Documentation

None

None

None

Table of contents

1. `daas_web` Documentation

Documentation for the `qweb` server code of the DaaS-DESIGN project: https://daas-design.de/

1.1 Where to start

- whole system overview
- high-level architecture explanation
- Authentication explanation
- low level Python API documentation of the `qweb` module (drill down to specific modules in the sub-navigation)

1.2 Building the docs

These docs use mkdocs.

- `mkdocs new [dir-name]` - Create a new project.
- `mkdocs serve` - Start the live-reloading docs server.
- `mkdocs build` - Build the documentation site.
- `mkdocs -h` - Print help message and exit.

Relevant files:

- `mkdocs.yml` contains configuration
- `docs/` contains documentation pages
- `src/app` contains the code from which API docs are created

2. Guide

2.1 Description

Backend Api and integration layer to control several services being used in the process to provide a desktop as a service (DaaS) solution.

2.1.1 Definitions

Initial definitions:

- Externally defined: GUI, Linux/Windows system, Object, Application, Client, Service, etc.
- Object: Any application
- Usable Objects: Any application that have a GUI and can be natively installed on a current Linux or Windows system.

General Design-specific definitions:

- Frontend: Website acting as user interface
- Frontend-Service: Required service to provide frontend functionality
- Frontend API: Communication interface between frontend and backend (Hosted as web service within the backend)
- Frontend-Database: Database to store session-related information and user configuration
- Backend: Core components implementing usecase functionality and Backend-Service interaction
- Backend-Service: Required service to provide Backend functionality
- Backend-Api: Communication interface between Integration-Layer and Backend-Services
- Backend-Database: Database to store state information for Apps, Deployments, Environments, Baseimages
- Frontend-Users: Endusers utilizing the frontend to interact with Usable Objects (THE user!).
- Frontend-Admin: Experienced Frontend-Users providing Design-Baseimages and Design-Environments for other Frontend-Users

Integration-Layer specific definitions:

- Design-App: Usable objects within Design-Deployments and provided through a web browser.
- Design-Deployment: Containerized or virtualized Design-Environment which contain Design-Apps.
- Design-Environment: Configured Design-Baseimage which contain Design-Deployments
- Design-Baseimage: Installed and configured platform hosted on a Backend-Service
- Design-Instance: A virtualized or containerized system with minimal communication infrastructure (ssh+pubkey-auth+known ip)
- Design-Node: A machine on which the Design-Appliance is installed and configured

Backend-Api:

- Instance: Running App, Deployment, Environment or Baseimage
- Connection: VNC or RDP connection created within Backend-Service guacamole
- VM: Virtual machine created within Backend-Service proxmox
- Container: TODO

2.1.2 Backend-API

The specified API is part of the backend. It is meant to be used by other components of the backend in order to encapsulate the required functionality into a dedicated component. This dedicated component currently acts as a wrapper for the three backend services, proxmox, guacamole, and dnsmasq but might perspectively integrate other services as well.

The API shall mainly be used by a higher order integration layer of the backend which maintains state and a separate communication channel between the backend and frontend. The API is designed to only indirectly interact with users of the DaaS appliance. It can therefore be seen as a broker between all relevant backend services and the integration layer or the frontend.

The API is composed of four major components that are put together within the superior ApiDesign component, representing the actual Backend-API as defined. This Backend-API can be configured by using dedicated configuration objects and is either invoked directly through the integration layer or for debugging purposes through the ApiCmd.py component.

There are currently four core components which are accompanied by one or more auxiliary modules and dedicated configuration objects.

For proxmox, these are:

- ProxmoxApi: Interacts with the proxmox service via REST (-> VM Management)
- ProxmoxRestConfig: Contains required configuration
- ProxmoxRestRequest: Utilizes HttpAdapter to perform Rest requests against the proxmox API

For nodes, these are:

- NodeApi: Interacts with the local operating system (-> Local services, e.g., dnsmasq)
- ProxmoxNetworkConfig: Contains required configuration.
- ProxmoxNetworkRequest: Utilizes SshAdapter to perform requests against the responsible DHCP server.
- TODO: ProxmoxNetworkConfig and ProxmoxNetworkRequest are WIP here! They are still part of ProxmoxApi, therefore the name. It shall be part of a NodeApi, which has to be implemented first.

For instances, these are:

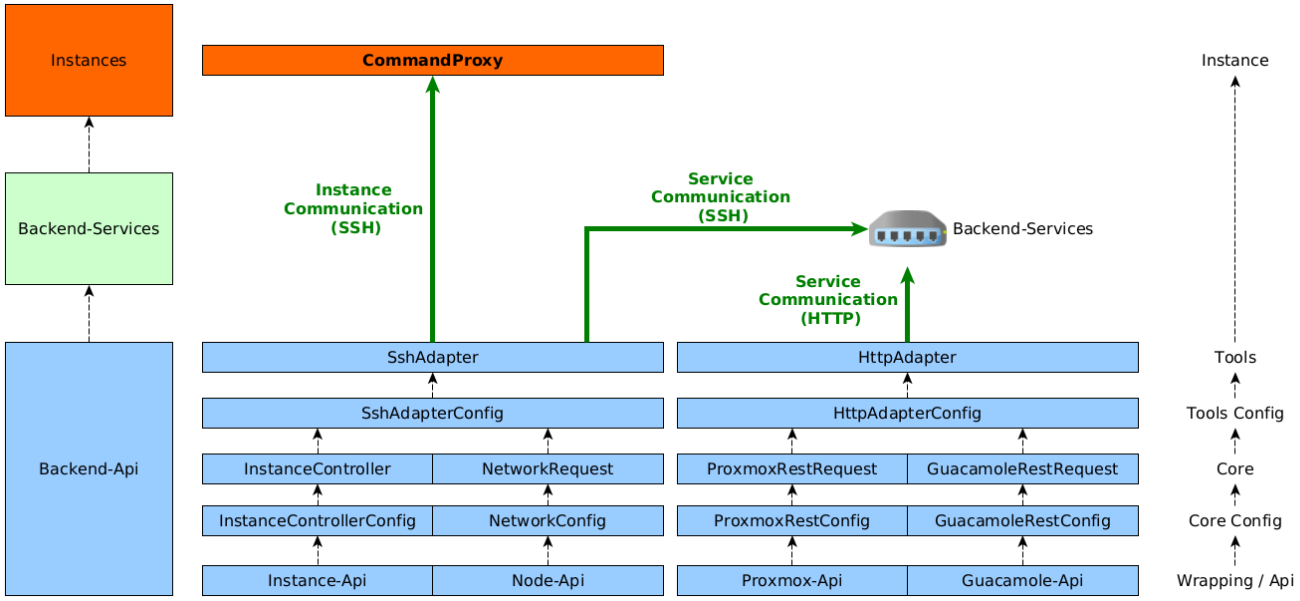
- InstanceApi: Interacts with VMs or containers by using a custom Python component (-> Guest OS and Application Management).
- InstanceControllerConfig: Contains required configuration.
- InstanceController: Utilizes SshAdapter to perform requests against the CommandProxy within a running instance.
- CommandProxy: Located within running BaseImages, Environments, or Deployments to establish a basic communication channel.

For guacamole, these are:

- GuacamoleApi: Interacts with the guacamole service via REST (-> Connection Management)
- GuacamoleRestConfig: Contains required configuration
- GuacamoleRestRequest: Utilizes HttpAdapter to perform Rest requests against the guacamole API.

Two additional helper classes are acting as adapters being used by other components of the Backend-API. These are:

- HttpAdapter: Request and response handling between Backend-Services and components of the Backend-API. Throws exceptions.
- SshAdapter: Utilized to control running Design-Instances. Throws exceptions.



2.1.3 Error handling

In general, appropriate return codes are given, and exceptions are thrown wherever required, but the plausibility of each invocation is not controlled at all. It is assumed that a high layer is able to invoke semantically correct sequences by using its own state information. Other than that, the API prevents syntactical user errors such as wrongly specified parameters.

As the component generally acts as a wrapper and, therefore, only forwards responses obtained by invocation of certain Backend-Services. The API currently avoids introducing its own notion of error codes or error handling. That implies that an invocation to the API resulting in an HTTP request would also yield an HTTP response. An invocation to the API resulting in the creation of a new subprocess would yield the response code of that subprocess.

Three examples:

- The API can be invoked to stop a VM with a certain vmid and will request that action from the backend service even though a particular VM might not exist. Therefore, the requested action against the proxmox services would end up with a 500 HTTP response code which the API would return. The ApiCmd component would treat everything other than a 200 response as an error and produce a response code on the command line which is unequal to zero.
- In another example, the API might be used to configure the IP for a particular VM which leads internally to the manipulation of a config file for dnsmasq and the restart of that service. If, for any reason, the service cannot be restarted, an appropriate error code from that subprocess is returned. In that case, the error code of the subprocess is returned by the API. The ApiCmd component would treat everything else than the return code zero as an error and yield on the command line a response code unequal to zero.
- In a third example, the API might be used to create a VNC connection within Guacamole, but the configuration specifies a wrong hostname within the URL for the Guacamole backend. In that case, the API would try to reach the malformed URL several times until a certain amount of reconnects is reached. If no connection could be made, an exception would be thrown, indicating the root of the problem. In that particular example, a requests.exceptions.ConnectionError exception was thrown with the reason 'No route to host.'

2.1.4 States

The API is designed to be stateless, which means that a caller has to take care that input parameters are valid. The caller has to keep a record of at least the identifiers assigned to usable objects being created within contained Backend-Services such as guacamole or proxmox. In that sense, the API provides methods to obtain that data and store them separately but does not automatically invoke them to assign such identifiers. In most cases, this boils down to only identifiers but can also occur in rarer use cases and for other parameters.

Another problem is the occurrence of race conditions. As the API does not maintain a state on its own, the API itself must, at some points, contact service and act in addition to the received response. Suppose the state of the Backend-Service changes after the response was sent to that caller. In that case, the API might act on behalf of outdated information resulting in an invalid state of the respective backend service. Therefore, a higher-order integration layer has to ensure that no other correlating state changes happen between requests depending on each other.

Two examples:

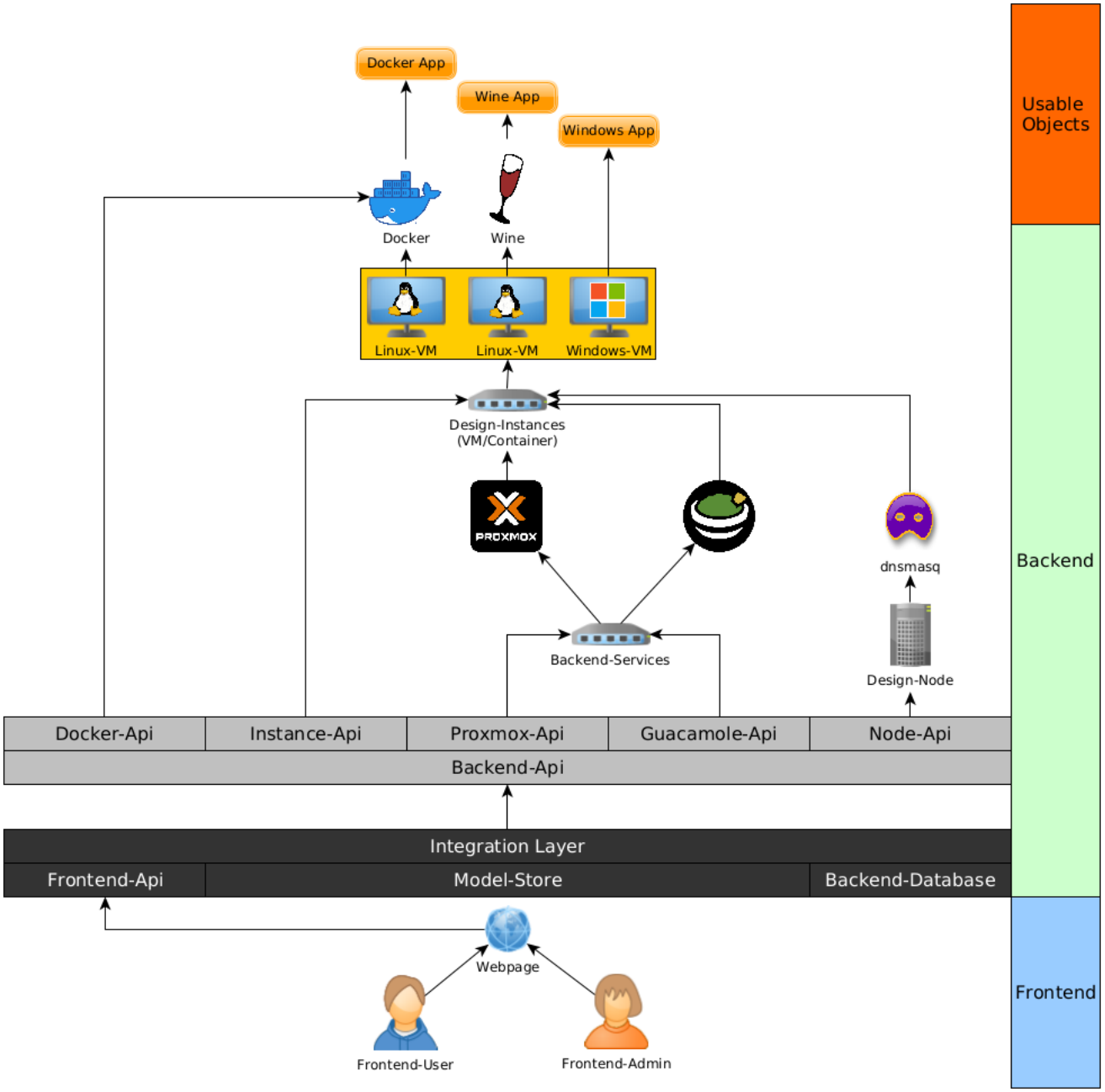
- If the API is invoked to change the configuration of a particular VM, then a request to do so is, in any case, forwarded to the responsible Backend-Service proxmox. This is also the case if that particular configuration is already present, and the request could be omitted theoretically.
- If the API is invoked to delete a VNC connection within guacamole, a first request might be issued beforehand, which lists all connections currently stored within guacamole. This list maps the vmid input parameter to the corresponding connection id, which is then used to delete the connection. The second request to delete the request would unintuitively fail if the connection was already deleted by another request in between.

2.1.5 Integration Layer

To integrate the API, an integration layer is needed to overcome the current prototype's limitations. On the one hand, error codes, error messages, and logs have to be managed in a well-defined manner, and exceptions have to be thrown more appropriately. On the other hand, the already mentioned state aspects have to be taken care of, which involves the persistence of data and the exclusion of multi-threading issues. On top of that, the most important aspect of object management has to be defined. As planned, the integration layer does not instantiate or destroy any usable objects on its own but only controls the lifetime of such objects within Backend-Services. The integration layer provides means of communication to the frontend by which state changes of such objects can be expressed in order to reflect the desired use-cases of frontend users. Direct interaction between Backend-Services and frontend users is appropriately prevented.

Major components:

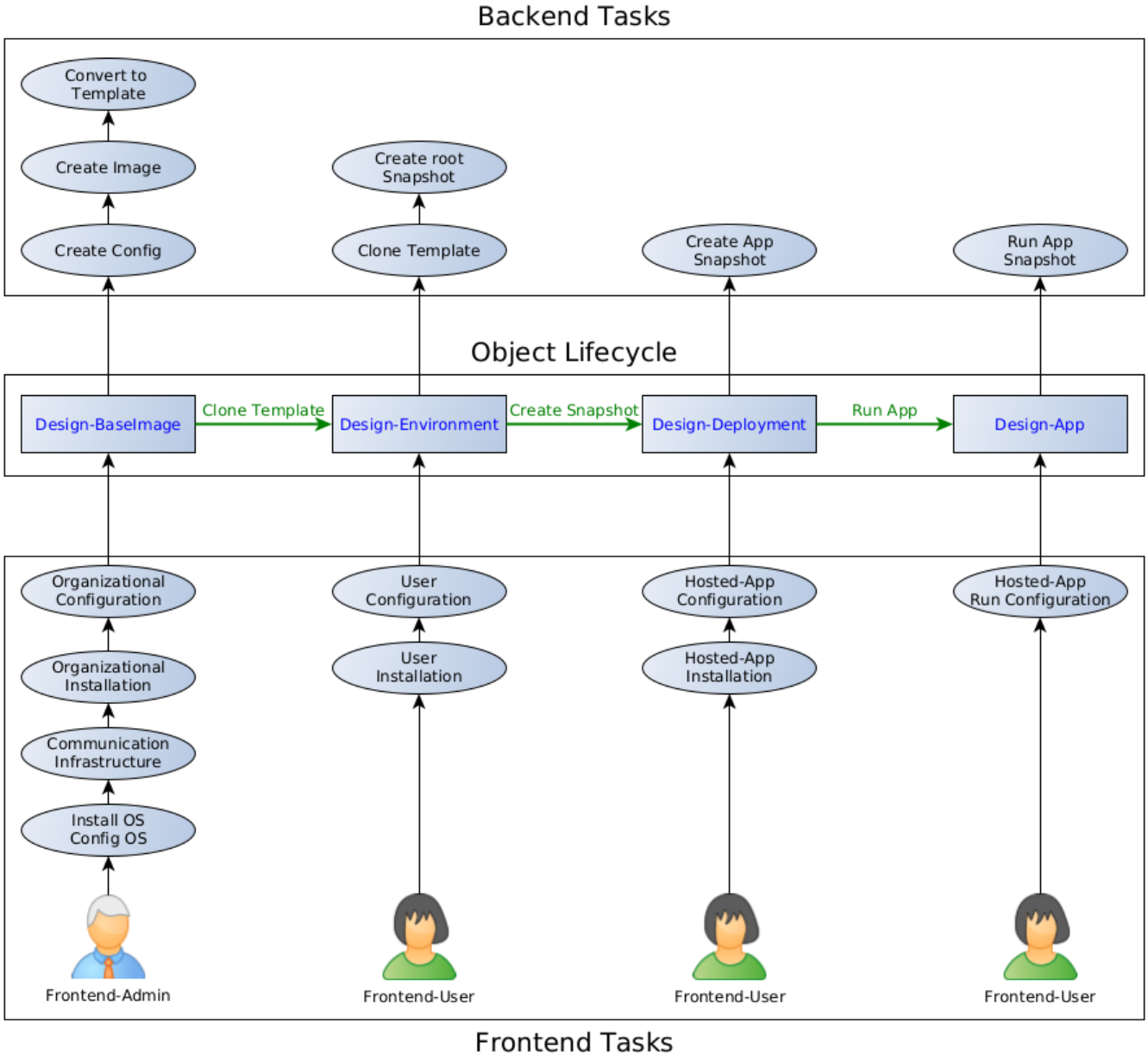
- Frontend-Api: Web-based service to receive and process invocations from the frontend.
- ?Model-Store?: Registry for abstract models of usable objects (Currently only kasm/docker, After merging also VMs)
- Backend-Database: persists state information for usable objects in a local file (currently sqlite)
- Backend-Api: interacts with backend Backend-Services



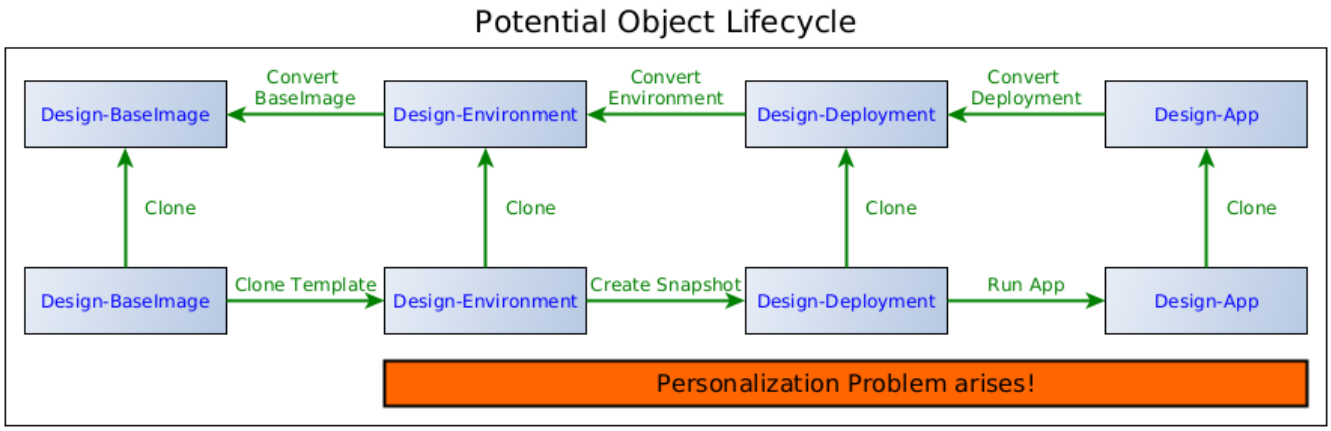
2.1.6 Integration Concept

The Integration Layer maintains four major entities which are able to reflect all necessary state and configuration for hosted applications to a particular point in time. They generally act as information source in order to create, manipulate or destroy resources within Backend-Services and on behalf of actual users or admins. Currently there are four phases which are coined 'BaseImage' (Phase0), 'Environment' (Phase1), 'Deployment' (Phase2), and 'App' (Phase3). In the current version they envisioned to inherit properties of prior phases and can be seen as nodes in a directed graph. The nodes are connected through directed

edges between Baseimage and Environment, between Environment and Deployment as well as between Deployment and App. In that order the edges are coined 'CloneTemplate' (Transition0), 'CreateSnapshot' (Transition1) and 'RunApp' (Transition).



Therefore any object starts at first as a Baseimage and progresses linearly through this directed graph until it is finally deleted. (In a future version this directed graph might also be turned into an undirected graph allowing for more flexibility and more sophisticated usecases. Some of these usecases might imply a crucial problem such that sensitive of one user might be stored within a vm which gets copied to another user which might then potentially obtain the sensitive information of the other user. This has to be prevented to any point in time and therefore future usecases have to be integrated while keeping that in mind. Therefore lets pretend its a directed graph for now!)



In addition to that, all but the last phase can be further partitioned into four subphases with the expressive names 'Create' (SubPhase0), 'ConfigAuto' (SubPhase1), 'ConfigInteractive' (SubPhase2) and 'Finalize' (SubPhase3). The configuration subphases ConfigAuto and ConfigInteractive allow users to contribute to the instantiation process at that particular point in time. The Create and Finalize subphases are allowing the backend to take care of all relevant steps to progress further through the graph. These phases and subphases are in general designed to be flexible enough to adopt to potentially contradicting concepts in order to reach the same ultimate goal of hosting an app through a web browser. These are:

- Containers vs. VM's -> Distinguished by type parameters in all major entities
- Windows vs. Linux vs. Linux+Wine -> Apps can be installed into all suited platforms
- VNC vs. RDP -> VNC for all instances, Optionally RDP for all (installed and configured!) Windows-instances
- Organizational Logic vs. User logic -> Admins ensure business constraints, Users ensure user constraints
- Installation vs. Configuration vs. Utilization -> Entities allow automated and interactive approaches

2.1.7 VM-Approach

The integration concept is mimicking workflows of the VM-driven approach within proxmox where virtual machines can be represented as regular plain virtual machines, virtual machine snapshots or virtual machine templates. On one hand this approach allows to provide very fast clone times from virtual machine templates into regular virtual machines but does not allow to create template snapshots. On the other hand regular virtual machines can contain trees of snapshots to reflect specific states of running and stopped virtual machines but are not very efficient in data storage and access times as well as the time required to clone into another virtual machine. Regular plain virtual machines without snapshots are possible and represent a virtual machine in its most lightweight manner. As no snapshots are contained cloning is as fast as clong from a VM-Template but does not offer possibilities to represent managable states.

In general, a similar approach is also possible in order to reflect container-based environments. Container-based environments are in general more lightweight compared to VM's but must still be prepared in a comparable manner if they are used to host applications in an automated DaaS approach (e.g. Basic Communication, Organizational Config, User Config, App Config, etc.).

This includes the installation, configuration and customization of operating systems, specific user-environments or individual applications. Steps in that process must either be completely or partially automated while also offering interactive approaches if necessary. For each of these steps a majority of the work can be done automatically or on-the-fly under the assumption that required configuration and installation options are configured before within the frontend. It is assumed that a Frontend-User and a Frontend-Admin will each individually contribute to distinct phases of the objects life-cycle. Essentially, a Frontend-Admin might provide Baseimages or Environments containing all required business-logic and organizational constraints. In contrast to that a Frontend-User might contribute more to the customization and individualization of Environments, Deployments or the running app itself.

With that in mind the four mentioned entities reflect individual aspects for the same objects life-cycle while also offering possibilities to distinguish between virtualization and containerization approaches. Additionally this data organization allows to leverage the advantages of the underlying virtual machine backend while also minimizing the disadvantages of it partially by turning Baseimages into fast clonable templates and by representing deployments as fast awakening snapshots.

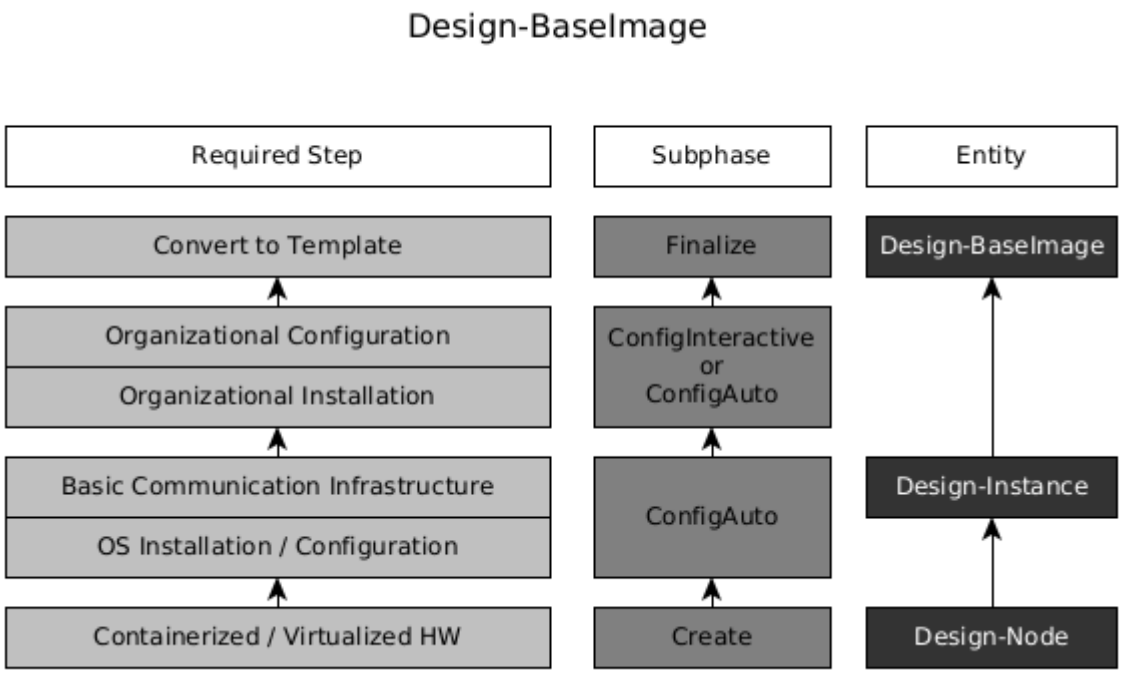
2.1.8 Container-Approach

TODO: Describe how the lightweight container-concept can be extended to match the described phases and subphases of the VM-Approach. Both approaches should be conceptionally aligned to behave similarly such that a common entity concept can be provided from the Frontend-Api to the Frontend-User or Frontend-Admin. As a result of that alignment the provided entity concept shall be be agnostic to the underlying VM or Container implementation!

Entities/Phases

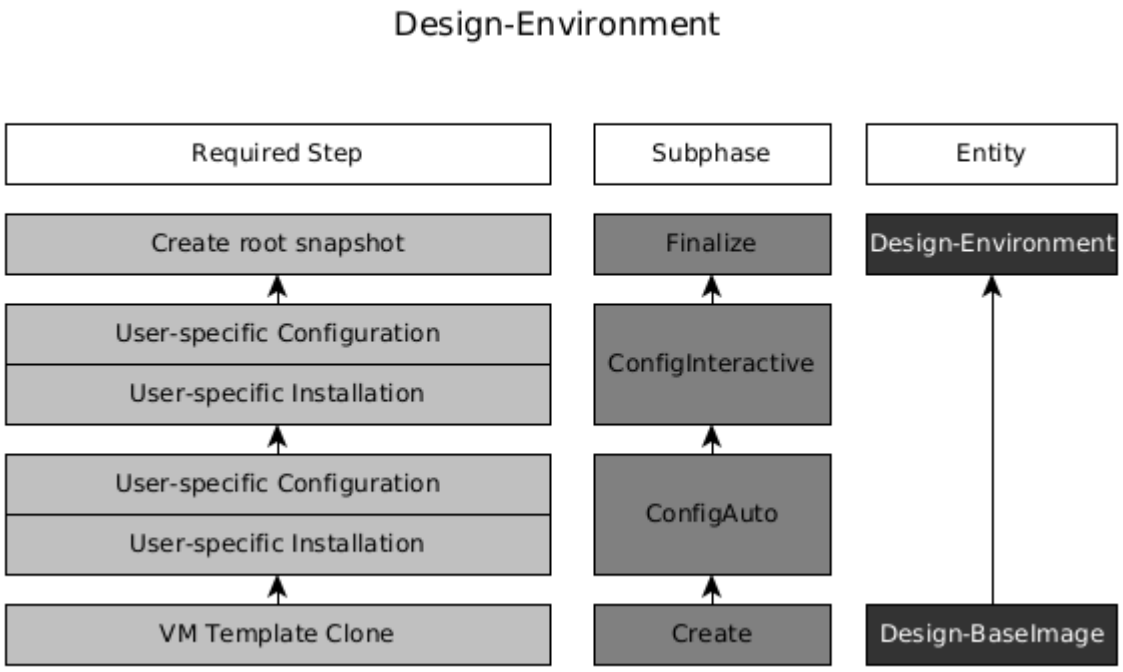
DESIGN-BASEIMAGE

A Design-Baseimage is the first stage a usable object can be in. It offers possibilities to store metadata with respect to the installation and configuration of the object and must at least contain means of basic interaction between the instance and the Backend-Api in order to provide a rudimentary access for further configurations and installations. In the current state this includes at least a properly configured ssh-server including a list of authorized keys for authentication. Further a specific IP-address has to be assigned. Currently it is assumed that this ip never changes and is uniquely tied to the underlying vm identifier or container identifier. The phase as a whole is therefore meant to be used to install the operating system, to include the basic communication infrastructure as well as for organizational configuration (e.g. Windows Active Directory, ACL's, Usernames, Hostnames, etc). In the case of a vm it is represented as template with fast clone times and efficient usage of storage. In the case of a container it might be represented as regular container image. An admin user might append organizational configuration and installation and provide such prepared images to other users.



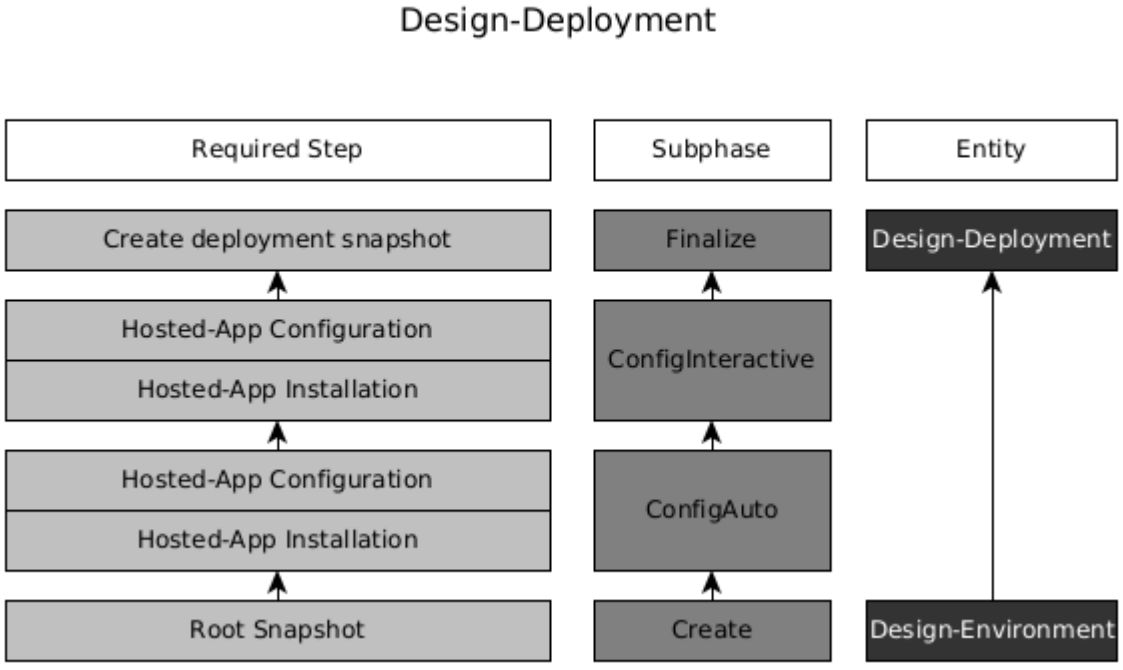
DESIGN-ENVIRONMENT

A Design-Environment is derived from a Design-Baseimage. As a result the environment initially inherits properties from the Baseimage but can also be updated to reflect different configurations (Network connection, Ip, Hostname, mapped storage, etc.) The environment therefore always contains a proper communication infrastructure and potentially organizational software and configurations. In the current stage no required steps have to be taken but users can optionally customize their environments as a preconfigured base for one or multiple deployments. Hence, the phase is meant to be used to configure for example user credentials (Windows-Live, Git, Confluence, etc.). By our own constraints the limit of environments per user is currently always one but from a technical perspective more would be possible if sufficient hardware resources are available. In the case of a vm it is represented as regular vm with at least one snapshot to offer user customizations. In the case of a container it might be represented as staged image on top of a container-based Design-Baseimage. An user might append custom configuration and required services as a common base for deployments.



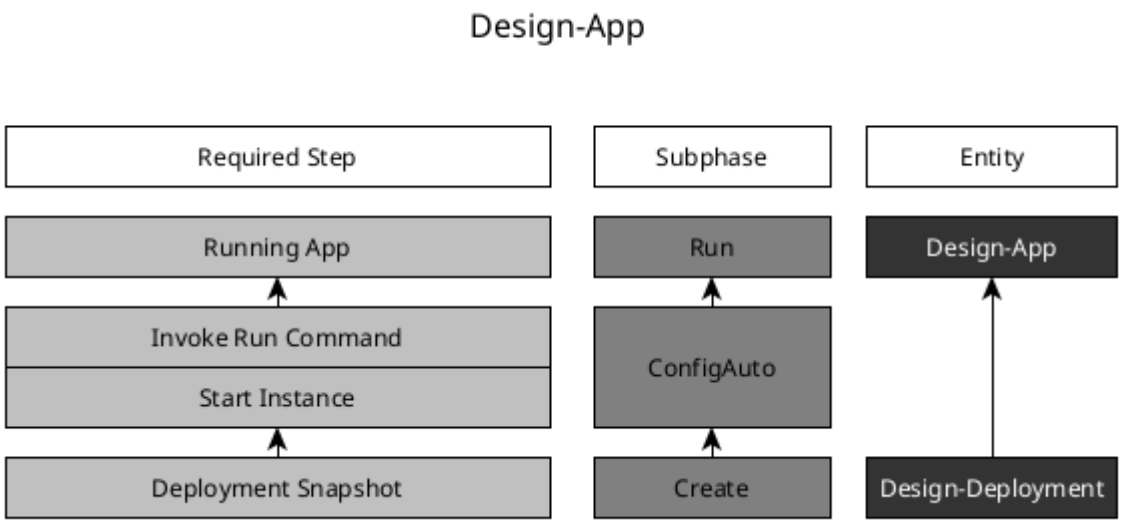
DESIGN-DEPLOYMENT

A Design-Deployment is derived from a Design-Environment and inherits it's properties initially. The deployment contains all relevant communication infrastructure, all organizational configurations as well as the user related customizations. Besides that, the deployment enables to install and preconfigure applications being hosted. The only thing missing for an deployment to run is the actual application being hosted. Hence, the phase is meant to be used to install and configure the hosted app only(e.g. Mircosoft Office Suite). In the case of a vm this might involve the utilization of live snapshots containing state of a running operating system and application but it can also be an offline snapshot with longer loading time but more efficient utilization of data storage. In the case of a container this might be represented as staged image on top of a container-based Design-Environment. Any finalized deployment can be started and stopped in a lightweight manner and does not involve additional steps prior to their start.



DESIGN-APP

A Design-App is derived from a Design-Deployment and inherits it's properties and state initially. Similarly to Design-Deployments the Design-App contains all relevant installations and configurations and is ready-to-use with respect to it's instantiation and can therefore be started, restarted and stopped to any point in time. The phase is meant to be used on top of a finalized Design-Deployment and contains a preconfigured command to run the hosted app. Cloning such a prepared Design-App might additionally be possible from a technical point of view but is undesirable as users are currently constraint to posses only one environment. Hint: In the case of a vm this would also take an inappropriate amount of time and in most cases cloning from a Design-Baseimage into a Design-Environment would be much faster if the configuration for that environment is non-interactive. In the best case the difference would be a few seconds for cloning from the template in contrast to a couple of minutes for cloning from a regular vm of the same size. For containers a similar approach is generally not as resource-consuming and often desirable by users. The solution might therefore allow clones of container-based Design-Apps.



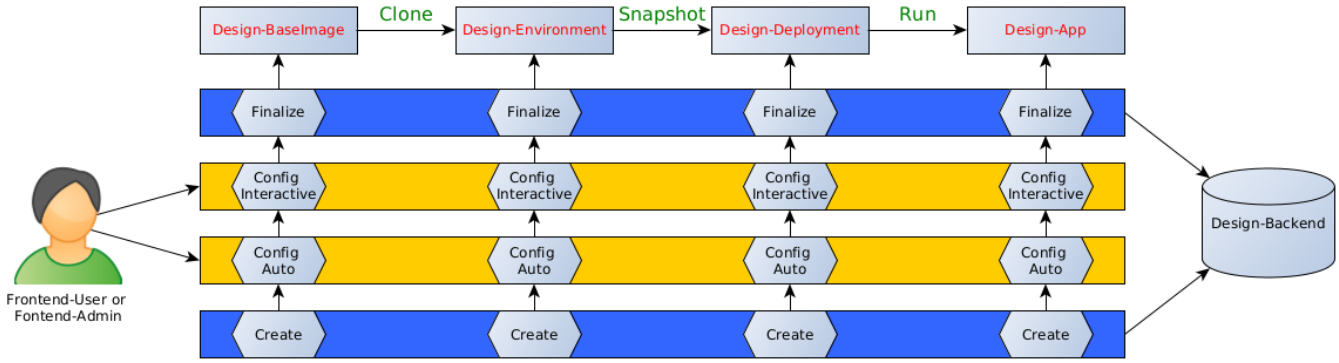
Instantiation Time

By following the described principles a running app can always be instantiated from scratch in only 3 major steps. The numbers mentioned in the following are not very scientific yet! These numbers should just be seen as a rule of thumb for now as no other data is available currently! The chosen setup follows the best practices given by proxmox and involves a virtio-stack for at least storage, RAM and network drivers. Link: https://pve.proxmox.com/wiki/Windows_10_guest_best_practices The approximated amounts might differ drastically if no virtio drivers are used!

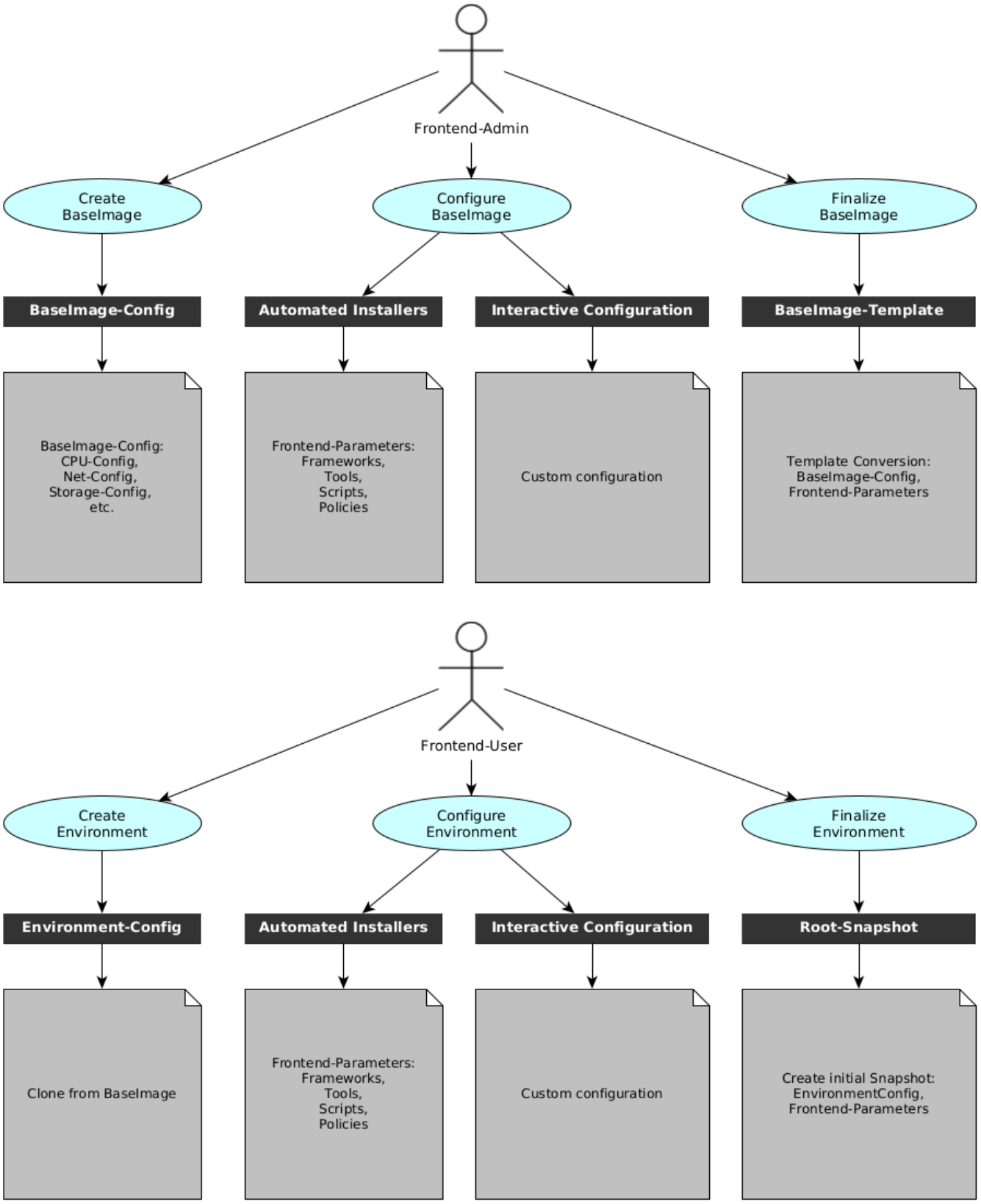
Approximated time for a full usecase: A very rough approximation for a fully automated approach would be that a simple app might completely be installed, configured and created 'ready-to-use' in under one minute. If the app was already prepared before

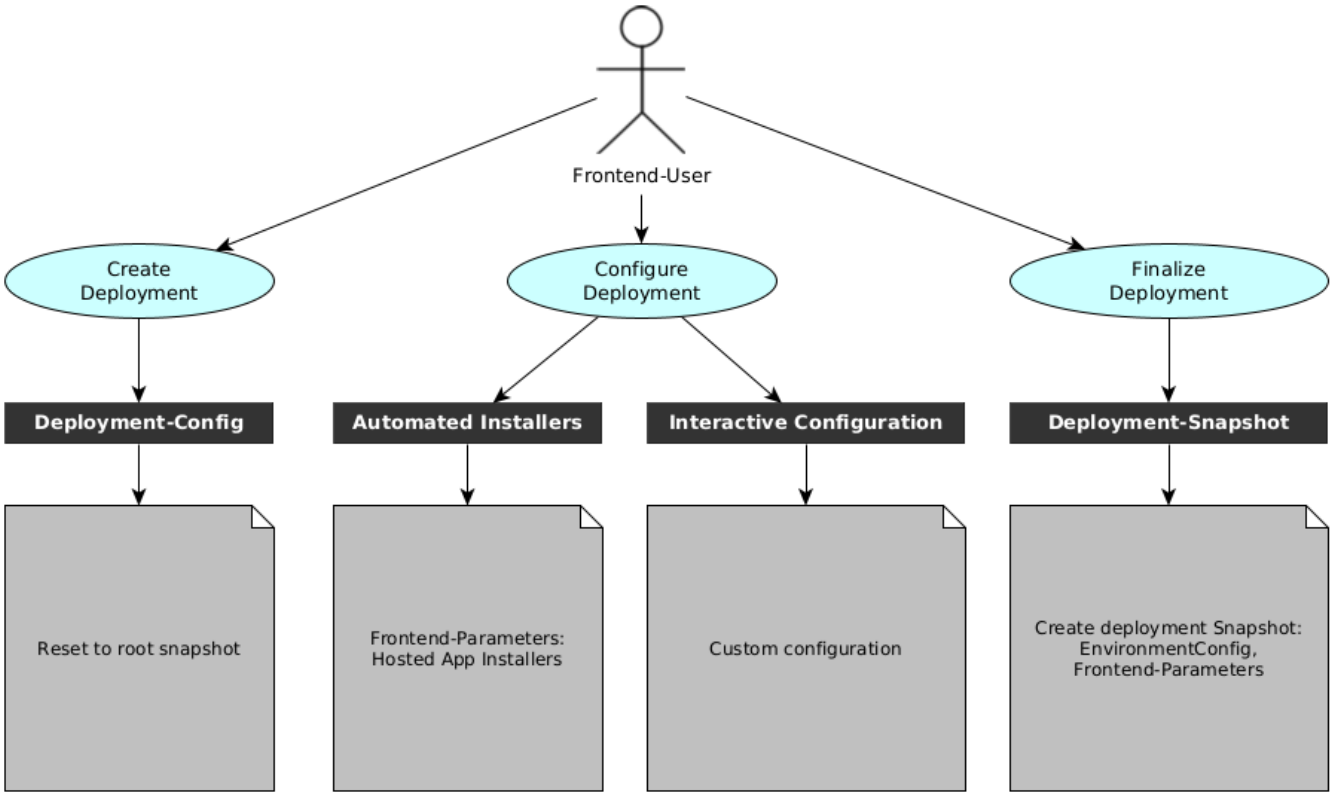
(Baseimage+Environment+Deployment), than the time to instantiate might shrink down to under 10 seconds. This is achieved by following the aforementioned principles and the described phases and subphases.

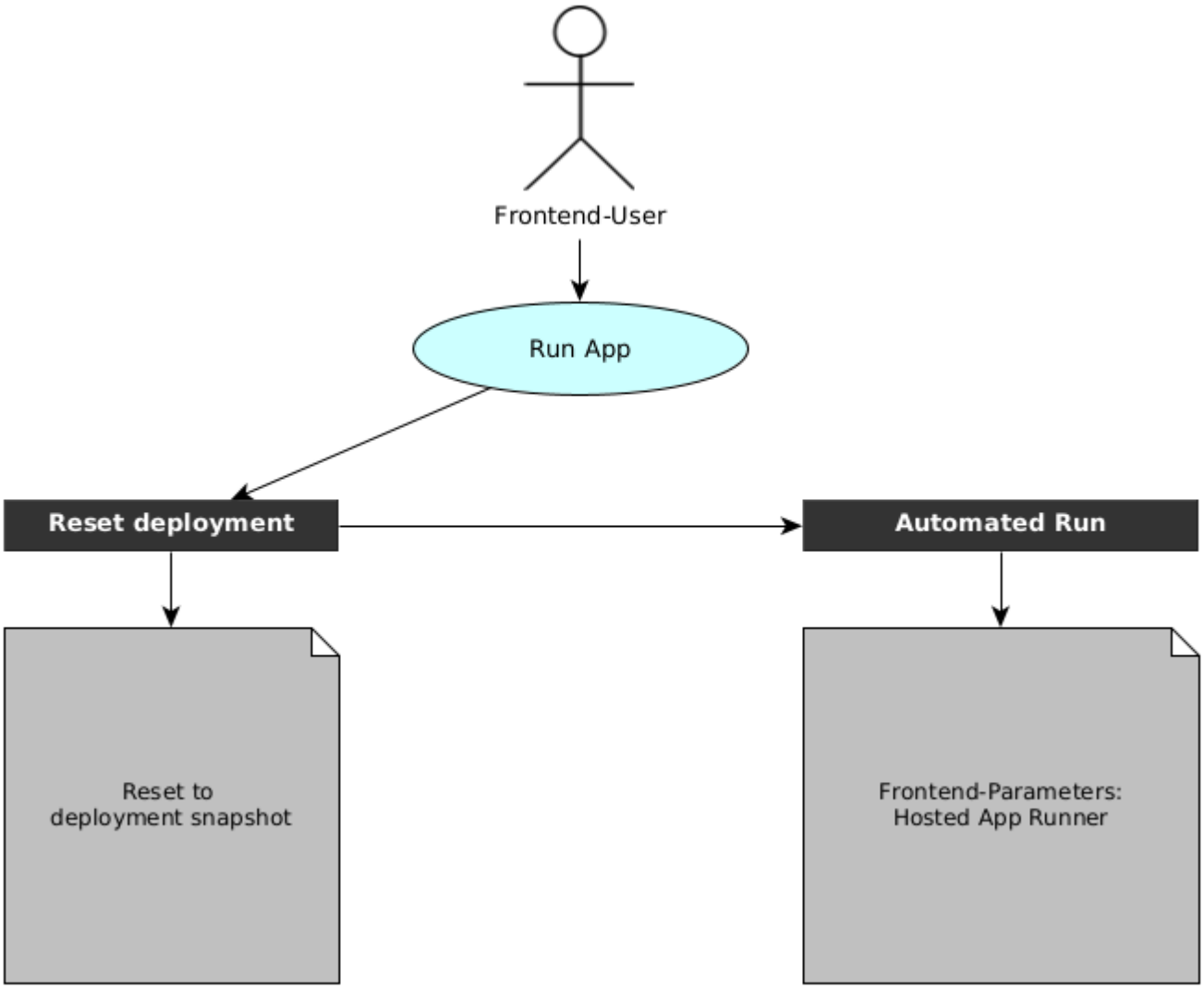
- Baseimage: By Cloning from Baseimages the most time consuming aspect of the whole process can be delegated into a preparation phase which does not affect instantiation time of Frontend-Users but only by Frontend-Admins and only once prior actual usage. Even if automated, the pure installation of Windows might take up to one or more hours, depending on the available internet connection and if a full-fledged windows updates has to be done. Nevertheless, on the current development hardware clon1ing of such an installed image can be done in approximately 1 to 5 seconds (64GB image size).
- Environment: By Creation of dedicated user environments the most time consuming aspect for Frontend-Users can be delegated into a configuration phase which is in the best case only executed once per application. Any Environment therefore contaions at least one root snapshot containing user specific installation and configuration. On the current development hardware the creation of this snapshot can be done in under a minute.
- Deployment: By instantiation of live snapshots the most time consuming aspect of booting the operating system can be delegated into a prior preparation phase. If the environment does not enforce interactive installation steps than the phase can be finished in a couple of seconds. For the snapshot the hosted app has to be installed and configured which can later be invoked through a Design-App. For a live snapshot the RAM and Storage have to be appended to the snapshot which creates a non-marginal overhead in disksize. On the current development hardware this can be done in a few seconds if the application does not enforce interactive installation steps. The test system required approximately 10 seconds for a live snapshot containing an additional 17GB disk which includes 8GB of RAM.
- App: As Apps are preconfigured Design-Deployments any app is by definition ready-to-use. An app can be based either on a live snapshot which contains the running hosted app and which can be instantiated in 3 to 5 seconds. Alternatively, the app can be based on an offline-snapshot where the hosted app is configured but not yet running as a process. Therefore offline-snapshots still need a preconfigured command-line in order to invoke the hosted app as soon as the operating system has successfully booted. On the current development hardware the time to boot into Windows10 requires approximately 30 seconds and the time to run the app is under 1 second.



2.1.9 Usecases

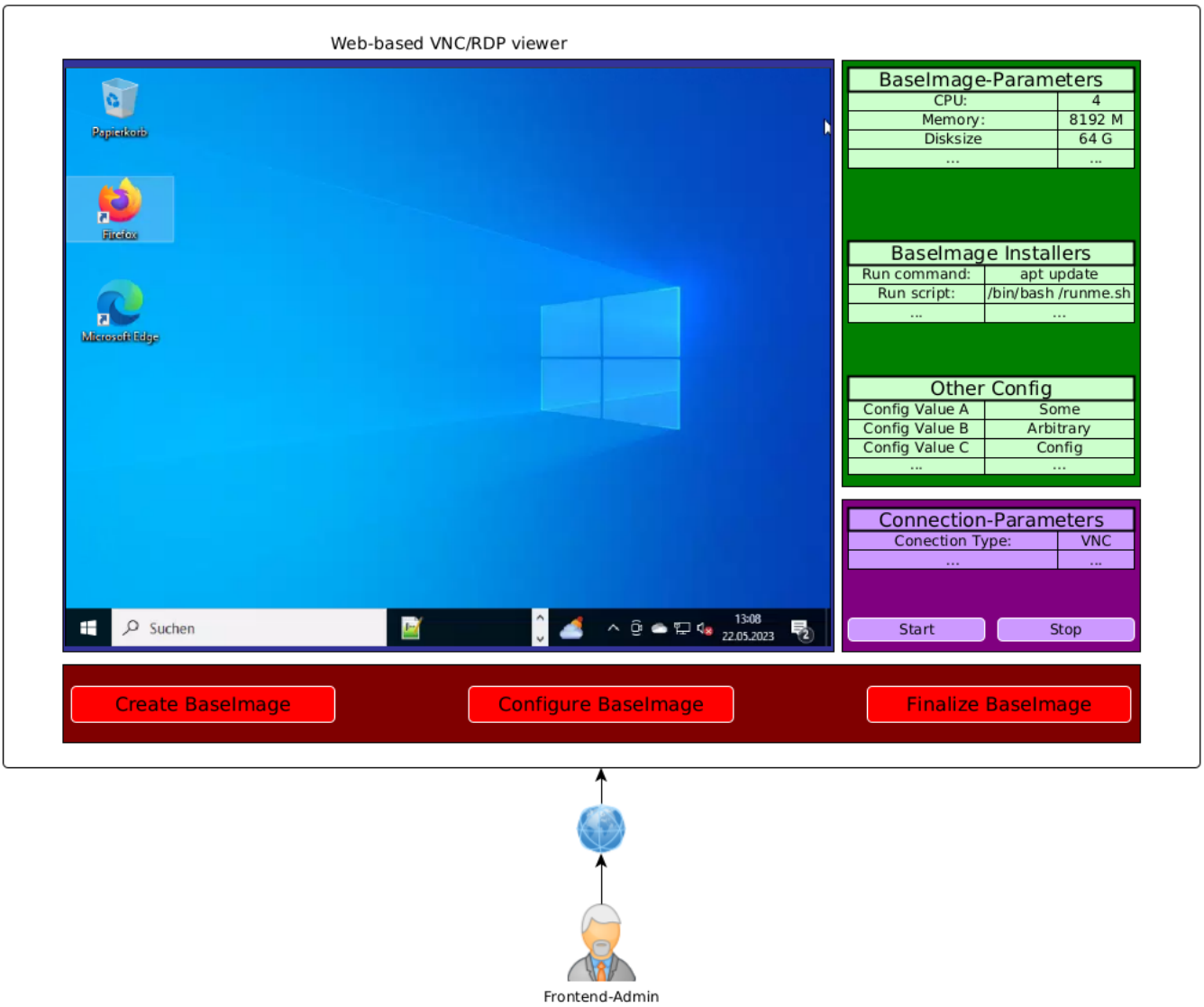



Frontend-User
Create Deployment
Configure Deployment
Finalize Deployment
Deployment-Config
Automated Installers
Interactive Configuration
Deployment-Snapshot
Reset to root snapshot
Frontend-Parameters: Hosted App Installers
Custom configuration
Create deployment Snapshot: EnvironmentConfig, Frontend-Parameters

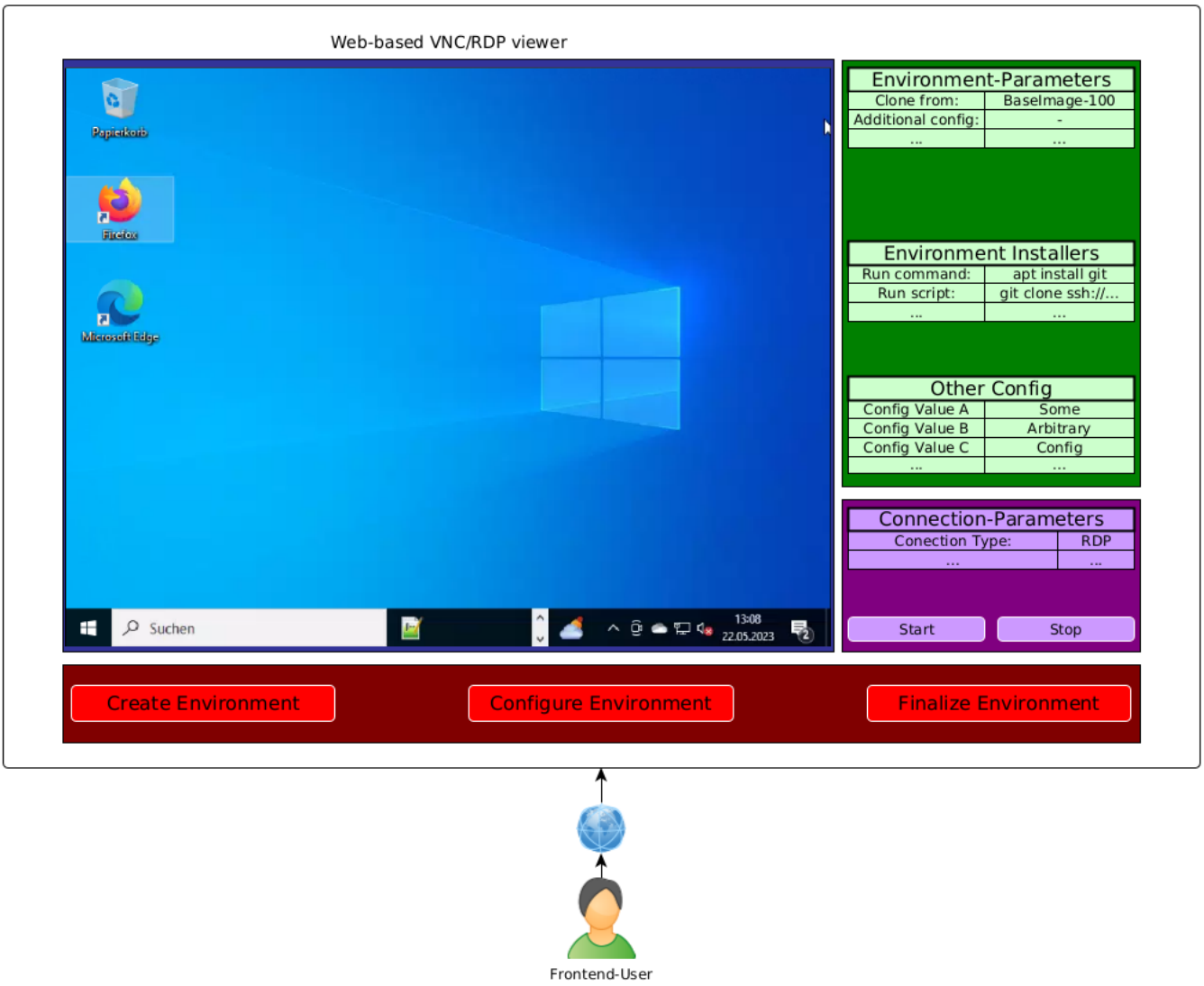

Frontend-User
Run App
Reset deployment
Automated Run
Reset to
deployment snapshot
Frontend-Parameters:
Hosted App Runner

2.1.10 Views

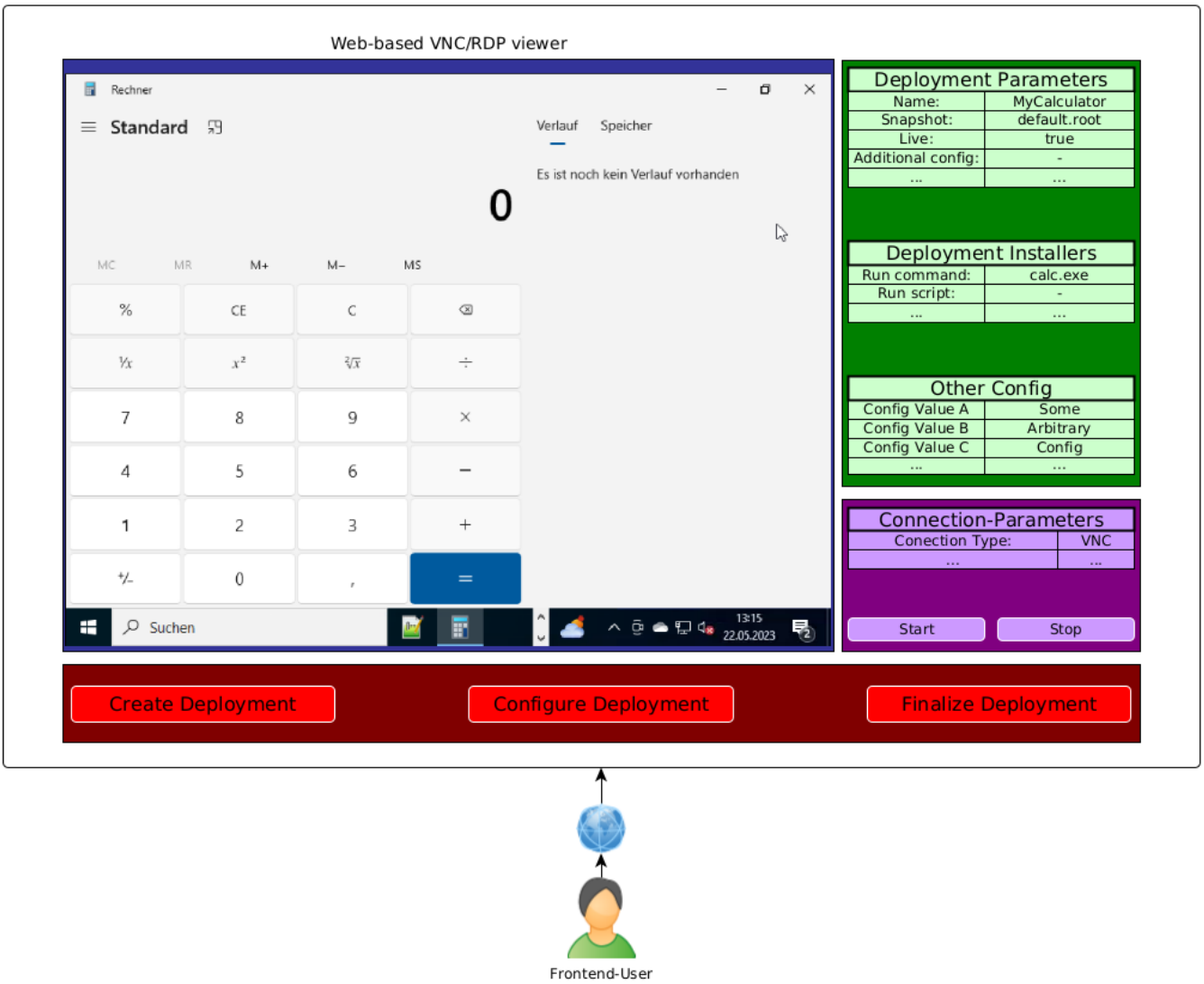
BaseImage View



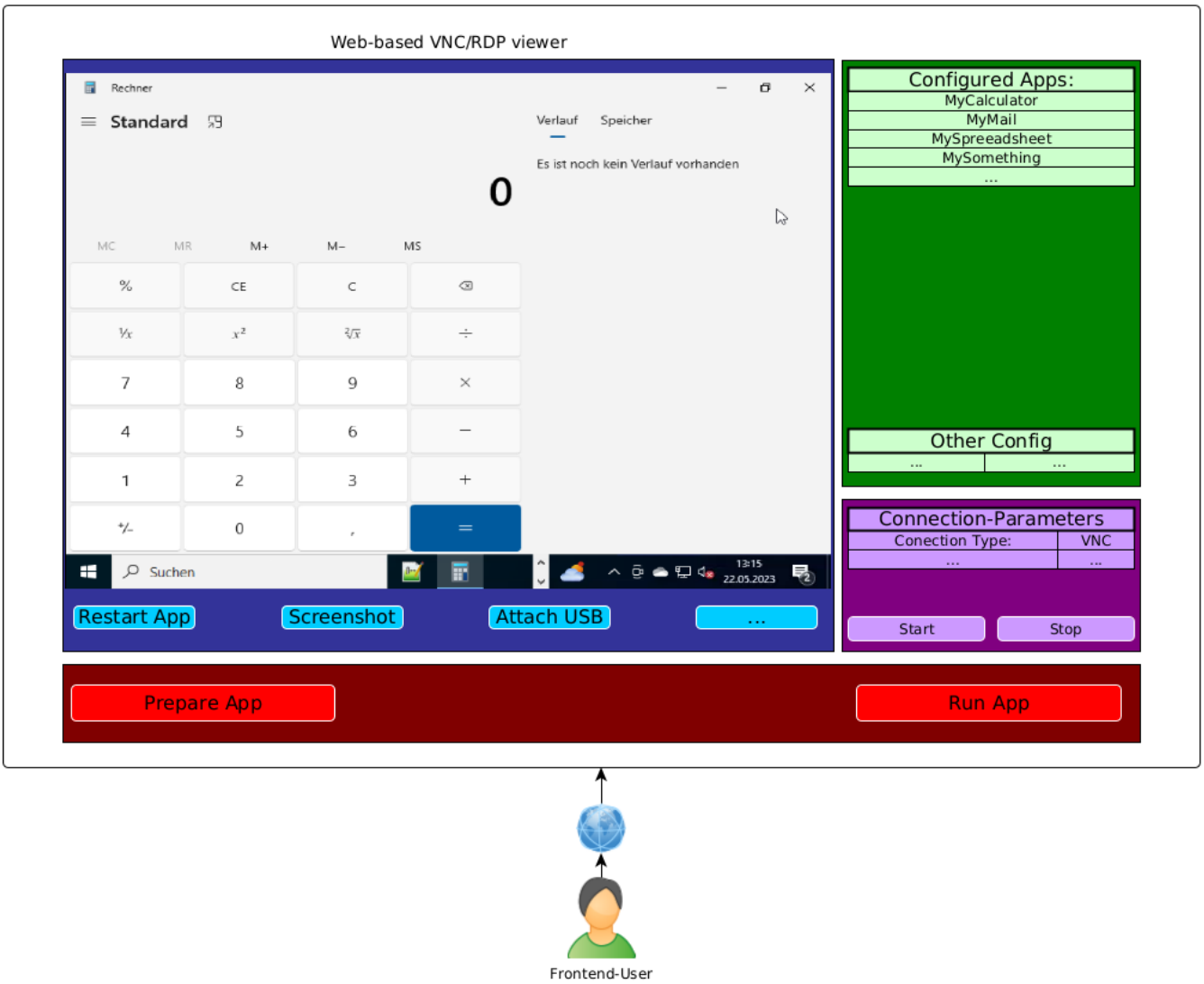
Environment View



Deployment View



App View


Web-based VNC/RDP viewer
Rechner
Standard
Verlauf
Speicher
Es ist noch kein Verlauf vorhanden
0
MC
MR
M+
M−
MS
%
CE
C
1/x
x²
²√x
÷
7
8
9
×
4
5
6
−
1
2
3
+
+/-
0
,
=
Suchen
13:15
22.05.2023
Restart App
Screenshot
Attach USB
...
Configured Apps:
MyCalculator
MyMail
MySpreeadsheet
MySomething
...
Other Config
...
...
Connection-Parameters
Conection Type:
VNC
...
...
Start
Stop
Prepare App
Run App
Frontend-User

3. Architecture

3.1 Architecture overview

The `qweb` application is a Python web application providing the following aspects:

- it provides a **minimal user interface** for demonstrating DaaS functionality
- it provides a **REST API** for browser-side code, e.g. for stopping a container
- it provides a **proxy** so that clients can directly view remote desktop contents in the browser, without having to manage their own VNC connections
- it manages the **application lifecycle** and stores necessary data in a **database**
- it contains **business logic** for managing **containers** and **VMs**

This document explains some of the more high-level architecture aspects in details.

3.1.1 Vision

This document describes the *status quo* of the system, but that is shaped by concerns about the future evolution.

In the end, this software should be able to manage containers and VMs and provide a Guacamole proxy for those instances:

```
graph LR
    Client --- Api[Management API] --> Docker & Proxmox;
    Client --- Proxy;
    Docker --- D1[instance] & D2[instance];
    Proxmox --- P1[instance] & P2[instance];
    D1 & D2 & P1 & P2 --- Guacd;
    Proxy ----- Guacd;
    subgraph daas_web
      Client; Api; Proxy
    end
    subgraph instances
      D1; D2; P1; P2
    end
```

This should eventually also support the lifecycle of *creating* new applications, but right now the focus is just on start/stop of application instances.

3.1.2 Web Application

The web application uses the Quart web framework. This is a lot like the popular `Flask` framework, but supports async operations – which we need for feeding the browser client via a WebSocket connection. Configuration is currently retrieved from environment variables or any plugin component, see the related module for details.

The routes are defined in the loaded plugin components as well. This contains all of the UI code and the REST API – maybe this will be spread across multiple modules in the future. Jinja templates are defined in `src/app/webroot/templates`. Assets (such as necessary JS code) are in `src/app/webroot/static`.

Most of the routes just figure out which data to retrieve from the DB and which class to invoke. Where access to external components is needed (Guacd, Docker, Database), these connections are mediated through the qweb environment.

All of the routes *must* be non-blocking. The application runs on a single thread. If one request blocks progress, other requests cannot continue. In particular, the VNC feed via WebSockets will be delayed.

3.1.3 Database

The database is made available through the `sqlalchemy` ORM component. This class represents a kind of repository or data access layer, isolating the rest of the code from DB-specific details. Currently, the system uses a MariaDB or SQLite database.

3.1.4 Proxy

See: Proxy Architecture

3.1.5 Business logic

Business logic involves juggling containers and VMs, applications and instances.

Concepts

Currently, the system uses three central concepts, as defined in the central [`src.app.daas.common.model`][]: applications, instances, and connections.

An **application** ([`DaasObject`][src.app.daas.model.DaasObject]) is configuration for something that can be started. It controls the lifecycle of its instances. There are subclasses for concrete scenarios, e.g. `ContainerApp` or `MachineApp`

A DaaS-Object is anything that can be started and then viewed by the user, but it might play the role of an environment/deployment/application depending on its state.

An **instance** ([`Instance`][src.app.daas.model.Instance]) is a running application with an associated connection. Technically, an `Instance` is created just before the corresponding resources are launched. This is necessary so that resources like port numbers can be allocated for that instance. Once actually started, the instance is updated with information like a Docker container ID.

A **connection** ([`Connection`][src.app.daas.model.GuacamoleConnection]) describes how the user can be shown the remote desktop contents. There are classes for specific types (e.g. [`GuacamoleConnection`][src.app.daas.common.model.GuacamoleConnection]), and they contain information such as connection passwords. Connection details are used by the proxy, but never exposed to the user.

Lifecycle

For containers, the application lifecycle is fairly simple: Apps can start instances, and then later stop them:

```
stateDiagram-v2
    [*] --> Instance: app.start()
    Instance --> [*]: app.stop(instance)
```

These state transition methods are responsible for effecting the actual change, and for tracking any necessary state in the database.

3.1.6 Limitations

See: Limitations

3.2 Proxy Architecture

The proxy components are needed to support the remote desktop connections of various connection types. However, there are significant differences depending on backend.

Historically, the Kasm proxy came first. It was implemented in order to test proxy support (including WebSockets) without also having to solve the Guacamole protocol problems at the same time. In the final deployed system, Kasm is not expected to be used. Instead, the focus is on the Guacamole proxy.

3.2.1 Guacamole Proxy

For Guacamole connections, any `/instance/*` request renders the guacamole-viewer template, which then establishes a WebSockets connection on same URL for transferring input events and screen contents. The details of the Guacamole WebSocket proxy are contained in the [`src.app.daas.proxy.guacamole_proxy`][] module, with the [`proxy_guacamole_ws()`] [src.app.daas.proxy.guacamole_proxy.proxy_guacamole_ws] function being the main entrypoint.

To illustrate this in context:

```
graph TD
    Client -- "Guacamole over WebSocket"  --- Proxy;
    Proxy -- "Guacamole over TCP" --- Guacd;
    Guacd -- "VNC" --- Container;
```

The proxy translates the streams of Guacamole protocol events in *both* directions, just changing the transport protocol.

This corresponds to the normal Guacamole system architecture, except that we have swapped out the Guacamole web front end for the `qweb` proxy. It is necessary to replace the Guacamole frontend so that connections can be created to the managed Docker containers without the users having to interact with details like connections. Users just start an application and then directly get a VNC screen.

In the Client → Guacd direction, the proxy has capabilities for additional filtering, which is needed for handling ping messages and screen resize events.

More involvement is needed during the protocol's handshake phase, where authentication and connection details have to be provided to Guacd. Details can be found in [`src.app.daas.proxy.handshake`][].

Utilities for the proxy are implemented in other modules:

- [`src.app.daas.proxy.syntax`][] – parser for the Guacamole protocol syntax.
- [`src.app.daas.proxy.streams`][] – abstraction over Guacamole streams, including both TCP and WebSocket support.

3.2.2 Chrome support

The demo has always worked under Firefox, but Chrome was refusing the WebSocket request. This is solved now.

Hypothesis 1: Security problems due to testing with `ws://` instead of `wss://` urls. This could be disproven experimentally.

Hypothesis 2: WebSocket subprotocols are causing problems. This failure mode could be demonstrated experimentally.

When the JS client creates a WebSocket, there is an optional subprotocol parameter:

```
new WebSocket("ws://...", "guacamole")
```

When a subprotocol is used, this causes a `Sec-WebSocket-Protocol: guacamole` header to be sent. Both Firefox and Chrome do this.

However, the server is supposed to confirm the subprotocol by sending it back. Our server didn't do this. Firefox ignored the response header, but Chrome does not (and fails with a completely nondescriptive message).

The solution is to explicitly invoke `websocket.accept(subprotocol="...")` in Quart.

Sources:

- WebSocket constructor on MDN
- `Sec-WebSocket-Protocol` on MDN
- `Sec-WebSocket-Protocol` in RFC 6455
- `WebSocket.accept()` method in Quart

3.3 Authentication

In order to authenticate requests from our frontend to the backend, a decentralized approach was chosen. We added a dedicated authentication component to our system architecture, which implements OAuth2 as it is one of the most popular industry-standard protocols for authorization. The component was implemented using the PHP programming language and is hosted as a dedicated and external service. The service offers a public and well-documented REST API and stores the necessary information to offer its authentication services in a local database.

3.3.1 Data Structure

The contained database generally divides users into the two major groups: "Users" and "Administrators". Each user entity provides at least the attributes id (unique), enabled, groups, email, name, and scopes. Additionally, each user or admin might be part of an arbitrary number of groups currently containing the two properties id (unique) and a description.

3.3.2 Access Methods

The system provides a well-documented public REST API to interact with the service. For all major entities, the appropriate GET, POST, and PATCH methods are provided. Furthermore, the system provides several auxiliary methods for testing and debugging, such as, for example, the "oauth2/user/session" method, which can be used to check the privileges of the currently logged-in user. To log in to the system, a user is required to send their credentials to the authentication system and receives back an OAuth token. With that token, a user can verify that they are currently logged in. The token has a default expiration timer, which can also be extended during its lifetime.

3.3.3 Current Verification Approach

Currently, only the authentication mechanism itself is integrated into our DaaS backend system. The backend system currently follows the approach of receiving an OAuth token from the frontend user, which has to be checked and verified against the authentication system. This can be done either by acting on behalf of the frontend user and supplying the frontend user's token to the "oauth2/user/session" method. Alternatively, a dedicated endpoint could be implemented, allowing a backend user to utilize their own legitimate session token and providing the frontend user's token only as a parameter. In this way, users would not be "impersonated" by the backend user. Therefore, the latter solution is generally preferable.

3.3.4 Privilege System

As described before, currently only the approach to verify logged-in users is implemented. Additionally, it is planned to provide a more granular method to grant or deny access to the system on a functional level. With that in mind, the overall aim of such a solution should be on one hand to minimize the additional integration efforts needed to adjust the database layout or endpoint specifications. On the other hand, the solution should aim to maximize a wide range of possible configuration use cases in order to provide convenient and appropriate management efforts when handling large amounts of groups and users. In any case, the decision for a specific approach will ultimately also influence the performance of the whole system when taking into account the number of requests needed to verify a particular user action. In the following, a draft version is described, which aims to optimize all three stated problem dimensions.

Prerequisites

In order to provide a privilege-based verification on a functional level, the initial step would be to assign a list of unique identifiers to each method callable by system users. This list of currently 71 unique identifiers is the ground truth information on which backend and authentication component agree upon initially. The endpoint privileges must then be distributed over plausible user and group entities in the form of static integer lists for both acceptance and denial of a particular right. Further, an appropriate endpoint to intially request all static groups at once is required. In a debugging context it might make sense to additionally provide an endpoint to set all groups and users at once.

Privilege Hierarchy

Based on this information, individual accept privileges can then initially be stored within each user entity in the authentication system to allow specific actions. With the aim to also allow the denial of a specific privilege, the user entity might additionally store a list of denied privileges. In order to provide a more convenient way to handle large user amounts, such accept and deny lists can also be added to all group entities within the authentication system. As a further optimization, certain groups can also be marked as "static" by using a dedicated property within a group entity. This property is aimed to mark groups that do not change too frequently and allows the backend to cache contents of such groups. In total, three major rules are required to reflect such constraints:

- By storing accept and deny lists within each user entity, arbitrary privilege use cases can be handled on a user-specific level.
- By storing accept and deny lists within each group entity, arbitrary privilege use cases can be handled on a group-specific level.
- By caching such groups, the total number of requests needed to verify a particular action can be further reduced.

Privilege Precedence

As privileges might be part of user or group specifications and since privileges are also deniable, a certain precedence or evaluation order has to be defined. The precedence is therefore defined as:

- Level 1.1: User Accept Rules
- Level 1.2: User Deny Rules
- Level 2.1: Group Accept Rules (Static)
- Level 2.2: Group Deny Rules (Static)
- Level 3.1: Group Accept Rules
- Level 3.2: Group Deny Rules

Verification Requests

Combining all mentioned requirements, an approach to verify specific user actions can therefore start by evaluating all user-specific access and deny lists. If the request cannot be verified by accounting that information, the authenticator can advance to the next level and account all static group-specific access and deny lists. If the request cannot be verified by accounting that information, the authenticator can advance to the next level and account all non-static group-specific access and deny lists.

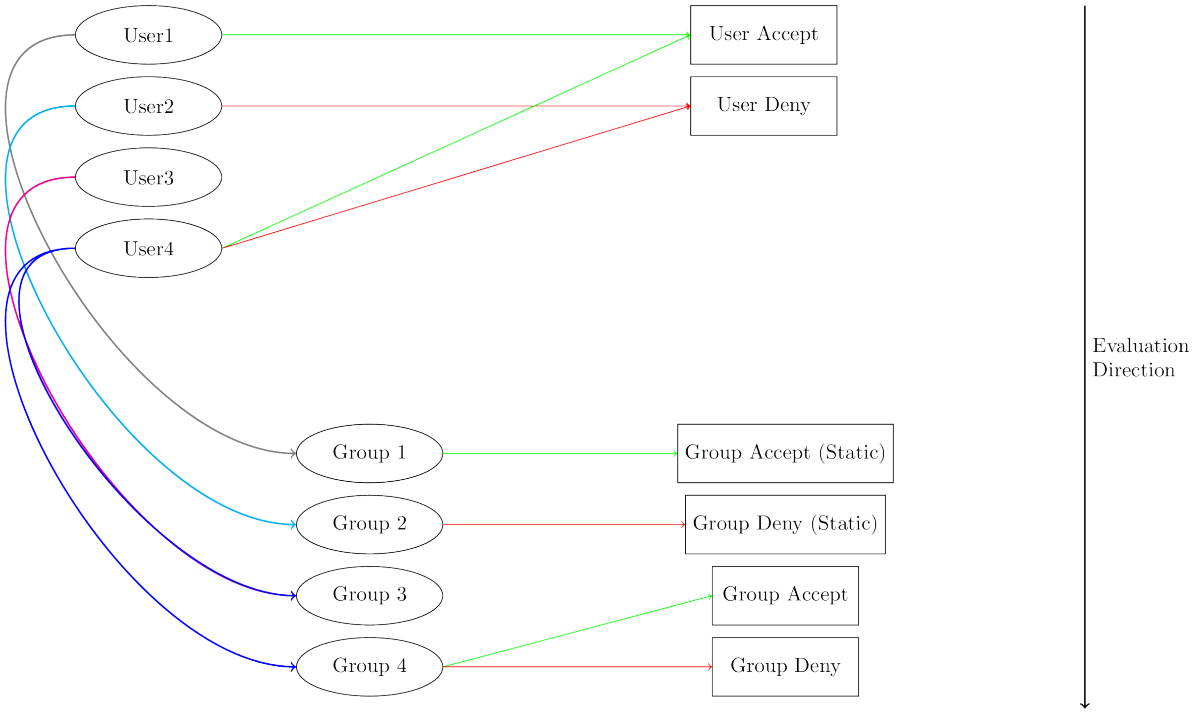
- If the request can be verified in the first step, then only one request is needed in total to fetch all information.
- If the request can be verified in the second step, then no additional request is needed to fetch all information as they are cached by the backend system.
- If the request can only be verified in the third step, then one request per assigned non-static user group is needed in total to fetch all information.
- The algorithm ommits non-static group requests if a static deny rule is found immidiately (higher precedence of deny rules).
- If the request cannot be verified in the third step, the overall result of the verification process is negative.

3.3.5 Essential Verification Requests

Depending on the actual structure of the user and group privileges, it might be possible, in an optimal case, to verify actions by only issuing one request to the backend (User request only). In a worst-case scenario, each non-static group specification has to be requested individually. The total number of requests needed is then 1 + n, where n is the number of non-static group memberships. It is therefore highly advisable to keep most group specifications in static groups. The number of non-static groups should be kept as small as possible. Static groups should usually be used to define regular users. User privileges are meant to be used to temporarily overwrite the default group privileges. Static groups with active deny rules can be used to reflect longer lasting privilege revocations.

\newpage

Privilege Hierarchy

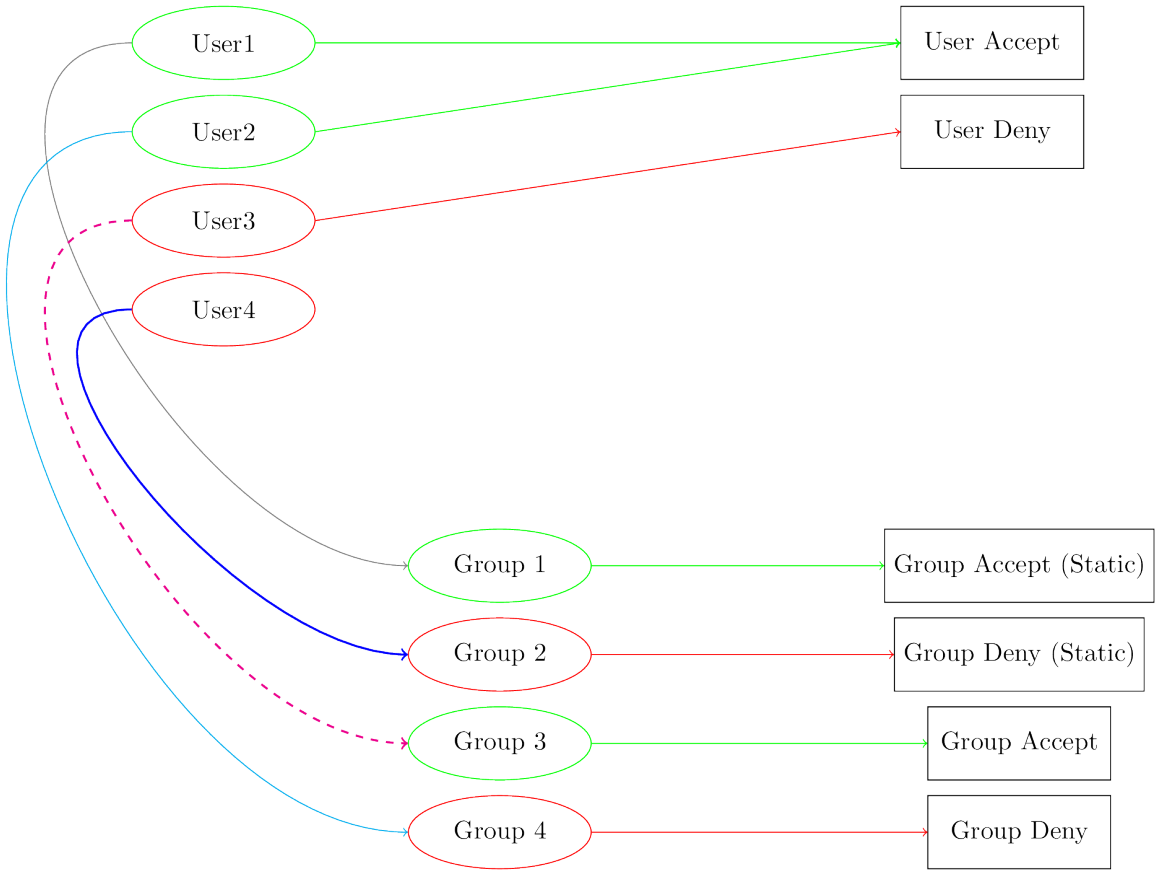


Users and groups:

- Users might have access or deny privileges at once (Deny over Accept)
- Groups might have access or deny privileges at once (Deny over Accept)
- Users can be have 0 to many group memberships
- Privileges are evaluated from top (users) to bottom (static first, then non-static groups)

\newpage

Best Case Scenarios

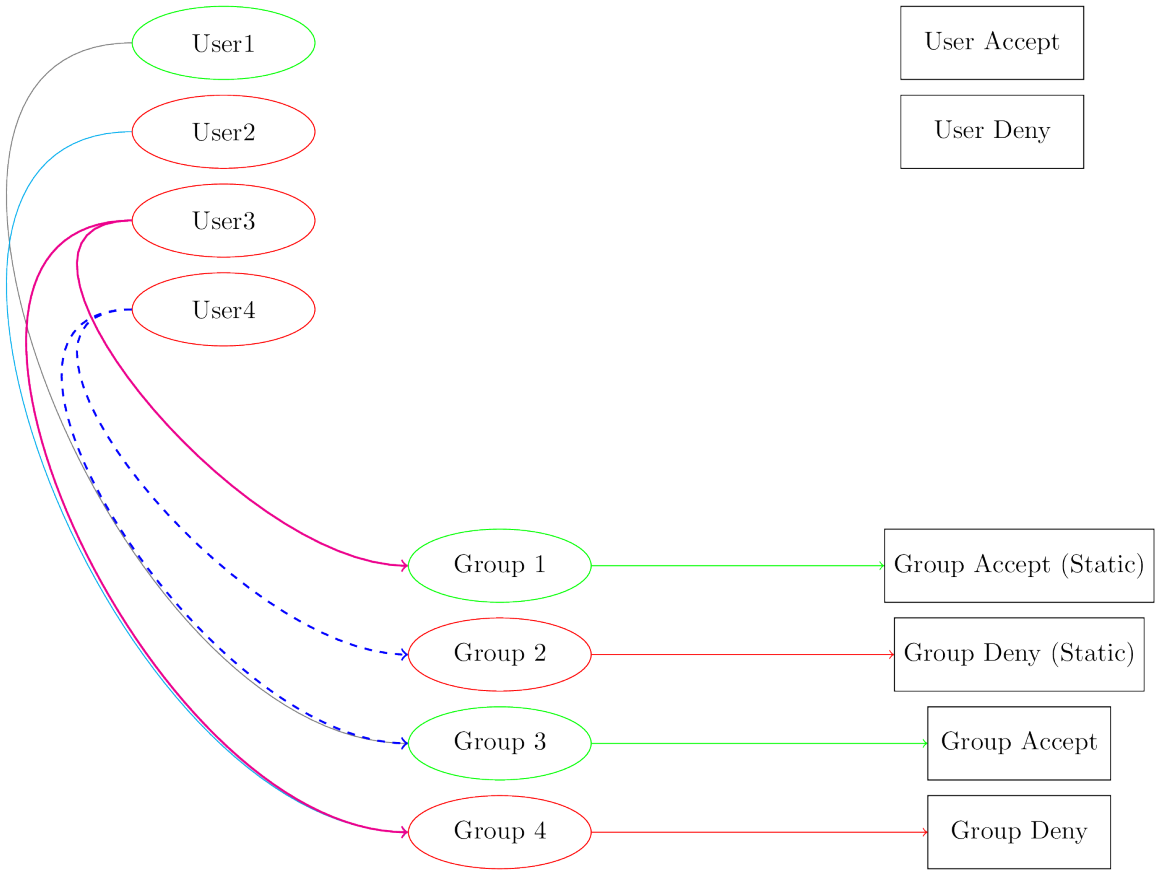


All actions need only one request to verify:

- User1 is accepted by user privilege
- User2 is accepted by user privilege (User over group)
- User3 is denied by user privilege (User over group)
- User4 is denied by static group privilege

\newpage

Worst Case Scenarios



Three worst case scenarios and one optimal case:

- User1 is accepted by non-static group privilege
- User2 is denied by non-static group privilege
- User3 is denied by non-static group privilege (Deny over Accept)
- User4 is denied by static group privilege (No worst case!)

3.4 Limitations of the architecture

This project is mostly prototype-level code, so there are lots of limitations. However, there are clear paths to solve them, if that should become necessary.

3.4.1 Features

Lots of features intended to be part of the final DESIGN architecture are not yet available. Importantly:

- no user management
- no VM support (being worked on)
- no application lifecycle (environments, deployments, applications)

3.4.2 Scalability and Performance

The main workload in the DaaS system is in the actual containers and VMs running applications. Compared to that, the REST API requires negligible resources. Thus, scalability arguments are likely to be premature.

However, there are two potential problem areas in this architecture:

- the database is embedded
- the proxy component is on a performance-critical path

3.4.3 Database limitations

The current database uses SQLite. This might be awkward to work with. It also means that the database engine runs in-process, precluding horizontal scaling. Another consequence is that if other components want to store persistent data, APIs for this must be exposed – external access to the DB is not possible.

Having to write raw SQL can be avoided by switching to an ORM or query builder like Python's SQLAlchemy.

Scalability concerns could be addressed by switching to a different database. To do that, the SQLite-specific details would have to be removed from the database class, and swapped for a different database connector.

Careful: any replacements of the database must also support async operations in order to avoid locking up the proxy.

If any changes to the database have to be made, the good news is that only the database module has to be touched. No other code contains database-specific operations.

3.4.4 Proxy limitations

The proxy has limitations along two directions: the WebSocket connection relies on asynchronous operations for low latencies, and the Guacamole proxy must fully parse each individual message.

3.4.5 Proxy must be asynchronous

It is unavoidable that the proxy component must be asynchronous: the server must push events such as screen updates to the client. Since the client is browser-based, this must be a web protocol. The only available protocol for that is WebSocket.

Writing WebSocket-capable servers is more difficult than an ordinary HTTP server, since it must be event-based, must be asynchronous. In Python, this requires a web framework that builds upon the ASGI ecosystem. Here, we've picked Quart.

The limitation in this context is that `async` code is infectious. In order for the async proxy to behave as expected, there must not be synchronous code blocking the event loop. In particular, all requests to external resources (Docker, Proxmox, databases) must be performed on a background thread. Quart offers the `run_sync()` utility for this purpose, but it's easy to forget.

There's also the problem that Guacamole proxy events are competing with other work in the event loop. If too much other work is happening, performance of the proxy might degrade, potentially leading to stuttering or other latency effects.

If this becomes a problem, the proxy and the management-API components have to be split up into separate servers. To do this, the management server would have to provide internal APIs from which the proxy server can retrieve credentials etc.

3.4.6 Guacamole message parsing

The Guacamole protocol was designed to be easy to parse by high-level languages like JavaScript and Java. But alas, Unicode makes everything more complex, and it is definitely not easy to handle correctly. While the protocol uses the sensible `LENGHT CONTENT` Hollerith-String encoding, the length is given in terms of Unicode codepoints, not in terms of bytes. Also, this encoding only relates to individual fields, but each message is made up of an indeterminate number of fields.

The proxy cannot just forward the stream of messages, but must parse each one of them:

- For the handshake, the proxy must participate in the protocol and read messages.
- The JS client has a weak assumption that it will only receive complete messages, thus messages from Guacd should be buffered until they can be forwarded in a single WebSocket frame.
- The JavaScript client can send `ping` messages. This ping command is an internal protocol extension of the JS client, and not handled by the Guacd component. Thus, our proxy must respond to the ping messages on its own.
- Other messages might have to be intercepted as well, e.g. to handle `size` messages which Guacd doesn't implement for VNC connections.

So, all messages have to be parsed.

Due to the protocol complexity, this requires byte-by-byte parsing. The Python code for this is absolutely not efficient. Maybe it can be optimized, maybe it can be rewritten to be JIT-compiled with Numba, maybe that code has to be replaced with a native component (e.g. using PyO3).

4. Configuration options

The web app and Proxmox adapters are configured via the `*.toml` files in the `src.config` folder. The corresponding code is in `src.app` and `src.scripts`.

4.1 Config file location

Configuration must be placed in a `*.toml` file in the config directory. There is a `*.toml` file for each component that can be adapted.

4.2 Config file syntax

This file uses the TOML format, which provides an INI-style syntax for a JSON-style data model. Here, the advantage is that the config can contain comments, and is neatly separated into multiple sections.

Example snippet:

```
# can have comments
[web]
database = "/some/path"
```

This corresponds to the JSON data structure:

```
{
  "web": {
    "database": "/some/path"
  }
}
```

4.3 Config file schema

The schema of the config files is validated, and unsupported keys will raise an error (to prevent typos).

5. Rest API

This document provides an overview of all routes.

Technically, most of these routes are not REST APIs.

5.1 Browser routes

Some routes provide HTML responses and are intended to be navigated to with the browser.

5.1.1 GET `/`

The user interface for listing, starting, and stopping apps/instances.

5.1.2 POST `/app/:id`

Creates a new instance of that application. It redirects to the instance URL.

5.1.3 GET `/instance/:id`

Shows a viewer for the instance.

5.1.4 GET `/login`

5.1.5 POST `/login`

Form for logging in. Other routes redirect to this endpoint if they require authentication. Upon successful authentication, this route redirects back.

5.1.6 POST `/logout`

Remove log-in information from the session cookie, logging the user out.

5.2 API routes

These routes are intended to be invoked via JS client code, e.g. via `fetch()`.

5.2.1 WS `/instance/:id`

Provide a websocket for the Guacamole proxy. Available if the instance uses a Guacamole connection.

5.2.2 DELETE `/instance/:id`

Stop the instance.

6. Internal API

6.1 src

6.1.1 app

daas

ADAPTER

`src.app.daas.adapter.adapter_http` module

A adapter to fetch Http requests

Classes

class `HttpAdapterConfig(verify_tls: bool, logging: bool, logrequests: bool, logresults: bool)` `dataclass`

HttpAdapterConfig

Attributes

attr `verify_tls: bool` `instance-attribute`

attr `logging: bool` `instance-attribute`

attr `logrequests: bool` `instance-attribute`

attr `logresults: bool` `instance-attribute`

Functions

class `HttpAdapter(cfg: HttpAdapterConfig)`

Bases: `Loggable`

Http adapter to retrieve local requests

Source code in `src/app/daas/adapter/adapter_http.py`

```
28  def __init__(self, cfg: HttpAdapterConfig) -> None:
29      Loggable.__init__(self, LogTarget.HTTP)
30      self.config = cfg
31      if self.config.verify_tls is False:
32          urllib3.disable_warnings()
```

Attributes

attr `config = cfg` `instance-attribute`

Functions

meth `request(method: str, url: str, header: dict[str, str] = {}, cookies: dict[str, str] = {}, params: dict[str, str] = {}, data: dict[str, Any] = {}, files: dict[str, Any] = {}) -> tuple[aiohttp.ClientResponse, dict]` `async`

Perform request with specified parameters

Source code in `src/app/daas/adapter/adapter_http.py`

```
35  async def request(
36      self,
37      method: str,
38      url: str,
39      header: dict[str, str] = {},
40      cookies: dict[str, str] = {},
41      params: dict[str, str] = {},
42      data: dict[str, Any] = {},
43      files: dict[str, Any] = {},
44  ) -> tuple[aiohttp.ClientResponse, dict]:
45      """
46      Perform request with specified parameters
47      """
48      encoded_data: dict[str, Any] | str = data
49      if "Content-Type" in header:
50          if header["Content-Type"] == "application/json":
51              encoded_data = json_parser.dumps(data)
52      if await self.__check_request_method(method):
53          args = {
54              "method": method.upper(),
55              "url": url,
56              "params": params,
57          }
58          if method.lower() != "get":
59              args.update(data=encoded_data)
60          return await self._request_aio(method, url, header, args, cookies, files)
61      self._log_error(f"Method {method} not known for {url}")
62      raise ValueError(f"Specified method was not known: '{method.upper()}'")
```

meth `__check_request_method(method: str) -> bool` async

Source code in `src/app/daas/adapter/adapter_http.py`

```
95  async def __check_request_method(self, method: str) -> bool:
96      if method.lower() in [
97          "get",
98          "post",
99          "put",
100         "delete",
101         "patch",
102         "options",
103         "head",
104     ]:
105         return True
106     return False
```

meth `__print_response(method: str, url: str, response) -> None` async

Source code in `src/app/daas/adapter/adapter_http.py`

```
108 async def __print_response(self, method: str, url: str, response) -> None:
109     if response is not None:
110         msg = ""
111         if self.config.logrequests:
112             msg = f"{method.upper():>6} {url}"
113         if self.config.logresults and response.status == 200:
114             body = json_parser.dumps(await response.json(), indent=1)
115             msg = f"{msg}\n{body}"
116         if msg != "":
117             self._log_info(msg, response.status)
```

`src.app.daas.adapter.adapter_ssh` module

Classes

class
`SshAdapterConfig(sshuser: str, sshhost: str, sshport: int, sshopt_timeout: int, sshopt_strictcheck: str, sshopt_nohostauth: str, logcmd: bool, logresult: bool)` `dataclass`

Config for SshAdapter

Attributes

attr `sshuser: str` `instance-attribute`

attr `sshhost: str` `instance-attribute`

attr `sshport: int` `instance-attribute`

attr `sshopt_timeout: int` `instance-attribute`

attr `sshopt_strictcheck: str` `instance-attribute`

attr `sshopt_nohostauth: str` `instance-attribute`

attr `logcmd: bool` `instance-attribute`

attr `logresult: bool` `instance-attribute`

Functions

class `SshAdapter(cfg: SshAdapterConfig)`

Bases: `Loggable`

Adapter to perform ssh requests

Source code in `src/app/daas/adapter/adapter_ssh.py`

```
25  def __init__(self, cfg: SshAdapterConfig):
26      Loggable.__init__(self, LogTarget.SSH)
27      self.config = cfg
```

Attributes

attr `config = cfg` `instance-attribute`

Functions

meth `__str__()`

Source code in `src/app/daas/adapter/adapter_ssh.py`

```
29  def __str__(self):
30      return f"{self.config}"
```

meth `scp_upload_call(src: str, dst: str) -> tuple[int, str, str]`

Copies files via scp and given arguments

Source code in `src/app/daas/adapter/adapter_ssh.py`

```
32  def scp_upload_call(self, src: str, dst: str) -> tuple[int, str, str]:
33      """
34      Copies files via scp and given arguments
35      """
36      remote_folder = os.path.dirname(dst)
37      folder_cmd = f"mkdir -p {remote_folder}"
38      folder_created, folder_out, folder_err = self.ssh_call(folder_cmd)
39      if folder_created != 0:
40          return folder_created, folder_out, folder_err
41
42      args = f"{src} {self.config.sshuser}@{self.config.sshhost}:{dst}"
43      full_command = [
44          "scp",
45          "-o",
46          f"ConnectTimeout={self.config.sshopt_timeout}",
47          "-o",
48          f"StrictHostKeyChecking={self.config.sshopt_strictcheck}",
49          "-o",
50          "UserKnownHostsFile=/dev/null",
51          "-o",
52          f"NoHostAuthenticationForLocalhost={self.config.sshopt_nohostauth}",
53          "-o",
54          "LogLevel=quiet",
55          "-P",
56          f"{self.config.sshport}",
57          src,
58          f"{self.config.sshuser}@{self.config.sshhost}:{dst}",
59      ]
60      try:
61          process = subprocess.run(
62              full_command, capture_output=True, text=True, check=True
63          )
64          std_out = ""
65          std_err = ""
66          if process.stdout is not None:
67              std_out = str(process.stdout)
68          if process.stderr is not None:
69              std_err = str(process.stderr)
70          self._print_result(args, process.returncode, std_out, std_err)
71          return (process.returncode, std_out, std_err)
72      except Exception as ex:
73          self._log_error(f"{str(ex)} -> {args}", 1)
74          raise OSError("Ssh execution failed for unknown reason!") from ex
```

meth `ssh_call(args: str) -> tuple[int, str, str]`

Spawns a process with given arguments

Source code in `src/app/daas/adapter/adapter_ssh.py`

```python
def ssh_call(self, args: str) -> tuple[int, str, str]:
    """
    Spawns a process with given arguments
    """
    full_command = [
        "ssh",
        "-o",
        f"ConnectTimeout={self.config.sshopt_timeout}",
        "-o",
        f"StrictHostKeyChecking={self.config.sshopt_strictcheck}",
        "-o",
        "UserKnownHostsFile=/dev/null",
        "-o",
        f"NoHostAuthenticationForLocalhost={self.config.sshopt_nohostauth}",
        "-o",
        "LogLevel=quiet",
        "-p",
        f"{self.config.sshport}",
        f"{self.config.sshuser}@{self.config.sshhost}",
        args,
        "&",
    ]
    # self.__log_info(f"SSHARGS: {full_command}")
    # pylint: disable=broad-exception-caught
    try:
        process = subprocess.run(
            full_command,
            capture_output=True,
            text=True,
            check=True,
            shell=False,
            encoding="cp850",
        )

        std_out = ""
        std_err = ""
        if process.stdout is not None:
            std_out = str(process.stdout)
        if process.stderr is not None:
            std_err = str(process.stderr)
        if std_out.endswith("\n"):
            std_out = std_out[:-1]
        if std_err.endswith("\n"):
            std_err = std_err[:-1]
        self._print_result(args, process.returncode, std_out, std_err)
        return (process.returncode, std_out, std_err)
    except UnicodeDecodeError as exe:
        self._log_error(f"UnicodeDecodeError Exception raised: {exe}", -1)
        return -1, "", f"UnicodeDecodeError Exception raised: {exe}"

    except Exception as exe:
        hdr = "SSHERR"
        msg = f"{hdr:>6} {str(exe)} -> {args} {type(exe)}"
        self._log_error(msg, -1)
        return -1, "", f"Exception raised: {exe}"
```

AUTH

`src.app.daas.auth.auth_daas` module

DaaS authenticator base

Classes

class `AuthenticatorConfig(url: str, clientid: str, scope: str, username: str, password: str, verify_endpoints: bool, remote_verification: bool, logrequests: bool)` `dataclass`

Config for authenticator

Attributes

attr `url: str` `instance-attribute`

attr `clientid: str` `instance-attribute`

attr `scope: str` `instance-attribute`

attr `username: str` `instance-attribute`

attr `password: str` `instance-attribute`

attr `verify_endpoints: bool` `instance-attribute`

attr `remote_verification: bool` `instance-attribute`

attr `logrequests: bool` `instance-attribute`

Functions

class `AuthToken(refresh_token: str = '', access_token: str = '', time_valid: int = 0, created_at: datetime = datetime.now(), valid_until: datetime = datetime.now())` `dataclass`

A token

Attributes

attr `refresh_token: str = ''` `class-attribute` `instance-attribute`

attr `access_token: str = ''` `class-attribute` `instance-attribute`

attr `time_valid: int = 0` `class-attribute` `instance-attribute`

attr `created_at: datetime = datetime.now()` `class-attribute` `instance-attribute`

attr `valid_until: datetime = datetime.now()` `class-attribute` `instance-attribute`

Functions

class `DaasAuthenticatorBase(cfg_auth: QwebAuthenticatorConfig)`

Bases: `QwebAuthenticatorBase`

DaaS authenticator base

Source code in `src/app/daas/auth/auth_daas.py`

```
51  def __init__(self, cfg_auth: QwebAuthenticatorConfig):
52      self.cfg_auth = cfg_auth
53      self.logger = logging.getLogger(LogTarget.AUTH.value)
54      QwebAuthenticatorBase.__init__(self, cfg_auth)
```

Attributes

attr `config: QwebAuthenticatorConfig` `instance-attribute`

attr `cfg_auth = cfg_auth` `instance-attribute`

attr `logger = logging.getLogger(LogTarget.AUTH.value)` `instance-attribute`

Functions

meth `verify_entity_ownership(entity: Optional[Any], owner: int) -> bool` `async`

Tests if owner has permissions to access entity properties

Source code in `src/app/daas/auth/auth_daas.py`

```
56  async def verify_entity_ownership(self, entity: Optional[Any], owner: int) -> bool:
57      """Tests if owner has permissions to access entity properties"""
58      if self.cfg_auth.enable_entity_verification:
59          if isinstance(entity, DaasObject):
60              return await self.verify_object(entity, owner)
61          elif isinstance(entity, InstanceObject):
62              return await self.verify_instance(entity, owner)
63          elif isinstance(entity, Environment):
64              return await self.verify_environment(entity, owner)
65          elif isinstance(entity, File):
66              return await self.verify_file(entity, owner)
67          elif isinstance(entity, Application):
68              return await self.verify_app(entity, owner)
69          elif isinstance(entity, RessourceInfo):
70              return await self.verify_limit(entity, owner)
71          elif isinstance(entity, dict):
72              return await self.verify_dict(entity, owner)
73          elif isinstance(entity, list):
74              return await self.verify_list(entity, owner)
75          if self.config.enable_log_entity_verification:
76              self._log_info("Verifying ownership error: Invalid entity type")
77          return False
78      if self.config.enable_log_entity_verification:
79          self._log_info(f"Verifying ownership disabled: {True} for owner {owner}")
80      return True
```

meth `verify_limit(entity: RessourceInfo, owner: int) -> bool` async

Source code in `src/app/daas/auth/auth_daas.py`

```
82  async def verify_limit(self, entity: RessourceInfo, owner: int) -> bool:
83      testid = entity.id_owner
84      result = testid in (-1, 0, owner)
85      if self.config.enable_log_entity_verification:
86          self._log_info(
87              f"Verifying ownership ( lmt): {result} for {owner} and {entity}"
88          )
89      return result
```

meth `verify_list(entity: list, owner: int) -> bool` async

Source code in `src/app/daas/auth/auth_daas.py`

```
91  async def verify_list(self, entity: list, owner: int) -> bool:
92      result = True
93      for element in entity:
94          if isinstance(element, dict):
95              if "id_owner" in element:
96                  testid = element["id_owner"]
97                  result = testid in (0, owner)
98                  if result is False:
99                      break
100             else:
101                 result = True
102         else:
103             result = False
104             break
105     if self.config.enable_log_entity_verification:
106         self._log_info(
107             f"Verifying ownership (list): {result} for {owner} and {entity}"
108         )
109     return result
```

meth `verify_dict(entity: dict, owner: int) -> bool` async

Source code in `src/app/daas/auth/auth_daas.py`

```
111   async def verify_dict(self, entity: dict, owner: int) -> bool:
112       result = True
113       if self.config.enable_log_entity_verification:
114           self._log_info(
115               f"Verifying ownership (dict): {result} for {owner} and {entity}"
116           )
117       return result
```

meth `verify_object(entity: DaasObject, owner: int) -> bool` async

Source code in `src/app/daas/auth/auth_daas.py`

```
119   async def verify_object(self, entity: DaasObject, owner: int) -> bool:
120       testid = entity.id_owner
121       result = testid in (0, owner)
122       if self.config.enable_log_entity_verification:
123           self._log_info(
124               f"Verifying ownership ( obj): {result} for {owner} and {testid}"
125           )
126       return result
```

meth `verify_instance(entity: InstanceObject, owner: int) -> bool` async

Source code in `src/app/daas/auth/auth_daas.py`

```
128   async def verify_instance(self, entity: InstanceObject, owner: int) -> bool:
129       testid = entity.app.id_owner
130       result = testid in (0, owner)
131       if self.config.enable_log_entity_verification:
132           self._log_info(
133               f"Verifying ownership (inst): {result} for {owner} and {testid}"
134           )
135       return result
```

meth `verify_environment(entity: Environment, owner: int) -> bool` async

Source code in `src/app/daas/auth/auth_daas.py`

```
137   async def verify_environment(self, entity: Environment, owner: int) -> bool:
138       from app.daas.db.database import Database
139
140       dbase = await get_database(Database)
141       testobj = await dbase.get_daas_object(entity.id_object)
142       assert testobj is not None
143       testid = testobj.id_owner
144       result = testid in (0, owner)
145       if self.config.enable_log_entity_verification:
146           self._log_info(
147               f"Verifying ownership (inst): {result} for {owner} and {testid}"
148           )
149       return result
```

meth `verify_file(entity: File, owner: int) -> bool` async

Source code in `src/app/daas/auth/auth_daas.py`

```
151   async def verify_file(self, entity: File, owner: int) -> bool:
152       testid = entity.id_owner
153       result = testid in (0, owner)
154       if self.config.enable_log_entity_verification:
155           self._log_info(
156               f"Verifying ownership (file): {result} for {owner} and {testid}"
157           )
158       return result
```

meth `verify_app(entity: Application, owner: int) -> bool` async

Source code in `src/app/daas/auth/auth_daas.py`

```python
async def verify_app(self, entity: Application, owner: int) -> bool:
    from app.daas.db.database import Database

    dbase = await get_database(Database)
    testid = entity.id_owner
    result = testid in (0, owner)
    if result and entity.id_template is not None and entity.id_template != "":
        testfile = await dbase.get_daas_object(entity.id_template)
        if testfile is not None:
            result = testfile.id_owner in (0, owner)
    if result and entity.id_file is not None and entity.id_file != "":
        testfile = await dbase.get_file(entity.id_file)
        if testfile is not None:
            result = testfile.id_owner in (0, owner)
    if self.config.enable_log_entity_verification:
        self._log_info(
            f"Verifying ownership ( app): {result} for {owner} and {testid}"
        )
    return result
```

meth `return_default_user(authenticated: bool = True) -> QwebUser` async

Source code in `src/app/daas/auth/auth_daas.py`

```python
async def return_default_user(self, authenticated: bool = True) -> QwebUser:
    return QwebUser(
        self.config.local.default_userid,
        self.config.local.default_username,
        authenticated,
    )
```

meth `initialize() -> bool` async

Initialize component

Source code in `src/app/daas/auth/auth_daas.py`

```python
async def initialize(self) -> bool:
    """Initialize component"""
    return True
```

`src.app.daas.auth.auth_daas_local` module

DaaS local authenticator

Attributes

attr `DELAY_AUTH = 0.02` module-attribute

Classes

class `DaasLocalAuthenticator(config: QwebAuthenticatorConfig)`

Bases: `DaasAuthenticatorBase`

DaaS local authenticator

Source code in `src/app/daas/auth/auth_daas_local.py`

```
15  def __init__(self, config: QwebAuthenticatorConfig):
16      DaasAuthenticatorBase.__init__(self, config)
```

Functions

meth `verify_user(req: ProcessorRequest, info: BlueprintInfo) -> QwebUser` async

Verifies only user

Source code in `src/app/daas/auth/auth_daas_local.py`

```
18  async def verify_user(self, req: ProcessorRequest, info: BlueprintInfo) -> QwebUser:
19      """Verifies only user"""
20      result = QwebUser()
21      if self.config.enable_auth_user:
22          # self._log_info(f"200 -> DaaS Local Auth for bearer {req.bearer} {info.url}")
23          result = await self.return_default_user()
24      else:
25          result = await self.return_default_user()
26      return result
```

meth `verify_endpoint(req: ProcessorRequest, info: BlueprintInfo) -> QwebUser` async

Verifies only endpoint

Source code in `src/app/daas/auth/auth_daas_local.py`

```
28  async def verify_endpoint(
29      self, req: ProcessorRequest, info: BlueprintInfo
30  ) -> QwebUser:
31      """Verifies only endpoint"""
32      result = QwebUser()
33      if self.config.enable_auth_endpoint:
34          # self._log_info(
35          #     f"200 -> DaaS Local Auth for endpoint {req.bearer} ({info.url})"
36          # )
37          result = await self.return_default_user()
38      else:
39          result = await self.return_default_user()
40      return result
```

`src.app.daas.auth.auth_daas_remote` module

Authenticates requests

Classes

class `DaasRemoteAuthenticator(cfg_auth: QwebAuthenticatorConfig, cfg_http: HttpAdapterConfig)`

Bases: `DaasAuthenticatorBase`

Remote authenticator

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
20  def __init__(self, cfg_auth: QwebAuthenticatorConfig, cfg_http: HttpAdapterConfig):
21      self.cfg_http = cfg_http
22      self.adapter = HttpAdapter(self.cfg_http)
23      self.token: Optional[AuthToken] = None
24      DaasAuthenticatorBase.__init__(self, cfg_auth)
```

Attributes

attr `config: QwebAuthenticatorConfig` `instance-attribute`

attr `cfg_http = cfg_http` `instance-attribute`

attr `adapter = HttpAdapter(self.cfg_http)` `instance-attribute`

attr `token: Optional[AuthToken] = None` `instance-attribute`

Functions

meth `initialize() -> bool` `async`

Initialize component

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
26  async def initialize(self) -> bool:
27      """Initialize component"""
28      return True
```

meth `verify_user(req: ProcessorRequest, info: BlueprintInfo) -> QwebUser` `async`

Verifies only user

Source code in `src/app/daas/auth/auth_daas_remote.py`

```python
async def verify_user(self, req: ProcessorRequest, info: BlueprintInfo) -> QwebUser:
    """Verifies only user"""

    result = QwebUser()
    if self.cfg_auth.enable_auth_token:
        if (
            info.auth_params == AuthenticationMode.TOKEN
            or info.auth_params == AuthenticationMode.ALL
        ):
            result = await self.get_user_by_viewer_token(req)
            if result.authenticated:
                return result

    if self.config.enable_auth_user:
        userdict = await self.get_user_session(req.bearer)
        if userdict is not None:
            if "user_id" in userdict and "name" in userdict:
                result = QwebUser(
                    id_user=userdict["user_id"],
                    name=userdict["name"],
                    authenticated=True,
                )
                self._log_request(
                    f"200 ->   HTTP Remote Auth success for user '{result.id_user}'"
                )
                return result
    self._log_error("HTTP Remote Auth error", 1)
    return result
```

meth `verify_endpoint(req: ProcessorRequest, info: BlueprintInfo) -> QwebUser` async

Verifies endpoint

Source code in `src/app/daas/auth/auth_daas_remote.py`

```python
async def verify_endpoint(
    self, req: ProcessorRequest, info: BlueprintInfo
) -> QwebUser:
    """Verifies endpoint"""
    result = QwebUser()
    user = await self.verify_user(req, info)
    if user.authenticated and self.config.enable_auth_endpoint:

        response_remote = await self.get_endpoint_permission(
            info.name, user.id_user
        )
        if (
            response_remote is not None
            and "result" in response_remote
            and "user" in response_remote
        ):
            if response_remote["result"] == "allow":
                name = response_remote["user"]["name"]
                userid = response_remote["user"]["id"]
                result = QwebUser(id_user=userid, name=name, authenticated=True)
                self._log_request(
                    f"200 ->   HTTP Remote Auth success for endpoint '{info.name}'"
                )
                return result
        self._log_error(" -1 ->   HTTP Remote Auth error")
    return result
```

meth `get_user_session(bearer: str) -> dict` async

Verify bearer

Source code in `src/app/daas/auth/auth_daas_remote.py`

```python
async def get_user_session(self, bearer: str) -> dict:
    """Verify bearer"""
    url = f"{self.cfg_auth.remote.url}/oauth2/user/session"
    response, data = await self.__request_api("get", url, access_token=bearer)
    if response is None or response.status != 200:
        self._log_error(f"Backend-Auth error on {url}", response.status)
        return {}
    return data
```

meth `get_user_by_viewer_token(req: ProcessorRequest) -> QwebUser` async

Verify viewer token

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
95  async def get_user_by_viewer_token(self, req: ProcessorRequest) -> QwebUser:
96      """Verify viewer token"""
97      from app.daas.db.database import Database
98
99      dbase = await get_database(Database)
100     result = await self.return_default_user(False)
101     if self.cfg_auth.enable_auth_token:
102         reqargs = req.request_context.request_args
103         token = ""
104         if "token" in reqargs:
105             token = reqargs["token"]
106             con = await dbase.get_guacamole_connection_by_token(token)
107             if con is not None:
108                 inst = await dbase.get_instance_by_conid(con.id)
109                 if inst is not None:
110                     userid = inst.id_owner
111                     result = QwebUser(userid, token, True)
112             self._log_request(
113                 f"200 ->   HTTP Remote Auth success for user '{result.id_user}'"
114             )
115             return result
116     return result
```

meth `get_endpoint_permission(endpoint: str, userid: int) -> dict` async

Get user details

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
118 async def get_endpoint_permission(self, endpoint: str, userid: int) -> dict:
119     """Get user details"""
120     endpoint = "debug"
121     url = f"{self.cfg_auth.remote.url}/permissions/{endpoint}/{userid}"
122     response, data = await self.__request_api("get", url)
123     if response is None or response.status != 200:
124         self._log_error(f"Backend-Auth error on {url}", response.status)
125         return {}
126     return data
```

meth `get_endpoint_permission_by_viewer_token(endpoint: str, token: str) -> dict` async

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
128 async def get_endpoint_permission_by_viewer_token(
129     self, endpoint: str, token: str
130 ) -> dict:
131     return {}
```

meth `get_endpoint_permission_by_bearer(endpoint: str, bearer: str) -> dict` async

Get user details

Source code in `src/app/daas/auth/auth_daas_remote.py`

```python
async def get_endpoint_permission_by_bearer(
    self, endpoint: str, bearer: str
) -> dict:
    """Get user details"""
    url = f"{self.cfg_auth.remote.url}/permissions_info"
    endpoint_name = endpoint
    if endpoint_name.startswith("/"):
        endpoint_name = endpoint_name[1:]

    # data = {"token": bearer, "function_name": endpoint_name}
    data = {"token": bearer, "function_name": "debug"}
    response, resp_data = await self.__request_api(
        "post", url, access_token=bearer, data=data
    )
    if response is None or response.status != 200:
        if response is not None:
            resp = await response.json()
            self._log_error(f"Bearer: {bearer}")
            self._log_error(f"Response: {resp}")
        self._log_error(f"Backend-Auth error on {url}")
        return {}
    result = resp_data
    return result
```

meth `refresh_session() -> tuple` `async`

Obtaines new token from authentication service

Source code in `src/app/daas/auth/auth_daas_remote.py`

```python
async def refresh_session(self) -> tuple:
    """Obtaines new token from authentication service"""

    # Check existing token
    if self.token is not None:
        self.__check_token()

    # Fetch new if required
    if self.token is None:
        response, data = await self.__request_token()
        if response is None or response.status != 200:
            self._log_error("Error on token request")
            return 1, "", "Authentication error"
        # data = await response.json()
        self.__create_token(data)

    # Create response
    if self.token is None:
        self._log_error("Error token is None")
        return 1, "", "Authentication error"
    return 0, "", ""
```

meth `__request_api(method: str, url: str, *, access_token: str = '', data: dict = {}) -> tuple[aiohttp.ClientResponse, dict]` `async`

Source code in `src/app/daas/auth/auth_daas_remote.py`

```python
async def __request_api(
    self, method: str, url: str, *, access_token: str = "", data: dict = {}
) -> tuple[aiohttp.ClientResponse, dict]:
    chosen = await self.__choose_bearer_token(access_token)
    header = {
        "Authorization": f"Bearer {chosen}",
        "accept": "application/json",
    }
    if len(data) > 0:
        header["Content-Type"] = "application/json"
    return await self.adapter.request(method, url, header=header, data=data)
```

meth `__choose_bearer_token(access_token: str) -> str` `async`

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
192  async def __choose_bearer_token(self, access_token: str) -> str:
193      if access_token == "":
194          if self.token is None:
195              await self.refresh_session()
196          if self.token is not None:
197              return self.token.access_token
198          return ""
199      return access_token
```

meth `__request_token()` `async`

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
201  async def __request_token(self):
202      # self.__log_request("Creating new token!")
203      url = f"{self.cfg_auth.remote.url}/oauth2/user/token"
204      response, data = await self.adapter.request(
205          method="post",
206          url=url,
207          header={"accept": "application/json"},
208          cookies={},
209          params={},
210          data={
211              "username": self.cfg_auth.remote.username,
212              "password": self.cfg_auth.remote.password,
213              "client_id": self.cfg_auth.remote.clientid,
214              "grant_type": "password",
215              "scope": self.cfg_auth.remote.scope,
216          },
217      )
218      return response, data
```

meth `__check_token()`

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
220  def __check_token(self):
221      if self.token is not None:
222          if datetime.now() < self.token.valid_until:
223              pass
224          else:
225              obj = self.token.valid_until
226              fmt = obj.strftime("%Y-%m-%d %H:%M:%S")
227              self._log_request(f"Token expired at: {fmt}")
228              self.token = None
```

meth `__create_token(data: dict)`

Source code in `src/app/daas/auth/auth_daas_remote.py`

```
230  def __create_token(self, data: dict):
231      if "token_type" in data and "expires_in" in data:
232          if "refresh_token" in data and "access_token" in data:
233              now = datetime.now()
234              self.token = AuthToken(
235                  created_at=now,
236                  valid_until=now + timedelta(seconds=data["expires_in"]),
237                  time_valid=data["expires_in"],
238                  refresh_token=data["refresh_token"],
239                  access_token=data["access_token"],
240              )
241              obj = self.token.valid_until
242              fmt = obj.strftime("%Y-%m-%d %H:%M:%S")
243              self._log_request(f"New token expires at: {fmt}")
```

COMMON

`src.app.daas.common.config` module

Classes

class `DatabaseConfig(*, database: str, data_path: str, db_name: str, db_user: str, db_pass: str, db_host: str, db_port: int, db_type: str, enable_echo: bool = False, admin_credentials: Optional[str] = None, include_samples: bool)` `dataclass`

Config for databases

Attributes

attr `database: str` `instance-attribute`

File path to the database.

Example: `daas.sqlite`

The file will be created if it doesn't exist.

If a transient DB is needed, specify a tempfile. The special SQlite `:memory:` name does technically work, but since the application creates multiple connections they'd all have fresh database instances which doesn't work.

attr `data_path: str` `instance-attribute`

attr `db_name: str` `instance-attribute`

attr `db_user: str` `instance-attribute`

attr `db_pass: str` `instance-attribute`

attr `db_host: str` `instance-attribute`

attr `db_port: int` `instance-attribute`

attr `db_type: str` `instance-attribute`

attr `enable_echo: bool = False` `class-attribute` `instance-attribute`

attr `admin_credentials: Optional[str] = None` `class-attribute` `instance-attribute`

`username:password` for an admin account.

Example value: `"admin:s3cret!"`

If this variable is set, the application will store these credentials in the database for password authentication. This is great for test environments and for initial configuration.

attr `include_samples: bool` `instance-attribute`

Functions

class `InstanceControllerConfig(sshuser: str, sshport: int, sshopt_timeout: int, sshopt_strictcheck: str, sshopt_nohostauth: str, prefixip: str, proxyfile: str, logcmd: bool, logresult: bool, enable_message_queue: bool, test_method_online_state: str)` `dataclass`

Config object for InstanceController

Attributes

attr `sshuser: str` `instance-attribute`

attr `sshport: int` `instance-attribute`

attr `sshopt_timeout: int` `instance-attribute`

attr `sshopt_strictcheck: str` `instance-attribute`

attr `sshopt_nohostauth: str` `instance-attribute`

attr `prefixip: str` `instance-attribute`

attr `proxyfile: str` `instance-attribute`

attr `logcmd: bool` `instance-attribute`

attr `logresult: bool` `instance-attribute`

attr `enable_message_queue: bool` `instance-attribute`

attr `test_method_online_state: str` `instance-attribute`

Functions

`src.app.daas.common.ctx` module

CTX helpers

Attributes

attr `T = TypeVar('T')` module-attribute

attr `log = Loggable(LogTarget.ANY)` module-attribute

Functions

func `get_backend_component(ctx: QwebProcessorContexts, name: str, ret_type: Type[T]) -> T`

Source code in `src/app/daas/common/ctx.py`

```python
def get_backend_component(
    ctx: QwebProcessorContexts, name: str, ret_type: Type[T]
) -> T:
    backend = ctx.backends.get(name)
    if backend is not None and backend.component is not None:
        if isinstance(backend.component, ret_type):
            return backend.component
    raise TypeError(f"Backend has invalid type: {name} -> {ret_type}")
```

func `get_request_backend(ctx: QwebProcessorContexts, name: str, ret_type: Type[T]) -> T`

Source code in `src/app/daas/common/ctx.py`

```python
def get_request_backend(ctx: QwebProcessorContexts, name: str, ret_type: Type[T]) -> T:
    node = ctx.backends.get(name)
    if node is not None:
        if isinstance(node, ret_type):
            return node
    raise TypeError(f"Backend has invalid type: {name} -> {ret_type}")
```

func `get_request_object(ctx: QwebProcessorContexts, name: str, ret_type: Type[T]) -> T`

Source code in `src/app/daas/common/ctx.py`

```python
def get_request_object(ctx: QwebProcessorContexts, name: str, ret_type: Type[T]) -> T:
    req_objects = ctx.objects.objects
    if name in req_objects:
        entity = req_objects[name]
        if isinstance(entity, ret_type):
            return entity
        raise TypeError(f"LayerObject has invalid type: {name} {type(entity)}")
    raise TypeError(f"LayerObject is not available: {name} {ret_type}")
```

func `get_request_object_optional(ctx: QwebProcessorContexts, name: str, ret_type: Type[T]) -> Optional[T]`

Source code in `src/app/daas/common/ctx.py`

```python
def get_request_object_optional(
    ctx: QwebProcessorContexts, name: str, ret_type: Type[T]
) -> Optional[T]:
    req_objects = ctx.objects.objects
    if name in req_objects:
        entity = req_objects[name]
        if isinstance(entity, ret_type):
            return entity
        raise TypeError(f"LayerObject has invalid type: {name} {type(entity)}")
    return None
```

func `create_response_by_data_attribute(data: dict, response: Optional[aiohttp.ClientResponse], include_data: bool = True) -> QwebResult` async

Source code in `src/app/daas/common/ctx.py`

```python
async def create_response_by_data_attribute(
    data: dict, response: Optional[aiohttp.ClientResponse], include_data: bool = True
) -> QwebResult:
    if response is not None:
        if response.status != 200:
            return await create_response_by_http(response, data, include_data)
    if "data" in data:
        if include_data:
            return QwebResult(200, data)
        return QwebResult(200, {})
    return QwebResult(500, {}, 1, "No data in api response")
```

func `create_response_by_data_length(data: dict, include_data: bool = True) -> QwebResult` `async`

Source code in `src/app/daas/common/ctx.py`

```python
async def create_response_by_data_length(
    data: dict, include_data: bool = True
) -> QwebResult:
    if len(data) > 0:
        if include_data:
            return QwebResult(200, data)
        return QwebResult(200, {})
    return QwebResult(500, {}, 1, "No data elements in api response")
```

func `create_response_by_exitstatus(data: dict, response: Optional[aiohttp.ClientResponse], include_data: bool = True) -> QwebResult` `async`

Source code in `src/app/daas/common/ctx.py`

```python
async def create_response_by_exitstatus(
    data: dict, response: Optional[aiohttp.ClientResponse], include_data: bool = True
) -> QwebResult:
    if response is not None:
        if response.status != 200:
            return await create_response_by_http(response, data, include_data)
    if "exitstatus" in data:
        if data["exitstatus"] == "OK":
            if include_data:
                return QwebResult(200, data, 0, "")
            return QwebResult(200, {}, 0, "")
        return QwebResult(400, {}, 1, data["exitstatus"])
    return QwebResult(500, {}, 1, "No exitstatus in api response")
```

func `create_response_by_http(response: aiohttp.ClientResponse, data: dict, include_data: bool = True) -> QwebResult` `async`

Source code in `src/app/daas/common/ctx.py`

```python
async def create_response_by_http(
    response: aiohttp.ClientResponse, data: dict, include_data: bool = True
) -> QwebResult:
    if response.status == 200:
        if include_data:
            return QwebResult(200, data, 0, "")
        return QwebResult(200, {}, 0, "")
    return QwebResult(response.status, {}, 2, f"{response.reason}")
```

func `create_response_url_stop(stopped: bool) -> QwebResult` `async`

Source code in `src/app/daas/common/ctx.py`

```
106  async def create_response_url_stop(stopped: bool) -> QwebResult:
107      if stopped:
108          return QwebResult(200, {}, response_url="-")
109      return QwebResult(400, {}, 1, "Not stopped")
```

func `create_response_url_start(connect: bool, id_instance: str) -> QwebResult` `async`

Source code in `src/app/daas/common/ctx.py`

```
112  async def create_response_url_start(connect: bool, id_instance: str) -> QwebResult:
113      from app.daas.db.database import Database
114
115      dbase = await get_database(Database)
116      inst = await dbase.get_instance_by_id(id_instance)
117      if connect:
118          if inst is not None and inst.id_con is not None:
119              con = await dbase.get_guacamole_connection(inst.id_con)
120              if con is not None:
121                  return QwebResult(
122                      200, {}, response_url=con.viewer_url, id_instance=inst.id
123                  )
124          return QwebResult(400, {}, 1, "instance or connection missing")
125      return QwebResult(200, {})
```

func `remove_args(args: dict, exclude_params: list[str]) -> dict` `async`

Source code in `src/app/daas/common/ctx.py`

```
128  async def remove_args(args: dict, exclude_params: list[str]) -> dict:
129      for param in exclude_params:
130          if param in args:
131              args.pop(param)
132      return args
```

func `log_task_arguments(ctx: QwebProcessorContexts, req: ProcessorRequest, info: BlueprintInfo, user: QwebUser)`

Source code in `src/app/daas/common/ctx.py`

```
135  def log_task_arguments(
136      ctx: QwebProcessorContexts,
137      req: ProcessorRequest,
138      info: BlueprintInfo,
139      user: QwebUser,
140  ):
141      msg = f"TaskArgs({ctx}, {req}, {info}, {user})"
142      log._log_debug(msg)
```

`src.app.daas.common.enums` module

Classes

class `PluginName`

Bases: `Enum`

Attributes

attr `DB = 'db'` `class-attribute` `instance-attribute`

attr `PROXY = 'proxy'` `class-attribute` `instance-attribute`

attr `EXTENSIONS = 'extensions'` `class-attribute` `instance-attribute`

attr `MESSAGING = 'messaging'` `class-attribute` `instance-attribute`

attr `INFO = 'info'` `class-attribute` `instance-attribute`

attr `ADMIN = 'admin'` `class-attribute` `instance-attribute`

attr `AUTH = 'auth'` `class-attribute` `instance-attribute`

attr `NODE = 'node'` `class-attribute` `instance-attribute`

attr `FILE = 'file'` `class-attribute` `instance-attribute`

attr `CEPH = 'ceph'` `class-attribute` `instance-attribute`

attr `CONTAINER = 'container'` `class-attribute` `instance-attribute`

attr `VM = 'vm'` `class-attribute` `instance-attribute`

attr `PHASES = 'phases'` `class-attribute` `instance-attribute`

attr `INSTANCES = 'instances'` `class-attribute` `instance-attribute`

attr `TESTING = 'testbackend'` `class-attribute` `instance-attribute`

class `BackendName`

Bases: `Enum`

Attributes

attr `EXTENSIONS = 'extensions'` `class-attribute` `instance-attribute`

attr `DB = 'db'` `class-attribute` `instance-attribute`

attr `TASK = 'task'` `class-attribute` `instance-attribute`

attr `PROXY = 'proxy'` `class-attribute` `instance-attribute`

attr `MESSAGING = 'messaging'` `class-attribute` `instance-attribute`

attr `INFO = 'info'` `class-attribute` `instance-attribute`

attr `ADMIN = 'admin'` `class-attribute` `instance-attribute`

attr `AUTH = 'auth'` `class-attribute` `instance-attribute`

attr `NODE = 'node'` `class-attribute` `instance-attribute`

attr `LIMITS = 'limits'` `class-attribute` `instance-attribute`

attr `FILE = 'file'` `class-attribute` `instance-attribute`

attr `CEPH = 'ceph'` `class-attribute` `instance-attribute`

attr `CONTAINER = 'container'` `class-attribute` `instance-attribute`

attr `VM = 'vm'` `class-attribute` `instance-attribute`

attr `PHASES = 'phases'` `class-attribute` `instance-attribute`

attr `INSTANCES = 'instances'` `class-attribute` `instance-attribute`

attr `TESTING = 'testbackend'` `class-attribute` `instance-attribute`

class `LayerName`

Bases: `Enum`

Attributes

attr `DB = 'db'` `class-attribute` `instance-attribute`

attr `LIMIT = 'limits'` `class-attribute` `instance-attribute`

attr `TASK = 'task'` `class-attribute` `instance-attribute`

attr `TESTING = 'testing'` `class-attribute` `instance-attribute`

class `ConfigFile`

Bases: `Enum`

Attributes

attr `EXTENSIONS = 'extensions.toml'` class-attribute instance-attribute

attr `CONTAINER = 'container.toml'` class-attribute instance-attribute

attr `PROXY = 'proxy.toml'` class-attribute instance-attribute

attr `VM = 'vm.toml'` class-attribute instance-attribute

attr `FILE = 'files.toml'` class-attribute instance-attribute

attr `CEPH = 'ceph.toml'` class-attribute instance-attribute

attr `DB = 'db.toml'` class-attribute instance-attribute

attr `NODE = 'node.toml'` class-attribute instance-attribute

attr `AUTH = 'auth.toml'` class-attribute instance-attribute

attr `INFO = 'info.toml'` class-attribute instance-attribute

attr `LIMITS = 'limits.toml'` class-attribute instance-attribute

attr `LOG = 'logging.toml'` class-attribute instance-attribute

attr `MESSAGING = 'messaging.toml'` class-attribute instance-attribute

class `ConfigSections`

Bases: `Enum`

Attributes

attr `EXTENSIONS = 'extensions'` class-attribute instance-attribute

attr `PROXY_VIEWER = 'viewer'` class-attribute instance-attribute

attr `MESSAGING = 'instances'` class-attribute instance-attribute

attr `MESSAGING_QUEUE = 'inst_queue'` class-attribute instance-attribute

attr `MESSAGING_SSH = 'inst_ssh'` class-attribute instance-attribute

attr `LIMITS = 'limits'` class-attribute instance-attribute

attr `DB = 'db'` class-attribute instance-attribute

attr `SAMPLES = 'samples'` class-attribute instance-attribute

attr `LOG = 'log'` class-attribute instance-attribute

attr `AUTH = 'auth'` class-attribute instance-attribute

attr `AUTH_LOCAL = 'local'` class-attribute instance-attribute

attr `AUTH_REMOTE = 'remote'` class-attribute instance-attribute

attr `HOST_DNS = 'host_dns'` class-attribute instance-attribute

attr `HOST_SSH = 'host_ssh'` class-attribute instance-attribute

attr `INFO_SYS = 'sys'` class-attribute instance-attribute

attr `VM_API = 'vm_api'` class-attribute instance-attribute

attr `VM_REST = 'vm_rest'` class-attribute instance-attribute

attr `VM_HTTP = 'vm_http'` class-attribute instance-attribute

attr `CONTAINER_REQUEST = 'container_request'` class-attribute instance-attribute

attr `CONTAINER_SERVICES = 'service_containers'` class-attribute instance-attribute

attr `FILES_STORE = 'filestore'` class-attribute instance-attribute

attr `CEPH_STORE = 'cephstore'` class-attribute instance-attribute

attr `CEPH_DAASFS = 'cephstore.daasfs'` class-attribute instance-attribute

`src.app.daas.common.errors` module

Common error handlers

Classes

class `ObjectProcessingError`

Bases: `Exception`

Provides details on object errors

`src.app.daas.common.model` module

Extra data model definitions.

Classes

class `DaaSEntity()`

Bases: `ABC`, `Loggable`

Source code in `src/app/daas/common/model.py`

```
18  def __init__(self):
19      Loggable.__init__(self, LogTarget.OBJECT)
```

Attributes

attr `__orm_etype = ORMMappingType.Unknown` `class-attribute` `instance-attribute`

Functions

meth `get_data()`

Source code in `src/app/daas/common/model.py`

```
21  def get_data(self):
22      data = vars(self)
23      if "logger" in data:
24          data.pop("logger")
25      if "log_target" in data:
26          data.pop("log_target")
27      return data
```

class `DaasObject(id: str, id_user: str, id_owner: int, id_proxmox: int, id_docker: str, object_type: str, object_mode: str, object_mode_extended: str, object_state: str, object_tasks: list, object_apps: list, object_target: dict, object_storage: str, os_wine: bool, os_type: str, hw_cpus: int, hw_memory: int, hw_disksize: int, os_username: str = 'root', os_password: str = 'root', os_installer: str = '', ceph_public: bool = True, ceph_shared: bool = True, ceph_user: bool = True, vnc_port_system: int = -1, vnc_port_instance: int = 5900, vnc_username: str = 'user1234', vnc_password: str = 'user1234', viewer_contype: str = 'sysvnc', viewer_resolution: str = '1280x800', viewer_resize: str = 'both', viewer_scale: str = 'both', viewer_dpi: int = 96, viewer_colors: int = 32, viewer_force_lossless: bool = False, extra_args: str = '')` `dataclass`

Bases: `DaaSEntity`

An application configuration or container or VM that can be launched.

Attributes

attr `__orm_etype = ORMMappingType.Unknown` `class-attribute` `instance-attribute`

attr `id: str` `instance-attribute`

An unique identifier for this app – should be UUID.

attr `id_user: str` `instance-attribute`

An arbitrary specification given by the user

attr `id_owner: int` `instance-attribute`

The owner id

attr `id_proxmox: int` `instance-attribute`

An unique identifier used by the proxmox backend.

attr `id_docker: str` `instance-attribute`

An unique identifier used by the docker backend.

attr `object_type: str` instance-attribute

Type of object

attr `object_mode: str` instance-attribute

Current mode of operation

attr `object_mode_extended: str` instance-attribute

Current submode of operation

attr `object_state: str` instance-attribute

Current object state

attr `object_tasks: list` instance-attribute

Current object tasks

attr `object_apps: list` instance-attribute

Current object applist

attr `object_target: dict` instance-attribute

Current object target

attr `object_storage: str` instance-attribute

object data storage location

attr `os_wine: bool` instance-attribute

Has wine capabilities

attr `os_type: str` instance-attribute

Type of operating system

attr `hw_cpus: int` instance-attribute

Amount of cpus

attr `hw_memory: int` instance-attribute

Amount of memory in bytes

attr `hw_disksize: int` instance-attribute

Amount of bytes on disk

attr `os_username: str = 'root'` class-attribute instance-attribute

Username within the installed os

attr `os_password: str = 'root'` class-attribute instance-attribute

Password within the installed os

attr `os_installer: str = ''` class-attribute instance-attribute

The initial image the object is based on

attr `ceph_public: bool = True` class-attribute instance-attribute

Public cephfs share

attr `ceph_shared: bool = True` `class-attribute` `instance-attribute`

Shared cephfs share

attr `ceph_user: bool = True` `class-attribute` `instance-attribute`

User cephfs share

attr `vnc_port_system: int = -1` `class-attribute` `instance-attribute`

The port used for default vnc connections

attr `vnc_port_instance: int = 5900` `class-attribute` `instance-attribute`

The port used for connections directly initiated against the instance

attr `vnc_username: str = 'user1234'` `class-attribute` `instance-attribute`

The username used for vnc connections

attr `vnc_password: str = 'user1234'` `class-attribute` `instance-attribute`

The password used for vnc connections

attr `viewer_contype: str = 'sysvnc'` `class-attribute` `instance-attribute`

Connection protocol for viewer

attr `viewer_resolution: str = '1280x800'` `class-attribute` `instance-attribute`

Default resolution

attr `viewer_resize: str = 'both'` `class-attribute` `instance-attribute`

Resize method

attr `viewer_scale: str = 'both'` `class-attribute` `instance-attribute`

Scale method

attr `viewer_dpi: int = 96` `class-attribute` `instance-attribute`

Viewer dpi

attr `viewer_colors: int = 32` `class-attribute` `instance-attribute`

Viewer color depth

attr `viewer_force_lossless: bool = False` `class-attribute` `instance-attribute`

Force viewer to use lossless compression if applicable

attr `extra_args: str = ''` `class-attribute` `instance-attribute`

Optional arguments

Functions

meth `get_data()`

Source code in `src/app/daas/common/model.py`

```
21  def get_data(self):
22      data = vars(self)
23      if "logger" in data:
24          data.pop("logger")
25      if "log_target" in data:
26          data.pop("log_target")
27      return data
```

meth `__repr__()`

Source code in `src/app/daas/common/model.py`

```
147  def __repr__(self):
148      return (
149          f"{self.__class__.__qualname__}"
150          f"(id={self.id},name={self.id_user},"
151          f"obj_type={self.object_type},os_type={self.os_type})"
152      )
```

class `File(id: str, id_owner: int, name: str, version: str, os_type: str, localpath: str, remotepath: str, filesize: int, created_at: datetime)` `dataclass`

Bases: `DaaSEntity`

A uploadable file object

Attributes

attr `__orm_etype = ORMMappingType.Unknown` `class-attribute` `instance-attribute`

attr `id: str` `instance-attribute`

Unique ID for this application

attr `id_owner: int` `instance-attribute`

Unique ID for the owning user or -1 for a global application

attr `name: str` `instance-attribute`

Name of the application

attr `version: str` `instance-attribute`

The applicatiuon version

attr `os_type: str` `instance-attribute`

The type of operating system

attr `localpath: str` `instance-attribute`

The local path opf the file

attr `remotepath: str` `instance-attribute`

The remote path opf the file

attr `filesize: int` `instance-attribute`

The filesize in bytes

attr `created_at: datetime` `instance-attribute`

When this application was first created.

Functions

meth `get_data()`

Source code in `src/app/daas/common/model.py`

```
def get_data(self):
    data = vars(self)
    if "logger" in data:
        data.pop("logger")
    if "log_target" in data:
        data.pop("log_target")
    return data
```

meth `__repr__()`

Source code in `src/app/daas/common/model.py`

```
def __repr__(self):
    return (
        f"{self.__class__.__qualname__}"
        f"(id={self.id},name={self.name},os_type={self.os_type})"
    )
```

class
`Application(id: str, id_owner: int, name: str, version: str, id_file: Optional[str], id_template: Optional[str], os_type: str, installer: str, installer_args: str, installer_type: str, target: str, target_args: str, created_at: datetime)` `dataclass`

Bases: `DaaSEntity`

A configurable application object

Attributes

attr `__orm_etype = ORMMappingType.Unknown` `class-attribute` `instance-attribute`

attr `id: str` `instance-attribute`

Unique ID for this application

attr `id_owner: int` `instance-attribute`

Unique ID for the owning user or -1 for a global application

attr `name: str` `instance-attribute`

Name of the application

attr `version: str` `instance-attribute`

The applicatiuon version

attr `id_file: Optional[str]` `instance-attribute`

The id of the file to copy

attr `id_template: Optional[str]` `instance-attribute`

The id of the source object for cloning

attr `os_type: str` `instance-attribute`

The type of operating system

attr `installer: str` `instance-attribute`

The related installer path

attr `installer_args: str` `instance-attribute`

Installer arguments

attr `installer_type: str` `instance-attribute`

The related installer type

attr `target: str` `instance-attribute`

The installed executable path

attr `target_args: str` `instance-attribute`

executable args

attr `created_at: datetime` `instance-attribute`

When this application was first created.

Functions

meth `get_data()`

Source code in `src/app/daas/common/model.py`

```
21  def get_data(self):
22      data = vars(self)
23      if "logger" in data:
24          data.pop("logger")
25      if "log_target" in data:
26          data.pop("log_target")
27      return data
```

meth `__repr__()`

Source code in `src/app/daas/common/model.py`

```
403  def __repr__(self):
404      return (
405          f"{self.__class__.__qualname__}"
406          f"(id={self.id},name={self.name},os_type={self.os_type})"
407      )
```

class `Environment(id: str, id_backend: str, id_object: str, name: str, state: str, env_tasks: list, env_apps: list, env_target: dict, created_at: datetime)` `dataclass`

Bases: `DaaSEntity`

An environment derived from a daas_object

Attributes

attr `__orm_etype = ORMMappingType.Unknown` `class-attribute` `instance-attribute`

attr `id: str` `instance-attribute`

Unique ID for this environemnt.

attr `id_backend: str` `instance-attribute`

Unique ID for the latest snapshot.

attr `id_object: str` `instance-attribute`

key to Object configuration that is being instantiated.

attr `name: str` `instance-attribute`

Name of the environment.

attr `state: str` `instance-attribute`

State of the environment.

attr `env_tasks: list` `instance-attribute`

Current env tasks

attr `env_apps: list` `instance-attribute`

Current env applist

attr `env_target: dict` `instance-attribute`

Current env target

attr `created_at: datetime` `instance-attribute`

When this environment was first created.

Functions

meth `get_data()`

Source code in `src/app/daas/common/model.py`

```
21  def get_data(self):
22      data = vars(self)
23      if "logger" in data:
24          data.pop("logger")
25      if "log_target" in data:
26          data.pop("log_target")
27      return data
```

meth `__repr__()`

Source code in `src/app/daas/common/model.py`

```
442   def __repr__(self):
443       return f"{self.__class__.__qualname__}" f"(id={self.id},name={self.name})"
```

class `Instance(id: str, id_owner: int, app: DaasObject, host: str, container: str, created_at: datetime, connected_at: datetime, booted_at: datetime, id_con: Optional[str], id_app: str, id_env: Optional[str], env: Optional[Environment] = None)` `dataclass`

Bases: `DaaSEntity`

An instance of some application

Attributes

attr `__orm_etype = ORMMappingType.Unknown` `class-attribute` `instance-attribute`

attr `id: str` `instance-attribute`

Unique ID for this instance.

attr `id_owner: int` `instance-attribute`

Id of the owner

attr `app: DaasObject` `instance-attribute`

App configuration that is being instantiated.

attr `host: str` `instance-attribute`

Local available Ip or resolvable hostname assigned by backend services

attr `container: str` `instance-attribute`

Container short_id assigned by docker backend

attr `created_at: datetime` `instance-attribute`

When this instance was first created.

attr `connected_at: datetime` `instance-attribute`

When this instance was first connected.

attr `booted_at: datetime` `instance-attribute`

When this instance was first connected.

attr `id_con: Optional[str]` `instance-attribute`

Connection

attr `id_app: str` `instance-attribute`

App id

attr `id_env: Optional[str]` `instance-attribute`

Env id

attr `env: Optional[Environment] = None` `class-attribute` `instance-attribute`

Environment configuration that is being instantiated.

Functions

meth `get_data()`

Source code in `src/app/daas/common/model.py`

```
21  def get_data(self):
22      data = vars(self)
23      if "logger" in data:
24          data.pop("logger")
25      if "log_target" in data:
26          data.pop("log_target")
27      return data
```

meth `__repr__()`

Source code in `src/app/daas/common/model.py`

```
490  def __repr__(self):
491      return f"{self.__class__.__qualname__}" f"(id={self.id})"
```

class `GuacamoleConnection(*, id: str, user: str, password: str, protocol: str, hostname: str, port: int, viewer_url: str, viewer_token: str)` `dataclass`

Bases: `DaaSEntity`

Connection details for Guacamole.

Attributes

attr `__orm_etype = ORMMappingType.Unknown` `class-attribute` `instance-attribute`

attr `id: str` `instance-attribute`

Unique ID

attr `user: str` `instance-attribute`

Username for the connection auth.

attr `password: str` `instance-attribute`

Password for the connection auth.

attr `protocol: str` `instance-attribute`

Protocol used by guacd, either `vnc` or `rdp`.

attr `hostname: str` `instance-attribute`

Hostname where guacd can find the connection's server.

attr `port: int` `instance-attribute`

Port where guacd can find the connection's server.

attr `viewer_url: str` `instance-attribute`

Absolute Endpoint URL for this connection

attr `viewer_token: str` `instance-attribute`

Authentication token for this connection

Functions

meth `get_data()`

Source code in `src/app/daas/common/model.py`

```
21  def get_data(self):
22      data = vars(self)
23      if "logger" in data:
24          data.pop("logger")
25      if "log_target" in data:
26          data.pop("log_target")
27      return data
```

meth `__repr__()`

Source code in `src/app/daas/common/model.py`

```
528  def __repr__(self):
529      return f"{self.__class__.__qualname__}" f"(id={self.id})"
```

class `RessourceInfo(id_owner: int, vm_max: int, container_max: int, obj_max: int, cpu_max: int, mem_max: int, dsk_max: int)` `dataclass`

Bases: `DaaSEntity`

Ressource limits

Attributes

attr `__orm_etype = ORMMappingType.Unknown` `class-attribute` `instance-attribute`

attr `id_owner: int` `instance-attribute`

attr `vm_max: int` `instance-attribute`

attr `container_max: int` `instance-attribute`

attr `obj_max: int` `instance-attribute`

attr `cpu_max: int` instance-attribute

attr `mem_max: int` instance-attribute

attr `dsk_max: int` instance-attribute

Functions

meth `get_data()`

Source code in `src/app/daas/common/model.py`

```
21  def get_data(self):
22      data = vars(self)
23      if "logger" in data:
24          data.pop("logger")
25      if "log_target" in data:
26          data.pop("log_target")
27      return data
```

meth `add(other)`

Adds other ressources

Source code in `src/app/daas/common/model.py`

```
545 def add(self, other):
546     """Adds other ressources"""
547     self.vm_max += other.vm_max
548     self.container_max += other.container_max
549     self.obj_max += other.obj_max
550     self.cpu_max += other.cpu_max
551     self.mem_max += other.mem_max
552     self.dsk_max += other.dsk_max
```

meth `__repr__() -> str`

Source code in `src/app/daas/common/model.py`

```
554 def __repr__(self) -> str:
555     return (
556         f"{self.__class__.__qualname__}"
557         f"(id_owner={self.id_owner},"
558         f"objs=({self.obj_max},{self.vm_max},{self.container_max}),"
559         f"res=({self.cpu_max},{self.mem_max},{self.dsk_max})"
560         ")"
561     )
```

class `AccessToken(*, id: str, id_user: int, description: str, created_at: datetime)` dataclass

Metadata on an access token that can be used for API access.

Note that the `secret_token` field is not part of this object, since it must only be used during token creation.

Attributes

attr `id: str` instance-attribute

ID for this token, useful for revocation.

attr `id_user: int` instance-attribute

Permission scope.

attr `description: str` instance-attribute

Textual description to remember what this token was created for.

attr `created_at: datetime` instance-attribute

When the token was created.

Functions

CONTAINER

docker

`src.app.daas.container.docker.DockerRequest` module

Docker facade

Classes

class `DockerImageInfo(name: str, attrs: dict, tag: str)` `dataclass`

A info object for Docker images

Attributes

attr `name: str` `instance-attribute`

attr `attrs: dict` `instance-attribute`

attr `tag: str` `instance-attribute`

Functions

class `DockerServicesConfig(service_containers: list[list])` `dataclass`

Attributes

attr `service_containers: list[list]` `instance-attribute`

Functions

class `ContainerConfigDocker(image: str, name: str, privileged: bool, bind_ports: list[str], volumes: list[str], cpus: int, memory: int)` `dataclass`

Attributes

attr `image: str` `instance-attribute`

attr `name: str` `instance-attribute`

attr `privileged: bool` `instance-attribute`

attr `bind_ports: list[str]` `instance-attribute`

attr `volumes: list[str]` `instance-attribute`

attr `cpus: int` `instance-attribute`

attr `memory: int` `instance-attribute`

Functions

class `DockerContainerInfo(name: str, id_container: str, status: str, info_image: DockerImageInfo, stats_raw: dict, stats_cpu: dict, stats_mem: dict, stats_disk: dict, stats_net: dict)` `dataclass`

A info object for Docker container

Attributes

attr `name: str` `instance-attribute`

attr `id_container: str` `instance-attribute`

attr `status: str` `instance-attribute`

attr `info_image: DockerImageInfo` `instance-attribute`

attr `stats_raw: dict` `instance-attribute`

attr `stats_cpu: dict` `instance-attribute`

attr `stats_mem: dict` `instance-attribute`

attr `stats_disk: dict` `instance-attribute`

attr `stats_net: dict` `instance-attribute`

Functions

class `DockerRequestConfig(host: str, wait_services_ms: int, logrequest: bool)` `dataclass`

A configuration for the DockerRequest component

Attributes

attr `host: str` `instance-attribute`

attr `wait_services_ms: int` `instance-attribute`

attr `logrequest: bool` `instance-attribute`

Functions

class `DockerRequest(cfg: DockerRequestConfig, cfg_services: DockerServicesConfig)`

Bases: `Loggable`

A component to wrap various docker commands

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
77  def __init__(
78      self,
79      cfg: DockerRequestConfig,
80      cfg_services: DockerServicesConfig,
81  ):
82      Loggable.__init__(self, LogTarget.CONTAINER)
83      self.config = cfg
84      self.config_services = cfg_services
85      self.docker = self.__connect()
```

Attributes

attr `config = cfg` `instance-attribute`

attr `config_services = cfg_services` `instance-attribute`

attr `docker = self.__connect()` `instance-attribute`

Functions

meth `__repr__()`

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
87  def __repr__(self):
88      return (
89          f"{self.__class__.__qualname__}"
90          f"(host={self.config.host},"
91          f"logrequest={self.config.logrequest})"
92      )
```

meth `connect()` `async`

Connects the component

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
94  async def connect(self):
95      """Connects the component"""
96      self.docker = self.__connect()
97      if self.docker is not None:
98          services = self._read_service_config()
99          for svc in services:
100             await self.docker_service_start(svc)
101         if self.config.wait_services_ms > 0:
102             self._log_info(
103                 f"Waiting {self.config.wait_services_ms}ms for started services"
104             )
105             time.sleep(self.config.wait_services_ms / 1000)
106         self.connected = True
107     return self.connected
```

meth `disconnect() -> bool` `async`

Disconnects the component

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
async def disconnect(self) -> bool:
    """Disconnects the component"""
    if self.docker is not None:
        services = self._read_service_config()
        self._log_info(f"Stopping {len(services)} services:")
        for svc in services:
            self._log_info(f"Stopping {svc.name}")
            await self.docker_service_stop(svc)
    self.docker = None
    self.connected = False
    return True
```

meth `docker_image_build(dockerfile: str, tag: str) -> tuple[int, str, str]` async

Build image

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
async def docker_image_build(
    self, dockerfile: str, tag: str
) -> tuple[int, str, str]:
    """Build image"""
    if self.docker is None:
        return 1, "", "No docker env"
    if os.path.exists(dockerfile) is False:
        return 2, "", f"File is not present: '{dockerfile}'"
    path = os.path.dirname(dockerfile)

    str_out = ""
    try:
        image, build_log = self.docker.images.build(
            path=path.lower(),
            dockerfile=dockerfile,
            tag=tag,
            rm=True,
            forcerm=True,
        )
        if image is None:
            return 2, "", "Image was not created"

        str_out = self._read_docker_log(build_log)
    except Exception as exe:
        self._log_error("Exception on image build:", 3, exe)
        return 3, "", "Error on build"
    return 0, str_out, ""
```

meth `docker_image_list(detailed: bool = False) -> list[DockerImageInfo]` async

Lists all docker images

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
async def docker_image_list(self, detailed: bool = False) -> list[DockerImageInfo]:
    """
    Lists all docker images
    """
    result: list[DockerImageInfo] = []
    if self.docker is None:
        return result
    imglist = self.docker.images.list()
    for image in imglist:
        infos = self.__create_image_info(image, detailed)
        for single in infos:
            result.append(single)
    return result
```

meth `docker_image_delete(tag: str = '') -> tuple` async

Deletes docker images

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
179  async def docker_image_delete(self, tag: str = "") -> tuple:
180      """
181      Deletes docker images
182      """
183      result = 1, "", "Unknown error"
184      if self.docker is None:
185          return result
186      try:
187          self.docker.images.remove(image=tag, force=True)
188          result = 0, "", ""
189      except docker.errors.ImageNotFound:
190          result = 1, "", "Image not found"
191      return result
```

meth `docker_service_start(cfg: ContainerConfigDocker) -> tuple` async

Starts a new service container based on specified image

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
193  async def docker_service_start(self, cfg: ContainerConfigDocker) -> tuple:
194      """
195      Starts a new service container based on specified image
196      """
197      if self.docker is None:
198          return 1, "", "No docker env"
199      container_config = self._create_run_config(
200          cfg.image,
201          cfg.name,
202          cfg.cpus,
203          cfg.memory,
204          cfg.privileged,
205          cfg.bind_ports,
206          cfg.volumes,
207      )
208      try:
209          await self.docker_container_stop(cfg.name, True)
210          self._log_info(f"Running service: {cfg.name}")
211          container = self.docker.containers.run(**container_config)
212          if container is not None:
213              name = container.name
214              image = container.image
215              self._log_info(f"Started service {name} from {image}")
216              return 0, name, ""
217      except Exception as exe:
218          self._log_error(f"  1 -> Exception raised1: {exe}")
219      return 1, "", "Service was not created!"
```

meth `docker_service_stop(cfg: ContainerConfigDocker) -> tuple` async

Stops a service container

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
221  async def docker_service_stop(self, cfg: ContainerConfigDocker) -> tuple:
222      """Stops a service container"""
223      if self.docker is None:
224          return 1, "", "No docker env"
225      try:
226          await self.docker_container_stop(cfg.name, True)
227          self._log_info(f"Stopped service {cfg.name} from {cfg.image}")
228          return 0, "", ""
229      except Exception as exe:
230          self._log_error(f"  1 -> Exception raised1: {exe}")
231      return 1, "", "Service was not stopped!"
```

meth
`docker_container_start(image: str, name: str, cpus: int, memory_b: int, privileged: bool = False, ports: list[str] = [], volumes: list[str] = []) -> tuple` async

Starts a new container based on specified image

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
233  async def docker_container_start(
234      self,
235      image: str,
236      name: str,
237      cpus: int,
238      memory_b: int,
239      privileged: bool = False,
240      ports: list[str] = [],
241      volumes: list[str] = [],
242  ) -> tuple:
243      """
244      Starts a new container based on specified image
245      """
246      if self.docker is None:
247          return 1, "", "No docker env"
248      container_config = self._create_run_config(
249          image, name, cpus, memory_b, privileged, ports, volumes
250      )
251      try:
252          await self.docker_container_stop(name, True)
253          container = self.docker.containers.run(**container_config)
254          if container is not None:
255              name = container.name
256              image = container.image
257              self._log_info(f"Started container {name} from {image}")
258              return 0, name, ""
259      except Exception as exe:
260          self._log_error(f"  1 -> Exception raised1: {exe}")
261      return 1, "", "Container was not created!"
```

meth `docker_container_stop(name: str, force: bool) -> tuple` async

Starts a new container based on specified image

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
263  async def docker_container_stop(self, name: str, force: bool) -> tuple:
264      """
265      Starts a new container based on specified image
266      """
267      if self.docker is None:
268          return 1, "", "No docker env"
269      container = None
270      contlist = self.docker.containers.list()
271      for cont in contlist:
272          if name == cont.name:
273              container = self.docker.containers.get(container_id=cont.name)
274              try:
275                  if container is not None:
276                      if force:
277                          container.kill()
278                          self._log_info(f"Killed {container.short_id}")
279                      else:
280                          container.stop()
281                          self._log_info(f"Stopped {container.short_id}")
282                      while await self.docker_container_get(name) is not None:
283                          time.sleep(0.1)
284                      return 0, name, ""
285              except Exception as exe:
286                  self._log_error(f"  1 -> Exception raised2: {exe}")
287      return 2, "", "No container"
```

meth `docker_get_container_ip(container_id: str) -> str` async

Obtain an IP address for the given container.

Can fail e.g. if the container doesn't exist or has no IP addresses. Doesn't guarantee that the IP address will be routable.

Source code in `src/app/daas/container/docker/DockerRequest.py`

```python
async def docker_get_container_ip(self, container_id: str) -> str:
    """
    Obtain an IP address for the given container.

    Can fail e.g. if the container doesn't exist
    or has no IP addresses.
    Doesn't guarantee that the IP address will be routable.
    """
    if self.docker is None:
        return ""
    inspect = self.docker.api.inspect_container(container_id)

    addresses = [
        net["IPAddress"] for net in inspect["NetworkSettings"]["Networks"].values()
    ]

    assert addresses, f"container {container_id} must have at least one IP address"

    # if there are multiple IPs, log them all and return the first
    if len(addresses) > 1:
        self._log_info(f"container {container_id} has multiple IPs: {addresses}")
    return addresses[0]
```

meth `docker_container_list(include_stats: bool = False) -> list[DockerContainerInfo]` `async`

List containers

Source code in `src/app/daas/container/docker/DockerRequest.py`

```python
async def docker_container_list(
    self, include_stats: bool = False
) -> list[DockerContainerInfo]:
    """
    List containers
    """
    result: list[DockerContainerInfo] = []
    if self.docker is None:
        return result
    contlist = self.docker.containers.list()
    for cont in contlist:
        info = self.__create_container_info(cont, include_stats)
        result.append(info)
    return result
```

meth `docker_container_logs(name: str) -> tuple[int, str, str]` `async`

Container logs

Source code in `src/app/daas/container/docker/DockerRequest.py`

```python
async def docker_container_logs(self, name: str) -> tuple[int, str, str]:
    """
    Container logs
    """
    container = await self.docker_container_get(name)
    if container is not None:
        return 0, f"{container.logs()}", ""
    code = 1
    msg = f"No container with specified name: {name}"
    self._log_error(msg)
    return (code, "", msg)
```

meth `docker_container_get(name: str) -> Optional[Container]` `async`

Returns container specified by name

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
340  async def docker_container_get(self, name: str) -> Optional[Container]:
341      """Returns container specified by name"""
342      result = None
343      if self.docker is None:
344          return result
345      try:
346          result = self.docker.containers.get(name)
347      except docker.errors.NotFound:
348          result = None
349      return result
```

meth `__docker_image_get(tag: str) -> Optional[Image]`

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
444  def __docker_image_get(self, tag: str) -> Optional[Image]:
445      result = None
446      if self.docker is None:
447          return result
448      try:
449          result = self.docker.images.get(tag)
450      except docker.errors.ImageNotFound:
451          result = None
452      return result
```

meth `__create_image_info(image: Image, detailed: bool = False) -> list[DockerImageInfo]`

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
454  def __create_image_info(
455      self, image: Image, detailed: bool = False
456  ) -> list[DockerImageInfo]:
457      ret: list[DockerImageInfo] = []
458      for tag in image.tags:
459          name = tag
460          if ":" in tag:
461              name = tag.split(":")[0]
462          result = DockerImageInfo(name, image.attrs, tag)
463          ret.append(result)
464
465      return ret
```

meth `__create_container_info(cont, detailed: bool = False) -> DockerContainerInfo`

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
467   def __create_container_info(
468       self, cont, detailed: bool = False
469   ) -> DockerContainerInfo:
470       result = {}
471       name = cont.name
472       status = cont.status
473       stats_raw = {}
474       stats_cpu = {}
475       stats_mem = {}
476       stats_disk = {}
477       stats_net = {}
478       imginfo = DockerImageInfo(name, cont.image.attrs, cont.image.tags)
479       if detailed is True:
480           stats_raw = cont.stats(stream=False)
481           stats_cpu = stats_raw["cpu_stats"]
482           stats_mem = stats_raw["memory_stats"]
483           stats_disk = stats_raw["blkio_stats"]
484           stats_net = stats_raw["networks"]
485       result = DockerContainerInfo(
486           name,
487           cont.id,
488           status,
489           imginfo,
490           stats_raw,
491           stats_cpu,
492           stats_mem,
493           stats_disk,
494           stats_net,
495       )
496       return result
```

meth `__connect()`

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
498   def __connect(self):
499       return docker.from_env()
```

meth `docker_get_daemoninfo() -> str` async

Returns info() output from currently used docker instance

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
501   async def docker_get_daemoninfo(self) -> str:
502       """Returns info() output from currently used docker instance"""
503       if self.docker is not None:
504           return self.docker.info()
505       return "No docker env"
```

meth `docker_image_inspect(tag: str) -> dict` async

Inspect image

Source code in `src/app/daas/container/docker/DockerRequest.py`

```
507   async def docker_image_inspect(self, tag: str) -> dict:
508       """Inspect image"""
509       image = self.__docker_image_get(tag)
510       if image is not None:
511           return image.attrs
512       return {}
```

DB

`src.app.daas.db.database` module

Provide state management via the `Database` class.

Attributes

attr `DEFAULT_ADMIN_ID = 5` module-attribute

Classes

class `Database()`

Bases: `LimitRepository`, `FileRepository`, `ApplicationRepository`, `ConnectionRepository`, `EnvironmentRepository`, `ObjectRepository`, `InstanceRepository`

A wrapper for database operations. Uses SQLAlchemy (ORM) for storing state.

Source code in `src/app/daas/db/database.py`

```
41  def __init__(self) -> None:
42      super().__init__()
43      self.auth = AuthDatabase()
```

Attributes

attr `auth = AuthDatabase()` instance-attribute

Functions

meth `connect() -> Database` async

Connects database

Source code in `src/app/daas/db/database.py`

```
45  async def connect(
46      self,
47  ) -> Database:
48      """Connects database"""
49      self.connected = await self._connect()
50      return self
```

meth `disconnect() -> None` async

Disconnects database

Source code in `src/app/daas/db/database.py`

```
52  async def disconnect(self) -> None:
53      """Disconnects database"""
54      await self._disconnect()
```

meth `load_demo_apps() -> None` async

Prime the database with some apps that are known to be available for a demo.

Source code in `src/app/daas/db/database.py`

```
async def load_demo_apps(self) -> None:
    """
    Prime the database with some apps
    that are known to be available for a demo.
    """
    from app.daas.objects.object_container import ContainerObject
    from app.daas.objects.object_machine import MachineObject

    cfg = await self._get_config_samples()
    logger = logging.getLogger(LogTarget.DB.value)
    registry = SampleRegistry(cfg=cfg)

    # if cfg.persist_session:
    #     session["userid"] = registry.config.default_owner
    #     session["user"] = registry.config.default_owner_name

    limitlist = registry.create_demo_limits()
    for testlimit in limitlist:
        if not await self.get_limit(testlimit.id_owner):
            msg = f"Import limit       : ({testlimit.id_owner})"
            logger.info(msg)
            await self.create_limit(testlimit)

    filelist = registry.create_demo_files()
    for testfile in filelist:
        if not await self.get_file(testfile.id):
            msg = f"Import file       : {testfile.name} ({testfile.id})"
            logger.info(msg)
            await self.create_file(testfile)

    objlist = registry.create_demo_objects()
    for obj in objlist:
        if not await self.get_daas_object(obj.id):
            msg = f"Import DB samples : {obj.id_user} ({obj.id})"
            if isinstance(obj, ContainerObject):
                logger.info(msg)
                await self.create_daas_object(obj)
            if isinstance(obj, MachineObject):
                logger.info(msg)
                await self.create_daas_object(obj)

    envs = registry.create_demo_environments()
    for env in envs:
        obj = await self.get_environment(env.id)
        if obj is None:
            await self.create_environment(env)

    cons = registry.create_demo_connection()
    for con in cons:
        obj = await self.get_guacamole_connection(con.id)
        if obj is None:
            await self.create_guacamole_connection(con)
    if len(objlist) > 0 and len(envs) > 0 and len(cons) > 0:
        insts = registry.create_demo_instances(objlist[0], envs[0], cons[0])
        for inst in insts:
            obj = await self.get_instance_by_id(inst.id)
            if obj is None:
                await self.create_instance(inst)

    applist = registry.create_demo_applications()
    for testapp in applist:
        if isinstance(testapp, ApplicationObject):
            if not await self.get_application(testapp.id):
                await self.create_application(testapp)
```

class `AuthDatabase()`

Auth-specific database operations.

Source code in `src/app/daas/db/database.py`

```
def __init__(self) -> None:
    self.creds = {}
```

Attributes

attr `creds = {}` `instance-attribute`

Functions

meth `save_password_hash(*, username: str, pwhash: str) -> None` `async`

Save or update the password hash in the database.

Source code in `src/app/daas/db/database.py`

```
363   async def save_password_hash(self, *, username: str, pwhash: str) -> None:
364       """
365       Save or update the password hash in the database.
366       """
367
368       values = {"id": -1, "password_hash": pwhash}
369       if username == "admin":
370           values["id"] = f"{DEFAULT_ADMIN_ID}"
371       self.creds[username] = values
```

meth `get_password_hash(username: str) -> tuple[int, str]` async

Retrieve a password hash for the given user.

Source code in `src/app/daas/db/database.py`

```
373   async def get_password_hash(self, username: str) -> tuple[int, str]:
374       """Retrieve a password hash for the given user."""
375       if username in self.creds:
376           return (
377               self.creds[username]["id"],
378               self.creds[username]["password_hash"],
379           )
380       return -1, ""
```

`src.app.daas.db.db_api` module

Database api

Classes

class `DatabaseApiBase()`

Bases: `Loggable`

Baseclass for Database api

Source code in `src/app/daas/db/db_api.py`

```
22  def __init__(self):
23      Loggable.__init__(self, LogTarget.DB)
```

Functions

class `DatabaseApi(engine: Engine, metadata: MetaData)`

Bases: `DatabaseApiBase`

Handles calls to the database

Source code in `src/app/daas/db/db_api.py`

```
60  def __init__(self, engine: Engine, metadata: MetaData):
61      super().__init__()
62      self.engine = engine
63      self.metadata = metadata
64      self.connected = False
```

Attributes

attr `engine = engine` `instance-attribute`

attr `metadata = metadata` `instance-attribute`

attr `connected = False` `instance-attribute`

Functions

meth `connect()`

Creates new session

Source code in `src/app/daas/db/db_api.py`

```
66  def connect(self):
67      """Creates new session"""
68      self.session = Session(self.engine)
69      self.connected = True
```

meth `disconnect()`

Closes session

Source code in `src/app/daas/db/db_api.py`

```
71  def disconnect(self):
72      """Closes session"""
73      if self.session is not None:
74          self.session.close()
75          self.session = None
76      self.connected = False
```

meth `orm_to_model(orm: ORMEntity)` async

Converts orm object to daas object

Source code in `src/app/daas/db/db_api.py`

```
78  async def orm_to_model(self, orm: ORMEntity):
79      """Converts orm object to daas object"""
80      return create_model(orm)
```

meth `model_to_orm(model: DaaSEntity)` async

Converts model object to orm object

Source code in `src/app/daas/db/db_api.py`

```
82  async def model_to_orm(self, model: DaaSEntity):
83      """Converts model object to orm object"""
84      return create_orm(model)
```

meth `db_session_flush() -> bool` async

Flush all changes to the db

Source code in `src/app/daas/db/db_api.py`

```
86  async def db_session_flush(
87      self,
88  ) -> bool:
89      """Flush all changes to the db"""
90      if self.session is None:
91          return False
92      self.session.flush()
93      return True
```

meth `db_session_create(dbname: str) -> bool`

Flush all changes to the db

Source code in `src/app/daas/db/db_api.py`

```
95   def db_session_create(
96       self,
97       dbname: str,
98   ) -> bool:
99       """Flush all changes to the db"""
100      if self.session is None:
101          return False
102      with self.engine.connect() as connection:
103          self._log_info(f"Create Database {dbname}")
104          connection.execute(text(f"CREATE DATABASE IF NOT EXISTS {dbname};"))
105          connection.commit()
106      return True
```

meth `db_session_query(mapping: Type, filter: Optional[TextClause] = None) -> Optional[Query]` async

Query the session by using specified mapping

Source code in `src/app/daas/db/db_api.py`

```
async def db_session_query(
    self,
    mapping: Type,
    filter: Optional[TextClause] = None,
) -> Optional[Query]:
    """Query the session by using specified mapping"""
    if self.session is None:
        return None
    qresult = self.session.query(mapping)
    if filter is not None:
        qresult = qresult.filter(filter)
    return qresult
```

meth `db_session_upsert(model: DaaSEntity) -> bool` async

Inserts or update given domain object

Source code in `src/app/daas/db/db_api.py`

```
async def db_session_upsert(self, model: DaaSEntity) -> bool:
    """Inserts or update given domain object"""
    orm = await self.model_to_orm(model)
    if orm is not None and self.session is not None:
        self._log_info(f"SQL (Upsert) {orm}")
        self.session.merge(orm)
        self.session.commit()
        return True
    return False
```

meth `db_session_delete(model: DaaSEntity) -> bool` async

Delete specified domain object

Source code in `src/app/daas/db/db_api.py`

```
async def db_session_delete(self, model: DaaSEntity) -> bool:
    """Delete specified domain object"""
    if self.session is None:
        return False
    tab: Optional[Table] = await self.get_table_by_domain(model)
    pk = await self._get_table_pk(tab)
    col = await self._get_pk_column_name(tab, pk)
    mapping = await self._get_table_mapping_orm(tab)
    data = model.get_data()
    if model is None or tab is None or pk is None or mapping is None:
        return False
    if col not in data:
        self._log_error(f"Column {col} not contained in {model.get_data()}")
        return False

    entity = self.session.query(mapping).filter(pk == data[col]).first()
    if entity:
        self._log_info(f"SQL (Delete) {entity}")
        self.session.delete(entity)
        self.session.commit()
    return True
```

meth `db_session_select_all(tablename: Tablenames) -> list[ORMEntity]` async

Select all from specified table

Source code in `src/app/daas/db/db_api.py`

```
153  async def db_session_select_all(self, tablename: Tablenames) -> list[ORMEntity]:
154      """Select all from specified table"""
155      return await self.db_session_select(tablename)
```

meth `db_session_select_one(tablename: Tablenames, filter: Optional[TextClause] = None) -> Optional[ORMEntity]` async

Select one from specified table

Source code in `src/app/daas/db/db_api.py`

```
157  async def db_session_select_one(
158      self, tablename: Tablenames, filter: Optional[TextClause] = None
159  ) -> Optional[ORMEntity]:
160      """Select one from specified table"""
161      result = await self.db_session_select(tablename, filter)
162      if len(result) >= 1:
163          return result[0]
164      return None
```

meth `db_session_select_max(tablename: Tablenames, column_int: Colnames) -> Optional[ORMEntity]` async

Select max value, from specified table and column

Source code in `src/app/daas/db/db_api.py`

```
166  async def db_session_select_max(
167      self, tablename: Tablenames, column_int: Colnames
168  ) -> Optional[ORMEntity]:
169      """Select max value, from specified table and column"""
170      mapping: Optional[Type[object]] = await self.get_mapping_orm(tablename.value)
171      if mapping is None:
172          return None
173      flt = text(f"{column_int.value} > 0 ORDER BY {column_int.value} Desc")
174      return await self.db_session_select_one(tablename, flt)
```

meth `db_session_select(tablename: Tablenames, filter: Optional[TextClause] = None) -> list[ORMEntity]` async

Select from specified table

Source code in `src/app/daas/db/db_api.py`

```
176  async def db_session_select(
177      self, tablename: Tablenames, filter: Optional[TextClause] = None
178  ) -> list[ORMEntity]:
179      """Select from specified table"""
180      mapping: Optional[Type[object]] = await self.get_mapping_orm(tablename.value)
181      if mapping is None:
182          return []
183      self._log_info(f"SQL (Select) {tablename}")
184      qresult = await self.db_session_query(mapping, filter)
185      if qresult is None:
186          return []
187      return [x for x in qresult if isinstance(x, ORMEntity)]
```

meth `get_table(name: str) -> Optional[Table]` async

Returns associated table object if available

Source code in `src/app/daas/db/db_api.py`

```
189   async def get_table(self, name: str) -> Optional[Table]:
190       """Returns associated table object if available"""
191       if name in self.metadata.tables:
192           return self.metadata.tables[name]
193       return None
```

meth `get_table_by_orm(orm: ORMEntity) -> Optional[Table]` `async`

Gets Table associated with specified orm type

Source code in `src/app/daas/db/db_api.py`

```
195   async def get_table_by_orm(self, orm: ORMEntity) -> Optional[Table]:
196       """Gets Table associated with specified orm type"""
197       return orm.get_table()
```

meth `get_table_by_domain(model: DaaSEntity) -> Optional[Table]` `async`

Gets Table associated with specified model type

Source code in `src/app/daas/db/db_api.py`

```
199   async def get_table_by_domain(self, model: DaaSEntity) -> Optional[Table]:
200       """Gets Table associated with specified model type"""
201       orm = await self.model_to_orm(model)
202       if orm is not None:
203           return orm.get_table()
204       return None
```

meth `get_mapping_orm(name: str) -> Optional[ORMEntity]` `async`

Get mapping orm

Source code in `src/app/daas/db/db_api.py`

```
206   async def get_mapping_orm(self, name: str) -> Optional[ORMEntity]:
207       """Get mapping orm"""
208       tab: Optional[Table] = await self.get_table(name)
209       return await self._get_table_mapping_orm(tab)
```

meth `get_mapping_domain(name: str) -> Optional[DaaSEntity]` `async`

Get mapping orm

Source code in `src/app/daas/db/db_api.py`

```
211   async def get_mapping_domain(self, name: str) -> Optional[DaaSEntity]:
212       """Get mapping orm"""
213       tab: Optional[Table] = await self.get_table(name)
214       return await self._get_table_mapping_orm(tab)
```

`src.app.daas.db.db_manager` module

Database manager

Classes

class `DatabaseManager(config: DatabaseConfig)`

Bases: `Loggable`

Manages connections to the database

Source code in `src/app/daas/db/db_manager.py`

```
16  def __init__(self, config: DatabaseConfig):
17      Loggable.__init__(self, LogTarget.DB)
18      self.cfg = config
19      self.connected = False
```

Attributes

attr `cfg = config` `instance-attribute`

attr `connected = False` `instance-attribute`

Functions

meth `initialize()`

Source code in `src/app/daas/db/db_manager.py`

```
21  def initialize(self):
22      self.engine = self.__create_engine()
23      if self.engine is not None:
24          self.metadata = ORMEntity.metadata
25          self.metadata.create_all(self.engine)
26          self.api = self._create_api(self.engine, self.metadata)
27      else:
28          raise ValueError("DB-Engine is None")
```

meth `__create_engine() -> Optional[Engine]`

Source code in `src/app/daas/db/db_manager.py`

```
30  def __create_engine(self) -> Optional[Engine]:
31      if self.cfg.db_type == "mariadb":
32          return self.__create_engine_mariadb()
33      if self.cfg.db_type == "sqlite":
34          return self.__create_engine_sqlite()
35      return None
```

meth `__create_engine_mariadb()`

Source code in `src/app/daas/db/db_manager.py`

```
37  def __create_engine_mariadb(self):
38      self._log_info(
39          f"Prepare engine: mariadb ({self.cfg.db_host}:{self.cfg.db_port})"
40      )
41      constring = self.__create_connection_string_mariadb(False)
42      engine = create_engine(constring, echo=self.cfg.enable_echo)
43      try:
44          self.api = self._create_api(engine, ORMEntity.metadata)
45          self.api.connect()
46          self.api.db_session_create(self.cfg.db_name)
47          self.api.disconnect()
48          constring = self.__create_connection_string_mariadb()
49          return create_engine(constring, echo=self.cfg.enable_echo)
50      except Exception as exe:
51          raise Exception(f"Error on db connect: {exe}")
```

meth `__create_engine_sqlite()`

Source code in `src/app/daas/db/db_manager.py`

```
53  def __create_engine_sqlite(self):
54      try:
55          self._log_info(f"Prepare engine: sqlite ({self.cfg.db_name}.sqlite)")
56          constring = self.__create_connection_string_sqlite()
57          return create_engine(constring, echo=self.cfg.enable_echo)
58      except Exception as exe:
59          raise Exception(f"Error on db connect: {exe}")
```

meth `__create_connection_string_mariadb(with_db: bool = True)`

Source code in `src/app/daas/db/db_manager.py`

```
61  def __create_connection_string_mariadb(self, with_db: bool = True):
62      dbtype = self.cfg.db_type
63      datapath = f"{self.cfg.data_path}/{dbtype}"
64      os.makedirs(datapath, exist_ok=True)
65      dbname = self.cfg.db_name
66      user = self.cfg.db_user
67      password = self.cfg.db_pass
68      host = self.cfg.db_host
69      port = self.cfg.db_port
70      if with_db is True:
71          return f"mysql+pymysql://{user}:{password}@{host}:{port}/{dbname}"
72      return f"mysql+pymysql://{user}:{password}@{host}:{port}/"
```

meth `__create_connection_string_sqlite()`

Source code in `src/app/daas/db/db_manager.py`

```
74  def __create_connection_string_sqlite(self):
75      dbtype = self.cfg.db_type
76      datapath = f"{self.cfg.data_path}/{dbtype}"
77      os.makedirs(datapath, exist_ok=True)
78      dbname = f"{self.cfg.db_name}.sqlite3"
79      full = f"{datapath}/{dbname}"
80      constring = f"sqlite:///{full}"
81      return constring
```

meth `connect() -> bool` async

Connects all databases

Source code in `src/app/daas/db/db_manager.py`

```
86  async def connect(self) -> bool:
87      """Connects all databases"""
88      if self.connected is False:
89          self.initialize()
90          self.api.connect()
91          self.connected = True
92          return self.connected
93      return False
```

meth `disconnect() -> bool` `async`

Disconnect all databases

Source code in `src/app/daas/db/db_manager.py`

```
95  async def disconnect(self) -> bool:
96      """Disconnect all databases"""
97      if self.connected is True:
98          self.api.disconnect()
99          self.connected = False
100         return True
101     return False
```

`src.app.daas.db.db_mappings` module

Database persistance objects

Attributes

attr `TableMapping = namedtuple('TableMapping', ['id', 'table'])` `module-attribute`

Classes

class `ORMMappingType`

Bases: `Enum`

Attributes

attr `Unknown = TableMapping(0, '')` `class-attribute` `instance-attribute`

attr `Object = TableMapping(1, 'daas_objects')` `class-attribute` `instance-attribute`

attr `Machine = TableMapping(2, 'daas_objects')` `class-attribute` `instance-attribute`

attr `Container = TableMapping(3, 'daas_objects')` `class-attribute` `instance-attribute`

attr `File = TableMapping(4, 'files')` `class-attribute` `instance-attribute`

attr `Application = TableMapping(5, 'template_applications')` `class-attribute` `instance-attribute`

attr `Environment = TableMapping(6, 'environments')` `class-attribute` `instance-attribute`

attr `Instance = TableMapping(7, 'instances')` `class-attribute` `instance-attribute`

attr `GuacamoleConnection = TableMapping(8, 'guacamole_connections')` `class-attribute` `instance-attribute`

attr `Limit = TableMapping(9, 'limits')` `class-attribute` `instance-attribute`

attr `id` `property`

attr `table` `property`

class `TableEntityMapping`

Functions

meth `__new__(*args, **kwargs)`

Source code in `src/app/daas/db/db_mappings.py`

```
58  def __new__(cls, *args, **kwargs):
59      if not cls._instance:
60          cls._instance = super(TableEntityMapping, cls).__new__(cls, *args, **kwargs)
61          cls._instance._mappings_orm = {}
62          cls._instance._mappings_domain = {}
63      return cls._instance
```

meth `add_mapping_orm(table, orm_entity)`

Source code in `src/app/daas/db/db_mappings.py`

```
65  def add_mapping_orm(self, table, orm_entity):
66      if self._mappings_orm is not None and table not in self._mappings_orm:
67          self.__log_info(f"Adding orm {table} with {orm_entity}")
68          self._mappings_orm[table] = orm_entity
```

meth `add_mapping_domain(table, domain_entity)`

Source code in `src/app/daas/db/db_mappings.py`

```
70  def add_mapping_domain(self, table, domain_entity):
71      if self._mappings_domain is not None and table not in self._mappings_domain:
72          self.__log_info(f"Adding mod {table} with {domain_entity}")
73          self._mappings_domain[table] = domain_entity
```

meth `get_orm(table) -> Optional[Type[object]]`

Source code in `src/app/daas/db/db_mappings.py`

```
75  def get_orm(self, table) -> Optional[Type[object]]:
76      if self._mappings_orm is not None and table in self._mappings_orm.keys():
77          return self._mappings_orm[table]
78      return None
```

meth `get_domain(table) -> Optional[Type[object]]`

Source code in `src/app/daas/db/db_mappings.py`

```
80  def get_domain(self, table) -> Optional[Type[object]]:
81      if self._mappings_domain is not None and table in self._mappings_domain.keys():
82          return self._mappings_domain[table]
83      return None
```

meth `__log_info(msg: str)`

Source code in `src/app/daas/db/db_mappings.py`

```
85  def __log_info(self, msg: str):
86      print(f"Mappings: {msg}")
```

Functions

func `ORMModelDomain(etype: ORMMappingType)`

Source code in `src/app/daas/db/db_mappings.py`

```
32  def ORMModelDomain(etype: ORMMappingType):
33      def decorator(cls):
34          cls.__orm_etype = etype
35          mappings = TableEntityMapping()
36          mappings.add_mapping_domain(cls.__orm_etype.table, cls)
37          return cls
38
39      return decorator
```

func `ORMModelPersistance(etype: ORMMappingType)`

Source code in `src/app/daas/db/db_mappings.py`

```
42  def ORMModelPersistance(etype: ORMMappingType):
43      def decorator(cls):
44          cls.__orm_etype = etype
45          mappings = TableEntityMapping()
46          mappings.add_mapping_orm(cls.__orm_etype.table, cls)
47          return cls
48
49      return decorator
```

`src.app.daas.db.db_model` module

Database model objects

Attributes

attr `Base = declarative_base()` `module-attribute`

Classes

class `Tablenames`

Bases: `Enum`

Enum with configured tablenames

Attributes

attr `Obj = 'daas_objects'` `class-attribute` `instance-attribute`

attr `Con = 'guacamole_connections'` `class-attribute` `instance-attribute`

attr `File = 'files'` `class-attribute` `instance-attribute`

attr `Application = 'template_applications'` `class-attribute` `instance-attribute`

attr `Env = 'environments'` `class-attribute` `instance-attribute`

attr `Inst = 'instances'` `class-attribute` `instance-attribute`

attr `Limit = 'limits'` `class-attribute` `instance-attribute`

class `Colnames`

Bases: `Enum`

Enum with configured column names

Attributes

attr `Id = 'id'` `class-attribute` `instance-attribute`

attr `Name = 'name'` `class-attribute` `instance-attribute`

attr `ObjectId = 'id_object'` `class-attribute` `instance-attribute`

attr `BackendId = 'id_backend'` `class-attribute` `instance-attribute`

attr `AppId = 'id_app'` `class-attribute` `instance-attribute`

attr `EnvId = 'id_env'` `class-attribute` `instance-attribute`

attr `ConnectionId = 'id_con'` `class-attribute` `instance-attribute`

attr `DockerId = 'id_docker'` `class-attribute` `instance-attribute`

attr `ProxmoxId = 'id_proxmox'` `class-attribute` `instance-attribute`

attr `Owner = 'id_owner'` `class-attribute` `instance-attribute`

attr `Host = 'host'` `class-attribute` `instance-attribute`

attr `ViewerToken = 'viewer_token'` `class-attribute` `instance-attribute`

class `JsonType`

Bases: `TypeDecorator`

Wrapper to convert json objects into string

Attributes

attr `impl = Text` `class-attribute` `instance-attribute`

Functions

meth `process_bind_param(value, dialect)`

Source code in `src/app/daas/db/db_model.py`

```
53  def process_bind_param(self, value, dialect):
54      if value is not None:
55          value = json.dumps(value)
56      return value
```

meth `process_result_value(value, dialect)`

Source code in `src/app/daas/db/db_model.py`

```
58  def process_result_value(self, value, dialect):
59      if value is not None:
60          value = json.loads(value)
61      return value
```

class `ORMEntity(**kwargs)`

Bases: `Base`, `Loggable`

Baseclass for every orm-mapped object

Source code in `src/app/daas/db/db_model.py`

```
73  def __init__(self, **kwargs):
74      Base.__init__(self)
75      Loggable.__init__(self, LogTarget.DB)
76      self.parentcls: Type[object] = self.__class__
77      super(ORMEntity, self).__init__(**kwargs)
```

Attributes

attr `__abstract__ = True` `class-attribute` `instance-attribute`

attr `parentcls: Type[object] = self.__class__` `instance-attribute`

Functions

meth `get_table() -> Optional[Table]`

Source code in `src/app/daas/db/db_model.py`

```
79  def get_table(self) -> Optional[Table]:
80      if self.__tablename__ in self.metadata.tables:
81          return self.metadata.tables[self.__tablename__]
82      return None
```

meth `get_data(logme: bool = False)`

Source code in `src/app/daas/db/db_model.py`

```
84  def get_data(self, logme: bool = False):
85      result = {k: v for k, v in self.__dict__.items() if k not in ORMEntity.__dict__}
86      return self.clean_dict(result)
```

meth `clean_dict(data: dict)`

Source code in `src/app/daas/db/db_model.py`

```
88  def clean_dict(self, data: dict):
89      if "_sa_instance_state" in data:
90          data.pop("_sa_instance_state")
91      if "__orminfo__" in data:
92          data.pop("__orminfo__")
93      return data
```

meth `__repr__()`

Source code in `src/app/daas/db/db_model.py`

```
95   def __repr__(self):
96       data = self.get_data()
97       clsname = self.__class__.__qualname__
98       info = f"{clsname}("
99       parts = []
100      if "id" in data:
101          parts.append(f"name={data['id']}")
102      if "name" in data:
103          parts.append(f"name={data['name']}")
104      if "id_user" in data:
105          parts.append(f"id_user={data['id_user']}")
106      if "id_owner" in data:
107          parts.append(f"owner={data['id_owner']}")
108      if "os_type" in data:
109          parts.append(f"os={data['os_type']}")
110      if "object_type" in data:
111          parts.append(f"type={data['object_type']}")
112      if "instance" in data:
113          parts.append(f"instance={data['instance']}")
114      return f"{info}{','.join(parts)})"
```

class `ORMObject(**kwargs)`

Bases: `ORMEntity`

Source code in `src/app/daas/db/db_model.py`

```
73   def __init__(self, **kwargs):
74       Base.__init__(self)
75       Loggable.__init__(self, LogTarget.DB)
76       self.parentcls: Type[object] = self.__class__
77       super(ORMEntity, self).__init__(**kwargs)
```

Attributes

attr `__abstract__ = True` `class-attribute` `instance-attribute`

attr `parentcls: Type[object] = self.__class__` `instance-attribute`

attr `__tablename__ = 'daas_objects'` `class-attribute` `instance-attribute`

attr `instances: Mapped[List[ORMInstance]] = relationship(back_populates='app', cascade='all, delete-orphan')` `class-attribute` `instance-attribute`

attr `environments: Mapped[List[ORMEnvironment]] = relationship(back_populates='app', cascade='all, delete-orphan')` `class-attribute` `instance-attribute`

attr `id: Mapped[str] = mapped_column(String(128), primary_key=True)` `class-attribute` `instance-attribute`

attr `id_user: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `id_owner: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `id_proxmox: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `id_docker: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `object_type: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `object_mode: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `object_mode_extended: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `object_state: Mapped[str] = mapped_column(String(1024))` `class-attribute` `instance-attribute`

attr `object_tasks: Mapped[JsonType] = mapped_column(JsonType)` `class-attribute` `instance-attribute`

attr `object_apps: Mapped[JsonType] = mapped_column(JsonType)` `class-attribute` `instance-attribute`

attr `object_target: Mapped[JsonType] = mapped_column(JsonType)` `class-attribute` `instance-attribute`

attr `object_storage: Mapped[str] = mapped_column(String(1024))` `class-attribute` `instance-attribute`

attr `os_type: Mapped[str] = mapped_column(String(32))` `class-attribute` `instance-attribute`

attr `os_wine: Mapped[str] = mapped_column(String(32))` `class-attribute` `instance-attribute`

attr `os_username: Mapped[str] = mapped_column(String(32))` `class-attribute` `instance-attribute`

attr `os_password: Mapped[str] = mapped_column(String(32))` `class-attribute` `instance-attribute`

attr `os_installer: Mapped[str] = mapped_column(String(4096))` `class-attribute` `instance-attribute`

attr `hw_cpus: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `hw_memory: Mapped[int] = mapped_column(BigInteger)` `class-attribute` `instance-attribute`

attr `hw_disksize: Mapped[int] = mapped_column(BigInteger)` class-attribute instance-attribute

attr `ceph_public: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `ceph_shared: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `ceph_user: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `vnc_port_system: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `vnc_port_instance: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `vnc_username: Mapped[str] = mapped_column(String(128))` class-attribute instance-attribute

attr `vnc_password: Mapped[str] = mapped_column(String(128))` class-attribute instance-attribute

attr `viewer_contype: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

attr `viewer_resolution: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

attr `viewer_resize: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

attr `viewer_scale: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

attr `viewer_dpi: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `viewer_colors: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `viewer_force_lossless: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `extra_args: Mapped[str] = mapped_column(String(128))` class-attribute instance-attribute

Functions

meth `get_table() -> Optional[Table]`

Source code in `src/app/daas/db/db_model.py`

```
79  def get_table(self) -> Optional[Table]:
80      if self.__tablename__ in self.metadata.tables:
81          return self.metadata.tables[self.__tablename__]
82      return None
```

meth `get_data(logme: bool = False)`

Source code in `src/app/daas/db/db_model.py`

```
84  def get_data(self, logme: bool = False):
85      result = {k: v for k, v in self.__dict__.items() if k not in ORMEntity.__dict__}
86      return self.clean_dict(result)
```

meth `clean_dict(data: dict)`

Source code in `src/app/daas/db/db_model.py`

```
88  def clean_dict(self, data: dict):
89      if "_sa_instance_state" in data:
90          data.pop("_sa_instance_state")
91      if "__orminfo__" in data:
92          data.pop("__orminfo__")
93      return data
```

meth `__repr__()`

Source code in `src/app/daas/db/db_model.py`

```
95  def __repr__(self):
96      data = self.get_data()
97      clsname = self.__class__.__qualname__
98      info = f"{clsname}("
99      parts = []
100     if "id" in data:
101         parts.append(f"name={data['id']}")
102     if "name" in data:
103         parts.append(f"name={data['name']}")
104     if "id_user" in data:
105         parts.append(f"id_user={data['id_user']}")
106     if "id_owner" in data:
107         parts.append(f"owner={data['id_owner']}")
108     if "os_type" in data:
109         parts.append(f"os={data['os_type']}")
110     if "object_type" in data:
111         parts.append(f"type={data['object_type']}")
112     if "instance" in data:
113         parts.append(f"instance={data['instance']}")
114     return f"{info}{','.join(parts)})"
```

class `ORMEnvironment(**kwargs)`

Bases: `ORMEntity`

Source code in `src/app/daas/db/db_model.py`

```
73  def __init__(self, **kwargs):
74      Base.__init__(self)
75      Loggable.__init__(self, LogTarget.DB)
76      self.parentcls: Type[object] = self.__class__
77      super(ORMEntity, self).__init__(**kwargs)
```

Attributes

attr `__abstract__ = True` `class-attribute` `instance-attribute`

attr `parentcls: Type[object] = self.__class__` `instance-attribute`

attr `__tablename__ = 'environments'` `class-attribute` `instance-attribute`

attr `fk_objs = f'{Tablenames.Obj.value}.id'` `class-attribute` `instance-attribute`

attr `app: Mapped[ORMObject] = relationship(back_populates='environments')` `class-attribute` `instance-attribute`

attr `inst: Mapped[ORMInstance] = relationship(back_populates='env')` `class-attribute` `instance-attribute`

attr `id: Mapped[str] = mapped_column(String(128), primary_key=True)` `class-attribute` `instance-attribute`

attr `id_object: Mapped[str] = mapped_column(String(128), ForeignKey(fk_objs))` `class-attribute` `instance-attribute`

attr `id_backend: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `name: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `state: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `env_tasks: Mapped[JsonType] = mapped_column(JsonType)` `class-attribute` `instance-attribute`

attr `env_apps: Mapped[JsonType] = mapped_column(JsonType)` `class-attribute` `instance-attribute`

attr `env_target: Mapped[JsonType] = mapped_column(JsonType)` `class-attribute` `instance-attribute`

attr `created_at: Mapped[str] = mapped_column(String(32))` `class-attribute` `instance-attribute`

Functions

meth `get_table() -> Optional[Table]`

Source code in `src/app/daas/db/db_model.py`

```
79  def get_table(self) -> Optional[Table]:
80      if self.__tablename__ in self.metadata.tables:
81          return self.metadata.tables[self.__tablename__]
82      return None
```

meth `get_data(logme: bool = False)`

Source code in `src/app/daas/db/db_model.py`

```
84  def get_data(self, logme: bool = False):
85      result = {k: v for k, v in self.__dict__.items() if k not in ORMEntity.__dict__}
86      return self.clean_dict(result)
```

meth `clean_dict(data: dict)`

Source code in `src/app/daas/db/db_model.py`

```
88  def clean_dict(self, data: dict):
89      if "_sa_instance_state" in data:
90          data.pop("_sa_instance_state")
91      if "__orminfo__" in data:
92          data.pop("__orminfo__")
93      return data
```

meth `__repr__()`

Source code in `src/app/daas/db/db_model.py`

```
95  def __repr__(self):
96      data = self.get_data()
97      clsname = self.__class__.__qualname__
98      info = f"{clsname}("
99      parts = []
100     if "id" in data:
101         parts.append(f"name={data['id']}")
102     if "name" in data:
103         parts.append(f"name={data['name']}")
104     if "id_user" in data:
105         parts.append(f"id_user={data['id_user']}")
106     if "id_owner" in data:
107         parts.append(f"owner={data['id_owner']}")
108     if "os_type" in data:
109         parts.append(f"os={data['os_type']}")
110     if "object_type" in data:
111         parts.append(f"type={data['object_type']}")
112     if "instance" in data:
113         parts.append(f"instance={data['instance']}")
114     return f"{info}{','.join(parts)})"
```

class `ORMGuacamoleConnection(**kwargs)`

Bases: `ORMEntity`

Source code in `src/app/daas/db/db_model.py`

```
73  def __init__(self, **kwargs):
74      Base.__init__(self)
75      Loggable.__init__(self, LogTarget.DB)
76      self.parentcls: Type[object] = self.__class__
77      super(ORMEntity, self).__init__(**kwargs)
```

Attributes

attr `__abstract__ = True` `class-attribute` `instance-attribute`

attr `parentcls: Type[object] = self.__class__` `instance-attribute`

attr `__tablename__ = 'guacamole_connections'` `class-attribute` `instance-attribute`

attr `obj = relationship('ORMInstance', back_populates='con')` `class-attribute` `instance-attribute`

attr `id: Mapped[str] = mapped_column(String(128), primary_key=True)` `class-attribute` `instance-attribute`

attr `viewer_url: Mapped[str] = mapped_column(String(256))` `class-attribute` `instance-attribute`

attr `viewer_token: Mapped[str] = mapped_column(String(256))` `class-attribute` `instance-attribute`

attr `user: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `password: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `protocol: Mapped[str] = mapped_column(String(32))` `class-attribute` `instance-attribute`

attr `hostname: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `port: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

Functions

meth `get_table() -> Optional[Table]`

Source code in `src/app/daas/db/db_model.py`

```
79  def get_table(self) -> Optional[Table]:
80      if self.__tablename__ in self.metadata.tables:
81          return self.metadata.tables[self.__tablename__]
82      return None
```

meth `get_data(logme: bool = False)`

Source code in `src/app/daas/db/db_model.py`

```
84  def get_data(self, logme: bool = False):
85      result = {k: v for k, v in self.__dict__.items() if k not in ORMEntity.__dict__}
86      return self.clean_dict(result)
```

meth `clean_dict(data: dict)`

Source code in `src/app/daas/db/db_model.py`

```
88  def clean_dict(self, data: dict):
89      if "_sa_instance_state" in data:
90          data.pop("_sa_instance_state")
91      if "__orminfo__" in data:
92          data.pop("__orminfo__")
93      return data
```

meth `__repr__()`

Source code in `src/app/daas/db/db_model.py`

```
95   def __repr__(self):
96       data = self.get_data()
97       clsname = self.__class__.__qualname__
98       info = f"{clsname}("
99       parts = []
100      if "id" in data:
101          parts.append(f"name={data['id']}")
102      if "name" in data:
103          parts.append(f"name={data['name']}")
104      if "id_user" in data:
105          parts.append(f"id_user={data['id_user']}")
106      if "id_owner" in data:
107          parts.append(f"owner={data['id_owner']}")
108      if "os_type" in data:
109          parts.append(f"os={data['os_type']}")
110      if "object_type" in data:
111          parts.append(f"type={data['object_type']}")
112      if "instance" in data:
113          parts.append(f"instance={data['instance']}")
114      return f"{info}{','.join(parts)})"
```

class `ORMInstance(**kwargs)`

Bases: `ORMEntity`

Source code in `src/app/daas/db/db_model.py`

```
209   def __init__(self, **kwargs):
210       if "config_inst" in kwargs:
211           kwargs.pop("config_inst")
212       if "config_ssh" in kwargs:
213           kwargs.pop("config_ssh")
214       super().__init__(**kwargs)
```

Attributes

attr `__abstract__ = True` class-attribute instance-attribute

attr `parentcls: Type[object] = self.__class__` instance-attribute

attr `__tablename__ = 'instances'` class-attribute instance-attribute

attr `fk_objs = f'{Tablenames.Obj.value}.id'` class-attribute instance-attribute

attr `fk_envs = f'{Tablenames.Env.value}.id'` class-attribute instance-attribute

attr `fk_cons = f'{Tablenames.Con.value}.id'` class-attribute instance-attribute

attr `app: Mapped[ORMObject] = relationship(back_populates='instances')` class-attribute instance-attribute

attr `env: Mapped[Optional[ORMEnvironment]] = relationship(back_populates='inst', single_parent=True)` class-attribute instance-attribute

attr `con: Mapped[Optional[ORMGuacamoleConnection]] = relationship(back_populates='obj', single_parent=True, cascade='all, delete-orphan')` class-attribute instance-attribute

attr `id: Mapped[str] = mapped_column(String(128), primary_key=True)` class-attribute instance-attribute

attr `id_app: Mapped[str] = mapped_column(String(128), ForeignKey(fk_objs))` class-attribute instance-attribute

attr `id_env: Mapped[str] = mapped_column(String(128), ForeignKey(fk_envs), nullable=True)` class-attribute instance-attribute

attr `id_con: Mapped[str] = mapped_column(String(128), ForeignKey(fk_cons), nullable=True)` class-attribute instance-attribute

attr `id_owner: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `host: Mapped[str] = mapped_column(String(128), nullable=True)` class-attribute instance-attribute

attr `container: Mapped[str] = mapped_column(String(128), nullable=True)` class-attribute instance-attribute

attr `created_at: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

attr `connected_at: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

attr `booted_at: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

Functions

meth `get_table() -> Optional[Table]`

Source code in `src/app/daas/db/db_model.py`

```
79    def get_table(self) -> Optional[Table]:
80        if self.__tablename__ in self.metadata.tables:
81            return self.metadata.tables[self.__tablename__]
82        return None
```

meth `get_data(logme: bool = False)`

Source code in `src/app/daas/db/db_model.py`

```
84    def get_data(self, logme: bool = False):
85        result = {k: v for k, v in self.__dict__.items() if k not in ORMEntity.__dict__}
86        return self.clean_dict(result)
```

meth `clean_dict(data: dict)`

Source code in `src/app/daas/db/db_model.py`

```
88  def clean_dict(self, data: dict):
89      if "_sa_instance_state" in data:
90          data.pop("_sa_instance_state")
91      if "__orminfo__" in data:
92          data.pop("__orminfo__")
93      return data
```

meth `__repr__()`

Source code in `src/app/daas/db/db_model.py`

```
95  def __repr__(self):
96      data = self.get_data()
97      clsname = self.__class__.__qualname__
98      info = f"{clsname}("
99      parts = []
100     if "id" in data:
101         parts.append(f"name={data['id']}")
102     if "name" in data:
103         parts.append(f"name={data['name']}")
104     if "id_user" in data:
105         parts.append(f"id_user={data['id_user']}")
106     if "id_owner" in data:
107         parts.append(f"owner={data['id_owner']}")
108     if "os_type" in data:
109         parts.append(f"os={data['os_type']}")
110     if "object_type" in data:
111         parts.append(f"type={data['object_type']}")
112     if "instance" in data:
113         parts.append(f"instance={data['instance']}")
114     return f"{info}{','.join(parts)})"
```

class `ORMFile(**kwargs)`

Bases: `ORMEntity`

Source code in `src/app/daas/db/db_model.py`

```
73  def __init__(self, **kwargs):
74      Base.__init__(self)
75      Loggable.__init__(self, LogTarget.DB)
76      self.parentcls: Type[object] = self.__class__
77      super(ORMEntity, self).__init__(**kwargs)
```

Attributes

attr `__abstract__ = True` `class-attribute` `instance-attribute`

attr `parentcls: Type[object] = self.__class__` `instance-attribute`

attr `__tablename__ = 'files'` `class-attribute` `instance-attribute`

attr `id: Mapped[str] = mapped_column(String(128), primary_key=True)` `class-attribute` `instance-attribute`

attr `id_owner: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `os_type: Mapped[str] = mapped_column(String(32))` `class-attribute` `instance-attribute`

attr `name: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `version: Mapped[str] = mapped_column(String(128))` `class-attribute` `instance-attribute`

attr `localpath: Mapped[str] = mapped_column(String(1024))` `class-attribute` `instance-attribute`

attr `remotepath: Mapped[str] = mapped_column(String(1024))` `class-attribute` `instance-attribute`

attr `filesize: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `created_at: Mapped[str] = mapped_column(String(32))` `class-attribute` `instance-attribute`

Functions

meth `get_table() -> Optional[Table]`

Source code in `src/app/daas/db/db_model.py`

```
79  def get_table(self) -> Optional[Table]:
80      if self.__tablename__ in self.metadata.tables:
81          return self.metadata.tables[self.__tablename__]
82      return None
```

meth `get_data(logme: bool = False)`

Source code in `src/app/daas/db/db_model.py`

```
84  def get_data(self, logme: bool = False):
85      result = {k: v for k, v in self.__dict__.items() if k not in ORMEntity.__dict__}
86      return self.clean_dict(result)
```

meth `clean_dict(data: dict)`

Source code in `src/app/daas/db/db_model.py`

```
88  def clean_dict(self, data: dict):
89      if "_sa_instance_state" in data:
90          data.pop("_sa_instance_state")
91      if "__orminfo__" in data:
92          data.pop("__orminfo__")
93      return data
```

meth `__repr__()`

Source code in `src/app/daas/db/db_model.py`

```
95   def __repr__(self):
96       data = self.get_data()
97       clsname = self.__class__.__qualname__
98       info = f"{clsname}("
99       parts = []
100      if "id" in data:
101          parts.append(f"name={data['id']}")
102      if "name" in data:
103          parts.append(f"name={data['name']}")
104      if "id_user" in data:
105          parts.append(f"id_user={data['id_user']}")
106      if "id_owner" in data:
107          parts.append(f"owner={data['id_owner']}")
108      if "os_type" in data:
109          parts.append(f"os={data['os_type']}")
110      if "object_type" in data:
111          parts.append(f"type={data['object_type']}")
112      if "instance" in data:
113          parts.append(f"instance={data['instance']}")
114      return f"{info}{','.join(parts)})"
```

class `ORMApplication(**kwargs)`

Bases: `ORMEntity`

Source code in `src/app/daas/db/db_model.py`

```
73  def __init__(self, **kwargs):
74      Base.__init__(self)
75      Loggable.__init__(self, LogTarget.DB)
76      self.parentcls: Type[object] = self.__class__
77      super(ORMEntity, self).__init__(**kwargs)
```

Attributes

attr `__abstract__ = True` class-attribute instance-attribute

attr `parentcls: Type[object] = self.__class__` instance-attribute

attr `__tablename__ = 'template_applications'` class-attribute instance-attribute

attr `fk_files = f'{Tablenames.File.value}.id'` class-attribute instance-attribute

attr `fk_objs = f'{Tablenames.Obj.value}.id'` class-attribute instance-attribute

attr `id: Mapped[str] = mapped_column(String(128), primary_key=True)` class-attribute instance-attribute

attr `id_owner: Mapped[int] = mapped_column()` class-attribute instance-attribute

attr `id_file: Mapped[str] = mapped_column(String(128), ForeignKey(fk_files), nullable=True)` class-attribute instance-attribute

attr `id_template: Mapped[str] = mapped_column(String(128), ForeignKey(fk_objs), nullable=True)` class-attribute instance-attribute

attr `os_type: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

attr `name: Mapped[str] = mapped_column(String(128))` class-attribute instance-attribute

attr `installer: Mapped[str] = mapped_column(String(1024))` class-attribute instance-attribute

attr `installer_args: Mapped[str] = mapped_column(String(1024))` class-attribute instance-attribute

attr `installer_type: Mapped[str] = mapped_column(String(1024))` class-attribute instance-attribute

attr `target: Mapped[str] = mapped_column(String(1024))` class-attribute instance-attribute

attr `target_args: Mapped[str] = mapped_column(String(1024))` class-attribute instance-attribute

attr `version: Mapped[str] = mapped_column(String(128))` class-attribute instance-attribute

attr `created_at: Mapped[str] = mapped_column(String(32))` class-attribute instance-attribute

Functions

meth `get_table() -> Optional[Table]`

Source code in `src/app/daas/db/db_model.py`

```
79  def get_table(self) -> Optional[Table]:
80      if self.__tablename__ in self.metadata.tables:
81          return self.metadata.tables[self.__tablename__]
82      return None
```

meth `get_data(logme: bool = False)`

Source code in `src/app/daas/db/db_model.py`

```
84  def get_data(self, logme: bool = False):
85      result = {k: v for k, v in self.__dict__.items() if k not in ORMEntity.__dict__}
86      return self.clean_dict(result)
```

meth `clean_dict(data: dict)`

Source code in `src/app/daas/db/db_model.py`

```
88  def clean_dict(self, data: dict):
89      if "_sa_instance_state" in data:
90          data.pop("_sa_instance_state")
91      if "__orminfo__" in data:
92          data.pop("__orminfo__")
93      return data
```

meth `__repr__()`

Source code in `src/app/daas/db/db_model.py`

```
95  def __repr__(self):
96      data = self.get_data()
97      clsname = self.__class__.__qualname__
98      info = f"{clsname}("
99      parts = []
100     if "id" in data:
101         parts.append(f"name={data['id']}")
102     if "name" in data:
103         parts.append(f"name={data['name']}")
104     if "id_user" in data:
105         parts.append(f"id_user={data['id_user']}")
106     if "id_owner" in data:
107         parts.append(f"owner={data['id_owner']}")
108     if "os_type" in data:
109         parts.append(f"os={data['os_type']}")
110     if "object_type" in data:
111         parts.append(f"type={data['object_type']}")
112     if "instance" in data:
113         parts.append(f"instance={data['instance']}")
114     return f"{info}{','.join(parts)})"
```

class `ORMLimit(**kwargs)`

Bases: `ORMEntity`

Source code in `src/app/daas/db/db_model.py`

```
73  def __init__(self, **kwargs):
74      Base.__init__(self)
75      Loggable.__init__(self, LogTarget.DB)
76      self.parentcls: Type[object] = self.__class__
77      super(ORMEntity, self).__init__(**kwargs)
```

Attributes

attr `__abstract__ = True` `class-attribute` `instance-attribute`

attr `parentcls: Type[object] = self.__class__` `instance-attribute`

attr `__tablename__ = 'limits'` `class-attribute` `instance-attribute`

attr `id_owner: Mapped[int] = mapped_column(primary_key=True)` `class-attribute` `instance-attribute`

attr `vm_max: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `container_max: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `obj_max: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `cpu_max: Mapped[int] = mapped_column()` `class-attribute` `instance-attribute`

attr `mem_max: Mapped[int] = mapped_column(BigInteger)` `class-attribute` `instance-attribute`

attr `dsk_max: Mapped[int] = mapped_column(BigInteger)` `class-attribute` `instance-attribute`

Functions

meth `get_table() -> Optional[Table]`

Source code in `src/app/daas/db/db_model.py`

```
79  def get_table(self) -> Optional[Table]:
80      if self.__tablename__ in self.metadata.tables:
81          return self.metadata.tables[self.__tablename__]
82      return None
```

meth `get_data(logme: bool = False)`

Source code in `src/app/daas/db/db_model.py`

```
84  def get_data(self, logme: bool = False):
85      result = {k: v for k, v in self.__dict__.items() if k not in ORMEntity.__dict__}
86      return self.clean_dict(result)
```

meth `clean_dict(data: dict)`

Source code in `src/app/daas/db/db_model.py`

```python
def clean_dict(self, data: dict):
    if "_sa_instance_state" in data:
        data.pop("_sa_instance_state")
    if "__orminfo__" in data:
        data.pop("__orminfo__")
    return data
```

meth `__repr__()`

Source code in `src/app/daas/db/db_model.py`

```python
def __repr__(self):
    data = self.get_data()
    clsname = self.__class__.__qualname__
    info = f"{clsname}("
    parts = []
    if "id" in data:
        parts.append(f"name={data['id']}")
    if "name" in data:
        parts.append(f"name={data['name']}")
    if "id_user" in data:
        parts.append(f"id_user={data['id_user']}")
    if "id_owner" in data:
        parts.append(f"owner={data['id_owner']}")
    if "os_type" in data:
        parts.append(f"os={data['os_type']}")
    if "object_type" in data:
        parts.append(f"type={data['object_type']}")
    if "instance" in data:
        parts.append(f"instance={data['instance']}")
    return f"{info}{','.join(parts)})"
```

Functions

func `create_model(orm: ORMEntity) -> Optional[DaaSEntity]`

Converts orm object to model object

Source code in `src/app/daas/db/db_model.py`

```python
def create_model(orm: ORMEntity) -> Optional[DaaSEntity]:
    """Converts orm object to model object"""

    if hasattr(orm, "__tablename__"):
        tabname: str = getattr(orm, "__tablename__")
        mappings = TableEntityMapping()
        mapped_cls = mappings.get_domain(tabname)
        if mapped_cls is not None:
            data = orm.get_data()
            data = check_data(orm, data)
            obj = mapped_cls(**data)
            if isinstance(obj, DaaSEntity):
                return obj
```

func `check_data(orm: ORMEntity, data: dict) -> dict`

Source code in `src/app/daas/db/db_model.py`

```python
def check_data(orm: ORMEntity, data: dict) -> dict:
    from app.daas.objects.object_container import ContainerObject
    from app.daas.objects.object_machine import MachineObject

    if isinstance(orm, ORMInstance):
        testapp = getattr(orm, "app")
        testenv = getattr(orm, "env")
        for attr in vars(orm):
            val = getattr(orm, attr)
            if isinstance(val, ORMEntity):
                newval = create_model(val)
                if isinstance(val, ORMObject):
                    if newval is not None:
                        if val.object_type == "vm":
                            data[attr] = MachineObject(**newval.get_data())
                        else:
                            data[attr] = ContainerObject(**newval.get_data())
                else:
                    data[attr] = newval
    return data
```

func `check_model_childs(model: dict) -> dict`

Source code in `src/app/daas/db/db_model.py`

```python
def check_model_childs(model: dict) -> dict:

    for k, v in model.items():
        if isinstance(v, ORMEntity):
            print(f"Found child model {model}")
            sub = create_model(v)
            model[k] = sub
        if isinstance(v, List):
            newv = []
            for el in v:
                if isinstance(el, ORMEntity):
                    sub = create_orm(el)
                    newv.append(sub)
            model[k] = newv
    return model
```

func `create_orm(model: DaaSEntity) -> Optional[ORMEntity]`

Converts model object to orm object

Source code in `src/app/daas/db/db_model.py`

```python
def create_orm(model: DaaSEntity) -> Optional[ORMEntity]:
    """Converts model object to orm object"""
    if hasattr(model.__class__, "__orm_etype"):
        etype: ORMMappingType = getattr(model.__class__, "__orm_etype")
        mappings = TableEntityMapping()
        mapped_cls = mappings.get_orm(etype.table)
        if mapped_cls is not None:
            data = model.get_data()
            data = check_orm_childs(data)
            obj = mapped_cls(**data)
            if isinstance(obj, ORMEntity):
                return obj
    else:
        raise ValueError("Object is not persistable")
```

func `check_orm_childs(orm: dict) -> dict`

Source code in `src/app/daas/db/db_model.py`

```
368  def check_orm_childs(orm: dict) -> dict:
369
370      for k, v in orm.items():
371          if isinstance(v, DaaSEntity):
372              sub = create_orm(v)
373              orm[k] = sub
374          if isinstance(v, List):
375              newv = []
376              for el in v:
377                  if isinstance(el, DaaSEntity):
378                      sub = create_orm(el)
379                      newv.append(sub)
380                  else:
381                      newv.append(el)
382              orm[k] = newv
383      return orm
```

`src.app.daas.db.db_repos` module

Repository components reflecting database tables

Attributes

attr `T = TypeVar('T', bound='DaaSEntity')` `module-attribute`

Classes

class `RepositoryBase()`

Bases: `Loggable`

Grants access to a certain part of teh database

<details>
<summary>Source code in <code>src/app/daas/db/db_repos.py</code></summary>

```
30  def __init__(self):
31      Loggable.__init__(self, LogTarget.DB)
32      self.dbman = None
33      self.connected = False
```

</details>

Attributes

attr `dbman = None` `instance-attribute`

attr `connected = False` `instance-attribute`

Functions

class `FileRepository()`

Bases: `RepositoryBase`

Grants access to file table

<details>
<summary>Source code in <code>src/app/daas/db/db_repos.py</code></summary>

```
30  def __init__(self):
31      Loggable.__init__(self, LogTarget.DB)
32      self.dbman = None
33      self.connected = False
```

</details>

Attributes

attr `dbman = None` `instance-attribute`

attr `connected = False` `instance-attribute`

Functions

meth `create_file(model: File)` `async`

Insert file

<details>
<summary>Source code in <code>src/app/daas/db/db_repos.py</code></summary>

```
152 async def create_file(self, model: File):
153     """Insert file"""
154     return await self._upsert(model)
```

</details>

meth `update_file(model: File) -> bool` `async`

Update file

Source code in `src/app/daas/db/db_repos.py`

```
156   async def update_file(self, model: File) -> bool:
157       """Update file"""
158       return await self._upsert(model)
```

meth `delete_file(id_pk: str) -> bool` async

Remove file

Source code in `src/app/daas/db/db_repos.py`

```
160   async def delete_file(self, id_pk: str) -> bool:
161       """Remove file"""
162       filter = await self._get_filter_id(id_pk)
163       orm = await self._select_one(Tablenames.File, filter)
164       model = await self._to_model(orm, File)
165       return await self._delete(model)
```

meth `get_file(id_app: str) -> Optional[File]` async

Fetch file by app id

Source code in `src/app/daas/db/db_repos.py`

```
167   async def get_file(self, id_app: str) -> Optional[File]:
168       """Fetch file by app id"""
169       filter = await self._get_filter_id(id_app)
170       api = await self._get_api()
171       orm = await api.db_session_select_one(Tablenames.File, filter)
172       return await self._to_model(orm, File)
```

meth `get_all_files(id_owner: int) -> list[File]` async

Fetch all available files

Source code in `src/app/daas/db/db_repos.py`

```
174   async def get_all_files(self, id_owner: int) -> list[File]:
175       """Fetch all available files"""
176       filter = await self._get_filter_owner_shared(id_owner)
177       selected = await self._select(Tablenames.File, filter)
178       return await self._to_model_list(selected, File)
```

meth `get_user_files(id_owner: int) -> list[File]` async

Fetch user files

Source code in `src/app/daas/db/db_repos.py`

```
180   async def get_user_files(self, id_owner: int) -> list[File]:
181       """Fetch user files"""
182       filter = await self._get_filter_owner(id_owner)
183       selected = await self._select(Tablenames.File, filter)
184       return await self._to_model_list(selected, File)
```

meth `get_shared_files() -> list[File]` async

Fetch shared files

Source code in `src/app/daas/db/db_repos.py`

```
186   async def get_shared_files(self) -> list[File]:
187       """Fetch shared files"""
188       filter = await self._get_filter_column(Colnames.Owner, 0)
189       selected = await self._select(Tablenames.File, filter)
190       return await self._to_model_list(selected, File)
```

class `LimitRepository()`

Bases: `RepositoryBase`

Grants access to limit table

Source code in `src/app/daas/db/db_repos.py`

```
30   def __init__(self):
31       Loggable.__init__(self, LogTarget.DB)
32       self.dbman = None
33       self.connected = False
```

Attributes

attr `dbman = None` `instance-attribute`

attr `connected = False` `instance-attribute`

Functions

meth `create_limit(model: RessourceInfo) -> bool` `async`

Insert limit

Source code in `src/app/daas/db/db_repos.py`

```
196   async def create_limit(self, model: RessourceInfo) -> bool:
197       """Insert limit"""
198       return await self._upsert(model)
```

meth `delete_limit(id_owner: int) -> bool` `async`

Remove limit

Source code in `src/app/daas/db/db_repos.py`

```
200   async def delete_limit(self, id_owner: int) -> bool:
201       """Remove limit"""
202       filter = await self._get_filter_owner(id_owner)
203       orm = await self._select_one(Tablenames.Limit, filter)
204       model = await self._to_model(orm, RessourceInfo)
205       return await self._delete(model)
```

meth `get_limit(id_owner: int) -> Optional[RessourceInfo]` `async`

Fetch limit by owner

Source code in `src/app/daas/db/db_repos.py`

```
207   async def get_limit(self, id_owner: int) -> Optional[RessourceInfo]:
208       """Fetch limit by owner"""
209       filter = await self._get_filter_owner(id_owner)
210       api = await self._get_api()
211       orm = await api.db_session_select_one(Tablenames.Limit, filter)
212       return await self._to_model(orm, RessourceInfo)
```

meth `get_all_limits() -> list[RessourceInfo]` `async`

Fetch all available limits

Source code in `src/app/daas/db/db_repos.py`

```
214   async def get_all_limits(self) -> list[RessourceInfo]:
215       """Fetch all available limits"""
216       selected = await self._select_all(Tablenames.Limit)
217       return await self._to_model_list(selected, RessourceInfo)
```

class `ApplicationRepository()`

Bases: `RepositoryBase`

Grants access to application table

Source code in `src/app/daas/db/db_repos.py`

```
30   def __init__(self):
31       Loggable.__init__(self, LogTarget.DB)
32       self.dbman = None
33       self.connected = False
```

Attributes

attr `dbman = None` `instance-attribute`

attr `connected = False` `instance-attribute`

Functions

meth `create_application(model: ApplicationObject)` `async`

Insert application

Source code in `src/app/daas/db/db_repos.py`

```
223   async def create_application(self, model: ApplicationObject):
224       """Insert application"""
225       return await self._upsert(model)
```

meth `update_application(model: ApplicationObject) -> bool` `async`

Update application

Source code in `src/app/daas/db/db_repos.py`

```
227   async def update_application(self, model: ApplicationObject) -> bool:
228       """Update application"""
229       return await self._upsert(model)
```

meth `delete_application(id_pk: str) -> bool` async

Remove application

Source code in `src/app/daas/db/db_repos.py`

```
231  async def delete_application(self, id_pk: str) -> bool:
232      """Remove application"""
233      filter = await self._get_filter_id(id_pk)
234      orm = await self._select_one(Tablenames.Application, filter)
235      model = await self._to_model(orm, ApplicationObject)
236      return await self._delete(model)
```

meth `get_application(id_app: str) -> Optional[ApplicationObject]` async

Fetch application by app id

Source code in `src/app/daas/db/db_repos.py`

```
238  async def get_application(self, id_app: str) -> Optional[ApplicationObject]:
239      """Fetch application by app id"""
240      filter = await self._get_filter_id(id_app)
241      api = await self._get_api()
242      orm = await api.db_session_select_one(Tablenames.Application, filter)
243      return await self._to_model(orm, ApplicationObject)
```

meth `get_all_applications(id_owner: int) -> list[ApplicationObject]` async

Fetch all available applications

Source code in `src/app/daas/db/db_repos.py`

```
245  async def get_all_applications(self, id_owner: int) -> list[ApplicationObject]:
246      """Fetch all available applications"""
247      filter = await self._get_filter_owner_shared(id_owner)
248      selected = await self._select(Tablenames.Application, filter)
249      return await self._to_model_list(selected, ApplicationObject)
```

meth `get_user_applications(id_owner: int) -> list[ApplicationObject]` async

Fetch user applications

Source code in `src/app/daas/db/db_repos.py`

```
251  async def get_user_applications(self, id_owner: int) -> list[ApplicationObject]:
252      """Fetch user applications"""
253      filter = await self._get_filter_owner(id_owner)
254      selected = await self._select(Tablenames.Application, filter)
255      return await self._to_model_list(selected, ApplicationObject)
```

meth `get_shared_applications() -> list[ApplicationObject]` async

Fetch shared applications

Source code in `src/app/daas/db/db_repos.py`

```
257   async def get_shared_applications(self) -> list[ApplicationObject]:
258       """Fetch shared applications"""
259       filter = await self._get_filter_column(Colnames.Owner, 0)
260       selected = await self._select(Tablenames.Application, filter)
261       return await self._to_model_list(selected, ApplicationObject)
```

class `ConnectionRepository()`

Bases: `RepositoryBase`

Grants access to connections table

Source code in `src/app/daas/db/db_repos.py`

```
30   def __init__(self):
31       Loggable.__init__(self, LogTarget.DB)
32       self.dbman = None
33       self.connected = False
```

Attributes

attr `dbman = None` `instance-attribute`

attr `connected = False` `instance-attribute`

Functions

meth `create_guacamole_connection(model: GuacamoleConnection) -> bool` `async`

Insert connection

Source code in `src/app/daas/db/db_repos.py`

```
267   async def create_guacamole_connection(self, model: GuacamoleConnection) -> bool:
268       """Insert connection"""
269       return await self._upsert(model)
```

meth `update_guacamole_connection(model: GuacamoleConnection) -> bool` `async`

Update connection

Source code in `src/app/daas/db/db_repos.py`

```
271   async def update_guacamole_connection(self, model: GuacamoleConnection) -> bool:
272       """Update connection"""
273       return await self._upsert(model)
```

meth `delete_connection(instance: InstanceObject) -> bool` `async`

Remove connection

Source code in `src/app/daas/db/db_repos.py`

```
async def delete_connection(self, instance: InstanceObject) -> bool:
    """Remove connection"""
    id_con = ""
    if instance.id_con is not None:
        id_con = instance.id_con
        filter = await self._get_filter_id(id_con)
        orm = await self._select_one(Tablenames.Con, filter)
        model = await self._to_model(orm, GuacamoleConnection)
        return await self._delete(model)
    return False
```

meth `get_guacamole_connection(id_pk: str) -> Optional[GuacamoleConnection]` `async`

Fetch connection by id

Source code in `src/app/daas/db/db_repos.py`

```
async def get_guacamole_connection(
    self, id_pk: str
) -> Optional[GuacamoleConnection]:
    """Fetch connection by id"""
    filter = await self._get_filter_id(id_pk)
    api = await self._get_api()
    orm = await api.db_session_select_one(Tablenames.Con, filter)
    return await self._to_model(orm, GuacamoleConnection)
```

meth `get_guacamole_connection_by_token(token: str) -> Optional[GuacamoleConnection]` `async`

Fetch connection by token

Source code in `src/app/daas/db/db_repos.py`

```
async def get_guacamole_connection_by_token(
    self, token: str
) -> Optional[GuacamoleConnection]:
    """Fetch connection by token"""
    filter = await self._get_filter_column(Colnames.ViewerToken, token)
    selected = await self._select_one(Tablenames.Con, filter)
    return await self._to_model(selected, GuacamoleConnection)
```

meth `all_connections() -> list[GuacamoleConnection]` `async`

Fetch all available connections

Source code in `src/app/daas/db/db_repos.py`

```
async def all_connections(self) -> list[GuacamoleConnection]:
    """Fetch all available connections"""
    selected = await self._select_all(Tablenames.Con)
    return await self._to_model_list(selected, GuacamoleConnection)
```

meth `update_connection_protocol(connection: GuacamoleConnection) -> Optional[GuacamoleConnection]` `async`

Update connection

Source code in `src/app/daas/db/db_repos.py`

```
308   async def update_connection_protocol(
309       self, connection: GuacamoleConnection
310   ) -> Optional[GuacamoleConnection]:
311       """Update connection"""
312       id_con = connection.id
313       filter = await self._get_filter_id(id_con)
314       orm = await self._select_one(Tablenames.Con, filter)
315       model = await self._to_model(orm, GuacamoleConnection)
316       if model is not None:
317           model.protocol = connection.protocol
318           model.protocol = connection.protocol
319           model.hostname = connection.hostname
320           model.port = connection.port
321           model.user = connection.user
322           model.password = connection.password
323           if await self._upsert(model) is True:
324               return model
325       return None
```

class `EnvironmentRepository()`

Bases: `RepositoryBase`

Grants access to environments table

Source code in `src/app/daas/db/db_repos.py`

```
30   def __init__(self):
31       Loggable.__init__(self, LogTarget.DB)
32       self.dbman = None
33       self.connected = False
```

Attributes

attr `dbman = None` `instance-attribute`

attr `connected = False` `instance-attribute`

Functions

meth `create_environment(model: Environment) -> bool` `async`

Create environment

Source code in `src/app/daas/db/db_repos.py`

```
331   async def create_environment(self, model: Environment) -> bool:
332       """Create environment"""
333       return await self._upsert(model)
```

meth `update_environment(model: Environment) -> bool` `async`

Update environment

Source code in `src/app/daas/db/db_repos.py`

```
335   async def update_environment(self, model: Environment) -> bool:
336       """Update environment"""
337       return await self._upsert(model)
```

meth `get_environment(id_pk: str) -> Optional[Environment]` `async`

Fetch environment by id

Source code in `src/app/daas/db/db_repos.py`

```
async def get_environment(self, id_pk: str) -> Optional[Environment]:
    """Fetch environment by id"""
    filter = await self._get_filter_id(id_pk)
    api = await self._get_api()
    orm = await api.db_session_select_one(Tablenames.Env, filter)
    return await self._to_model(orm, Environment)
```

meth `get_environments_by_object(obj_id: str) -> list[Environment]` async

Fetch environment by name

Source code in `src/app/daas/db/db_repos.py`

```
async def get_environments_by_object(self, obj_id: str) -> list[Environment]:
    """Fetch environment by name"""
    filter_id = await self._get_filter_statement(Colnames.ObjectId, obj_id)
    filter = await self._get_filter_str(filter_id)
    api = await self._get_api()
    ormlist = await api.db_session_select(Tablenames.Env, filter)
    return await self._to_model_list(ormlist, Environment)
```

meth `get_environment_by_name(obj_id: str, name: str) -> Optional[Environment]` async

Fetch environment by name

Source code in `src/app/daas/db/db_repos.py`

```
async def get_environment_by_name(
    self, obj_id: str, name: str
) -> Optional[Environment]:
    """Fetch environment by name"""
    filter_obj = await self._get_filter_statement(Colnames.ObjectId, obj_id)
    filter_name = await self._get_filter_statement(Colnames.Name, name)
    filter = await self._get_filter_str(f"{filter_name} AND {filter_obj}")
    api = await self._get_api()
    orm = await api.db_session_select_one(Tablenames.Env, filter)
    return await self._to_model(orm, Environment)
```

meth `get_environment_by_backend_id(obj_id: str) -> Optional[Environment]` async

Fetch environment state from its backend id

Source code in `src/app/daas/db/db_repos.py`

```
async def get_environment_by_backend_id(self, obj_id: str) -> Optional[Environment]:
    """Fetch environment state from its backend id"""
    filter = await self._get_filter_column(Colnames.BackendId, obj_id)
    api = await self._get_api()
    orm = await api.db_session_select_one(Tablenames.Env, filter)
    return await self._to_model(orm, Environment)
```

meth `all_environments() -> list[Environment]` async

Fetch all available envs

Source code in `src/app/daas/db/db_repos.py`

```
372   async def all_environments(self) -> list[Environment]:
373       """Fetch all available envs"""
374       selected = await self._select_all(Tablenames.Env)
375       return await self._to_model_list(selected, Environment)
```

meth `all_object_environments(id_env: str) -> list[Environment]` async

Fetch all available envs

Source code in `src/app/daas/db/db_repos.py`

```
377   async def all_object_environments(self, id_env: str) -> list[Environment]:
378       """Fetch all available envs"""
379       flt = await self._get_filter_id(id_env)
380       selected = await self._select(Tablenames.Env, flt)
381       return await self._to_model_list(selected, Environment)
```

meth `delete_environment(env: Environment) -> bool` async

Remove environment

Source code in `src/app/daas/db/db_repos.py`

```
383   async def delete_environment(self, env: Environment) -> bool:
384       """Remove environment"""
385       filter = await self._get_filter_id(env.id)
386       orm = await self._select_one(Tablenames.Env, filter)
387       model = await self._to_model(orm, Environment)
388       return await self._delete(model)
```

class `ObjectRepository()`

Bases: `RepositoryBase`

Grants access to object table

Source code in `src/app/daas/db/db_repos.py`

```
30   def __init__(self):
31       Loggable.__init__(self, LogTarget.DB)
32       self.dbman = None
33       self.connected = False
```

Attributes

attr `dbman = None` instance-attribute

attr `connected = False` instance-attribute

Functions

meth `create_daas_object(model: DaasObject) -> bool` async

Create object

Source code in `src/app/daas/db/db_repos.py`

```
394   async def create_daas_object(self, model: DaasObject) -> bool:
395       """Create object"""
396       return await self._upsert(model)
```

meth `update_daas_object(model: DaasObject) -> bool` `async`

Update object

Source code in `src/app/daas/db/db_repos.py`

```
398  async def update_daas_object(self, model: DaasObject) -> bool:
399      """Update object"""
400      return await self._upsert(model)
```

meth `get_daas_object(id_pk: str) -> Optional[DaasObject]` `async`

Fetch object by id

Source code in `src/app/daas/db/db_repos.py`

```
402  async def get_daas_object(self, id_pk: str) -> Optional[DaasObject]:
403      """Fetch object by id"""
404
405      filter = await self._get_filter_id(id_pk)
406      api = await self._get_api()
407      orm = await api.db_session_select_one(Tablenames.Obj, filter)
408      return await self.__convert_by_object_type(orm)
```

meth `get_daas_object_by_docker_id(id_docker: str) -> Optional[DaasObject]` `async`

Fetch object by docker id

Source code in `src/app/daas/db/db_repos.py`

```
410  async def get_daas_object_by_docker_id(
411      self, id_docker: str
412  ) -> Optional[DaasObject]:
413      """Fetch object by docker id"""
414      filter_id = await self._get_filter_statement(Colnames.DockerId, id_docker)
415      filter = await self._get_filter_str(filter_id)
416      api = await self._get_api()
417      orm = await api.db_session_select_one(Tablenames.Obj, filter)
418      return await self.__convert_by_object_type(orm)
```

meth `get_daas_object_by_proxmox_id(id_proxmox: str) -> Optional[DaasObject]` `async`

Fetch object by proxmox id

Source code in `src/app/daas/db/db_repos.py`

```
420  async def get_daas_object_by_proxmox_id(
421      self, id_proxmox: str
422  ) -> Optional[DaasObject]:
423      """Fetch object by proxmox id"""
424      filter_id = await self._get_filter_statement(Colnames.ProxmoxId, id_proxmox)
425      filter = await self._get_filter_str(filter_id)
426      api = await self._get_api()
427      orm = await api.db_session_select_one(Tablenames.Obj, filter)
428      return await self.__convert_by_object_type(orm)
```

meth `get_daas_objects_by_owner(id_owner: int) -> list[DaasObject]` `async`

Fetch all objects

Source code in `src/app/daas/db/db_repos.py`

```
430  async def get_daas_objects_by_owner(self, id_owner: int) -> list[DaasObject]:
431      """Fetch all objects"""
432      filter = await self._get_filter_owner(id_owner)
433      api = await self._get_api()
434      ormlist = await api.db_session_select(Tablenames.Obj, filter)
435      return await self._to_model_list(ormlist, DaasObject)
```

meth `get_daas_objects_available(id_owner: int) -> list[DaasObject]` async

Fetch all objects

Source code in `src/app/daas/db/db_repos.py`

```
437  async def get_daas_objects_available(self, id_owner: int) -> list[DaasObject]:
438      """Fetch all objects"""
439      filter_obj = await self._get_filter_statement(Colnames.Owner, id_owner)
440      filter_name = await self._get_filter_statement(Colnames.Owner, 0)
441      filter = await self._get_filter_str(f"{filter_name} OR {filter_obj}")
442      api = await self._get_api()
443      ormlist = await api.db_session_select(Tablenames.Obj, filter)
444      return await self.__convert_by_object_types(ormlist)
```

meth `all_daas_objects() -> list[DaasObject]` async

Fetch all available objects

Source code in `src/app/daas/db/db_repos.py`

```
446  async def all_daas_objects(self) -> list[DaasObject]:
447      """Fetch all available objects"""
448      ormlist = await self._select_all(Tablenames.Obj)
449      return await self.__convert_by_object_types(ormlist)
```

meth `delete_daas_object(obj: DaasObject) -> bool` async

Remove object

Source code in `src/app/daas/db/db_repos.py`

```
451  async def delete_daas_object(self, obj: DaasObject) -> bool:
452      """Remove object"""
453      filter = await self._get_filter_id(obj.id)
454      orm = await self._select_one(Tablenames.Obj, filter)
455      model = await self.__convert_by_object_type(orm)
456      return await self._delete(model)
```

meth `suggest_vmid() -> int` async

Suggests new vmid for a new virtual machine. Must be unique and between 100 and 254.

Source code in `src/app/daas/db/db_repos.py`

```
458   async def suggest_vmid(self) -> int:
459       """
460       Suggests new vmid for a new virtual machine.
461       Must be unique and between 100 and 254.
462       """
463       orm = await self._select_max(Tablenames.Obj, Colnames.ProxmoxId)
464       if orm is None:
465           return 100
466       model = await self._to_model(orm, DaasObject)
467       if model is None:
468           self._log_info(f"Error converting orm to model: {orm}")
469           return -1
470       if model.id_proxmox < 100:
471           self._log_info(f"Error invalid vmid found: {model.id_proxmox}")
472           return -1
473       if model.id_proxmox >= 100 and model.id_proxmox < 254:
474           return model.id_proxmox + 1
475       self._log_info("Not enough proxmox ids left")
476       return -1
```

meth `__convert_by_object_type(orm: ORMEntity) -> Optional[DaasObject]` async

Source code in `src/app/daas/db/db_repos.py`

```
478   async def __convert_by_object_type(self, orm: ORMEntity) -> Optional[DaasObject]:
479       from app.daas.objects.object_container import ContainerObject
480       from app.daas.objects.object_machine import MachineObject
481
482       if orm is not None and isinstance(orm, ORMObject):
483           if orm.object_type == "vm":
484               return await self._to_model(orm, MachineObject)
485           elif orm.object_type == "container":
486               contmodel = await self._to_model(orm, DaasObject)
487               if contmodel is not None:
488                   contmodel = ContainerObject(**contmodel.get_data())
489                   return contmodel
490       return None
```

meth `__convert_by_object_types(ormlist: list[ORMEntity]) -> list[DaasObject]` async

Source code in `src/app/daas/db/db_repos.py`

```
492   async def __convert_by_object_types(
493       self, ormlist: list[ORMEntity]
494   ) -> list[DaasObject]:
495
496       result = []
497       for orm in ormlist:
498           result.append(await self.__convert_by_object_type(orm))
499       return result
```

class `InstanceRepository()`

Bases: `RepositoryBase`

Grants access to instance table

Source code in `src/app/daas/db/db_repos.py`

```
30    def __init__(self):
31        Loggable.__init__(self, LogTarget.DB)
32        self.dbman = None
33        self.connected = False
```

Attributes

attr `dbman = None` instance-attribute

attr `connected = False` instance-attribute

Functions

meth `create_instance(model: InstanceObject) -> bool` async

Create instance

Source code in `src/app/daas/db/db_repos.py`

```
505  async def create_instance(self, model: InstanceObject) -> bool:
506      """Create instance"""
507      return await self._upsert(model)
```

meth `update_instance(model: InstanceObject) -> bool` async

Update instance

Source code in `src/app/daas/db/db_repos.py`

```
509  async def update_instance(self, model: InstanceObject) -> bool:
510      """Update instance"""
511
512      model.app = await self._to_orm(model.app, ORMObject)
513      return await self._upsert(model)
```

meth `all_instances() -> list[InstanceObject]` async

Fetch all available instances

Source code in `src/app/daas/db/db_repos.py`

```
515  async def all_instances(self) -> list[InstanceObject]:
516      """Fetch all available instances"""
517      selected = await self._select_all(Tablenames.Inst)
518      return await self._to_model_list(selected, InstanceObject)
```

meth `get_instance_by_id(id_pk: str) -> Optional[InstanceObject]` async

Fetch all instances

Source code in `src/app/daas/db/db_repos.py`

```
520  async def get_instance_by_id(self, id_pk: str) -> Optional[InstanceObject]:
521      """Fetch all instances"""
522      filter = await self._get_filter_id(id_pk)
523      api = await self._get_api()
524      orm = await api.db_session_select_one(Tablenames.Inst, filter)
525      # print(f"INSTBYID_BEFORE: {id_pk} {filter}")
526      inst = await self._to_model(orm, InstanceObject)
527      # print(f"INSTBYID_AFTER: {inst} {type(inst.app)}")
528      return inst
```

meth `get_instance_by_objid(id_pk: str) -> Optional[InstanceObject]` async

Fetch instances by object id

Source code in `src/app/daas/db/db_repos.py`

```
530  async def get_instance_by_objid(self, id_pk: str) -> Optional[InstanceObject]:
531      """Fetch instances by object id"""
532      filter = await self._get_filter_column(Colnames.AppId, id_pk)
533      api = await self._get_api()
534      orm = await api.db_session_select_one(Tablenames.Inst, filter)
535      if orm is not None:
536          # print(f"ORM: {orm} {type(orm.app)} {id_pk} {filter}")
537          return await self._to_model(orm, InstanceObject)
538      return None
```

meth `get_instance_by_envid(id_pk: str) -> Optional[InstanceObject]` `async`

Fetch instances by environment id

Source code in `src/app/daas/db/db_repos.py`

```
540  async def get_instance_by_envid(self, id_pk: str) -> Optional[InstanceObject]:
541      """Fetch instances by environment id"""
542      filter = await self._get_filter_column(Colnames.EnvId, id_pk)
543      api = await self._get_api()
544      orm = await api.db_session_select_one(Tablenames.Inst, filter)
545      return await self._to_model(orm, InstanceObject)
```

meth `get_instance_by_conid(id_pk: str) -> Optional[InstanceObject]` `async`

Fetch instances by connection id

Source code in `src/app/daas/db/db_repos.py`

```
547  async def get_instance_by_conid(self, id_pk: str) -> Optional[InstanceObject]:
548      """Fetch instances by connection id"""
549      filter = await self._get_filter_column(Colnames.ConnectionId, id_pk)
550      api = await self._get_api()
551      orm = await api.db_session_select_one(Tablenames.Inst, filter)
552      return await self._to_model(orm, InstanceObject)
```

meth `get_instances_by_owner(id_owner: int = 0) -> list[InstanceObject]` `async`

Fetch all instances by owner id

Source code in `src/app/daas/db/db_repos.py`

```
554  async def get_instances_by_owner(self, id_owner: int = 0) -> list[InstanceObject]:
555      """Fetch all instances by owner id"""
556      api = await self._get_api()
557      flt = await self._get_filter_owner(id_owner)
558      ormlist = await api.db_session_select(Tablenames.Inst, flt)
559      return await self._to_model_list(ormlist, InstanceObject)
```

meth `get_instance_by_adr(adr: str) -> Optional[InstanceObject]` `async`

Fetch all instances by ip address

Source code in `src/app/daas/db/db_repos.py`

```
async def get_instance_by_adr(self, adr: str) -> Optional[InstanceObject]:
    """Fetch all instances by ip address"""
    api = await self._get_api()
    flt_host = await self._get_filter_statement(Colnames.Host, adr)
    flt = await self._get_filter_str(flt_host)
    ormlist = await api.db_session_select_one(Tablenames.Inst, flt)
    return await self._to_model(ormlist, InstanceObject)
```

meth `get_instances_available(id_owner: int = 0) -> list[InstanceObject]` async

Fetch all available instances

Source code in `src/app/daas/db/db_repos.py`

```
async def get_instances_available(self, id_owner: int = 0) -> list[InstanceObject]:
    """Fetch all available instances"""
    api = await self._get_api()
    filter_obj = await self._get_filter_statement(Colnames.Owner, id_owner)
    filter_name = await self._get_filter_statement(Colnames.Owner, 0)
    filter = await self._get_filter_str(f"{filter_name} OR {filter_obj}")
    ormlist = await api.db_session_select(Tablenames.Inst, filter)
    return await self._to_model_list(ormlist, InstanceObject)
```

meth `delete_instance(instance: InstanceObject) -> bool` async

Remove instance

Source code in `src/app/daas/db/db_repos.py`

```
async def delete_instance(self, instance: InstanceObject) -> bool:
    """Remove instance"""
    filter = await self._get_filter_id(instance.id)
    orm = await self._select_one(Tablenames.Inst, filter)
    model = await self._to_model(orm, InstanceObject)
    return await self._delete(model)
```

`src.app.daas.db.db_samples` module

Bootstrap objects

Classes

class
`SampleRegistryConfig(shared_owner: int, shared_owner_name: str, test_owner: int, test_owner_name: str, default_owner: int, default_owner_name: str, default_resolution: str, persist_files: bool, persist_apps: bool, persist_objects: bool, persist_session: bool)` `dataclass`

Config for SampleRegistry

Attributes

attr `shared_owner: int` `instance-attribute`

attr `shared_owner_name: str` `instance-attribute`

attr `test_owner: int` `instance-attribute`

attr `test_owner_name: str` `instance-attribute`

attr `default_owner: int` `instance-attribute`

attr `default_owner_name: str` `instance-attribute`

attr `default_resolution: str` `instance-attribute`

attr `persist_files: bool` `instance-attribute`

attr `persist_apps: bool` `instance-attribute`

attr `persist_objects: bool` `instance-attribute`

attr `persist_session: bool` `instance-attribute`

Functions

class `SampleRegistry(cfg: SampleRegistryConfig)`

Holds sample information

Source code in `src/app/daas/db/db_samples.py`

```
37  def __init__(self, cfg: SampleRegistryConfig):
38      self.config = cfg
```

Attributes

attr `config = cfg` `instance-attribute`

Functions

meth `create_demo_limits()`

Source code in `src/app/daas/db/db_samples.py`

```
40  def create_demo_limits(self):
41      owner = self.config.default_owner
42      return [
43          RessourceInfo(
44              id_owner=owner,
45              vm_max=-1,
46              container_max=-1,
47              obj_max=-1,
48              cpu_max=12,
49              mem_max=12 * 1024**3,
50              dsk_max=256 * 1024**3,
51          )
52      ]
```

meth `create_demo_instances(obj: DaasObject, env: Environment, con: GuacamoleConnection)`

Source code in `src/app/daas/db/db_samples.py`

```
54  def create_demo_instances(
55      self, obj: DaasObject, env: Environment, con: GuacamoleConnection
56  ):
57      owner = self.config.default_owner
58      return [
59          # InstanceObject(
60          #     id="randomkey",
61          #     id_owner=owner,
62          #     app=obj,
63          #     host="192.168.223.116",
64          #     container="daas.xxx",
65          #     created_at=datetime.now(),
66          #     connected_at=datetime.now(),
67          #     booted_at=datetime.now(),
68          #     id_con=con.id,
69          #     id_app=obj.id,
70          #     id_env=env.id,
71          #     env=env,
72          # )
73      ]
```

meth `create_demo_connection()`

Source code in `src/app/daas/db/db_samples.py`

```
75  def create_demo_connection(self):
76      return [
77          # GuacamoleConnection(
78          #     id="randomkey",
79          #     user="root",
80          #     password="root",
81          #     protocol="vnc",
82          #     hostname="192.168.223.116",
83          #     port=6016,
84          #     viewer_url="https://foo.bar.baz",
85          #     viewer_token="randomkey2",
86          # )
87      ]
```

meth `create_demo_environments()`

Create environmenst for testing

Source code in `src/app/daas/db/db_samples.py`

```
89   def create_demo_environments(self):
90       """Create environmenst for testing"""
91       owner = self.config.default_owner
92       envtasks = [{"args": "", "cmd": "/usr/bin/galculator", "type": "exec_cmd"}]
93       envapps = [{"args": "", "cmd": "/usr/bin/galculator", "name": "calc"}]
94       envtarget = {"args": "", "cmd": "/usr/bin/galculator", "name": "testname"}
95       return [
96           # Environment(
97           #     id="testenv",
98           #     id_object="vmDeb12",
99           #     name="envtest",
100          #     id_backend="foo",
101          #     env_tasks=envtasks,
102          #     env_apps=envapps,
103          #     env_target=envtarget,
104          #     state="environment-final",
105          #     created_at=datetime.now(),
106          # )
107      ]
```

meth `create_demo_objects() -> list`

Create demo objects

Source code in `src/app/daas/db/db_samples.py`

```
109  def create_demo_objects(self) -> list:
110      """Create demo objects"""
111      allobj = []
112      allobj.extend(self.create_demo_machines())
113      allobj.extend(self.create_demo_containers())
114      return allobj
```

meth `create_demo_files()`

Create files for testing

Source code in `src/app/daas/db/db_samples.py`

```
116  def create_demo_files(self):
117      """Create files for testing"""
118      owner = self.config.default_owner
119      return [
120          File(
121              id="win10DummyFile",
122              id_owner=0,
123              os_type="win10",
124              name="testfile.bat",
125              version="1.2.3",
126              localpath=f"/home/design-daas/src/data/shared/{owner}/win10DummyFile.file",
127              remotepath="C:/Users/root",
128              filesize=79,
129              created_at=datetime.now(),
130          ),
131          File(
132              id="win11DummyFile",
133              id_owner=0,
134              os_type="win11",
135              name="testfile.bat",
136              version="1.2.3",
137              localpath=f"/home/design-daas/src/data/shared/{owner}/win11DummyFile.file",
138              remotepath="C:/Users/root",
139              filesize=79,
140              created_at=datetime.now(),
141          ),
142          File(
143              id="deb12DummyFile",
144              id_owner=0,
145              os_type="l26",
146              name="testfile.bash",
147              version="1.2.3",
148              localpath=f"/home/design-daas/src/data/shared/{owner}/deb12DummyFile.file",
149              remotepath="/root",
150              filesize=79,
151              created_at=datetime.now(),
152          ),
153          File(
154              id="thunderbirdFile",
155              id_owner=0,
156              os_type="l26",
157              name="thunderbird.exe",
158              version="1.2.3",
159              localpath=f"/home/design-daas/src/data/shared/{owner}/thunderbirdFile.file",
160              remotepath="/root/shared/.wine64/drive_c",
161              filesize=79,
162              created_at=datetime.now(),
163          ),
164      ]
```

meth `create_demo_applications()`

Create applications for testing

Source code in `src/app/daas/db/db_samples.py`