Technical Documentation

WIP/outdated

Backups

With the help of the Tartarus backup solution, the data from the AWP directory is automatically backed up in Hetzner's Storage Box. This solution was chosen because the servers are already contracted with Hetzner and the backup storage was supplied free of charge.

The *tartarus* command starts the backup. A configuration file needs to be defined. A configuration file must still be specified. For different backup profiles, these are located under */etc/tartarus/*.conf* . They contain the information, which directory is backed up and where Tartarus stores the markers for the incremental backups. The marker files are maintained by Tartarus itself, but a path to the marker file must be created for each backup profile beforehand. For the backup profile of the Minetest directory it looks like this:

`/var/spool/tartarus/timestamps/backupprofilname` When creating a new backup profile, the path must be adjusted.

The *generic.inc* file contains the most important settings regarding the connection to the Hetzner server. In addition, the -i option can be used to switch to incremental instead of complete backups. For this, a complete backup must first be performed before an incremental backup can be started with the same profile.

`_sudo tartarus /etc/tartarus/Name der Backupprofilkofgurationsdatei_` creates a backup according to the specified configuration profile.

`_sudo tartarus -i /etc/tartarus/Name der Backupprofilkofgurationsdatei_` creates an incremental backup according to the specified configuration profile.

To perform a cleanup of the storage box after each backup, the script c*haron.ftp* invoked automatically under */usr/sbin* / after Tartarus. . This deletes all backups older than 13 days belonging to the same profile after the new backup is created. This ensures that the storage space of the Storage Box is cleared again. In the /etc/tartarus/`_generic.inc_` a hook was inserted for this. There you could also define the maximum age. Complete backups can only be deleted if all incremental backups based on them have been deleted. With the Hetzner administration robot https://robot.your-server.de/storage you could check the used storage space. It is important to note that Hetzner's robot only updates every 15 minutes.

To access the content from the server, the backup storage has been mounted at _/home/awp/backups_. Use _sudo crontab -l_ to view the current automation. With sudo crontab -e the automation can be adjusted.

To access the content from the server, the backup storage has been mounted at `_/home/awp/backups_`. Use`_sudo crontab -l_` to view the current automation. To adjust the automation use *sudo crontab -e* .

```
#
# m     h       dom     mon     dow     command
  30    22        *       *      mon-sat  /usr/sbin/tartarus -i /etc/tartarus/awp.conf
  30    22        *       *      sun      /usr/sbin/tartarus etc/tartarus/awp.conf
```

These two entries ensure that a backup is performed by Tartarus at 22:30. From Monday to Saturday these backups with the configuration profile are carried out incrementally. On Sundays a complete backup is performed with the configuration profile.

Dokumentation CheckMK

Use the CheckMK monitoring tool to monitor and check the most important services of the server. CheckMK can be operated via a web interface and can also be configured in this interface.

The web interface can only be used if monitoring has been started. Check-MK uses the „open monitoring distribution“ (omd) for monitoring. OMD can create instances, so called sites, with which you can monitor the server independently. Currently an instance has been created, which has been named „monitoring“. To start the monitoring on this instance, execute the command *omd start monitoring* . The instance can also be stopped again with the command *omd stop monitoring* .

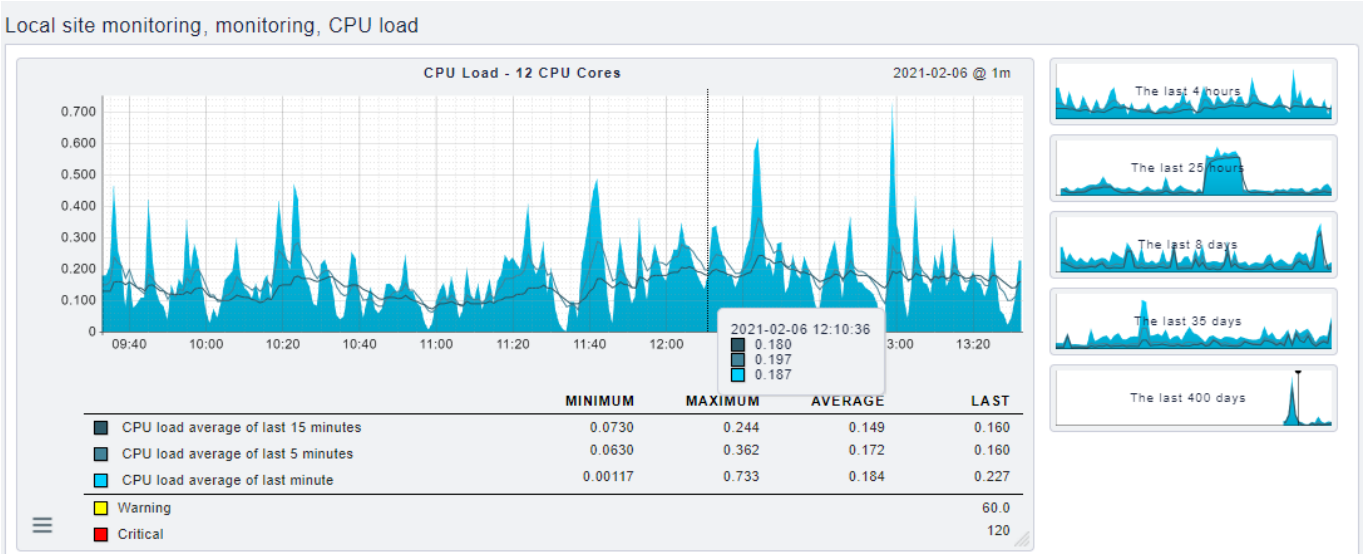
Once the instance is started, the web interface can be accessed via http://minetest.lmz-bw.de/monitoring.

Hosts:

CheckMK needs a host to monitor services. Here the Linux server with the IP address (188.40.113.24) was set up as host. Each host has a number of services that are automatically detected and monitored by CheckMK.

Services:

All currently monitored services are displayed on the home page. You can get the most important information about each monitored service immediately. If you click on the service name (for example, CPU load) you will get all the information about this service. Each service is assigned a state by the system which is derived from the rules defined for this service. The state is either ok, warn, crit, unknown or pend.
Klick on the menu button to automatically restart the check or display the check parameters. Click on with the graph icon to get one or more graphs that show you the evolution of the numbers for this service.



The graph shows an overview of the course of the service over the last 4 hours. You can change the graphic on the x-axis (time) as well as on the y-axis (CPU load) and also zoom in. So it is possible to display the graph you need. The threshold values are also displayed or when a value gets the status warning or critical. You can thus see how busy the service currently is.

Currently 20 services are monitored. To access all services or to monitor other services as well, you have to change the configuration in the Wato menu. To do this, click on Host in the Wato configuration and select the service icon for the „monitoring“ host. In this menu, all detected services are displayed and you can enable or disable services as you wish. To change the rule for the status of a service, you have to adjust the check parameters with a new rule. To do this, click on the parameter icon and create a new rule with new values in the „main folder“
It is important to note that if something is changed or something is added, you still have to confirm this change before the change is made.

User:

You can view all users in the Wato menu under the User item. In this menu, you can create new users as well as edit current users. Currently the users cmkadmin and aufseher are configured. Both users have full administration rights and for both a mail address is entered. You can find the login data for both users in the password database.

Notification:

For both active users a mail address is registered. A mail is sent to these addresses as soon as the status of a service changes. The mail contains information about which service has changed and a graph that shows the development of the service over the last day. Both users are currently registered in the contact group „Everything“. If you do not want to be notified about every service you have to change the contact group in the Wato menu or create a new contact group.

Logs:

The different logs are stored by omd. They are stored in the folder */omd/sites/monitoring/var/log* /.

Further Documentation:

You can find detailed documentation on all features of Checkmk here:
https://docs.checkmk.com/latest/de/index.html

Documentation of Service Architecture

2 Servers:

- Webhosting (www.blockalot.de):

Services:

- Website
- Mail Server

Access:

- KonsoleH on https:<nowiki>//</nowiki>konsoleh.your-server.de (access data: KonsoleH in the keepass)

Backend Server (minetest.lmz-bw.de):

Services:

1. Minetest
2. Postgres database
3. Backend
4. Keycloak
5. CheckMK

Access:

- ssh address: 188.40.113.24 or minetest.lmz-bw.de (access: *awp user server* or *root user server* in the keepass)

Documentation Backend

The backend provides the required functionalities for the frontend via an interface.

Interface

The interface is a rest interface that is used to provide required data and functionality using the HTTP methods.

Using https:<nowiki>//</nowiki>minetest.lmz-bw.de:1234/api-docs a documentation of the currently available endpoints based on the Open API 3.0.3 standard was created with the help of *swagger.* Authentication with a *bearer token* is required to use the interface.

Technology

The programming language used for the backend is //TypeScript// in version 4.1.3. For the interface

we used Express version 4.17.1. The DBMS we used is *PostgreSQL*, since it also serves as the database for the Minetest servers. For user management we use a *keycloak*server, which runs on version 12.

Versioning

The code for the backend was versioned using *Git*. You find the complete commit history on GitHub. It should be noted that the current state of development is in the *main* branch. In the *live* branch is the current running state. It is available here.

Pipeline

Two pipelines have been created using GitHub Actions.

The first is a quality assurance pipeline. It executes the following commands in the order given:

1. npm ci → check the deployed dependencies
2. npm run lint:ci → Check code quality using ESLint

With the help of the second pipeline, the code can be deployed on the currently used Linux machine at Hetzner. With each incoming commit on the branch *live*, a script is executed that transfers the code via FTP to the Linux server and executes the following commands:

1. *systemctl –user stop prod_backend* → stops the Linux service
2. *npm install* → download necessary dependencies
3. *npx tsc* → compiling with TypeScript Compiler
4. *systemctl –user start prod_backend* → restart the Linux service

In addition, a pre-commit hook was set up with *Husky*that executes the command *npm run lint:ci* before every attempted commit. This is to prevent code with insufficient code quality from ending up in the repository.

Authentification

In order to successfully authenticate on the backend, you must first fetch a token from our keycloak server here. This requires a POST request with the content type *application/x-www-form-urlencoded* with the following payload:

- grant_type: password
- username: <html><testuser></html>
- password: <html><password></html>
- client_id <html><client ID></html> → the client we currently use for the frontend is called *vue-client*

If you enter the above data correctly, you will receive a *JWT*in the response under the property *access_token* that must be sent with a request to the backend as *authorization header* in order to authenticate yourself.

Procedure of Incoming Requests

The following procedure is used for processing incoming requests:

1. Parsing the body using express.json()
2. Security check of the request using helmet
3. Authentication and extraction of the user ID from the token → Custom middleware under /src/security/validatejwt.ts
4. Execution of a handler that takes over the flow control for the processing of the request → Handler functions under /src/handler/<html><http-methode></html>-handler.ts
5. Running the various services by the handler → Service functions under /src/service
6. Error handling → with error handler under /src/handler/error-handler.ts
7. Sending a corresponding response

Code Documentation

The code was annotated with comments in JSDoc format following the recommendation from the Empfehlung aus der TypeScript documentation.

Documentation Website

Blockalot's frontend is the part of the application that end users operate to manage their BLOCKALOTSpaces.

Design

The website design is based on the corporate design of the LMZ: The colors, especially orange and olive green, of the LMZ homepage were adopted. The logo and some graphics were also adopted or modified. For some elements, for example, for the „cards" of the BLOCKALOTSpaces on the dashboard, we used *neumorphism*. While this provides a modern look, its suitability, especially in terms of contrast, should be further reviewed.

A mockup for the website was created using *Figma*. It can be viewed here. If changes need to be made, please contact *rech1033@hs-karlsruhe.de* for access.

The colors used on the website are defined as global variables in the *src/css/colors.css* file. This allows changes to these color codes to change the corresponding colors throughout the site. Thus, for example, the application could be adapted to a different corporate design with little effort. In addition, the implementation of a „darkmode" can be realized, for which an example can already be found in the mockup.

Technology

The frontend is written with the *JavaScript framework Vue* in version 3.0.0. *Vuex* was used for *state management*. We tried to avoid external code to reduce dependencies.

Versioning

You can view all dependencies and their versions in repository on Github or in the *package.json* file. In addition, a pipeline for quality assurance and automatic deployment has been set up there using *GitHub Actions*, analogous to the backend.

Structure

The root file is *src/App.vue*. This includes the *navbar*, the *footer* and the current *view*. The current view is determined by *Vue-Router*, the corresponding file is *src/router/index.js*. The views are located in *src/views* and correspond to the different pages of the application (Home, Dashboard, FAQ, BLOCKALOTSpaces, Imprint and Data Protection).

The views include different components that are stored in *src/components*. Some of these components, for example the button, can be reused throughout the frontend. *Vuese* can be used to generate more information about each component. *Vuese* automatically generates documentation for the components. The generated files are written in *Markdown* and are located in *website/components*. A website can be hosted locally that displays all *Markdown* files of the component documentation in a formatted way. For this, you need to install *Vuese* globally and then run *vuese serve -open* (see Vuese documentation).

The *store* (state management) is defined in *src/store/index.js*. This is where *state manipulations* are performed and *API calls* is activated from the *src/api/api.js* file. These are responsible for communication with the backend.

The password and PDF generation functions are stored in the *src/functions* folder to make the code more concise. All graphics are located in *src/assets*.

Documentation Keycloak

The keycloak server is used for the *user management* of the website. The name of the service on the Linux machine is *keycloak*. The *logs* can be found using the following path:
/keycloak-12.0.2/standalone/log/server.log
The keycloak installation is located in the folder keycloak-*akt. Version number*. Keycloak is run in *standalone* mode (the corresponding configuration is also located in this folder).
There is a *realm* (= collection of all settings and configurations) for testing (*lmz_dev*) and one that is live (*lmz_prod*). The configuration can also mostly be done via a user interface (e.g. creating new users):
https://minetest.lmz-bw.de:8443/auth/admin/lmz_prod/console/ (configuration of all realms possible)
https://minetest.lmz-bw.de:8443/auth/admin/lmz_prod/console/ You find the access data for this in the *keepass*. The appearance of the login page as well as the user profiles must be customized via Keycloak and not via the frontend (folder themes).

Documentation Postgres

Postgres is the *DBMS* we use for both our backend and Minetest servers. It does not have its own

service name, such as Keycloak. It is managed using the following command: *sudo pg_ctlcluster 13 main (reload|start|restart)* sudo pg_ctlcluster 13 main (reload|start|restart) The logs are located at _/var/log/postgresql/postgresql-13-main.log_ In Postgres, each world has its own database that contains seven tables. There is also another database where our backend manages its data on created BLOCKALOTSpaces.

The access to the Postgres database is done via SSH. With The access to the Postgres database is done via SSH. With sudo -u postgres psql the database can be accessed. With *sudo -u postgres psql* the database can be accessed. With \l all databases are displayed. With \c *databaseName* you create a connection to a specific database. With \d the contained tables can be displayed.

For a more convenient administration, the database can be accessed via *pgAdmin*. For this, a local installation as well as a depositing of the IP address in the Config file in */etc/postgresql/13/main/pg_hba.conf* is necessary. For this, the IP address must be attached below (following the pattern of the existing entries). After that a reload of the configuration with the command *sudo pg_ctlcluster 13 main reload* is necessary.

In our backend database, inconsistent data may have resulted from errors during testing. These can be traced with the following command:

SELECT * FROM (SELECT d.datacl, d.datname, w.* FROM pg_database AS d FULL JOIN worlds AS w ON „w“.„worldId“ = substring(d.datname, 2)::INTEGER
WHERE datistemplate = false AND datname LIKE 'k%') AS all_databases WHERE port IS NULL OR datname IS NULL ORDER BY datname;

The command matches the databases of the BLOCKALOTSpaces with the entries in our backend database and displays the deltas.

Minetest

The name of the services of each BLOCKALOTSpace is derived from the following service: _k_< *klassenzimmerID>*
The logs are located In the *debug.txt* file in the respective BLOCKALOTSpace folder in */home/awp/minetest-live/worlds/<html><userID></html>/<html><classroomID></html>*. Here, the userID is the ID of the account from Keycloak. The BLOCKALOTSpace folder also contains all the associated configuration files.

The current used Minetest installation is located in the home directory AWP (*/home/awp*) in the folder *minetest-live*. In the folder */mods*, you find the current globally installed mods. Find documentation about the installed mods in mod_uebersicht .

Each world has its own *postgres database* in which the ingame data (player inventory, access data, blocks of the world, etc.) is stored (the access data can be found in the respective world.mt (or in the password database see Postgres).

Each world has its own service in Systemd.

More information:

The installation is done as follows:

Navigate to the desired folder

```
export branch=$(printf "Enter Minetest version: ">&2;read r;echo "$r")
```

Current version: **stable-5**

```
(sudo apt-get install -y unzip g{it,cc,++} {c,}make
{zlib1g,lib{sqlite3,curl4-openssl,luajit-5.1,leveldb,pq}}-dev&& \
cd $(mktemp -dp /var/tmp)&&wget
downloads.sourceforge.net/irrlicht/irrlicht-1.8.4.zip -O irr.zip&& \
unzip -q irr.zip&&git clone -b $branch
https://github.com/minetest/minetest&&cd minetest&& \
git clone -b $branch {https://github.com/minetest,games}/minetest_game&& \
cmake . -
D{BUILD_CLIENT=0,{BUILD_SERVER,RUN_IN_PLACE,ENABLE_POSTGRESQL,ENABLE_CURSES}
=1,\
IRRLICHT_INCLUDE_DIR=$PWD/../irrlicht-1.8.4/include,POSTGRESQL_INCLUDE_DIR=/
usr/include/postgresql/,\
POSTGRESQL_LIBRARY=/usr/lib/x86_64-linux-
gnu/libpq.so.5.13,PostgreSQL_TYPE_INCLUDE_DIR=/usr/include/postgresql/}&& \
make -j$(nproc) package&&p=$(echo minetest-*.tar.gz)&&cp "$p" ~&& \
printf '\n\n\e[1;32mBuild successful, an archive called "%s" was placed in
your home folder.\nTo run the server, extract the archive\
and run the executable inside the 'bin/' folder\e[0m\n' "$p")||printf
'\n\n\e[1;31mBuild failed, review log output above to identify and fix the
issue.\e[0m\n
```

Certificates

The certificates for the backend were created with the following command:

sudo certbot –apache -d minetest.lmz-bw.de

Output:

Saving debug log to /var/log/letsencrypt/letsencrypt.log

Renewing an existing certificate Created an SSL vhost at /etc/apache2/sites-available/000-default-le-ssl.conf Enabled Apache socache_shmcb module Enabled Apache ssl module Deploying Certificate to VirtualHost /etc/apache2/sites-available/000-default-le-ssl.conf Enabling available site: /etc/apache2/sites-available/000-default-le-ssl.conf

Please choose whether or not to redirect HTTP traffic to HTTPS, removing HTTP access. - 1: No redirect - Make no further changes to the webserver configuration. 2: Redirect - Make all requests redirect to secure HTTPS access. Choose this for new sites, or if you're confident your site works on HTTPS. You can undo this change by editing your web server's configuration. - Select the appropriate number [1-2] then [enter] (press 'c' to cancel): 2 Redirecting vhost in /etc/apache2/sites-enabled/000-default.conf to ssl vhost in /etc/apache2/sites-available/000-default-le-ssl.conf

- Your existing certificate has been successfully renewed, and the new certificate has been installed.

The new certificate covers the following domains: minetest.lmz-bw.de

You should test your configuration at:
https://www.ssllabs.com/ssltest/analyze.html?d=minetest.lmz-bw.de -

IMPORTANT NOTES: - Congratulations! Your certificate and chain have been saved at: /etc/letsencrypt/live/minetest.lmz-bw.de/fullchain.pem Your key file has been saved at: /etc/letsencrypt/live/minetest.lmz-bw.de/privkey.pem Your cert will expire on 2021-04-07. To obtain a new or tweaked version of this certificate in the future, simply run certbot again with the „certonly“ option. To non-interactively renew *all* of your certificates, run „certbot renew“ - If you like Certbot, please consider supporting our work by:

Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate Donating to EFF: https://eff.org/donate-le

This certificate was used with the following instructions for Keycloak (possibly adjust path directly):

https://ordina-jworks.github.io/security/2019/08/14/Using-Lets-Encrypt-Certificates-In-Java.html

https://www.keycloak.org/docs/11.0/server_installation/index.html#_start_cli

openssl pkcs12 -export -in /etc/letsencrypt/live/minetest.lmz-bw.de/fullchain.pem -inkey /etc/letsencrypt/live/minetest.lmz-bw.de/privkey.pem -out /tmp/minetest.lmz-bw.de_2.p12 -name http://minetest.lmz-bw.de/ -CAfile /etc/letsencrypt/live/minetest.lmz-bw.de/fullchain.pem -caname „Let's Encrypt Authority X3“ -password pass:changeit

keytool -importkeystore -deststorepass changeit -destkeypass changeit -deststoretype pkcs12 -srckeystore /tmp/minetest.lmz-bw.de_2.p12 -srcstoretype PKCS12 -srcstorepass changeit -destkeystore /tmp/minetest.lmz-bw.de_2.keystore -alias minetest.lmz-bw.de/%%

Data Model

The application accesses data from four sources.

Source Name	**Description**
Keycloak data	All user data is stored here (name, school, password), this is usually not changed by the application
Database „Backendata“	This is where the information needed for (BLOCKALOTSpace) administration is stored.
Databases „k**“	Each world has its own database where game relevant information is stored (e.g. game passwords, inventory)
Directory minetest-live/worlds	This is where the necessary configuration information for the BLOCKALOTSpaces is stored (mods, spawnpoint, etc.)

The template database is marked as a template and connections are forbidden, because the database cannot be used as a template if a connection exists.

We have set this up with the following commands:

UPDATE pg_database SET datistemplate = true where datname = 'minetest_template1';

UPDATE pg_database SET datallowconn = false where datname = 'minetest_template1';

Documentation Tests

For testing the backend there is a Postman Collection with configured tests. These can be easily run with the collection runner (access must be requested – unfortunately, number of users is limited). The frontend requires manual testing at the moment.