" on the follow-up page can create duplicate records if two requests are raised by different users within about 1 second of each other.



If you experience this issue, and/or feel that there is a risk of multiple users requesting follow-up generation at the same time then one mitigation strategy is to reduce the batchSize in the *config.json* file from the default 10,000 to 500 as follows: -

“jobSettings": {
"generateFollowups": {
"batchSize": 500
}
}

Go.Data Security

Security

Go.Data has participated in three security audits before launch, two run by WHO and one by a member state. Both automated and manual (ethical hacking) tests were conducted.

The following is a list of details related to the security of Go.Data: -

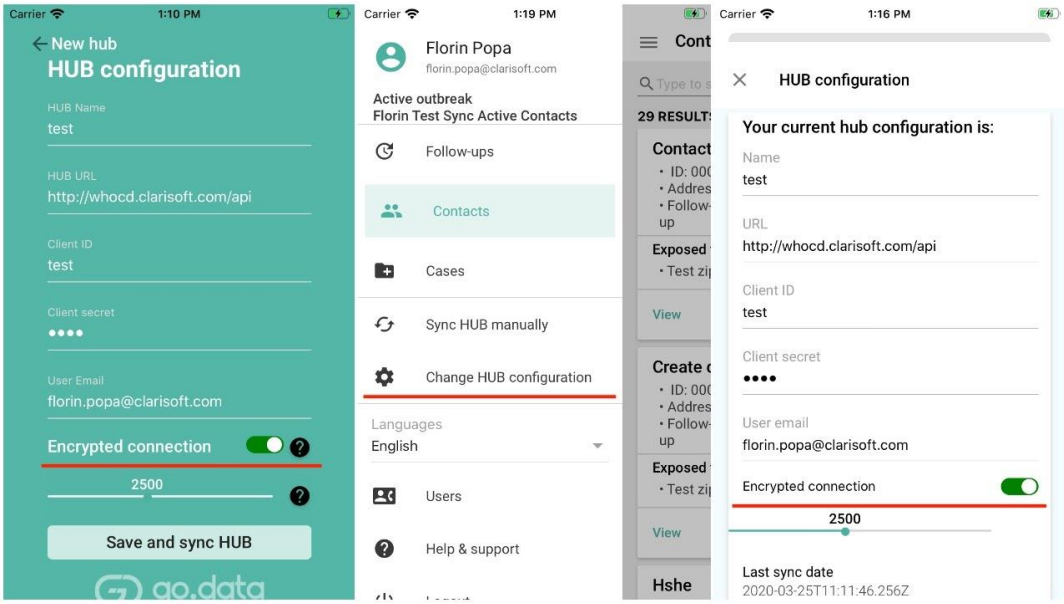
- Sensitive data like passwords is encrypted.
- Captcha can be optionally enabled on password-related screens.
- Two-factor authentication can be enabled on password-related screens. This proceeds via the dispatch of an email so a working SMTP server must be configured with Go.Data.
- By default, all authentication tokens have a time to live (TTL) of 10 minutes. You can reduce or increase this value by changing the *config.json* file. TTL of a token is taken in consideration from the last request made with this token. So, if you actively use the website then the token shouldn't expire during this time.

```
"authToken": {
  "ttl": 600 // seconds
}
```

- Only one authentication token per user account can be active at a time (if you sign in, previous tokens associated with that account are disabled). So, in case you use a third-party app to connect to the Go.Data API, it is recommended to login periodically to invalidate previous token. This way you reduce the chance for a valid token to be stolen.
- All backups can be encrypted by specifying the encryption key in *config.json* file:

```
"backUp": {
  "password": "key"
}
```

- Communication between mobile apps and API can be encrypted as well: Users have the option to exchange encrypted information between the app and the API. This option is present on the hub configuration screen as the "Encrypted connection" switch. By default, the encrypted connection is turned on. This option can later be turned off from the hub configuration screen.



- The MongoDB is not encrypted so should always be placed in a secure location ensuring that the port used by the database is only accessible to Go.Data. **See the next section on Ports.**
- Go.Data does not include its own web server, if you need to install a HTTPS security layer then you should place the tool behind a suitable reverse proxy (or load balancer).

Ports

Go.Data uses different ports for its web/API interface and for communication with MongoDB. If you experience connection issues, particularly from mobile phones connecting to Go.Data then you can validate firewall rules on the web port which may be preventing traffic.

By default, Go.Data will be available on Port 8000 or 3000

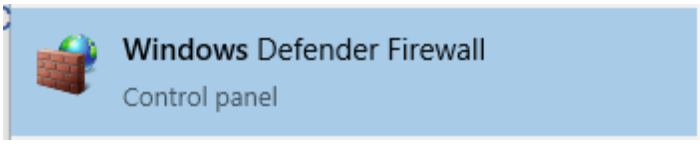
By default, MongoDB will be available on Port 27000

These ports appear when installing Go.Data and can be changed.

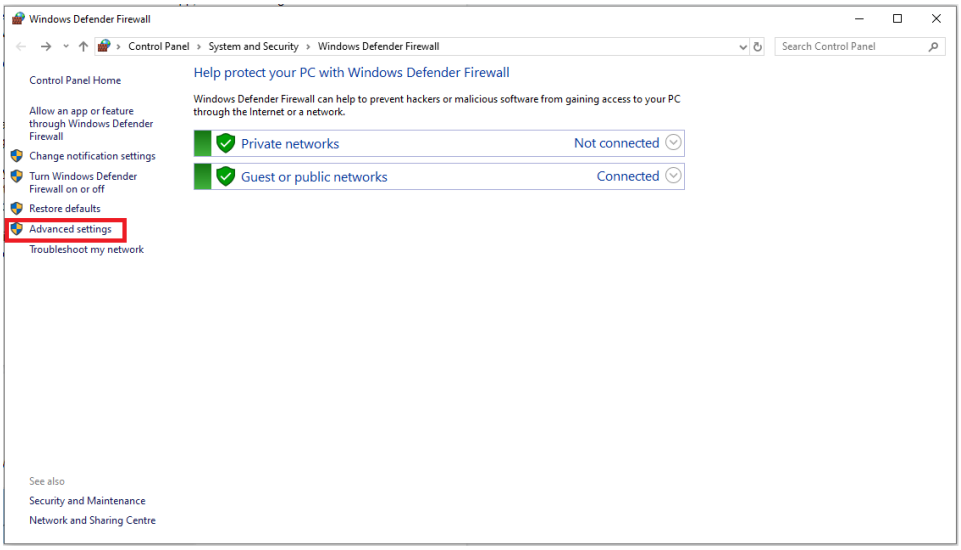
1. To be able to synchronise a Go.Data smartphone to the server copy – the smartphone must be able to access the Go.Data API endpoint on the configured port. Therefore, phone and the machine on which the Go.Data server are installed, must both be on the same Wi-Fi network or Go.Data must be exposed on a publicly-available URL via a reverse proxy. A first test is always to ping the Go.Data IP/address from a smartphone and see if the machine is visible to the phone.

2. If the issue still persists and Go.Data is installed on Windows, you may need to add an exception to the Windows Firewall. This can be done as follows:

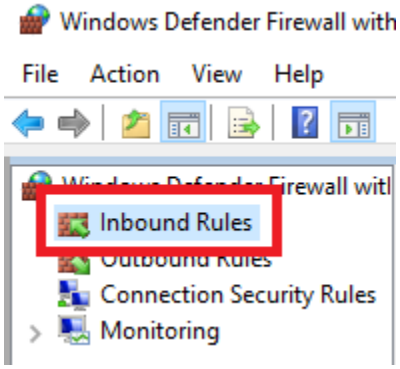
2.1 Search for the Windows Defender Firewall in Control Panel



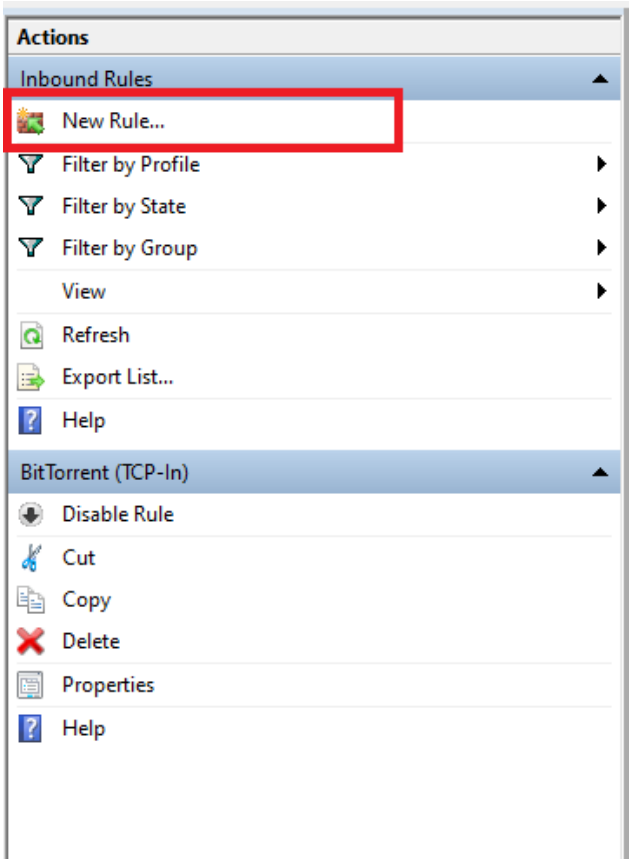
2.2 Go to Advanced settings



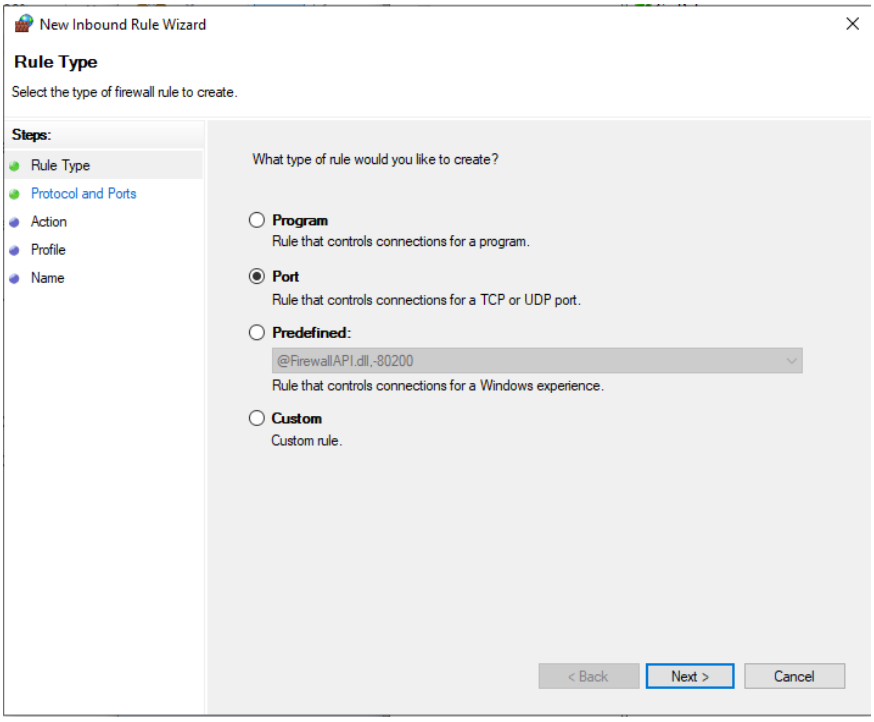
2.3 Select “Inbound rules”



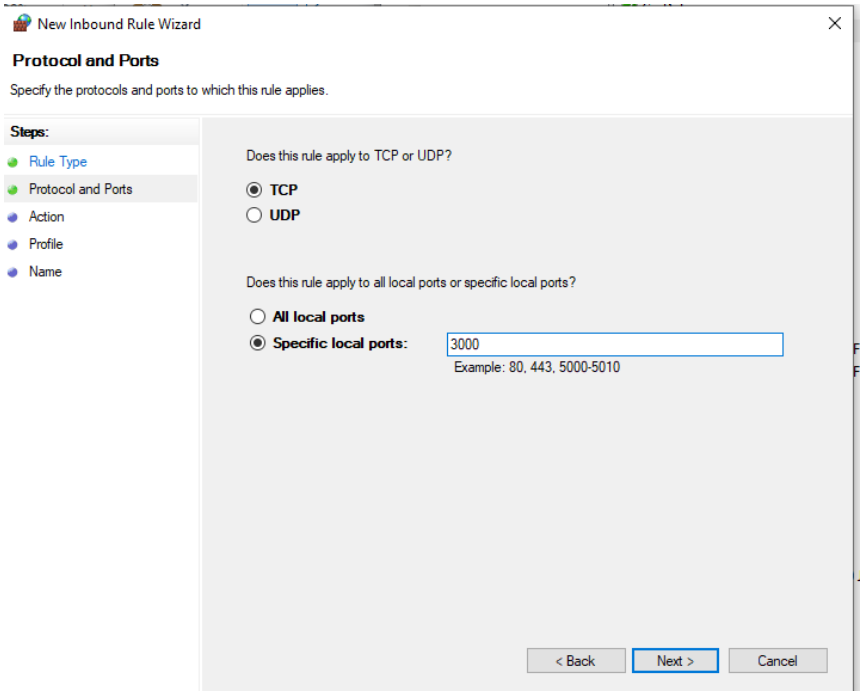
2.4 In the right panel click on “New Rule”



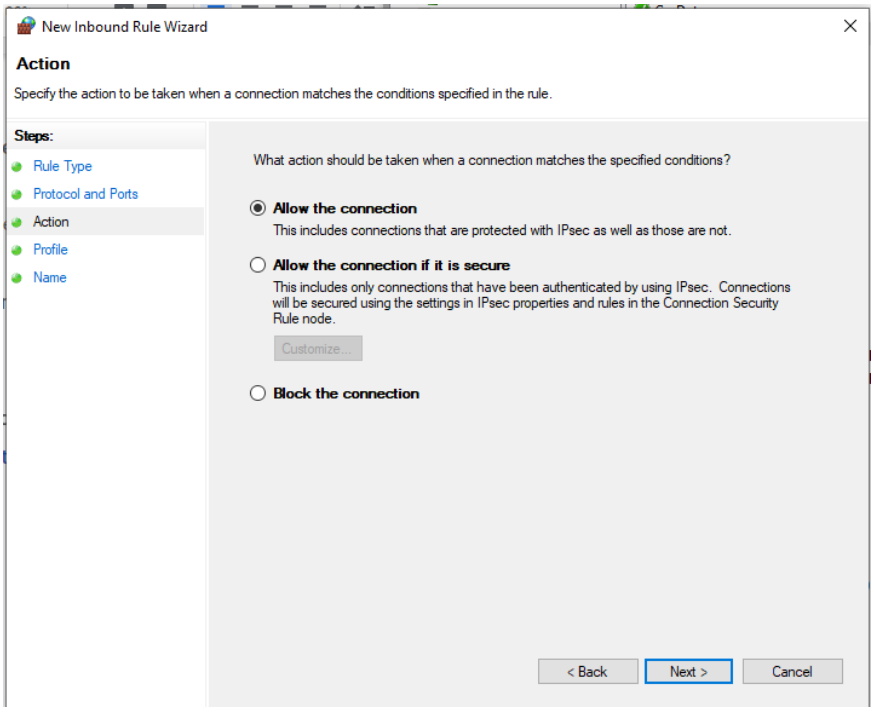
2.5 Select “Port”



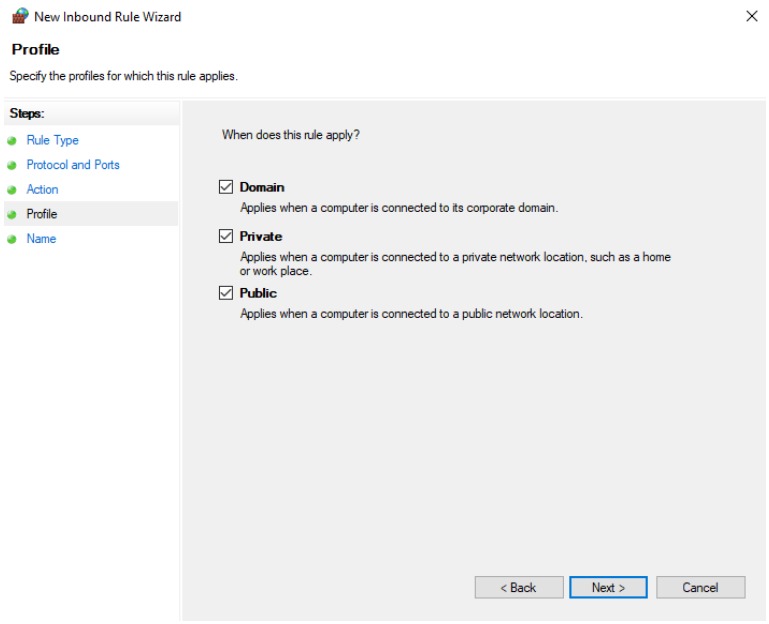
2.6 Select “TCP”, and add the port number (default 3000)



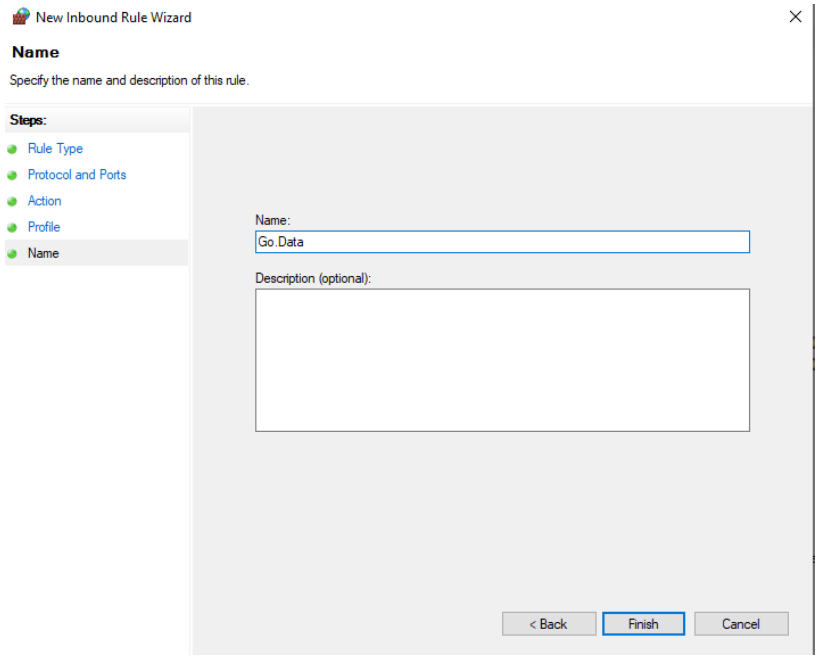
2.7 Select “Allow the connection”



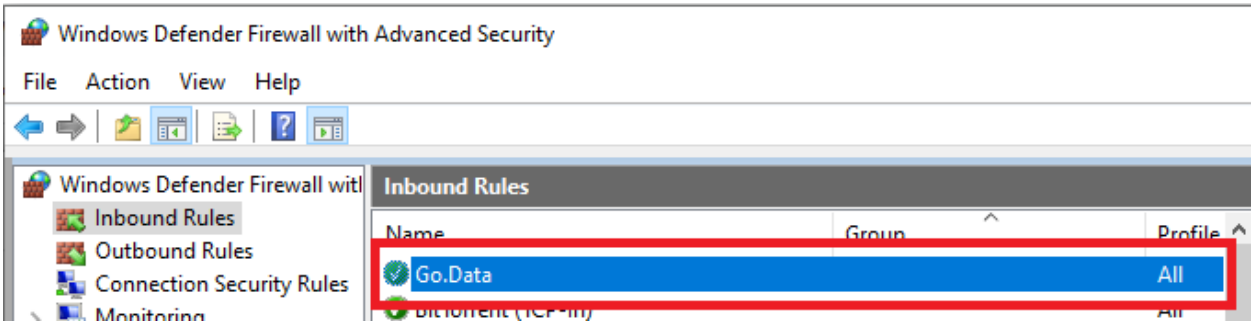
2.8 Select when the rule should apply



2.9 Give the new rule a name



2.10 Click “Finish”
2.11 Verify the new inbound rule



NOTE THAT: If you expose Go.Data to the public internet (behind a reverse proxy or directly on its own server) you should double-check that all ports are closed except the ones needed for clients to contact the Go.Data website, typically Port 443 for HTTPS communication. You should NEVER expose Port 27000 for the MongoDB to the outside world as hackers may detect that there is an "open" database and compromise your system.

For example, Amazon Web Services' standard Linux machine accept traffic on all ports by default and so these must be closed.

In future, most likely from Version 37 of Go.Data, the MongoDB port will probably change from 27000 for extra security.

HTTPS

For public-facing installations, you cannot, at present, enable https from the Go.Data application itself. This may be addressed in future versions of the tool.

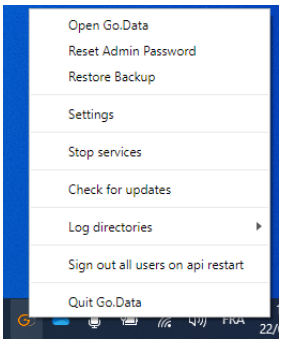
The URL present in the "public" properties in the ***config.json*** are used only by the reset password and the electron application. So consequently, changing the protocol property isn't enough. Users will need to have a proxy server in front of the Go.Data Service.

This can be achieved by configuring an Apache, Nginx, IIS server, load balancer etc. with an SSL certificate, which is visible from outside of the private network (a proxy). This server will route data to the private Go.Data server (which is accessible only in the private network). This setup can be done on the same computer too as long as everything is configured properly.

Until Go.Data will support applying SSL certificates directly, this is the only viable workaround to secure access.

Reset administrator password

Windows reset



To reset the administrator password on MS Windows, the administrator must have access to the Go.Data server itself and can then right-click on the Go.Data icon which appears in the system tray.

A list of options will be shown which includes, among other essential operations, the ability to “Reset Admin Password”.

This option is intended as a failsafe to gain control of a Go.Data instance if all other login options have been lost and a password reset email cannot be requested.

Linux reset

For the reset of the administrator password on Linux, the administrator must proceed as follows:

1. Login to Linux
2. Spawn a terminal console shell
3. Change directory to Go.Data installation directory
4. Execute the command <installdir>\go-data-reset-password-x64.sh --file <backupfile>
5. Once complete, this will reset the password to the default password. Logging in again, will prompt the user to modify it.

Go.Data API

Go.Data exposes an Application Programming Interface (API) which is used for all interactions between the web front-end, the smartphone applications and even between copies of Go.Data, if you configure multiple instances of the solution to exchange data in an “upstream sever/client application” model.

This means that the API is very flexible and any operation possible from the web interface/smartphone can also be made by calling the appropriate method direct.

Accessing

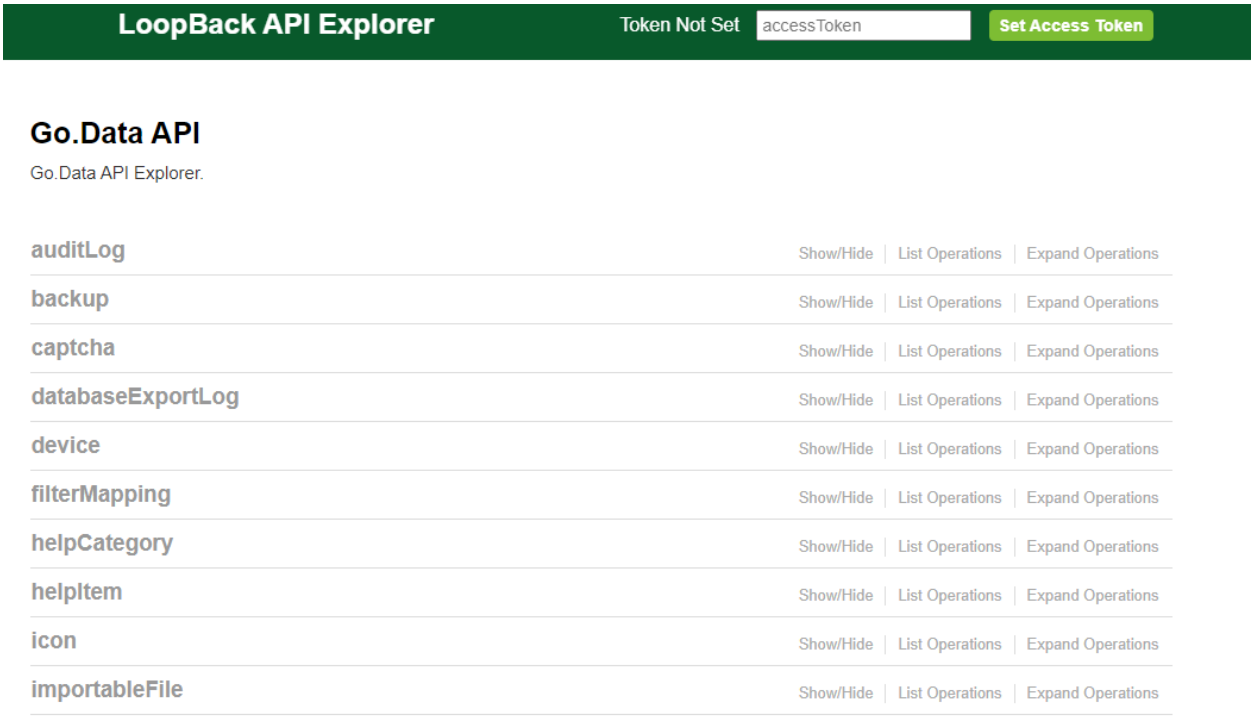
As stated earlier in the document, the self-documenting description of the API methods can be viewed using Loopback Explorer by adding /explorer to the end of any Go.Data URL. For example, when installing Go.Data on your local machine with default settings: -

http://localhost:8000/explorer

Loopback explorer provides some examples of possible operations, lists interface parameters and allows you to manually input json inputs and to test them against the API.

Methods are present to GET, POST, PASTE etc. for bidirectional exchange of data with Go.Data.

A familiarity with the types of data entities present within Go.Data is essential for understanding the API.



Authenticating

Almost all methods exposed by Go.Data require the request to be called by an authenticated user. Authentication works as follows: -

1) A call must be made to POST method /users/login passing in an email and password for a valid user within Go.Data. An example is given below: -

```
{
    "email": "email@test.com",
    "password": "your_password_here"
}
```

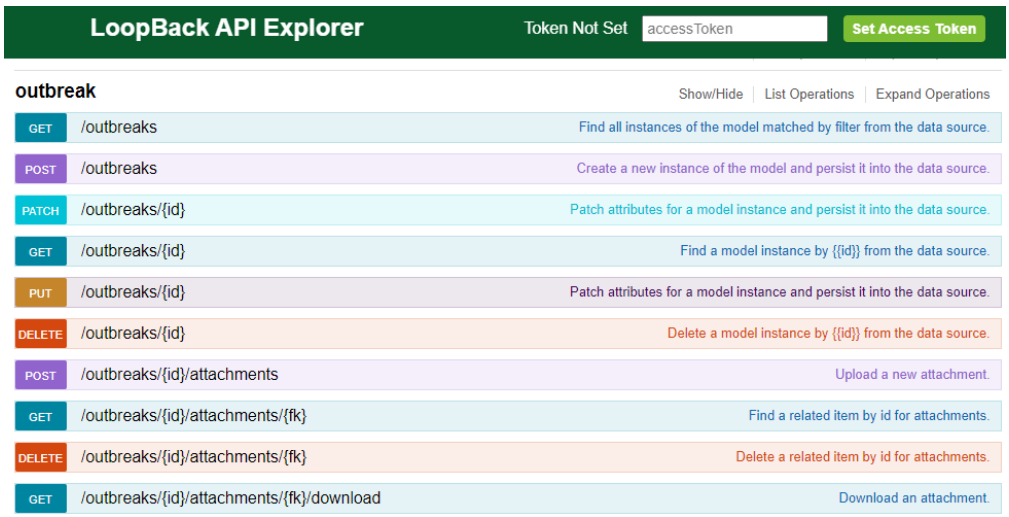
2) If the user details passed are accepted, then this method will return the following json response which contains an “id” property which is the authentication token to be used by this user for subsequent calls.


```
{
  "id": "Iv0S7Tj1F8RJaQSpBPHUlQ518KOUmK3SC7FlDvqLgbfbtjBt6ZFB77ViEQnKtqRq",
  "ttl": 600,
  "created": "2020-06-22T13:20:49.549Z",
  "userId": "18bf64ac-a36e-4890-a074-64aec702e21b",
  "createdAt": "2020-06-22T13:20:49.550Z",
  "createdBy": "system",
  "updatedAt": "2020-06-22T13:20:49.550Z",
  "updatedBy": "system",
  "deleted": false
}
```


3) The token received must be passed in the header of following calls. Within the loopback explorer interface, there is the option to enter this token at the top right and request that it is stored for testing further calls.

4) Note that all users in Go.Data have a single “active” outbreak, and this will be the one whose data is returned in subsequent calls using the authentication token received for the user. If you need to work across multiple outbreaks in your code, then you will either need to change users OR switch the active outbreak of the current user via an API call.

5) For this same reason, following call to work with outbreak data usually involve methods under the “outbreak” API path as shown in the next screenshot.



6) Other tools such as SoapUI or even constructing a query and submitting directly as a URL are possible.
7) For example, to retrieve a list of all outbreaks in Go.Data using an authentication token already received calling the API of my local instance, the call would be constructed as follows: -

- `http://localhost:8000/api/outbreaks?access_token=VUCA57YkMIcF5R2M8vl6PuaNHoMU0Q3Mr4cmGEOH06CPblx6xflnAw0AfQaMYdZP`

Note that is the response returns 422 “Invalid captcha” if received when attempting the call to the login API method, then this is a known issue linked to Go.Data’s use of cookies and you cab workaround as follows:

1. Captcha is saved in a session variable associated with the cookie provided by the browser only for a specific web page. Make sure that you don’t call '**GET** /captcha/generate-svg' with forComponent=login. If you want to use the login directly from the API, avoid calling this method, since this is the method that forces the login method to restrict access without entering captcha. If this is the case, restart the API, followed by calling login without calling this method first.

2. If you use API Explorer requests, in which case the browser attaches the cookie used by the session variable, with one usage example being: open login page, followed by accessing the explorer and try to login, then you will get the 422 “Invalid captcha” error, then these are the steps to follow:

- close all websites page, close browser
- open browser
- open explorer directly without going through the login page (in this case the browser resets the cookie variable, so it should work without having to restart the API).

Other potential solutions:

- use browser with incognito mode

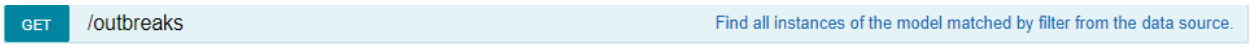
- use a different browser than the one used for Web UI
- close tab followed by clearing cookies cache related to this website

Working with data

Once authenticated, bidirectional exchange of data with Go.Data's API is then possible, and the lifetime of the authentication token is extended with each successful call.

Outbreak example

For example, to retrieve details of a single outbreak (the name of the outbreak, location etc.) a call can be made to the GET outbreaks method and would be constructed as follows by passing in both the access token (received from authentication) and the ID of the outbreak of interest.

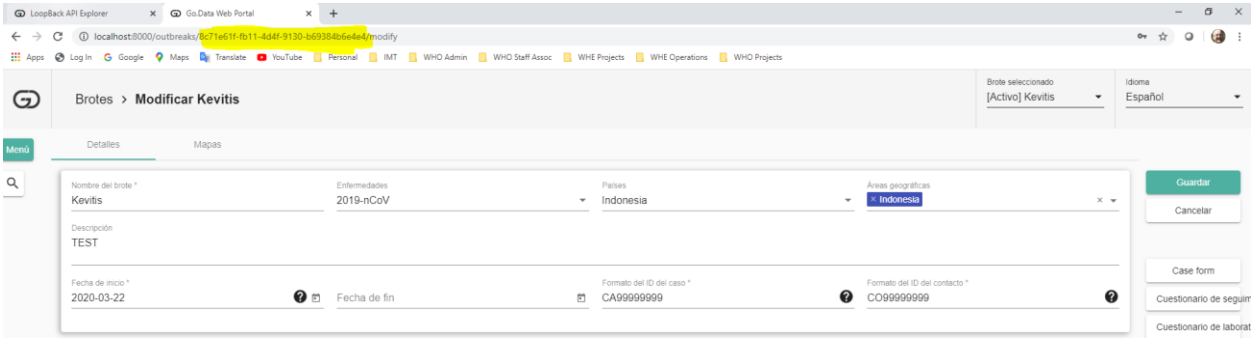



```
http://localhost:8000/api/outbreaks/8c71e61f-fb11-4d4f-9130-
b69384b6e4e4?access_token=VUCA57YkMIcF5R2M8vl6PuaNHoMU0Q3Mr4cmGEOH06CPblx6xfl
nAw0AfQaMYdZP
```

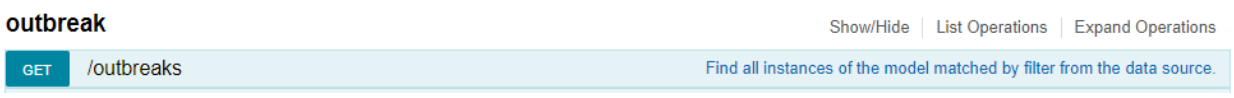
The IDs used in these calls are the internal Globally Unique Identifiers (GUIDs) used as primary keys for records in the MongoDB.

In the example above, there are two ways that a user could find the ID of the outbreak to use: -

1) If the user logins in Go.Data and navigates to a record, then the ID typically appears in the URL at the top of the page. An example is given below for viewing the example "Kevitis" outbreak and the ID that appears.



2) If a user needs to look up an ID, then it can always be done via a method also. In this example, the GET outbreaks method can be used to retrieve all outbreaks and then find the ID of the one in which you're interested.



Case example

As a second example, the call to retrieve all cases belonging to a given outbreak would be as follows: -


```
http://localhost:8000/api/outbreaks/8c71e61f-fb11-4d4f-9130-
b69384b6e4e4/cases?access_token=VUCA57YkMIcF5R2M8vl6PuaNHoMU0Q3Mr4cmGEOH06CPb
lx6xflnAw0AfQaMYdZP
```

Note that any security restrictions that limit the data which the user can see will also apply to calls via the API. Users should be careful to authenticate with a user account which has sufficient privilege and access for the data they wish to access/change.

The call to access the record for a specific case would then incorporate the access token, the outbreak ID and the ID of the case you wish to return: -


```
http://localhost:8000/api/outbreaks/8c71e61f-fb11-4d4f-9130-
b69384b6e4e4/cases/3cd71bf6-afac-40d3-a32d-
a1793cfe7638?access_token=HNm29JYiCIa0sNk5kjyTl8FeGKJmhFMiWAhGL6FOBVcBSCc2s2J
DQ3EnLH4dFt4l
```

For posting data back to Go.Data, the same json structures should be used but if there are any fields that are not to be changed, then those JSON properties should be omitted, not left blank.

For example, if we wished to update the first name for the Case that was retrieved in the previous call then a call would be made to the following PUT method: -

And the data passed in would be only the field to be changed: -

```
{
  "firstName": "TestDemo"
}
```

This constructs the following call: -

```
http://localhost:8000/api/outbreaks/8c71e61f-fb11-4d4f-9130-
b69384b6e4e4/cases/3cd71bf6-afac-40d3-a32d-
a1793cfe7638?access_token=HNm29JYiCIa0sNk5kjyTl8FeGKJmhFMiWAhGL6FOBVcBSCc2s2J
DQ3EnLH4dFt4l
```

Filter example

To use the filters provided with the method calls, the syntax is to use the keyword “where” and the sequence of elements for filtering: -

{"where":{"***fieldname***": "***filtervalue***"}}

So, for example, if filtering the method GET /outbreak/{id}/cases based on the Case ID,

Then the filter would be: -

{"where":{"visualId": "CA00000001"}}

This string will need URL encoding if passed as part of the URL. A full list of examples is given below.

JSON QUERY *{"where":{"createdAt":{"$gt":"2020-04-14T00:00:00Z"}}}*

URL ENCODED *%7B%22where%22%3A%7B%22createdAt%22%3A%7B%22%24gt%22%3A%222020-04-14T00%3A00%3A00Z%22%7D%7D%7D*

REQUEST COMMAND */outbreaks/* ***[[OUTBREAK TOKEN HERE]]*** */cases?filter=%7B%22where%22%3A%7B%22createdAt%22%3A%7B%22%24gt%22%3A%222020-04-14T00%3A00%3A00Z%22%7D%7D%7D&access_token=****[[ACCESS TOKEN HERE]]*

FINAL GET REQUEST *https://godata.gov.mt/api//outbreaks/* ***[[OUTBREAK TOKEN HERE]]*** */cases?filter=%7B%22where%22%3A%7B%22createdAt%22%3A%7B%22%24gt%22%3A%222020-04-14T00%3A00%3A00Z%22%7D%7D%7D&access_token=****[[ACCESS TOKEN HERE]]*

Sample code

To date, Go.Data partners have written integration code using C#, Python and R and the Go.Data community (https://community-godata.who.int) is a good place to engage with this community.

A repository with all sample code, further instructions on API usage and helpful libraries can be found on the WHO GitHub at: -

https://github.com/WorldHealthOrganization/godata

Appendix A: Security Audit

Introduction

As part of its development cycle, the Go.Data software has undergone two security audits administered by the WHO CyberSecurity team. These audits reviewed the tool against a standard list of common issues and also included an ethical hacking penetration test to validate its vulnerability.

The first security audit was performed against Go.Data in January 2019 against V34 and a second audit was performed one year later in January 2020 against the V36 release, to check that the issues identified from the first audit had been resolved correctly.

Summary results from the tests are included in this Appendix for information for IT administrators who wish to reassure themselves that the tool has been through a rigorous process of hardening and is not a cybersecurity risk when deployed to environments under their control.

Cybersecurity tests concentrated on the Go.Data "core" software and database (the web interface) and the API. Less attention was paid to the mobile application since this interacts with Go.Data via the same API.

In addition to the two cybersecurity tests performed by WHO, further tests have been run by other organisations which have deployed Go.Data to comply with their own processes and standards. The results of those tests are not available to WHO but it is worth noting that as of the time of writing (September 2021), the tool has been through multiple audits.

Cybersecurity Results – Jan 2019

The first cybersecurity scan performed against Go.Data noted a number of environment issues specific to the WHO machine on which the software was stored and the following issues which were addressed in the tool itself.

1) It was a strong recommendation that **two-factor authentication** be added for extra security and this was implemented as an optional feature that can be enabled in the *config.json* file.
2) It was a strong recommendation that there be a **cool down period** after a number of unsuccessful login attempts and this was implemented as an optional feature that can be enabled in the *config.json* file.
3) **JQuery V1.8.3** was being used which contained vulnerabilities and work has been ongoing to keep the software components used within Go.Data up to date.

All other issues detected at this time related to the environment in which Go.Data was running and not the tool itself.

Cybersecurity Results – Jan 2020

The cybersecurity scan performed in January 2020 was executed against a copy of V36 Go.Data hosted on a Virtual Machine controlled by WHO. This virtual machine was exposed via a reverse proxy to the public internet at address:

https//godata-r4.who.int

The cybersecurity scan examined both the web configuration of the deployment and the tool for vulnerabilities and particular attention was paid to ensure that the recommendations from January 2019 were implemented.

The details of the scan can be seen in Figure 1.

Scan of godata-r4.who.int

Scan details

Scan information	
Start time	29/01/2020, 10:39:30
Start url	https://godata-r4.who.int/
Host	godata-r4.who.int
Scan time	181 minutes, 36 seconds
Profile	Full Scan
Server information	Apache/2.4.37 (Unix) OpenSSL/1.0.2k-fips mod_perl/2.0.10 Perl/v5.16.3
Responsive	True
Server OS	Unix
Server technologies	Perl,Perl

Threat level

Acunetix Threat Level 3

One or more high-severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

Alerts distribution

Total alerts found	5
High	1
Medium	1
Low	0
Informational	3

Figure 1: Go.Data Cybersecurity Details – January 2020

The results of the scan can be seen in Figure 2. Since the scan involved details of WHO's environment, a full report cannot be listed here, only the summary.

Executive summary

Alert group	Severity	Alert count
CORS (Cross-Origin Resource Sharing) origin validation failure	High	1
TLS 1.0 enabled	Medium	1

Figure 2: Go.Data Cybersecurity Results – January 2020

The issues identified were: -

CORS – Cross Origin Resource Sharing had been left enabled at the time of the scan and was correctly identified as a vulnerability. As noted in this document in the section on Configuring CORS, this can be disabled by making a change in the *config.json* file for Go.Data which will resolve this issue by blocking cross origin requests.

TLS 1.0 Enabled – TLS Version 1.0 was enabled on the WHO web server on which Go.Data was installed, this was an environment vulnerability and not an issue with the Go.Data software. TLS 1.0 has since been disabled in WHO environments to resolve this issue.

Although it might block older browsers from using Go.Data, it is recommended that earlier versions of TLS are disabled on the server on which Go.Data runs and TLS 1.2 or higher is used.

Summary

At the time of writing, Go.Data is considered sufficiently hardened for common cybersecurity attacks provided that the configurable features such as CORS, multi-factor-authentication and token timeouts are set correctly.

The environment in which Go.Data is deployed must also be validated to ensure that there are not vulnerabilities.

It is recommended that each new deployment of Go.Data includes cybersecurity testing therefore.