ap-nimbus Documentation

Release 1.0.1

Geoff Williams

Aug 07, 2023

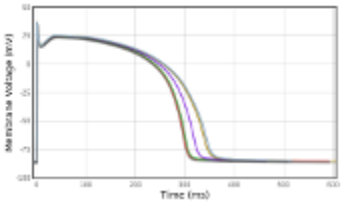
Contents

Project Home https://github.com/CardiacModelling/ap-nimbus

Documentation https://ap-nimbus.readthedocs.io/

Created Aug 07, 2023

Version 1.0.1



CHAPTER 1

IMPORTANT

This activity represents the next step in the development of the original AP-Portal work – towards a container-based / cloud solution.

`AP-Nimbus` is available at https://github.com/CardiacModelling/ap-nimbus and is being developed by the University of Nottingham's School of Mathematical Sciences .

CHAPTER 2

Preamble

- Unlike AP-Portal, this work includes the installation of `ApPredict`, the cardiac simulation software.
- Also unlike `AP-Portal`, this work, by nature of containerisation, means that `AP-Nimbus` work does not embody a single application, it is instead a collection of containers where each can operate in isolation, e.g. as a standalone `Docker` or `Singularity` container, or alternatively, orchestrated in a microservice architecture (e.g. `Kubernetes` or `Docker Compose`).

CHAPTER 3

Singularity

This documentation predominantly covers `Docker` container environments, however it has been relatively straightforward to create Singularity containers (e.g. `singularity build app-manager.img docker://cardiacmodelling/ap-nimbus-app-manager:0.0.10`) and use those[1].

Sample invocation scripts can be found at ap-predict-online's app-manager –> tools section.

[1] `Singularity` containers (or rather, a `Singularity` version of just `cardiacmodelling/ap-nimbus-app-manager` so far) have been trialled operating in isolation, not in an orchestrated environment.

CHAPTER 4

Diagrammatic Representation

For the role each of the containers has in the overall `AP-Nimbus` activity please see *Activity Overview*.

4.1 ApPredict containers

ApPredict is the underlying cardiac simulation engine.

Building or installing `ApPredict` is a complex and time-consuming process and by distributing in container form it's possible to have it installed in a fraction of the time[2].

4.2 `AP-Nimbus` containers

The following illustrates a Docker container setup.

It is equally feasible to run as .. :

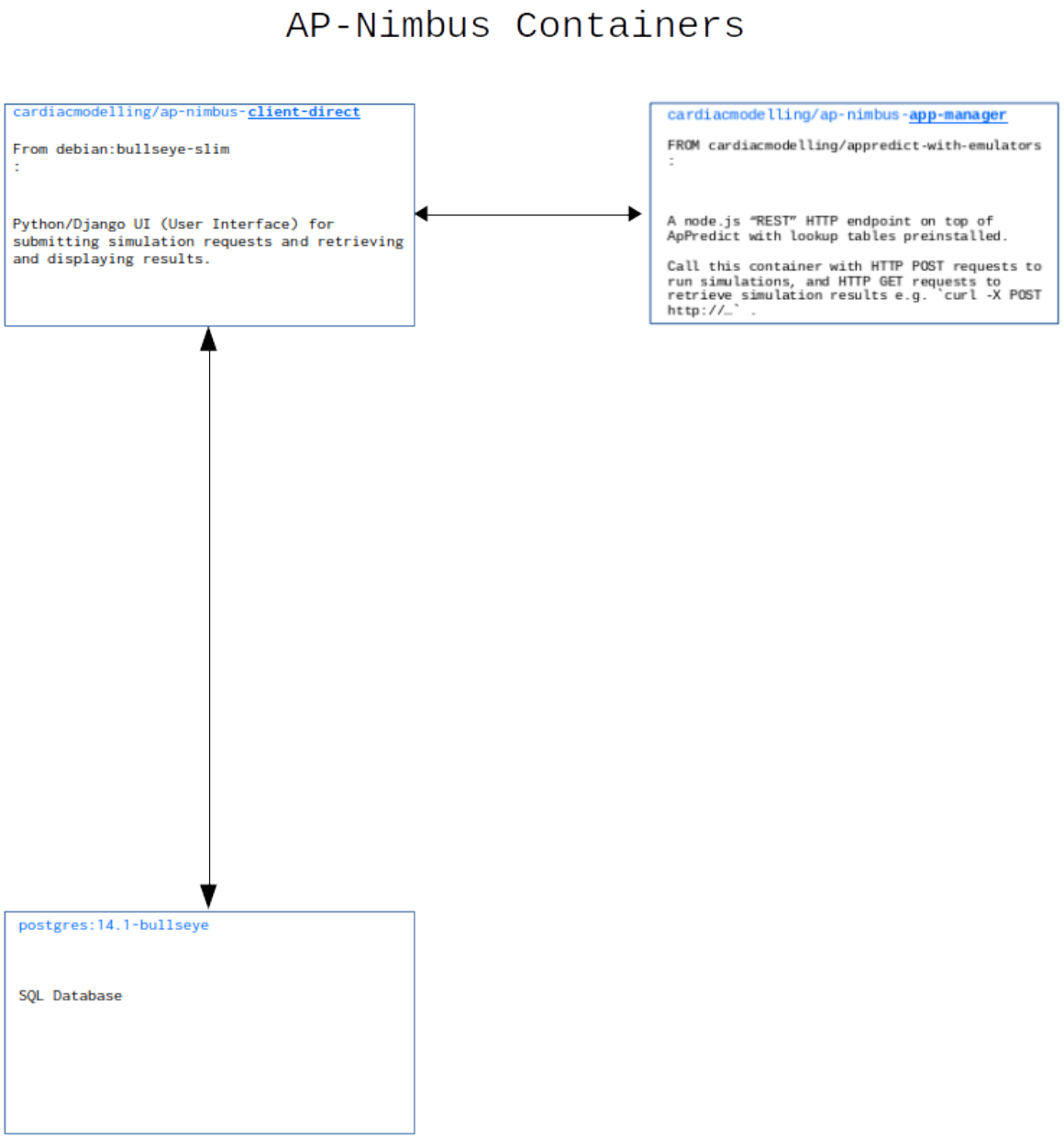
- Containerised
  - `docker run ..` a single `cardiacmodelling/ap-nimbus-app-manager` container and call it with HTTP (Hypertext Transfer Protocol) `POST` and `GET` requests, or;
  - `docker run ..` either of the `cardiacmodelling/appredict-with-emulators` or `cardiacmodelling/appredict-no-emulators` containers directly from a CLI (Command-line Interface) to run their internal `ApPredict`s, or;
- Non-containerised
  - Running the various components without using containers is technically possible, but not reccomended or supported. See *Developer Section* for suggestions on how to develop the components using containers.

See also:

For instructions on how to run containers, see the more detailed section on *Running*.

[2] So long as there's a container runtime, e.g. `Docker`, running somewhere!


ApPredict in Containers
cardiacmodelling/ap-nimbus-app-manager
FROM cardiacmodelling/appredict-with-emulators
:
A node.js “REST” HTTP endpoint on top of ApPredict with lookup tables preinstalled.
Call this container with HTTP POST requests to run simulations, and HTTP GET requests to retrieve simulation results e.g. `curl -X POST http://…` .
“REST” API to ApPredict
cardiacmodelling/appredict-with-emulators
FROM cardiacmodelling/appredict-no-emulators
:
ApPredict with lookup tables preinstalled.
Run simulations from the command-line, e.g. `docker run -it --rm ...` .
ApPredict with lookup tables installed
cardiacmodelling/appredict-no-emulators
FROM cardiacmodelling/appredict-chaste-libs
:
ApPredict only.
Run simulations from the command-line, e.g. `docker run -it --rm ...` .
ApPredict only
cardiacmodelling/appredict-chaste-libs
FROM debian:buster-slim
:
ApPredict’s dependencies
Debian
Base Operating System


AP-Nimbus Containers
cardiacmodelling/ap-nimbus-client-direct
From debian:bullseye-slim
:
Python/Django UI (User Interface) for submitting simulation requests and retrieving and displaying results.
cardiacmodelling/ap-nimbus-app-manager
FROM cardiacmodelling/appredict-with-emulators
:
A node.js "REST" HTTP endpoint on top of ApPredict with lookup tables preinstalled.
Call this container with HTTP POST requests to run simulations, and HTTP GET requests to retrieve simulation results e.g. `curl -X POST http://…` .
postgres:14.1-bullseye
SQL Database

CHAPTER 5

Activity Overview

5.1 Activity Overview

5.1.1 Current work

The information below relates to the current `AP-Nimbus` work being done in the project's CardiacModelling GitHub repository .

app-manager

`app-manager` is a Node .js (v18.12.1) "REST (Representational State Transfer)" (i.e. it's not fully REST) API (Application Programming Interface) to the `ApPredict` cardiac simulation engine.

- *Install cardiacmodelling/ap-nimbus-app-manager*
- *Running cardiacmodelling/ap-nimbus-app-manager*

`ApPredict` containers

Note: Docker containers `cardiacmodelling/appredict-chaste-libs`, `cardiacmodelling/appredict-no-emulators` and `cardiacmodelling/appredict-with-emulators` are derived from source code in https://github.com/CardiacModelling/appredict-docker (or via the `appredict-docker` submodule of https://github.com/CardiacModelling/ap-nimbus).

See also:

ApPredict containers for visual representation.

appredict-chaste-libs

`cardiacmodelling/appredict-chaste-libs` is a container which contains all of `ApPredict`'s library dependencies.

- Conventionally `cardiacmodelling/appredict-chaste-libs` isn't "installed" as it's just a Debian Docker image with all of `ApPredict`'s dependency packages installed, and as such just the foundation for `cardiacmodelling/appredict-no-emulators` to build on.
- Historically (before `cardiacmodelling/appredict-chaste-libs:0.0.5`) this container's contents were mostly built from source packages of dependent libraries.
- There's no "running" of `cardiacmodelling/appredict-chaste-libs`.

appredict-no-emulators

`cardiacmodelling/appredict-no-emulators` is a container which has `cardiacmodelling/appredict-chaste-libs` as its foundation, and it is the minimum necessary to run `ApPredict` simulations via CLI.

- *Install cardiacmodelling/appredict-no-emulators or cardiacmodelling/appredict-with-emulators*
- *Running cardiacmodelling/appredict-no-emulators or cardiacmodelling/appredict-with-emulators*

appredict-with-emulators

`cardiacmodelling/appredict-with-emulators` is a container which has `cardiacmodelling/appredict-no-emulators` as its foundation, with all the Lookup Tables (https://cardiac.nottingham.ac.uk/lookup_tables/appredict_lookup_table_manifest.txt) available at build time pre-installed to avoid the need to dynamically download them.

- *Install cardiacmodelling/appredict-no-emulators or cardiacmodelling/appredict-with-emulators*
- *Running cardiacmodelling/appredict-no-emulators or cardiacmodelling/appredict-with-emulators*

client-direct

`client-direct` is a Django 4 component of `AP-Nimbus`.

- The client is available at https://github.com/CardiacModelling/ap-nimbus-client.
- Conventionally `client-direct` isn't installed/run as stand-alone as it generally serves no purpose to install only the UI (User Interface). For installing details see *Install cardiacmodelling/ap-nimbus-client-direct* Developers may be interested in further reading in the developer section of *Developer client-direct section*.

5.1.2 Discontinued work

The following projects in are either discontinued or not currently being developed.

ap-nimbus-azure-durable-functions

This was a test activity to determine the viability of using Azure Durable Functions as a solution to enabling high-concurrency cardiac simulations.

Whereas with the orchestrated container option, e.g. `Kubernetes`, which managed the legacy `client-direct` (superceded by `cardiacmodelling/ap-nimbus-client-direct`), `cardiacmodelling/ap-nimbus-app-manager` and legacy "ap-nimbus-datastore" container deployment and intercommunication on an "always-on" basis, the Azure Durable Functions are supposed to rapidly spin up and down containers "on demand".

For our purposes therefore we were only investigating the ability of Azure Durable Functions to rapidly scale up and down `cardiacmodelling/ap-nimbus-app-manager`. The expectation being that by using the Async HTTP API technique we would have an "always on" HTTP endpoint that listened to simulation requests (e.g. as per simulationInput.json) and would spin up the corresponding number of `cardiacmodelling/ap-nimbus-app-manager`s and return the generated results.

- Azure Durable Functions is available in the ap-nimbus git repository via the ap-nimbus-azure-durable-functions submodule

Developers may be interested to read about some of the issues faced :

- *Azure Durable Functions*.

ap-nimbus-datastore

The "datastore" component, in the form of a Mongo Db, was used up to 2021. It has been replaced by a Postgres container.

ap-nimbus-deploy

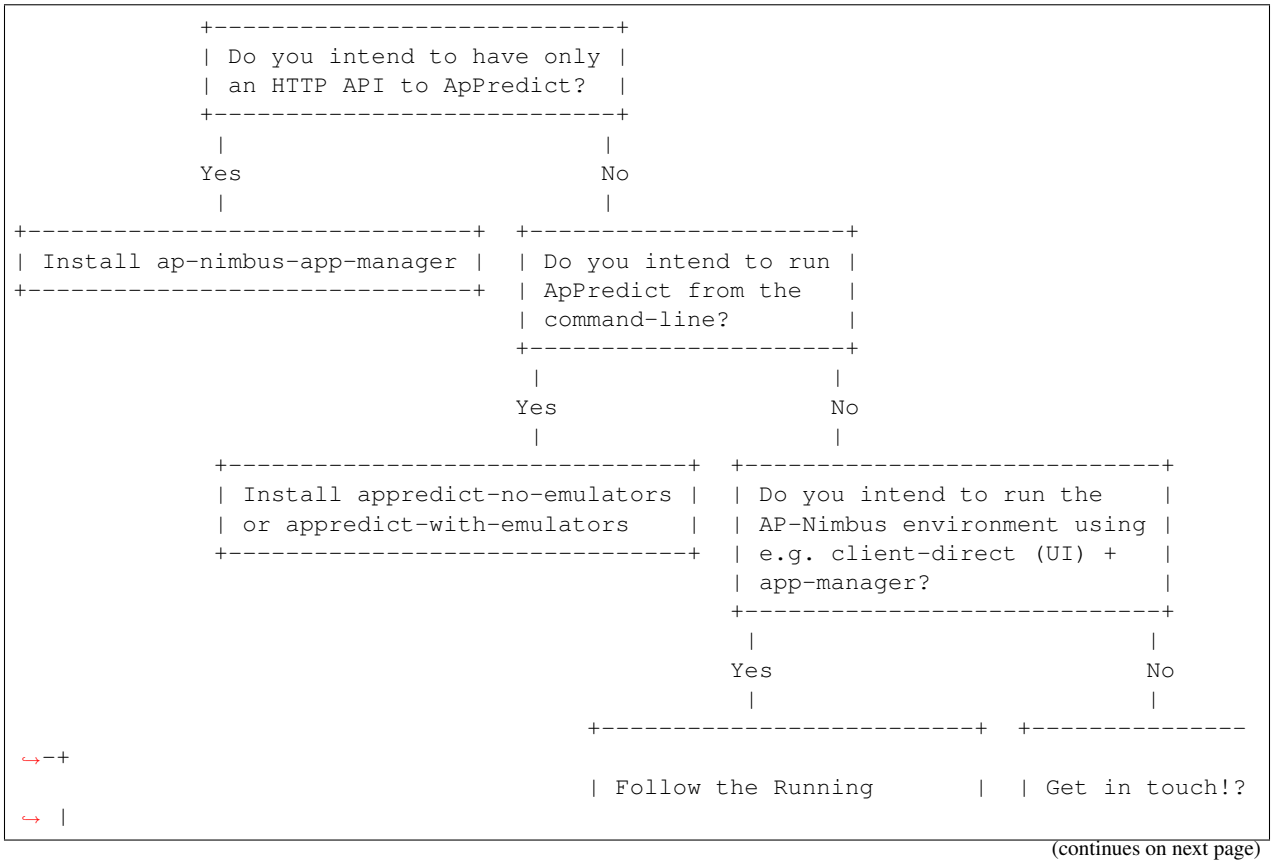
This activity ceased in 2021. Instead containers are run independently.

CHAPTER 6

Installation

6.1 Installation

6.1.1 Deployment Options



(continues on next page)

(continued from previous page)

```
                                        | instructions linked to   |  +---------------
↪-+
                                        | below                    |
                                        +--------------------------+
```

Install

See also:

For instructions on how to run these containers, e.g. in the context of `AP-Nimbus` environment, see the more detailed section on *Running*.

Containers generally, when attempting to run them for the first time, will be installed locally by being auto-downloaded from DockerHub and deployed to the local image collection by the container runtime.

Install `cardiacmodelling/ap-nimbus-app-manager`

```
docker run -it --rm -p 8080:8080 cardiacmodelling/ap-nimbus-app-manager:<version>
```

(See ap-nimbus-app-manager tags for available version numbers.) This command will automatically download the container from https://hub.docker.com/u/cardiacmodelling if it is not already available in the local `Docker` image collection, and by default will listen on `http://0.0.0.0:8080/` (`Cntl-C` to exit). For further instructions on running, including `POST`ing data to the running endpoint, see the section on *Running cardiacmodelling/ap-nimbus-app-manager*.

Install `cardiacmodelling/appredict-no-emulators` or `cardiacmodelling/appredict-with-emulators`

```
docker run -it --rm cardiacmodelling/appredict-no-emulators:<version> apps/
ApPredict/ApPredict.sh
```

(See appredict-no-emulators tags or appredict-with-emulators tags for available version numbers.) This command will automatically download the container from https://hub.docker.com/u/cardiacmodelling if it is not already available in the local `Docker` image collection, and by default will run `ApPredict` with no args (and therefore display the 'help' information). For further instructions on running see the section on *Running cardiacmodelling/appredict-no-emulators or cardiacmodelling/appredict-with-emulators*.

Install `cardiacmodelling/ap-nimbus-client-direct`

`client-direct` requires a number of configuration property values to be assigned (e.g. for database connectivity) before it will start, and therefore installation is best explained in the context of the provision of an operational `AP-Nimbus` environment.

For further instructions on running, see the section on *Running cardiacmodelling/ap-nimbus-client-direct*.

CHAPTER 7

Security

7.1 Security

`client-direct` makes use of built-in security measures of Django, in terms of accounts, logging-in and Cross site scripting, SQL injection, and Clickjacking protections.

It is important that `client-direct` is provided with a strong secret key that is kept private. This can be done using the *DJANGO_SECRET_KEY* environment variable see *Running cardiacmodelling/ap-nimbus-client-direct*.

Note: The docker component is not set-up to use https. Therefore it is strongly recommended that a webserver such as apache or nginx using https is used to proxy external requests to the `client-direct` component. Due to the lack of security features it is NOT recommended to expose the `app-manager` over the internet.

7.1.1 Shared Computers / Browser Memory

All simulation results are stored in browser memory until manually deleted, so can potentially be seen by all users of a shared computer.

CHAPTER 8

Running

8.1 Running

8.1.1 Running `cardiacmodelling/ap-nimbus-app-manager`

See also:

Offical `docker run` documentation.

You have two options for running `cardiacmodelling/ap-nimbus-app-manager`, either :

1. As a standalone container to call using a CLI, e.g. `curl` (see later *Start a Simulation* and *Query a Simulation* sections).

   `docker run -it --rm -p 8080:8080 cardiacmodelling/ap-nimbus-app-manager:<version>` (`Cntl-C` to exit)

2. As part of `AP-Nimbus` (see also *Running cardiacmodelling/ap-nimbus-client-direct*)

   ```
   docker run -d --name name-app-manager --net ap-nimbus-network --restart
   always cardiacmodelling/ap-nimbus-app-manager:<version>
   ```

   Note:

   - You could also define a volume here, e.g. `-v volume-app-manager:/home/appredict/apps/app-manager` if you wanted to see how simulations were being run and results stored ephemerally in the `run` and `res` folders at runtime.
   - The value specified in `--name` can be used as the hostname in the `client-direct` *Environment Variables* settings. **Please note**: DO NOT use underscores in these names as underscores in hostnames is not universally supported.

In Docker parlance, the `-p` will "publish" or "expose" the container port 8080 to port 8080 on the host machine by means of fiddling with the firewall (see *cardiacmodelling/ap-nimbus-app-manager* for a bit more information).

See also:

For supplementary developer information see *cardiacmodelling/ap-nimbus-app-manager*.

Start a Simulation

```
curl --header "expect: " --header "Content-Type:application/json" -X POST -d @request.
↪json http://<host>:8080/
```

In the response will be a simulation identifier for subsequent querying of simulation status and results.

Example `request.json` for `cardiacmodelling/ap-nimbus-app-manager:0.0.10+`

```
{
  "modelId":8,
  "credibleIntervalPctiles":[38,68,86,95],
  "pacingFrequency":1,
  "pacingMaxTime":5,
  "IKr":{
    "associatedData":[{"pIC50":"4.01","hill":"1.01","saturation":"0.01"}],
    "spreads":{"c50Spread":"0.18"}},
  "INa":{
    "associatedData":[{"pIC50":"4.02","hill":"1.02","saturation":"0.02"}],
    "spreads":{"c50Spread":"0.2"}},
  "ICaL":{
    "associatedData":[{"pIC50":"4.03","hill":"1.03","saturation":"0.03"}],
    "spreads":{"c50Spread":"0.15"}},
  "IKs":{
    "associatedData":[{"pIC50":"4.04","hill":"1.04","saturation":"0.04"}],
    "spreads":{"c50Spread":"0.17"}},
  "IK1":{
    "associatedData":[{"pIC50":"4.05","hill":"1.05","saturation":"0.05"}]},
  "Ito":{
    "associatedData":[{"pIC50":"4.06","hill":"1.06","saturation":"0.06"}]},
  "INaL":{
    "associatedData":[{"pIC50":"4.07","hill":"1.07","saturation":"0.07"}]},
  "plasmaMaximum":300,
  "plasmaMinimum":0,
  "plasmaIntermediatePointCount":10,
  "plasmaIntermediatePointLogScale":true
}
```

Example `request.json` for `cardiacmodelling/ap-nimbus-app-manager:0.0.6-0.0.9`

```
{
  "modelId":8,
  "pacingFrequency":1,
  "pacingMaxTime":5,
  "IKr":{
    "associatedData":[{"pIC50":"4.01","hill":"1.01","saturation":"0.01"}],
    "spreads":{"c50Spread":"0.18"}},
  "INa":{
    "associatedData":[{"pIC50":"4.02","hill":"1.02","saturation":"0.02"}],
```

(continues on next page)

(continued from previous page)

```
    "spreads":{"c50Spread":"0.2"}},
  "ICaL":{
    "associatedData":[{"pIC50":"4.03","hill":"1.03","saturation":"0.03"}],
    "spreads":{"c50Spread":"0.15"}},
  "IKs":{
    "associatedData":[{"pIC50":"4.04","hill":"1.04","saturation":"0.04"}],
    "spreads":{"c50Spread":"0.17"}},
  "IK1":{
    "associatedData":[{"pIC50":"4.05","hill":"1.05","saturation":"0.05"}]},
  "Ito":{
    "associatedData":[{"pIC50":"4.06","hill":"1.06","saturation":"0.06"}]},
  "INaL":{
     "associatedData":[{"pIC50":"4.07","hill":"1.07","saturation":"0.07"}]},
  "plasmaMaximum":300,
  "plasmaMinimum":0,
  "plasmaIntermediatePointCount":10,
  "plasmaIntermediatePointLogScale":true
}
```

Example `request.json` for `cardiacmodelling/ap-nimbus-app-manager`:0.0.5 and earlier

```
{
  "created":1576587107451,
  "modelId":3,
  "pacingFrequency":1,
  "pacingMaxTime":5,
  "pIC50IKr":"4",
  "pIC50INa":"",
  "pIC50ICaL":"",
  "pIC50IKs":"4",
  "pIC50IK1":"",
  "pIC50Ito":"",
  "plasmaMaximum":100,
  "plasmaMinimum":0,
  "plasmaIntermediatePointCount":4,
  "plasmaIntermediatePointLogScale":true
}
```

Query a Simulation

Generally, if you have a valid simulation identifier available from a prior `POST` request, you will see the query "help" by submitting :

```
user@host:~> curl http://0.0.0.0:8080/api/collection/8c0b04cc-6e2f-4307-ae17-
↪605cf81e0707/
{"error":"Valid data query options are: \"STOP\", \"voltage_traces\", \"voltage_
↪results\", \"progress_status\", \"q_net\" and \"messages\""}
```

To retrieve voltage results data :

```
curl http://<host>:8080/api/collection/<simulation_id>/voltage_results
```

8.1.2 Running `cardiacmodelling/appredict-no-emulators` or `cardiacmodelling/appredict-with-emulators`

See also:

Offical `docker run` documentation.

Without dynamic CellML (`--model <file>`) or PKPD (Pharmacokinetic-Pharmacodynamic) (`--pkpd-file <file>`)

```
user@host:~/tmp> mkdir testoutput
user@host:~/tmp> docker run --rm \
                            -u `id -u`:`id -g` \
                            -v `pwd`/testoutput:/home/appredict/apps/ApPredict/
↪testoutput:Z \
                            cardiacmodelling/appredict-no-emulators:<version> \
                            apps/ApPredict/ApPredict.sh --model 1 --plasma-conc-high␣
↪100
```

Warning:

- **Don't** be tempted to `Cntl-C` to finish early, otherwise you may be left with a `root`-owned directory to remove!
- For `cardiacmodelling/appredict-no-emulators:0.0.9` **only**, the last line should be `/bin/bash apps/ApPredict/ApPredict.sh <args>`

See also:

Sample ApPredict argument input

With dynamic CellML (`--model <file>`) or PKPD (`--pkpd-file <file>`)

For simplicity, the *annotated*[1] CellML file is called `this.cellml`, the PKPD file is called `this.pkpd`, and both are in the directory where the `docker run ...` command is being run.

```
user@host:~/tmp> mkdir testoutput
user@host:~/tmp> docker run --rm \
                            -u `id -u`:`id -g` \
                            -v `pwd`/testoutput:/home/appredict/apps/ApPredict/
↪testoutput:Z \
                            -v `pwd`/this.cellml:/home/appredict/apps/ApPredict/this.
↪cellml:ro \
                            -v `pwd`/this.pkpd:/home/appredict/apps/ApPredict/this.
↪pkpd:ro \
                            cardiacmodelling/appredict-no-emulators:<version> \
                            apps/ApPredict/ApPredict.sh --model apps/ApPredict/this.
↪cellml --pkpd-file apps/ApPredict/this.pkpd
```

[1] "Annotated" implies metadata tags for voltage, time, ionic current conductances should be added using the Cardiac Electrophysiology Web Lab tool, or pre-annotated models can be downloaded from GitHub Chaste/cellml.

Warning:

- **Don't** be tempted to `Cntl-C` to finish early, otherwise you may be left with a `root`-owned directory to remove!
- For `cardiacmodelling/appredict-no-emulators:0.0.9` **only**, the last line should be `/bin/bash apps/ApPredict/ApPredict.sh <args>`

See also:

Sample ApPredict argument input

Sample `ApPredict` argument input

The following is derived from `cardiacmodelling/appredict-no-emulators:0.0.9` (i.e. around December 2022)

```
*******************************************************************************************
* ApPredict::Please provide some of these inputs:
*
* EITHER --model
*   options: 1 = Shannon, 2 = TenTusscher (06), 3 = Mahajan,
*            4 = Hund-Rudy, 5 = Grandi, 6 = O'Hara-Rudy 2011 (endo),
*            7 = Paci (ventricular), 8 = O'Hara-Rudy CiPA v1 2017 (endo)
* OR --model <name of pre-compiled cellmlfile (without .cellml)>
* OR --model <file> (a CellML file, relative to current working directory or an␣
↪absolute path)
*
* SPECIFYING PACING:
* --pacing-freq            Pacing frequency (Hz) (optional - defaults to 1Hz)
* --pacing-max-time        Maximum time for which to pace the cell model in MINUTES
*                          (optional - defaults to time for 10,000 paces at this␣
↪frequency)
* --pacing-stim-duration   Duration of the square wave stimulus pulse applied (ms)
*                          (optional - defaults to stimulus duration from CellML)
* --pacing-stim-magnitude  Height of the square wave stimulus pulse applied (uA/cm^2)
*                          (optional - defaults to stimulus magnitude from CellML)
*
* SPECIFYING DRUG PROPERTIES dose-response properties for each channel:
* Channels are named:
* * herg (IKr current - hERG),
* * na (fast sodium current - NaV1.5),
* * nal (late/persistent sodium current - NaV1.5 (perhaps!)),
* * cal (L-type calcium current- CaV1.2),
* * iks (IKs current - KCNQ1 + MinK),
* * ik1 (IK1 current - KCNN4 a.k.a. KCa3.1),
* * ito ([fast] Ito current - Kv4.3 + KChIP2.2).
*
* For each channel you specify dose-response parameters [multiple entries for repeat␣
↪experiments]
*   EITHER with IC50 values (in uM), for example for 'hERG':
* --ic50-herg     hERG IC50    (optional - defaults to "no effect")
*   OR with pIC50 values (in log M):
* --pic50-herg    hERG pIC50   (optional - defaults to "no effect")
*     (you can use a mixture of these for different channels if you wish,
*     e.g. --ic50-herg 16600 --pic50-na 5.3 )
```

(continues on next page)

(continued from previous page)

```
*   AND specify Hill coefficients (dimensionless):
* --hill-herg     hERG Hill    (optional - defaults to "1.0")
*   AND specify the saturation effect of the drug on peak conductance (%):
* --saturation-herg   saturation level effect of drug (optional - defaults to 0%)
*
* SPECIFYING CONCENTRATIONS AT COMMAND LINE:
* --plasma-concs  A list of (space separated) plasma concentrations at which to test␣
↪(uM)
* OR alternatively:
* --plasma-conc-high  Highest plasma concentration to test (uM)
* --plasma-conc-low   Lowest  plasma concentration to test (uM)
*                     (optional - defaults to 0)
*
* both ways of specifying test concentrations have the following optional arguments
* --plasma-conc-count  Number of intermediate plasma concentrations to test
*                 (optional - defaults to 0 (for --plasma-concs) or 11 (for --plasma-
↪conc-high))
* --plasma-conc-logscale <True/False> Whether to use log spacing for the plasma␣
↪concentrations
*
* SPECIFYING CONCENTRATIONS IN A FILE (for PKPD runs):
* if you want to run at concentrations in a file instead of specifying at command␣
↪line, you can do:
* --pkpd-file <relative or absolute filepath>
*   To evaluate APD90s throughout a PKPD profile please provide a file with the data␣
↪format:
*   Time(any units)<tab>Conc_trace_1(uM)<tab>Conc_trace_2(uM)<tab>...Conc_trace_N(uM)
*   on each row.
*
* SECOND DRUG:
* To run a second compound, with independent binding model
* That is, total_block = block_drug_1 + block_drug_2 - block_drug_1*block_drug_2
* Supply the following argument:
* --drug-two-conc-factor  Factor to multiply the concentrations of drug 1 (specified␣
↪as above)
*                           to use when evaluating the block due to drug 2.
* To specify drug properties (and UQ below) use the same format, but just before the␣
↪channel name
* insert 'drug-two-' into the option names, for instance:  --pic50-drug-two-herg
*
* UNCERTAINTY QUANTIFICATION:
* --credible-intervals [x y z...] This flag must be present to do uncertainty␣
↪calculations.
*                     It can optionally be followed by a specific list of␣
↪percentiles that are required
*                     (not including 0 or 100, defaults to just the 95% intervals).
* Then to specify 'spread' parameters for assay variability - for use with Lookup␣
↪Tables:
* --pic50-spread-herg      (for each channel that you are providing ic50/pic50 values␣
↪for,
* --hill-spread-herg        herg is just given as an example)
*   (for details of what these spread parameters are see 'sigma' and '1/beta' in␣
↪Table 1 of:
*    Elkins et al. 2013  Journal of Pharmacological and Toxicological
*    Methods, 68(1), 112-122. doi: 10.1016/j.vascn.2013.04.007 )
* --brute-force <N>  Make credible intervals with brute force forward simulations,
*                    rather than using lookup tables, and do N samples each time.
```

(continues on next page)

(continued from previous page)

```
*
*
* OTHER OPTIONS:
* --no-downsampling  By default, we print downsampled output to create small action␣
↪potential
*                    traces, but you can switch this off by calling this option.
* --output-dir       By default output goes into '$CHASTE_TEST_OUTPUT/ApPredict_output
↪'
*                    but you can redirect it (useful for parallel scripting)
*                    with this argument.
* --version          Print out Chaste and ApPredict versions, along with dependency␣
↪versions
*                    and exit immediately (this info automatically goes to a
↪'provenance_info.txt'
*                    file on completion of a normal run without this flag).
```

8.1.3 Running `cardiacmodelling/ap-nimbus-client-direct`

See also:

Offical `docker run` documentation.

Note: `client-direct` requires a number of configuration property values / *Environment Variables* to be assigned (e.g. for database connectivity) before it will start. See later *Prerequisites* section for a more detailed running explanation.

```
docker run -d --name name-client --net ap-nimbus-network --restart always -v volume-
↪client:/opt/django/media -p 4240:80 --env-file env cardiacmodelling/ap-nimbus-
↪client-direct:<version>
```

Where:

1. `ap-nimbus-network` is a docker network.
2. `volume-client` is a docker volume.
3. `env` A file, e.g. docker/env, containing *Environment Variables* (which could also be passed as command line switches).
4. `<version>` is a valid dockerhub tag (as available at https://hub.docker.com/r/cardiacmodelling/ap-nimbus-client-direct).

Note: If you want to run `cardiacmodelling/ap-nimbus-client-direct` in "developer" mode (e.g. using client/config/develop_settings.py) then add the `-d DJANGO_SETTINGS_MODULE=config.develop_settings` to the `docker run` command.

In Docker parlance, the `-p 4240:80` will "publish" or "expose" the container port 80 to port 4240 on the host machine by means of fiddling with the firewall (see *cardiacmodelling/ap-nimbus-client-direct* for a bit more information).

Environment Variables

The environment variables used by the docker components for `AP-Nimbus` are listed below. They are listed as `VAR=VALUE` pairs (without any spaces).

Note: A template environment var file is provided in the repository, e.g. docker/env.

```
DJANGO_SECRET_KEY=
```

Please pick a secret key. A long random string is ideal.

```
DJANGO_SUPERUSER_EMAIL=
DJANGO_SUPERUSER_FULLNAME=
DJANGO_SUPERUSER_PASSWORD=
DJANGO_SUPERUSER_INSTITUTION=
```

Please assign the email, full name, password and institution of the user who will be the Django superuser on application initialisation. After initialisation, if you wish to change the identity of the Django superuser, you have the choice to either : 1. Create a new user (i.e. someone whose email does not already exist in the application), by specifying a completely new email, full name, password and institution in the above env vars, or, 2. Use the identity of an existing user (identified by email, case sensitive), who will have their existing full name, password and institution overwritten with the values specified in the above env vars. (In both cases, you will need to restart `cardiacmodelling/ap-nimbus-client-direct` for the changes to become effective.)

Warning: On system restart, a Django superuser will exist as defined by whatever data is specified in the `SUPERUSER` environment vars. See create_admin.py, which gets run on each restart. If the Django superuser (as defined by `DJANGO_SUPERUSER_EMAIL`) needs to update their email address, they must do so first via the UI "Account" option, and then via a corresponding update to `DJANGO_SUPERUSER_EMAIL` (any other details to update must be done through `SUPERUSER` environment vars **ONLY**), and then restart `cardiacmodelling/ap-nimbus-client-direct`.

```
subfolder=
```

This is intended for situations where the client is running behind a proxy, e.g. Apache, and a URL path is used to direct proxying requests, e.g. the `ActionPotentialPortal` in https://cardiac.nottingham.ac.uk/ActionPotentialPortal. This variable will ensure the urls used will correctly contain the relevant path.

```
ALLOWED_HOSTS=
```

Allowed hosts Django should be allowed to serve web pages for (comma separated). In production this should probably be set to the public facing hostname of your webpage to prevent security issues such as web cache poisoning. If left empty any host will be allowed (`'*'`)

```
smtp_server=
```

Set the SMTP (Simple Mail Transfer Protocol) server used to send emails from.

```
django_email_from_addr=
```

This email address is used as the `From:` address in any emails sent by the client.

```
#PYTHONUNBUFFERED=1
POSTGRES_PASSWORD=
PGPASSWORD=
PGDATABASE=django_ap_nimbus_client
PGPORT=5432
PGHOST=name-postgres
PGUSER=postgres
```

Database variables. Values above assume you are running a postgres container with name `name-postgres` in the docker network. `PGPASSWORD` is used by Django whereas `POSTGRES_PASSWORD` is used by the Postgres database, so these should have the same value.

```
AP_PREDICT_ENDPOINT=http://name-app-manager:8080
```

Location of the AP predict endpoint. Value above assumes you are running an app-manager container with the name `ap-nimbus-app-manager` in the docker network, but it could be set to be elsewhere.

```
HOSTING_INFO=
```

Supply a brief sentence about where this instance is hosted.

```
PRIVACY_NOTICE=""
```

A brief statement that will be shown at the start of the privacy notice

```
CONTACT_MAILTO=mailto:
```

Mailto link for contacting maintiners

```
CONTACT_TEXT=""
```

Contact text for contacting maintiners

```
AP_PREDICT_STATUS_TIMEOUT=1000
```

Status timeout (in ms). After this time the portal assumes something has gone wrong and stops trying to get a status update.

Prerequisites

In order for the `cardiacmodelling/ap-nimbus-client-direct` to call `app-manager` and display simulation results you at least need:

1. The `app-manager` (See *Running cardiacmodelling/ap-nimbus-app-manager*).
2. Communication: networking

   `app-manager` and `client-direct` need to be able to communicate with each other. There are two main ways of achieving this:

   1. You can refer to components by their docker IP address. To find this it needs to be started first, so you will have to start the database and `app-manager` first and find their IPs to use in the `client-direct` configuration.
   2. An easier way is to create a docker network (e.g. `docker network create ap-nimbus-network`) and add the database, `app-manager` and `client-direct` components to it and give each component a name.

3. Data persistence

   Important: By default data does NOT persist. This means that if you restart the `client-direct` any uploaded files (cellml models and PK data files) will be gone and if you restart the database all data will be gone, i.e. all accounts, simulations, cellml models etc. Data volumes can be used to make sure data persists.

   The following commands create and "inspect" a docker data volume called `volume-postgres`:

   ```
   docker volume create volume-postgres docker volume inspect volume-postgres
   ```

   `inspect` reveals information including the mount point (the actual location where the data is stored). This mountpoint can be backed up if desired.

4. Database

   The following starts a postgres 14.1 database using the official debian-bullseye postgres docker image.

   ```
   docker run -d --name name-postgres --net ap-nimbus-network --restart
   always --user postgres -v volume-postgres:/var/lib/postgresql/data
   --env-file env postgres:14.1-bullseye
   ```

   Parameters

   `-d detached mode` You could start with `-it` if you want to see output. However don't forget to leave the component running (detach rather than close).

   `--name name-postgres` This is the name given to the component, it can be seen when doing `docker ps` and is the hostname we use in settings for `client-direct`.

   `--net ap-nimbus-network` Make sure the component is part of our docker network.

   `--restart always` Should the component stop for whatever reason we want it to try and restart.

   `-v volume-postgres:/var/lib/postgresql/data` Link the docker volume `volume-postgres` to the path inside the container where the database data is stored.

   `--env-file env` Use the *Environment Variables* in the `env` file.

CHAPTER 9

Troubleshooting

9.1 Troubleshooting

9.1.1 Illegal Instruction

When running `cardiacmodelling/ap-nimbus-app-manager`, `cardiacmodelling/appredict-with-emulators` or `cardiacmodelling/appredict-no-emulators` there is a possibility that `ApPredict` will fail to run with the message indicating "illegal instruction", e.g. "PETSC ERROR: Caught signal number 4 Illegal instruction: Likely due to memory corruption".

This was encountered in early builds (i.e. builds before May 2020, or more specifically, since commit e560e78), due to the use of the `ApPredict` build flag `b=GccOptNative`, rather than `b=GccOpt`. The outcome of which was that when the container (and therefore `ApPredict`) was being built, at compile time it was picking up the server's native cpu flags but in some cases these were unlikely to be found in the container's deployment environment (e.g. `sse4_1` and `sse4_2` – use `cat /proc/cpuinfo` for examples).

9.1.2 `client-direct`

If the `client-direct` is not working check that all components are set-up: the `client-direct`, the `app-manager` and the database. Also check that all environment variables have been set appropriately, check that the components are on the same network and have access to eachother.

See also:

Running cardiacmodelling/ap-nimbus-client-direct

CHAPTER 10

Developer Section

10.1 Developer Section

Please keep in mind that some of the content below refers to discontinued activities. See *Activity Overview* for more information on the status of some activities.

10.1.1 Building Options

See also:

docker build reference documentation (Probably a safer choice for information!)

ApPredict-related Containers

This section refers to :

1. appredict-no-emulators
2. appredict-with-emulators

> **Warning:** It's important to note that even when using 32 processors it can 30+ mins to build! Equally, building appredict-with-emulators uses a lot of RAM as `ApPredict` loads files >1Gb into RAM. So better not to use all available processors for this step.

Note: Within the `Dockerfiles` we have not enforced version-controlling of images or dependencies (e.g. by specifying the base image tag with hash, or by using specific versions of software, e.g. libhdf5-dev version *X*), as the built container represents a unique snapshot in time. See No version-controlling in appredict-chaste-libs Dockerfile.

Note: These instructions cover the case for building and testing `ApPredict` containers locally, rather than for uploading to DockerHub.

```
user@host:~> mkdir tmp && cd tmp
user@host:~/tmp> git clone https://github.com/CardiacModelling/appredict-docker
user@host:~/tmp> cd appredict-docker/appredict-no-emulators
user@host:~/tmp/appredict-docker/appredict-no-emulators>

          Edit the Dockerfile to a new appredict tag value

user@host:~/tmp/appredict-docker/appredict-no-emulators> docker build --build-arg␣
↪build_processors=<processors> -t appredict-no-emulators:<new version, e.g. mytest1>␣
↪.
user@host:~/tmp/appredict-docker/appredict-no-emulators> docker run -it --rm␣
↪appredict-no-emulators:<new version, e.g. mytest1>

          Verify below the ApPredict commit value against the commit hashes at https:/
↪/github.com/Chaste/ApPredict/tags

bash-4.4$ apps/ApPredict/ApPredict 2> /dev/null | grep 'ApPredict is based on commit'
bash-4.4$ exit

user@host:~/tmp/appredict-docker/appredict-no-emulators> cd ../appredict-with-
↪emulators
user@host:~/tmp/appredict-docker/appredict-with-emulators>

          Edit the Dockerfile `FROM` clause to appredict-no-emulators:<new version, e.
↪g. mytest1>

user@host:~/tmp/appredict-docker/appredict-with-emulators> docker build --build-arg␣
↪build_processors=<processors> -t appredict-with-emulators:<new version, e.g.␣
↪mytest1> .
```

`cardiacmodelling/ap-nimbus-client-direct` Container

This section refers to :

1. ap-nimbus-client-direct

Perhaps the most important thing to keep in mind when building a local container is to **temporarily** modify the `docker/Dockerfile` to copy the content of the local dir (which you've probably just modified a bit for this change), e.g.

```
user@host:~> mkdir tmp && cd tmp
user@host:~/tmp> git clone https://github.com/CardiacModelling/ap-nimbus-client
user@host:~/tmp> cd ap-nimbus-client
user@host:~/tmp/ap-nimbus-client>

          Edit the docker/Dockerfile to copy the local content
          e.g. #RUN git clone --recursive --branch master --depth 1 https://github.
↪com/CardiacModelling/ap-nimbus-client.git
               COPY / /opt/django/ap-nimbus-client

user@host:~/tmp/ap-nimbus-client> docker build -f docker/Dockerfile -t ap-nimbus-
↪client-direct:<new version, e.g. mytest1> .
```

Why not use Chaste/chaste-docker?

It would be possible to use the "offical" Chaste-endorsed dependency container as a base image to build `cardiacmodelling/appredict-no-emulators`, etc., on – and up to around 2021 we'd been using an Alpine Linux distribution base. Now, however, we're using the preferred Debian container to enable easy install of dependency packages.

10.1.2 Developing with Containers

`cardiacmodelling/ap-nimbus-app-manager`

See also:

Running cardiacmodelling/ap-nimbus-app-manager

Developing in container

If you start a container as follows:

```
docker run -it --name name-app-manager --net ap-nimbus-network --restart always␣
↪cardiacmodelling/ap-nimbus-app-manager:<version> bash
```

You will get a command bash shell inside the container. You can make changes and manually start the `app-manager` by running `/home/appredict/apps/app-manager/kickoff.sh`. In order to try out changes, you can stop the `app-manager` by stopping `kick_off.sh` and killing `convert.sh` and manually restart it after changes have been applied.

Note: If you `exit` the container it will stop and it will lose the changes made. Instead of exiting the container you can detach with `CTRL+P+Q` and re-attach with `docker attach <ap-nimbus-app-manager container name>` (you may eventually need a `docker container stop <ap-nimbus-app-manager container name>`).

`cardiacmodelling/ap-nimbus-client-direct`

See also:

Running cardiacmodelling/ap-nimbus-client-direct

The instruction is to "publish" or "expose" the container ports (see Publish or expose port (-p, –expose) for more information). What this does on an `iptables`-based system is adjust the firewall such as :

```
user@host:~> iptables -L -n --line-numbers -t nat

:

Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    MASQUERADE  all  --  172.19.0.0/16       0.0.0.0/0
:
4    MASQUERADE  tcp  --  172.19.0.3          172.19.0.3          tcp dpt:80
```

(continues on next page)

(continued from previous page)

```
Chain DOCKER (2 references)
num  target     prot opt source               destination
1    RETURN     all  --  0.0.0.0/0            0.0.0.0/0
2    DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:4240␣
↪to:172.19.0.3:80

:
```

Django static file creation

Within ap-nimbus-client/client/static there are the static files which the Django UI references, e.g. images, css, js.

The following commands represent an example of how a new minified `main-min.js` can be generated.

Note : You need to have `npm` (so you probably need `node.js` installed - e.g. `apt install nodejs npm`).

> **Warning:** 1. The commands below also modify the version-controlled file `npm-shrinkwrap.json`, but don't commit this change! 2. They also fill up `node_modules` with almost 100Mb of stuff which you **really** don't want to be copied into a container! Better to delete the content of this directory before building the container.

```
user@host:~/git/ap-nimbus-client/client/static> npm install gulp
user@host:~/git/ap-nimbus-client/client/static> ./node_modules/gulp/bin/gulp.js
↪# <-- reads gulpfile.js
user@host:~/git/ap-nimbus-client/client/static> find ./ -mmin -1 -type f
./css/style-min.css
./build/js/main-min.js
```

If you're just wanting to create and use an unminified version of `main.js` for local testing, you can try the following.

```
user@host:~/git/ap-nimbus-client/client/static> sed -i "s/noSource: true/noSource:␣
↪true, ignoreFiles: ['main.js']/g" gulpfile.js
user@host:~/git/ap-nimbus-client/client/static> sed -i "s/main-min/main/g" ../
↪templates/includes/head.html
user@host:~/git/ap-nimbus-client/client/static> ./node_modules/gulp/bin/gulp.js
↪# <-- reads gulpfile.js
user@host:~/git/ap-nimbus-client/client/static> find ./ -mmin -1 -type f
./css/style-min.css
./build/js/main.js
```

Django migrations

A simplified section for developers unfamiliar with Python, Django and data migrations!

Requirements :

- A running postgres db
- env vars in your environment. Derived from `docker/env`, and containing valid values, particularly db data. In the commands below I've put an env file in `~/git/env` (and note that if there are spaces in env var values then you may need to wrap the values in double quotes, e.g. `ENVVAR="An env var val"`)

```
user@host:~/git/ap-nimbus-client> python3 -m venv ../tmpenv
user@host:~/git/ap-nimbus-client> source ../tmpenv/bin/activate
↪                 # <-- Activate virtual env
(tmpenv) user@host:~/git/ap-nimbus-client> pip install -r requirements/requirements.
↪txt                # <-- Load python libs into virtual env
(tmpenv) user@host:~/git/ap-nimbus-client> set -a; source ../env; set +a
↪                 # <-- Load env vars (e.g. db host/user/etc) into environment
(tmpenv) user@host:~/git/ap-nimbus-client> python client/manage.py showmigrations
↪                 # <-- Should show migrations on a "per app" basis
# We want to update db records (not alter table properties), so create a templated␣
↪migration (for the "files" app)
(tmpenv) user@host:~/git/ap-nimbus-client> python client/manage.py makemigrations␣
↪files --name fix_0005_auto --empty
(tmpenv) user@host:~/git/ap-nimbus-client> nano client/files/migrations/0008_fix_0005_
↪auto.py          # <-- Edit content according to migration
(tmpenv) user@host:~/git/ap-nimbus-client> deactivate
↪                 # <-- Leave virtual env
user@host:~/git/ap-nimbus-client>
↪                 # <-- As you were (except vars in your environment, ../env, and ../
↪tmpenv!!)
```

Developing in container

If you start a container using the `docker run -d --name name-client ....` you can access the running container using, for example :

```
docker exec -it $(docker inspect --format="{{.Id}}" name-client) /bin/bash
```

Therein you can access whichever parts of the application the user `appredict` has been granted access to (which would likely be determined by whatever has been assigned in the container's `/etc/sudoers`, as determined by the container Dockerfile).

10.1.3 Container Operations

cardiacmodelling/ap-nimbus-app-manager

1. Docker runs `kick_off.sh` when the image is run.
2. `kick_off.sh` firstly sets off `convert.sh` watching file creation/modification in a `run` directory.
   - If a file creation/modification event of interest takes place (as a result of an `ApPredict` invocation) the changed file is read from the `run` directory (probably a simulation-specific `ApPredict_output/` directory), processed, and written (usually in JSON (Javascript Object Notation) format) to a simulation-specific `res` directory.
3. `kick_off.sh` secondly sets off `server.js` listening on whichever host/port (e.g. `0.0.0.0:8080`).
4. `app-manager` (or rather `server.js`,) receives a `POST` request from `client-direct` to run `ApPredict`
5. `server.js` invokes `run_me.sh`
6. `run_me.sh` starts `ApPredict` which writes output to the `run` directory.
7. `convert.sh` sees results appearing in the `run` directory and, on events of interest, processes the content and writes it to the `res` directory.

8. `server.js` will be awaiting polling calls from `client-direct` requesting the results data and responding with content from files in the `res` directory if/when available.

10.1.4 Further Information

Azure Durable Functions

The following were based on observations as at March 9th 2020.

- Based on Allow selecting disk type when creating AKS cluster it appears that the default policy which can't be overridden is to use expensive disk. I want to use cheap storage for proof-of-concept, not premium SSD (Solid State Drive)!!! `az disk list`
- I want to test my stuff on Linux but when investigating the use of Async HTTP APIs I discover that it's not possible because as of March 2020 it's impossible to fully emulate core elements of Azure Storage (e.g. blob, file, queue and table). The nearest option is Azurite which, in version 3, does not offer Azure Tables emulation (e.g. `docker run -p 10000:10000 -p 10001:10001 mcr.microsoft.com/azure-storage/azurite`) because it doesn't listen for Table Storage on port 10002. Ironically it did in version 2, i.e. `docker run -d -t -p 10000:10000 -p 10001:10001 -p 10002:10002 -v /path/to/folder:/opt/azurite/folder arafato/azurite`, however when I tried to use it it failed – perhaps because Azure Functions arrived after version 2. Microsoft does kindly provide a storage emulator, but unkindly it only runs on Windows. `az storage account list`

10.1.5 Networking and persistence

In order to have components communicate and have data persist we suggest taking advantage of docker networking and docker volumes. see *Prerequisites*.

10.1.6 Backup

The `client-direct` component stores user log-ins and stores simulations users have run, CellML files and PKPD data files users have uploaded. In order to make sure that this data is not lost if the docker component needs to be restarted, make sure you use docker volumes for both the database used (if it's running as a docker component) and the `client-direct`. See *Install cardiacmodelling/ap-nimbus-client-direct*.

It is generally also a good idea to back-up this data in order to be able to recover from any issues with the host system, or that cannot be solved with a simple restart of the docker component. In terms of backing up, we suggest using docker to dump the database to a file, and to compress the local path the data volumes refer to a file as well, both on a regular schedule. In order to find out where the data is stores use the command `docker inspect volume-postgres` where `volume-postgres` is the name of the data volume and look for the path specified as *Mountpoint*.

An example of a backup set-up can be found at https://github.com/CardiacModelling/ap-nimbus-client/tree/master/backup, but this should be adjusted for your specific system.

10.1.7 Starting simulations via command line

Alternative Mechanisms

1. Start `cardiacmodelling/ap-nimbus-app-manager` and call its endpoint from the command line. See *Running cardiacmodelling/ap-nimbus-app-manager*.

2. Run `cardiacmodelling/appredict-with-emulators` or `cardiacmodelling/appredict-no-emulators` in isolation/standalone. See *Running cardiacmodelling/appredict-no-emulators or cardiacmodelling/appredict-with-emulators*.
3. Start `cardiacmodelling/ap-nimbus-app-manager` and a database as you normally would for `cardiacmodelling/ap-nimbus-client-direct` (see *Prerequisites*), but rather than use the `client-direct` UI, instead invoke `cardiacmodelling/ap-nimbus-client-direct` from the command line as below.

Invoking `cardiacmodelling/ap-nimbus-client-direct` as a CLI

Apart from using the web front-end in the form of *client-direct*, `cardiacmodelling/ap-nimbus-client-direct` also offers a command-line tool for starting simulations. In order to call this tool, it needs a setup with Python, the same installed components as the web front-end and access to the database (see *Prerequisites*). The most conveniant way to achieve this is to start a docker component specifically for the purpose of starting simulations in command line. The example below starts a docker component and shows the various command line options available.

Note: Any path to PK data files is local to the docker component.

```
sudo docker run -it \
                --rm \
                --net ap-nimbus-network \
                -v volume-client:/opt/django/media \
                --env-file env \
                cardiacmodelling/ap-nimbus-client-direct:<version> \
                python manage.py start_simulation -h

usage: manage.py start_simulation [-h] [--model_year MODEL_YEAR] [--model_version␣
↪MODEL_VERSION] [--notes NOTES] [--pacing_frequency PACING_FREQUENCY] [--maximum_
↪pacing_time MAXIMUM_PACING_TIME] [--ion_current_type ION_CURRENT_TYPE] [--ion_units␣
↪ION_UNITS] [--concentration_type PK_OR_CONCS]
                                  [--minimum_concentration MINIMUM_CONCENTRATION] [--
↪maximum_concentration MAXIMUM_CONCENTRATION] [--intermediate_point_count␣
↪INTERMEDIATE_POINT_COUNT] [--intermediate_point_log_scale INTERMEDIATE_POINT_LOG_
↪SCALE] [--PK_data_file PK_DATA] [--concentration_point CONCENTRATION_POINT]
                                  [--current_inhibitory_concentration <current>␣
↪concentration hill coefficient saturation level spread of uncertainty] [--version]␣
↪[-v {0,1,2,3}] [--settings SETTINGS] [--pythonpath PYTHONPATH] [--traceback] [--no-
↪color] [--force-color] [--skip-checks]
                                  title author_email model_name

positional arguments:
  title                 Title to identify sumulations. Please note: use quotes if the␣
↪title contains spaces or quotes.
  author_email          Email address of the author for which the simulation is run
  model_name            The name of the model to use. If the name is not unique,␣
↪please also specify year and/or version. Please note: use quotes if the model name␣
↪contains spaces or quotes.

optional arguments:
  -h, --help            show this help message and exit
  --model_year MODEL_YEAR
                        The year for a specified model, to tell models with the same␣
↪name apart e.g. 2020
```

(continues on next page)

(continued from previous page)

```
  --model_version MODEL_VERSION
                        The model version, where a model has multiple versions e.g.␣
↪CiPA-v1.0
  --notes NOTES         Textual notes for the simulation. Please note: use quotes if␣
↪the notes contain spaces or quotes.
  --pacing_frequency PACING_FREQUENCY
                        (in Hz) Frequency of pacing (between 0.05 and 5).
  --maximum_pacing_time MAXIMUM_PACING_TIME
                        (in mins) Maximum pacing time (between 0 and 120).
  --ion_current_type ION_CURRENT_TYPE
                        Ion current type: (pIC50 or IC50)
  --ion_units ION_UNITS
                        Ion current units. (-log(M), M, µM, or nM)
  --concentration_type PK_OR_CONCS, --pk_or_concs PK_OR_CONCS
                        Concentration specification type. (compound_concentration_
↪range, compound_concentration_points, or pharmacokinetics)
  --minimum_concentration MINIMUM_CONCENTRATION
                        (in µM) at least 0.
  --maximum_concentration MAXIMUM_CONCENTRATION
                        (in µM) > minimum_concentration.
  --intermediate_point_count INTERMEDIATE_POINT_COUNT
                        Count of plasma concentrations between the minimum and␣
↪maximum (between 0 and 10).
  --intermediate_point_log_scale INTERMEDIATE_POINT_LOG_SCALE
                        Use log scale for intermediate points.
  --PK_data_file PK_DATA, --PK_data PK_DATA
                        File format: tab-seperated values (TSV). Encoding: UTF-8␣
↪Column 1 : Time (hours) Columns 2-31 : Concentrations (µM).
  --concentration_point CONCENTRATION_POINT
                        Specify compound concentrations points one by one. For␣
↪example for points 0.1 and 0.2 specify as follows: --concentration_point 0.1 --
↪concentration_point 0.2
  --current_inhibitory_concentration <current> concentration hill coefficient␣
↪saturation level spread of uncertainty
                        Inhibitory concentrations, one by one e.g. --current_
↪inhibitory_concentration INa 0.5 1 0 0
  --version             Show program's version number and exit.
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output, 2=verbose␣
↪output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g. "myproject.
↪settings.main". If this isn't provided, the DJANGO_SETTINGS_MODULE environment␣
↪variable will be used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g. "/home/
↪djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions.
  --no-color            Don't colorize the command output.
  --force-color         Force colorization of the command output.
  --skip-checks         Skip system checks.
```

10.1.8 Updating `ApPredict`

This section explains what to do to get changes to the `ApPredict` *main* branch to trickle through to the (containerised) `AP-Portal`. The way the containerised portal works, is that `ApPredict` runs inside the

`cardiacmodelling/ap-nimbus-app-manager` container. Therefore we do not need to worry about the client or database components (unless `ApPredict` input or output formats have changed!).

Prerequisites

1. Write access to https://github.com/Chaste/ApPredict If you have a GitHub personal access token, you may need to run `git clone https://<user>:<token>@github.com/repo/project` or `git remote set-url origin https://<user>:<token>@github.com/repo/project`
2. Write access to https://github.com/CardiacModelling/ap-nimbus
3. Write access to https://github.com/CardiacModelling/ap-nimbus-app-manager
4. Write access to https://github.com/CardiacModelling/appredict-docker
5. Write access to https://hub.docker.com/u/cardiacmodelling You may need to run `docker login -u <user> -p <pwd> docker.io` at the command line to authenticate
6. (Optional) https://readthedocs.org/ access (to rebuild https://ap-nimbus.readthedocs.io/)
7. A multicore server with good upload speeds as a build machine!

Steps

Warning: For brevity only the cloning, updating and pushing to *master* branch is illustrated - new branches and Pull Requests are recommended. It would be better to **not** run the "build" instructions on servers hosting `AP-Nimbus` - to avoid inadvertently overwriting existing containers. If you are concerned about ingesting cached layers, then use the option `--no-cache` during docker building. (Alternative techniques are discussed in Stack Overflow - How to force Docker for a clean build of an image)

- Make changes to the underlying `ApPredict` and assign an annotated tag to the `ApPredict` default branch, e.g.

```
user@host:~/git> git clone https://github.com/Chaste/ApPredict
user@host:~/git> cd ApPredict
user@host:~/git/ApPredict> git tag -a <new_appredict_tag> -m 'some messages for␣
↪git history'
user@host:~/git/ApPredict> git push --tags
```

- **Optional Step!** : Only if there's been a change in `ApPredict`'s *chaste-libs* dependencies and we need a new `cardiacmodelling/appredict-chaste-libs`.
  - Determine the next available `cardiacmodelling/appredict-chaste-libs` container tag value ("*<new_appredict-chaste-libs_tag>*") See https://hub.docker.com/r/cardiacmodelling/appredict-chaste-libs
  - Build and upload to Docker Hub a new `cardiacmodelling/appredict-chaste-libs`.

```
user@host:~/git> git clone https://github.com/CardiacModelling/appredict-
↪docker
user@host:~/git> cd appredict-docker/appredict-chaste-libs
user@host:~/git/appredict-docker/appredict-chaste-libs>
# Edit Dockerfile (and if helpful, README.md)
user@host:~/git/appredict-docker/appredict-chaste-libs> docker build -t␣
↪cardiacmodelling/appredict-chaste-libs:<new_appredict-chaste-libs_tag> .    ␣
↪ # <-- Note trailing "."
```

(continues on next page)

(continued from previous page)

```
user@host:~/git/appredict-docker/appredict-chaste-libs> docker push␣
↪cardiacmodelling/appredict-chaste-libs:<new_appredict-chaste-libs_tag>
# The git committing, pushing, etc., will occur in a later step.
```

- Determine the next available container tag values.
  - `cardiacmodelling/appredict-no-emulators` tag value (“*<new_appredict-no-emulators_tag>*”) See https://hub.docker.com/r/cardiacmodelling/appredict-no-emulators/tags
  - `cardiacmodelling/appredict-with-emulators` tag value (“*<new_appredict-with-emulators_tag>*”) See https://hub.docker.com/r/cardiacmodelling/appredict-with-emulators/tags
  - `cardiacmodelling/ap-nimbus-app-manager` tag value (“*<new_ap-nimbus-app-manager_tag>*”) See https://hub.docker.com/r/cardiacmodelling/ap-nimbus-app-manager/tags
- Build and upload to Docker Hub a new `cardiacmodelling/appredict-no-emulators` and `cardiacmodelling/appredict-with-emulators` **Note** : If you don’t want to be uploading new versions to Docker Hub immediately as per the following instructions, see *ApPredict-related Containers* and *Running cardiacmodelling/appredict-no-emulators or cardiacmodelling/appredict-with-emulators* for building/testing locally.

```
user@host:~/git> cd appredict-docker/appredict-no-emulators
user@host:~/git/appredict-docker/appredict-no-emulators>
# In Dockerfile :
#   1. Optionally update "FROM cardiacmodelling/appredict-chaste-libs:<new_
↪appredict-chaste-libs_tag>"
#      if you're using a new version of chaste libs
#   2. Optionally update "ARG chaste_tag=<Chaste ver>" if you're using a new␣
↪version of Chaste
#   3. Update "ARG appredict_tag=<new_appredict_tag>"
user@host:~/git/appredict-docker/appredict-no-emulators> docker build --build-arg␣
↪build_processors=<processors> -t cardiacmodelling/appredict-no-emulators:<new_
↪appredict-no-emulators_tag> .   # <-- Note trailing "."
user@host:~/git/appredict-docker/appredict-no-emulators> docker push␣
↪cardiacmodelling/appredict-no-emulators:<new_appredict-no-emulators_tag>
user@host:~/git/appredict-docker/appredict-no-emulators> cd ../appredict-with-
↪emulators
# In Dockerfile :
#   1. Update "FROM cardiacmodelling/appredict-no-emulators:<new_appredict-no-
↪emulators_tag>"
user@host:~/git/appredict-docker/appredict-with-emulators> docker build --build-
↪arg build_processors=<processors> -t cardiacmodelling/appredict-with-emulators:
↪<new_appredict-with-emulators_tag> .  # <-- Note trailing "."
user@host:~/git/appredict-docker/appredict-with-emulators> docker push␣
↪cardiacmodelling/appredict-with-emulators:<new_appredict-with-emulators_tag>
# The git committing, pushing, etc., will occur in a later step.
```

- Test the newly-uploaded Docker Hub containers.

  See instructions at *Running cardiacmodelling/appredict-no-emulators or cardiacmodelling/appredict-with-emulators*
- Update documentation
  - Docker Hub, with version numbers and change logs, e.g.
    - (Optional) https://hub.docker.com/repository/docker/cardiacmodelling/appredict-chaste-libs/general
    - https://hub.docker.com/repository/docker/cardiacmodelling/appredict-no-emulators/general

        * https://hub.docker.com/repository/docker/cardiacmodelling/appredict-with-emulators/general

    - GitHub/RtD - Make any relevant changes to :

        * https://github.com/CardiacModelling/ap-nimbus/tree/master/docs/RtD

          You may need to install (Ubuntu) `sphinx`, `sphinx_rtd_theme` / (Fedora) `python3-sphinx`, `python3-sphinx_rtd_theme`.

```
user@host:~/git> git clone https://github.com/CardiacModelling/ap-nimbus
user@host:~/git> cd ap-nimbus/docs/RtD
user@host:~/git/ap-nimbus/docs/RtD>
# Edit .rst files (see https://www.sphinx-doc.org/en/master/usage/
↪restructuredtext/basics.html)
user@host:~/git/ap-nimbus/docs/RtD> make clean html           # <-- After␣
↪this, view content of _build/html/index.html in browser to verify!
```

        * `README.md` files anywhere (e.g. `appredict-docker`, `ap-nimbus`).

- Commit and push GitHub changes

```
user@host:~/git/appredict-docker> git commit -m "Commit message!"
user@host:~/git/appredict-docker> git push
user@host:~/git/appredict-docker> cd ../ap-nimbus
user@host:~/git/ap-nimbus> git submodule update appredict-docker --remote
user@host:~/git/ap-nimbus> git commit -m "Commit message!"
user@host:~/git/ap-nimbus> git push
```

- (Optional) Update `cardiacmodelling/ap-nimbus-app-manager`

```
user@host:~/git> git clone https://github.com/CardiacModelling/ap-nimbus-app-
↪manager
user@host:~/git> cd ap-nimbus-app-manager
user@host:~/git/ap-nimbus-app-manager>
# In Dockerfile
#   1. Update "FROM cardiacmodelling/appredict-with-emulators:<new_ap-nimbus-app-
↪manager_tag>"
user@host:~/git/ap-nimbus-app-manager> docker build --build-arg build_processors=
↪<processors> -t cardiacmodelling/ap-nimbus-app-manager:<new_ap-nimbus-app-
↪manager_tag> .   # <-- Note trailing "."
user@host:~/git/ap-nimbus-app-manager> docker push cardiacmodelling/ap-nimbus-app-
↪manager:<new_ap-nimbus-app-manager_tag>
```

    - Test the newly-uploaded Docker Hub containers.

      See instructions at *Running cardiacmodelling/ap-nimbus-app-manager*

    - Update documentation

        * Docker Hub, with version numbers and change logs, e.g.

            - https://hub.docker.com/repository/docker/cardiacmodelling/ap-nimbus-app-manager/general

        * GitHub/RtD - Make any relevant changes to :

            - https://github.com/CardiacModelling/ap-nimbus/tree/master/docs/RtD (See info above for how to verify changes locally)

            - `README.md` files anywhere.

    - Commit and push GitHub changes

```
user@host:~/git/ap-nimbus-app-manager> git commit -m "Commit message!"
user@host:~/git/ap-nimbus-app-manager> git push
user@host:~/git/ap-nimbus-app-manager> cd ../ap-nimbus
user@host:~/git/ap-nimbus> git submodule update ap-nimbus-app-manager --remote
user@host:~/git/ap-nimbus> git push
```

- Updating the test/production servers

  See internal documentation

Updating Git submodules

Easiest is updating all the submodules at once:

```
git clone --recursive https://github.com/cardiacModelling/ap-nimbus
cd ap-nimbus
git submodule update --remote
git add -A
git commit -m "Updated submodules"
git push
```

If you want to update just 1 submodule then add the path in the update line. E.g.

```
git submodule update --remote client-direct
```