

---

# **Bonsai**

***Release 0.1***

**Markus Johansson**

**Jan 09, 2024**



# CONTENTS

- 1 Installation 1**
  - 1.1 Data persistence . . . . . 2
- 2 Frequently asked questions 3**
- 3 Administration 5**
  - 3.1 Index database . . . . . 5
  - 3.2 Create users . . . . . 5
  - 3.3 Upload samples . . . . . 5
  - 3.4 Create and manage groups of samples . . . . . 6
- 4 Deployment into a production environment 7**
- 5 Bonsai developer documentation 9**
  - 5.1 Services . . . . . 9
- 6 Changelog 13**
  - 6.1 [Unreleased] . . . . . 13
  - 6.2 [v0.2.0] . . . . . 13
  - 6.3 [v0.1.0] . . . . . 14
  - 6.4 Previous release . . . . . 14
- Index 15**



## INSTALLATION

You can use `docker-compose.yml` file to quickly install Bonsai for demo and development purposes. This requires that you have `docker` and `docker-compose` installed. Depending on your environment you might need to edit the `docker-compose` file, `api` config, and `web client` config files.

To install Bonsai, first clone the repository and start the softwares.

```
# clone Bonsai
git clone git@github.com:Clinical-Genomics-Lund/cgviz.git
# navigate to project directory
cd cgviz
# start the bonsai software stack
docker-compose up -d
```

Use the Bonsai `api` command line interface to create the required database indexes.

```
docker-compose exec api bonsai_api index
```

Create an admin user with the CLI. There are three built in user roles (*user*, *uploader*, and *admin*). The *user* role has permission to retrieve data and comment on isolates and should be the default user role. *Uploader* has permission to create and modify data but cannot view isolates, this role is intended for uploading data to the database. The *admin* has full permission to view, create, modify and delete data.

```
docker-compose exec api bonsai_api create-user -u admin -p admin -r admin
```

Use the `upload_sample.py` script to add analysis result and genome signature file to the database.

```
./scripts/upload_sample.sh \
  --api localhost:8011 \
  --group <optional: group_id of group to associate sample with> \
  -u <username> \
  -p <password> \
  --input /path/to/input.json
```

## 1.1 Data persistence

The data is not persistent between docker container updates by default as all data is kept in the container. You have to mount the mongo database and the API genome signature database to the host OS to make the data persistent. The volume mounts can be configured in the `docker-compose.yml` file. If you mount the databases to the host OS you have to ensure that they have correct permissions so the container have read and write access to these files.

Use the following command to get the user and group id of the user in the container.

```
$ docker-compose run --rm mongodb id
# uid=1000(worker) gid=1000(worker) groups=1000(worker)
```

Use `chown -R /path/to/volume_dir 1000:1000` to change the permission of the folders you mount to the container.

The following are an example volume mount configuration. See the [docker-compose](#) documentation for more information on volume mounts.

```
services:
  mongodb:
    volumes:
      - "./volumes/mongodb:/data/db"

  api:
    volumes:
      - "./volumes/api/genome_signatures:/data/signature_db"
```

## FREQUENTLY ASKED QUESTIONS





## ADMINISTRATION

Users and admins administer Bonsai through either the API command line interface or the Bonsai front end. Bonsai has three predefined user roles (*user*, *uploader*, and *admin*) with different levels of privileges. The uploader role has only write permissions and is intended for automation scripts that upload new samples or create groups. A user with an uploader role cannot retrieve information from the database to minimize the threat vector.

The user can view samples and groups and post comments but not upload new results or manage sample groups. The user role should be enough for most users.

The admin user role grants full permission to create, modify, and read all data through the API entry points.

### 3.1 Index database

The database must be indexed for Bonsai to work correctly. The database indexes support and speed up common queries and enforces restriction on the data. For instance, will the indexes prevent duplicated sample IDs and sample group IDs? The indexes are created using the Bonsai API command line interface. Note that if you are running the containerized version of Bonsai, you must execute the commands in the container.

### 3.2 Create users

Users are also created through the API. You must specify the username, password, and user role when you create the user.

### 3.3 Upload samples

JASEN analysis results are uploaded to Bonsai using HTTP requests to the Bonsai API where the analysis result in JSON format and signature file is uploaded separately. Bonsai includes a simple upload bash script that can upload a sample and assign it to a group. The script is located in the scripts directory.

```
./scripts/upload_sample.sh
--api localhost:8011
--group <optional: group_id of group to associate sample with>
-u <username>
-p <password>
--input /path/to/input.json
```

The script demonstrates basic sample management using the API routes and these could be included in your automations for sample processing.

## 3.4 Create and manage groups of samples

Groups can be created and managed through the front end by admin users. Groups can be created and modified from the */groups/edit* view. Groups can also be managed through the API, which allows for automation and/ or integration into sample processing pipelines.

## DEPLOYMENT INTO A PRODUCTION ENVIRONMENT

The following should be done when deploying Bonsai to a production environment.

1. Change the *SECRET\_KEY* variable in the Bonsai app and API configuration. Use a production-ready WSGI server such as gunicorn instead of the default Flask development server.
2. Configure the *BONSAI\_API\_URL* in the frontend config or set it as an environmental variable.
3. It should be set to the URL where you host the Bonsai API.

You should use SSL certificates in a production environment to protect the data sent from being intercepted. We also recommend enabling authentication to your Mongo database and setting up specific accounts for database administrators and software to protect your data. See the [Mongo documentation](#) for more information on authentication and authorization mechanisms.



## BONSAI DEVELOPER DOCUMENTATION

Bonsai consists of a front end, an API layer, and several services that perform long-running or computationally intensive tasks. The front end uses the API to query or update the database and to create jobs for the services to execute. Redis is used as a queue and for caching the results of API queries. The services listen for submitted jobs, perform the task, and return the results to Redis. The front end can then request status updates of submitted jobs using the unique job id it received when it requested the job.

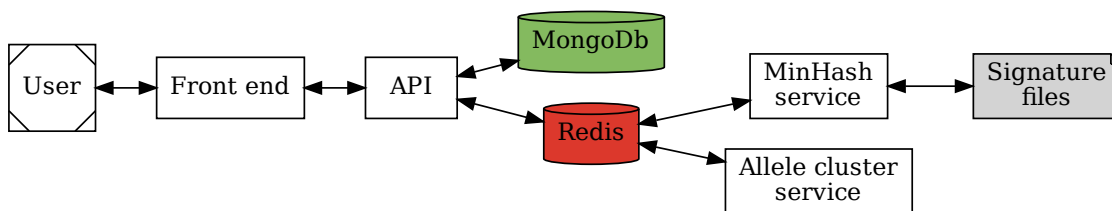


Fig. 1: Overview of Bonsai software stack.

## 5.1 Services

### 5.1.1 MinHash clustering

The MinHash clustering service subscribes to the Redis database and listens for jobs on the *minhash* queue.

#### Tasks

`minhash_service.tasks.add_signature(sample_id: str, signature) → str`

Find signatures similar to reference signature.

#### Parameters

- **str** (*sample\_id*) – the sample\_id
- **str** [*signature Dict[str, ...]*] – sourmash signature file in JSON format

#### Returns

path to the signature

**Return type**`str``minhash_service.tasks.remove_signature(sample_id: str) → Dict[str, str | bool]`

Remove a signature from the database and index.

**Parameters**

**str** (*sample\_id*) – the sample\_id of the signature to remove

**Returns**

The status of the removed job

**Return type**`Dict[str, str | bool]``minhash_service.tasks.index(sample_ids: List[str]) → str`

Add signatures to sourmash SBT index.

**Parameters**

**List[str]** (*sample\_ids*) – The path to multiple signature files

**Returns**

result message

**Return type**`str``minhash_service.tasks.similar(sample_id: str, min_similarity: float = 0.5, limit: int | None = None) → List[SimilarSignature]`

Find signatures similar to reference signature.

**Parameters**

- **str** (*sample\_id*) – The id of reference sample
- **float** (*min\_similarity*) – Minimum similarity score
- **None** (*limit int* /) – Limit the result to x samples, default to None

**Returns**

list of the similar signatures

**Return type**`SimilarSignatures``minhash_service.tasks.cluster(sample_ids: List[str], cluster_method: str = 'single') → str`

Cluster multiple sample on their sourmash signatures.

**Parameters**

- **List[str]** (*sample\_ids*) – The sample ids to cluster
- **int** (*cluster\_method*) – The linkage or clustering method to use, default to single

**Raises**

**ValueError** – raises an exception if the method is not a valid MStree clustering method.

**Returns**

clustering result in newick format

**Return type**`str`

`minhash_service.tasks.find_similar_and_cluster(sample_id: str, min_similarity: float = 0.5, limit: int | None = None, cluster_method: str = 'single') → str`

Find similar samples and cluster them on their minhash profile.

#### Parameters

- **str** (*sample\_id*) – The id of reference sample
- **float** (*min\_similarity*) – Minimum similarity score
- **None** (*limit int* ) – Limit the result to x samples, default to None
- **int** (*cluster\_method*) – The linkage or clustering method to use, default to single

#### Raises

**ValueError** – raises an exception if the method is not a valid MStree clustering method.

#### Returns

clustering result in newick format

#### Return type

str

## 5.1.2 Allele clustering service

The allele clustering service subscribes to the Redis database and listens for jobs on the *allele\_clustering* queue. The exposed functions are located in *tasks.py*.

### Tasks

`allele_cluster_service.tasks.cluster(profile: str, method: str) → str`

Cluster multiple sample on their allele profiles.

#### Parameters

- **str** (*method*) – a string representation of a tsv table of the allele profiles
- **str** – the MStree clustering method

#### Raises

**ValueError** – raises an exception if the method is not a valid MStree clustering method.

#### Returns

cluster in newick format

#### Return type

str





## CHANGELOG

Starting with version 2.0 will the project use [semantic versioning](#).

### 6.1 [Unreleased]

#### 6.1.1 Added

#### 6.1.2 Fixed

#### 6.1.3 Changed

### 6.2 [v0.2.0]

#### 6.2.1 Added

- Improved output of create\_user API CLI command
- bonsai\_api create-user command have options for mail, first name and last name.
- Open samples by clicking on labels in the similar samples card in the samples view.
- Optional “extended” HTTP argument to sample view to view extended prediction info

#### 6.2.2 Fixed

- Fixed crash in create\_user API CLI command
- Resistance\_report now render work in progress page
- Removed old project name from GrapeTree header
- Fixed issue that prevented node labels in GrapeTree from being displayed.

### 6.2.3 Changed

- The role “user” have permission to comment and classify QC
- Updated PRP to version 0.3.0

## 6.3 [v0.1.0]

### 6.3.1 Added

- Find similar samples by calculating MinHash distance
- Added async similarity searches and clustering
- User can choose clustering method from the sample basket
- Added spp prediction result to sample view
- Find similar samples default to 50
- Added STX typing for *Escherichia coli* samples
- CLI command for updating tags for all samples

### 6.3.2 Fixed

- Fixed calculation of missing and novel cgmlst alleles
- Fixed so sample metadata is displayed in GrapeTree

### 6.3.3 Changed

- Renamed client to frontend and server to api
- Renamed cgvis to Bonsai
- Complete rewrite of the project
- Removed N novel cgmlst alleles from cgmlst qc card
- Use data models from JASEN Pipeline Result Processing application
- Use pre-defined table columns in edit groups view

## 6.4 Previous release

Starting with version 2.0, the project was renamed from cgvis to Bonsai.

Analyze outbreak, antimicrobial resistance and virulence factors in bacteria.

---

**Note:** This project is under active development and Bonaai and the documentation is likely to change.

---

Intended to visualize results from [JASEN](#) pipeline.

Check out the [Installation](#) section for installation instructions.

## INDEX

### A

`add_signature()` (*in module minhash\_service.tasks*), 9

### C

`cluster()` (*in module allele\_cluster\_service.tasks*), 11

`cluster()` (*in module minhash\_service.tasks*), 10

### F

`find_similar_and_cluster()` (*in module minhash\_service.tasks*), 10

### I

`index()` (*in module minhash\_service.tasks*), 10

### R

`remove_signature()` (*in module minhash\_service.tasks*), 10

### S

`similar()` (*in module minhash\_service.tasks*), 10