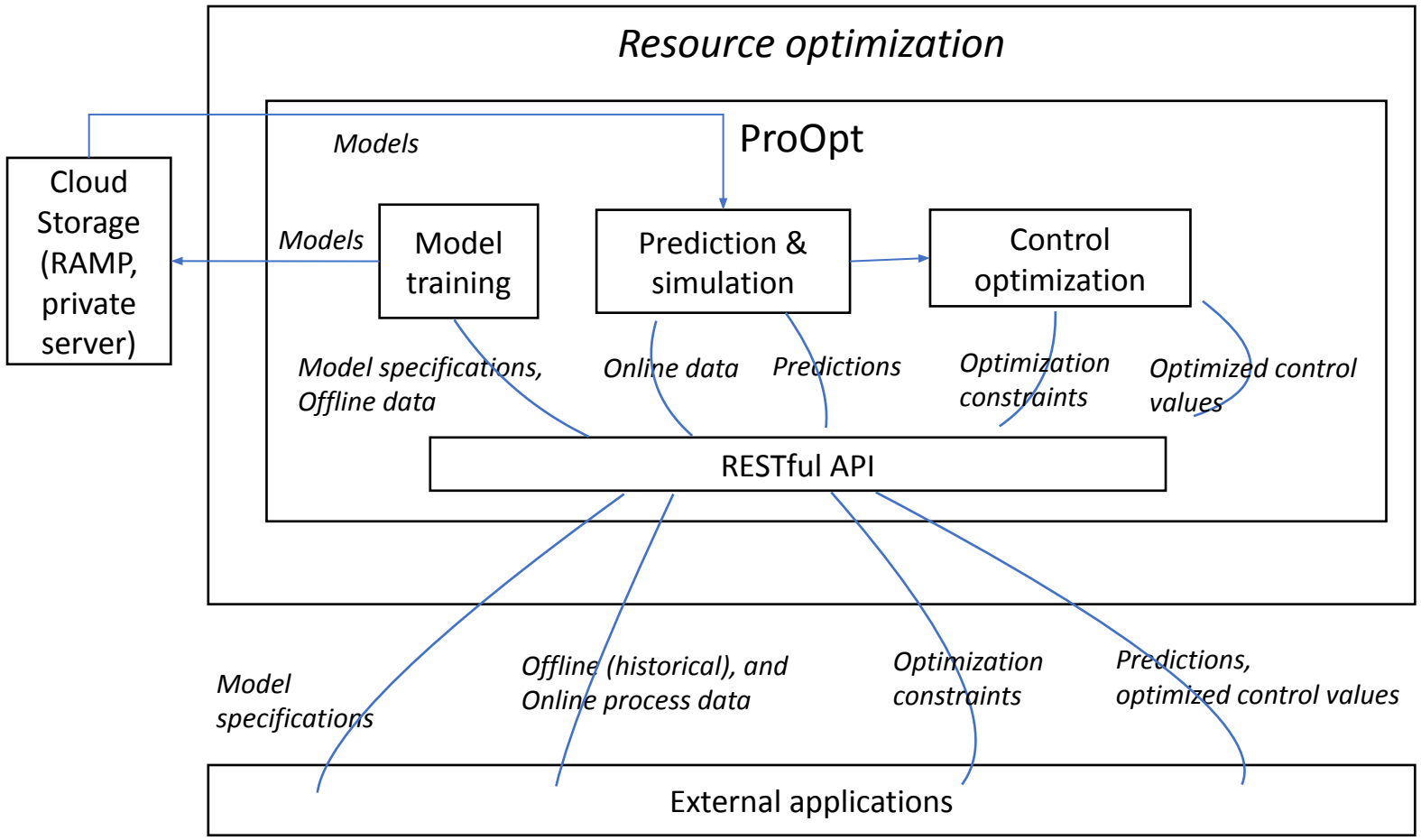
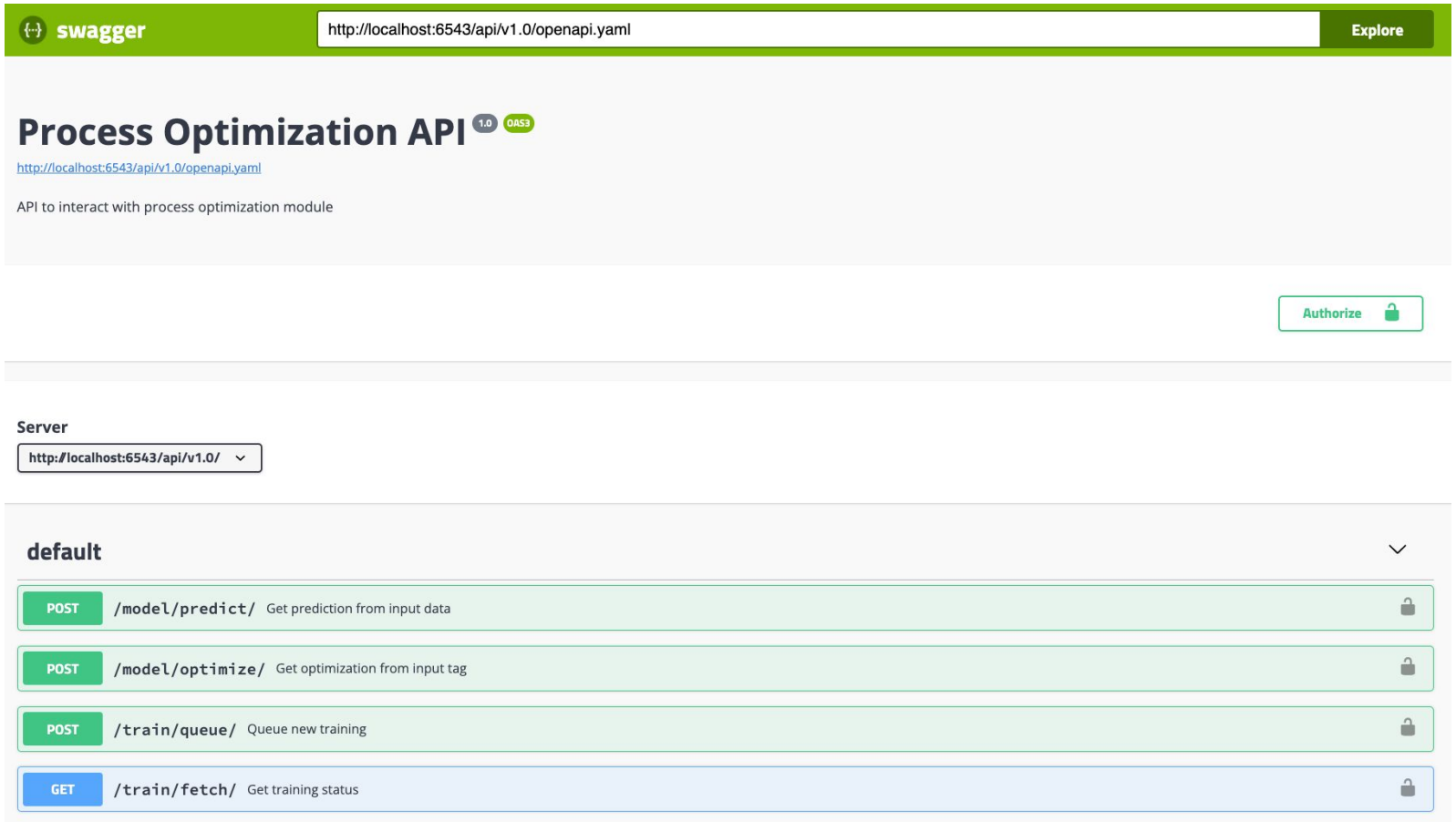
Process Optimization
Technical Architecture



ProOpt has three main functional modules and communicates with end-use applications through a RESTful API.

1. The **model training** module receives training dataset and model specification from external applications to run a training pipeline and results in predictive models. The models are stored in the RAMP storage for later use by other modules.
2. The **Prediction & simulation** module reads process data from external applications and returns the predictions made by the trained models. External applications can change process data and send the data to this module to simulate the target quality.
3. This **Control optimization** module is responsible for finding the optimal process setups to achieve the targeted quality. This module receives process data, the optimization constraints and the target quality from the external application and runs the optimization algorithm. The module returns the optimal values of the process parameters, together with the achieved quality

Process Optimization API



There are four API endpoints

1. External application sends process dataset and model configuration data to **`/train/queue/`** endpoint in JSON format
2. External application sends request to **`/train/fetch/`** endpoint to get the model training status
3. External applications send process data to **`/model/predict/`** endpoint and receive the predictions
4. External application send process data and optimization constraints to **`/model/optimize/`** endpoint and receives the optimized process tags and the achieved target quality

Examples (/model/predict/)

Request body specifies used model and recent data frame:

```
{
        model: string,
        data: Array<{
                timestamp: "2020-02-04T12:35:00.000Z",
                tag_name_1: float,
                tag_name_2: float,
                ...
                [key: string]: number, <--- // the amount of key and key names are dynamic
        }>
}
```

Response consisting of predictions (or simulations) in coming time windows:

```
{
        predictions: Array<float>
}
```

Examples (/model/optimize/)

Request body specifies list of tags to optimize:

```
{
        tags: Array<string>, // list of tags
        model: string,
        data: Array<{
                timestamp: "2020-02-04T12:35:00.000Z",
                tag_name_1: float,
                tag_name_2: float,
                ...
                [key: string]: number, <--- // the amount of key and key names are dynamic
        }>
}
```

Response returns optimal values of the tags:

```
{
        optimizations: {
                tag_name_1: float,
                tag_name_2: float,
                ...
                [key: string]: number, <--- // the amount of key and key names are dynamic
        }
}
```