

Eingereicht von:

Andreas Eigner

Mathias Hausleithner

Angefertigt am

Institut für

Wirtschaftsinformatik -

Communications

Engineering

BeurteilerIn:

Claudia Kaar, BSc MSc

Jänner, 2020

MONITORING SERVICE FÜR ULTIMAKER 3 & ULTIMAKER 5 3D-DRUCKER



Projektdokumentation

im Rahmen des Kurses

IT-Project

Communications Engineering

im Bachelorstudium

Wirtschaftsinformatik

Inhaltsverzeichnis

1.	Einleitung	4
2.	Auftrag	5
2.1.	Ausgangssituation	5
2.2.	Ziele	5
2.3.	Projektverlauf	6
3.	Technische Dokumentation	8
3.1.	Projektarchitektur	8
3.2.	Beschreibung der Komponenten	8
3.2.1.	Ultimaker 3D-Drucker	8
3.2.2.	Datenaufbereitungsservice	9
3.2.3.	Telegraf	15
3.2.4.	InfluxDB	16
3.2.5.	Grafana	18
3.3.	Einbindung in die Netzwerkinfrastruktur der Grand Garage	19
3.3.1.	Docker	20
4.	Installationsanleitung	21
4.1.	Herunterladen des Projekts	21
4.2.	Docker Images erstellen und starten	21
4.3.	Konfiguration der Dashboards	23
4.3.1.	Aufruf Grafana	23
4.3.2.	Konfiguration der Datenquelle	23
4.3.3.	Import der bestehenden Dashboards	24
4.4.	Wartung der Komponenten	25
4.4.1.	Datenaufbereitungsservice	25
4.4.2.	Telegraf	26
4.4.3.	Grafana	26
5.	Benutzerhandbuch	27
5.1.	Einstieg	27
5.2.	Dashboards	28
5.2.1.	Overview	28
5.2.2.	Hotend temperature	29
5.2.3.	Material extruded	29

5.2.4. Printer status	30
5.2.5. Printjob history.....	30
5.2.6. Printjob progress.....	31
5.2.7. Time spent hot.....	32
5.3. Dashboard Funktionen	32
5.3.1. Rolle „Viewer“	32
5.3.2. Rolle „Editor“	33

1. Einleitung

Dieses Dokument beinhaltet die technische Dokumentation, eine Installationsanleitung, sowie ein Benutzerhandbuch des Monitoring Services für Ultimaker 3D Drucker.

Das Monitoring Service wurde im Zuge eines IT-Projektes am Communication Engineering Institut an der Johannes-Kepler Universität Linz, in Zusammenarbeit mit der Grand Garage, erstellt und dient zur Aufzeichnung und grafischen Aufbereitung von 3D-Drucker-Daten von Geräten des Herstellers „Ultimaker“. Diese Daten werden in einer Datenbank gesammelt und über ein Dashboard Statistiken und Grafiken für jeden individuellen Drucker, als auch kumuliert angezeigt.

Folgende Metriken werden von der Applikation aufgezeichnet und grafisch dargestellt:

- Status der Drucker
- Aktuelle Temperatur der Hotends
- Kumulierte Zeit über 150° Celsius je Hotend
- Verbrauchtes Material
- Druckfortschritt eines Druckauftrages
- Historie der Druckaufträge

Wenn Sie die Applikation erweitern möchten oder Änderungen notwendig sind, finden Sie in dieser Anleitung einen Link zu einem Git-Repository, über welches Sie den Source-code herunterladen können und daraus ein neues Docker-image erstellen können.

2. Auftrag

2.1. Ausgangssituation

Die Grandgarage in der Tabakfabrik in Linz ist eine neu errichtete Innovationswerkstätte der Firma CAP.future GmbH. Diese Einrichtung ermöglicht es Unternehmen, Studenten und Technikbegeisterten auf ca. 4000m² an Geräten wie 3D-Druckern, Lasercuttern, CNC-Fräsmaschinen und viele mehr kostengünstig Prototypen zu fertigen. Es soll dadurch eine Maker-Community geschaffen werden, die sich kreativ entfalten kann und sich gegenseitig unterstützt. Um ständig neues Wissen aus den Geräten zu generieren, will man beginnen nach und nach Schnittstellen zu diesen zu entwickeln und die Daten für Monitoring- und Analysezwecke nutzen.

Aufgabe dieses Projektes ist es, eine API-Schnittstelle zu den 3D-Druckern des Typs Ultimaker3 und Ultimaker5 zu entwickeln, die Daten des Druckers aufzubereiten, in eine Datenbank zu persistieren und grafisch in einer Monitoring Applikation darzustellen. Die 3D-Drucker können über deren derzeit noch dynamischen IP-Adressen erreicht werden. Auf der Weboberfläche des Druckers ist eine detaillierte REST-API Doku einsehbar, die für die API-Anbindung als Grundlage dient. Des Weiteren steht eine technische Dokumentation für den Remote Access der Drucker zur Verfügung, welche Studenten im Zuge eines Kompetenztrainings am Institut für Communication Engineering erstellt haben.

Kernaufgabe dieses Projektes ist die Entwicklung eines Services, welches als Docker Container zur Verfügung gestellt werden soll und über die Schnittstelle des Druckers Daten sammelt, aufbereitet und an das Logging-Backend des vorhandenen TIG-Stacks weitergibt. Der TIG-Stack wird dafür verwendet, die Daten zu persistieren und grafisch darzustellen. Komponenten dieses Stacks sind Telegraf (=Agent für die Erfassung und Sammlung von Messdaten), InfluxDB (=Performante Zeitreihendatenbank) und Grafana (=Dashboard für Datenvisualisierung). Dieser Stack läuft auf dem Server der Grandgarage und wird bereits für das Monitoring von diversen Daten verwendet. Aufgabe unsererseits ist es, die Schnittstelle des Serveragenten Telegraf so zu erweitern, damit dieser zyklisch (z.B. jede 5s) Daten von dem 3D-Drucker Service abfragt und diese dann in die InfluxDB speichert. Schlussendlich muss das Grafana Dashboard so konfiguriert werden, damit die neu gewonnen Daten geeignet dargestellt werden können. Um den API-Druckerservice zu testen, wird der TIG-Stack zuerst in einer Testumgebung aufgesetzt, bevor das Projekt in das Produktivsystem übernommen wird. Eine Vorgabe, welche Daten genau aufgezeichnet werden sollen gibt es konkret nicht. Diese werden im Projektverlauf mit Herrn Döberl abgestimmt. Beispiele für Daten wären Materialverbrauch, Temperatur, Druckfortschritt, Status,..)

2.2. Ziele

- Erstellung eines API-Services zur Sammlung und Aufbereitung von Daten der Ultimaker3 und Ultimaker5 3D-Drucker
- Speicherung der Daten und grafische Darstellung unter Verwendung des TIG-Stacks
- Einbindung in die Netzwerkinfrastruktur der Grand Garage

Projektmeilensteinplan			
PSP-Code	Bezeichnung	Geplanter Termin (Soll)	Ist-Termin
1.1.1	Projekt beauftragt	29.10.19	29.10.19
1.1.4	Projekt abgeschlossen	31.01.19	
1.2.6	Projektplan erstellt	07.11.19	07.11.2019
1.3.11	Service ausführbar	19.12.19	12.12.2019
1.4.7	Projekt läuft im Testsystem	09.01.20	04.01.2020
1.4.10	Projekt läuft im Produktivsystem	16.01.20	14.01.2020
1.5.2	Dokumentation fertiggestellt	30.01.20	
1.5.4	Projekt abgenommen	31.01.20	21.01.2020

2.3. Projektverlauf

Nach Festlegung eines Projektplanes und Start der Umsetzung des Projektes, war der erste Schritt das Einarbeiten in die API der Ultimaker 3D-Drucker. Gleichzeitig wurden in Zusammenarbeit mit Mitarbeitern der Grand Garage bereits Metriken diskutiert, die für eine Analyse interessant sind. Bei den ersten Tests der Drucker API stellte sich heraus, dass die Drucker teilweise ihre IP-Adressen im Netzwerk verlieren und dadurch nicht mehr erreichbar sind, obwohl diese eingeschaltet sind. Durch Neukonfiguration der Netzwerkadresse kann dieses Problem für einige Zeit wieder behoben werden. Dieses Problem war bis zum Projektabschluss präsent und wurde von der Grand Garage noch nicht gelöst, da zurzeit andere Tasks eine höhere Priorität haben.

Für unseren Datenaufbereitungsservice wurde ein Java Projekt auf Basis des Spring Frameworks eingerichtet. Als Build-Management-Tool wurde Apache Maven genutzt. Da der Service sowohl auf andere APIs zugreift, als auch selbst eine API im Netzwerk zur Verfügung stellen muss, wurde der Service als Webanwendung implementiert, welche auf dem Open-Source Webserver Apache Tomcat aufgesetzt wurde.

Erster Schritt in der Implementierung war die Erstellung eines Client-Modells, um sämtliche Metriken der 3D-Drucker über deren API abrufen zu können. Parallel dazu wurde die Zeitreihendatenbank InfluxDB und der Server-Agent Telegraf, der die Daten von dem Service abgreift und an die Datenbank weitergibt, studiert. Wichtige Fragestellungen hierbei waren:

- Wie wird die Datenbank installiert und konfiguriert?
- Welches Datenformat erwartet die Datenbank?
- Wie wird der Agent Telegraf installiert und konfiguriert?

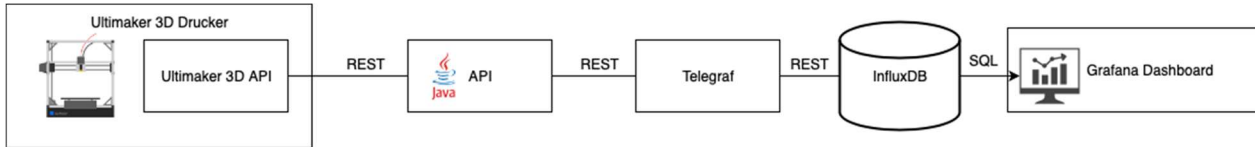
Nach Fertigstellung des Clients wurden das Konstrukt Service-Telegraf-InfluxDB getestet. Dabei gab es einige Schwierigkeiten, was einerseits an der Konfiguration von Telegraf und andererseits an einer inkorrekten Syntax der Datenpunkte lag. Bevor die APIs implementiert werden konnten war es wichtig, dass wir uns mit der Logik der Darstellung im Dashboard Grafana beschäftigt haben. Dadurch konnten wir festlegen welche Tags zu einer Metrik mitgegeben werden müssen, um die gewünschten Dashboards darstellen zu können.

Während der Implementation der APIs gab es immer wieder Herausforderungen die es zu lösen gab. Unsere API hat unter seltenen, aber nicht reproduzierbaren Umständen Werte von einem Drucker in die Antwort eines anderen Druckers kopiert und somit für verfälschte Daten gesorgt. Zuerst haben wir einen Fehler in der Telegraf Konfiguration vermutet. Der Fehler blieb auch durch intensives Testen bestehen und somit legten wir unseren Fokus bei der Fehlersuche auf die Java Applikation. Das Spring Framework übernimmt das Threading beim REST-Controller und sorgt dafür, dass jeder Request auf einem eigenen Thread läuft und es so zu keinen Concurrency Problemen kommen kann. Erst nach genauen Studieren des erstellten REST-Clients der 3D-Drucker haben wir bemerkt, dass dieser eine Klasse statisch instanziiert und dadurch alle Threads auf das selbe Objekt zugreifen. Bei zeitgleichen Anfragen kann dies dann genau zu der Problematik führen, dass Werte kopiert werden. Durch eine kleine Änderung des Clients konnte somit das Problem behoben werden. Die Tatsache, dass die Drucker teilweise nicht antworten, hat die Fehlersuche erheblich erschwert. Ein weiteres Problem ergab sich bei der Implementierung einer Metrik, die die verbleibende Materialmenge eines Druckers aufzeichnen soll. Während der Tests der erstellten Schnittstellen der Metriken wurde festgestellt, dass vom Drucker die verbleibende Materialmenge nicht korrekt an den Service übergibt. Fehlerursache hierbei war, dass an den Druckern keine original Extruder für den Austausch verwendet werden und dadurch diese Funktion nicht unterstützt wird.

Als das Zusammenspiel aus Java Applikation, Telegraf und InfluxDB nun soweit funktioniert hat, haben wir angefangen uns mit Docker zu beschäftigen, da schlussendlich das Service in einer Docker Umgebung laufen soll. Die initiale Erstellung des Docker Files ist mit viel Testaufwand verbunden, bis aus der Applikation schlussendlich ein einsetzbares Image wird. Parallel dazu wurden in Grafana Dashboards erweitert und neue angelegt. Nach einer zwei-tägigen Testphase haben wir gemeinsam mit unserem Ansprechpartner der Grand Garage unser Projekt in die Docker-Umgebung der Grand Garage integriert und getestet. Schlussendlich wurden alle vorgegeben Ziele unter Einhaltung der festgelegten Meilensteintermine und Zufriedenheit des Auftraggebers erreicht.

3. Technische Dokumentation

3.1. Projektarchitektur



Das **Java Webservice (Datenaufbereitungsservice)** erfüllt 2 grundlegende Aufgaben. Erstens dient dieses Service als Client für die Schnittstelle der 3D-Drucker und zweitens stellt es eine API für die Abfrage von Druckerdaten zur Verfügung. Das Format, in dem die Daten zur Verfügung gestellt werden, heißt **line-protocol**, und ist ein für die Datenbank verständliches Format. Genau diese Schnittstelle wird dann von dem vorkonfigurierten **Telegraf Agent** zyklisch abgefragt (die genauen Zyklen kann man aus der Telegraf Konfiguration entnehmen). Telegraf ist ein eigenständiges open-source Programm, welches als Schnittstelle zwischen Java Webservice und der InfluxDB fungiert. Telegraf überträgt diese Daten auf einer im Image enthaltenen InfluxDB Instanz. **InfluxDB** ist eine open-source time-series Datenbank, welche zu jedem Datenpunkt einen Zeitstempel hinzufügt. So ist es möglich, zeitlich bedingte Veränderungen von einem Datenpunkt zu speichern und abzurufen. Das **Grafana**-Dashboard kann die Datensätze der Datenbank abfragen und grafisch aufbereiten.

Diese Kombination aus Telegraf, InfluxDB und Grafana wird auch als „TIG – Stack“ bezeichnet und ist ein weit verbreiteter Technologie-Stack zum Monitoring von Datenpunkten.

3.2. Beschreibung der Komponenten

3.2.1. Ultimaker 3D-Drucker

Die Grand Garage verwendet für den 3D-Druck Geräte von verschiedenen Herstellern. Ein Großteil dieser Drucker ist von dem Hersteller „Ultimaker“. Diese verfügen über eine, nach dem Open API Standard definierte Schnittstelle und bieten daher die Möglichkeit, diverse Daten über Gerät und Druck zu überwachen und auszuwerten. Insgesamt werden Daten von drei Geräten des Typs „Ultimaker 3“ und zwei Geräten des Typs „Ultimaker 5“ gesammelt. Die Schnittstelle dieser beiden Typen ist identisch, das heißt, dass man bei der Abfrage nicht zwischen den Typen unterscheiden muss. Die Drucker sind über WLAN mit dem internen Netzwerk verbunden und besitzen eine statische IP Adresse.

3.2.1.1. Schnittstellenbeschreibung

Die Geräte stellen eine Schnittstellenbeschreibung zur Verfügung. Diese ist innerhalb des lokalen Netzwerks unter „**IP-ADRESSE/api/v1**“ (10.6.0.31/api/v1) erreichbar. In dieser findet man zu jedem erreichbaren Endpunkt eine Dokumentation zu den benötigten Parametern, Pfaden, Antwortstruktur, Fehlercodes usw. Man kann jeden Endpunkt auch direkt über diese Schnittstellenbeschreibung testen.

Der Endpunkt zur Abfrage der aktuellen Temperatur einer Düse sieht wie folgt aus:

GET /printer/heads/{head_id}/extruders/{extruder_id}/hotend/temperature

Implementation Notes

Returns temperature of extruder

Response Class (Status 200)

Temperature of the hotend

Model Example Value

```

{
  "target": 0,
  "current": 0
}

```

Response Content Type

Parameters

Parameter	Value	Description	Parameter Type	Data Type
head_id	<input type="text" value="(required)"/>	ID of head from which the hotend is fetched	path	long
extruder_id	<input type="text" value="(required)"/>	ID of extruder from which the hotend is fetched	path	long

Dieser Endpunkt wird unter anderem von unserem Datenaufbereitungsservice zyklisch pro Drucker und Extruder (Ein Drucker besitzt 2 Extruder) abgefragt und liefert dabei ein JSON File.

3.2.2. Datenaufbereitungsservice

Kernstück des Projekts ist eine Java Applikation, welche über das Sprint Boot Framework in der Java Version 11 realisiert wurde. Sie dient einerseits als REST Webservice und bietet eine API an, welche Daten von den 3D-Druckern bereitstellt. Diese API wird von dem Telegraf Agent zyklisch abgefragt. Andererseits ist die Applikation daher auch ein Client für die API der 3D-Drucker und leitet bei jeder Anfrage des Telegraf Agent die Anfrage an die Drucker API weiter und liefert diese im richtigen Format an Telegraf zurück.

3.2.2.1. Open-API 2.0 (Swagger)

Die API der 3D-Drucker ist im standardisierten Open-API 2.0 (Swagger) Format. Das hat den Vorteil, dass man anhand der Schnittstellendokumentation dieser API über den Swagger Code-

Generator einen API-Client für verschiedenste Programmiersprachen erstellen kann. Das vereinfacht die Abfrage der Endpunkte der Drucker.

Swagger Editor: <https://editor.swagger.io/>

Schnittstellendokumentation 3D-Drucker als JSON-File: IP-ADRESSE/api/v2/api-docs

Die Schnittstellendokumentation im JSON-Format muss in den Swagger Editor kopiert werden. Dieser stellt die Endpunkte für ein besseres Verständnis auch grafisch dar. Über den Button „*Generate Client*“ kann man für alle gängigen Programmiersprachen einen Client erstellen lassen. Dieser kümmert sich um die richtige Abfrage der Endpunkte und gegebenenfalls um die Fehlerbehandlung.

3.2.2.2. Schnittstellenbeschreibung

Die Schnittstellendokumentation des Datenaufbereitungsservice ist als JSON-File im Projektordner unter „api.json“ abgelegt. Diese kann über den Swagger Editor grafisch angezeigt werden.

Api Documentation ^{1.0}

[Base URL: localhost:8080/]

Api Documentation

[Terms of service](#)

[Apache 2.0](#)

api-controller Api Controller



GET /api/{ip}/hot-time timeSpentHot

GET /api/{ip}/hotend-temperatures hotendTemperatures

GET /api/{ip}/material-extruded materialExtruded

GET /api/{ip}/material-remaining materialRemaining

GET /api/{ip}/printjob-history printjobHistory

GET /api/{ip}/printjob-progress printjobProgress

GET /api/{ip}/status printerStatus

GET
/api/{ip}/hot-time
timeSpentHot

Parameters
Try it out

Name	Description
ip ★ required string (path)	ip <input type="text" value="ip - ip"/>

Responses
Response content type
/

Code	Description
200	OK <div> Example Value Model </div> <div>string</div>
401	Unauthorized
403	Forbidden
404	Not Found

3.2.2.3. Testfallspezifikation

In dieser Testfallspezifikation werden die REST Schnittstellen des Datenaufbereitungsservices mit den zu benutzenden Eingabewerten und den zu erwarteten Ausgabewerten dokumentiert. Dies soll dazu dienen, nach Fertigstellung des Services, dessen Korrektheit zu überprüfen. Die Schnittstellen des Ultimaker 3D-Druckers werden nicht mehr spezifisch in einem Testszenario getestet, da diese vom Hersteller freigegeben sind und des Weiteren werden diese implizit durch die Schnittstellen des Datenaufbereitungsservices getestet. Erwartetes Ausgabeformat der Schnittstellen ist in jedem Testfall das InfluxDB line protocol (https://docs.influxdata.com/influxdb/v1.7/write_protocols/line_protocol_reference/). Die Syntax dieses Protokolls muss genau eingehalten werden, da die Daten sonst nicht in die Datenbank gespeichert werden können. In den verschiedenen Testfällen wird die Schnittstelle immer nur bei einem Drucker getestet, da die Funktionen bei den restlichen Druckern ident sind.

Testfallbeschreibung	Time spent hot
3D-Drucker	10.6.0.31
Anmerkung	Einige Ausgabewerte können nicht als erwartet bestimmt werden, da diese sich dynamisch verändern und von verschiedenen Umgebungsbedingungen abhängen. Daher sind teilweise nur Datentypen angegeben, die erwartet werden. Wichtig ist, dass die Syntax

	der erwarteten Ausgabe eingehalten wird, da diese von dem Logging Agenten Telegraf erwartet wird.
Test 1	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss online sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/hot-time
Erwartete Ausgabe	time_spent_hot,printer=31,extruder=1,nozzleType=String,weekday=String,month=String,year=String time-spent-hot=Integer time_spent_hot,printer=31,extruder=2,nozzleType=String,weekday=String,month=String,year=String time-spent-hot=Integer
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	time_spent_hot,printer=31,extruder=1,nozzleType=AA_0.4,weekday=Saturday,month=January,year=2020 time-spent-hot=7784940 time_spent_hot,printer=31,extruder=2,nozzleType=AA_0.4,weekday=Saturday,month=January,year=2020 time-spent-hot=3227220
Kommentar	Erfolgreich
Test 2	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss offline sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/hot-time
Erwartete Ausgabe	(null)
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	null
Kommentar	Erfolgreich

Testfallbeschreibung	Temperature hotend
3D-Drucker	10.6.0.31
Anmerkung	Einige Ausgabewerte können nicht als erwartet bestimmt werden, da diese sich dynamisch verändern und von verschiedenen Umgebungsbedingungen abhängen. Daher sind teilweise nur Datentypen angeben, die erwartet werden. Wichtig ist, dass die Syntax der erwarteten Ausgabe eingehalten wird, da diese von dem Logging Agenten Telegraf erwartet wird.
Test 1	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss online sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/hotend-temperatures
Erwartete Ausgabe	temperature_hotend,printer=31,extruder=1,nozzleType=string temperature=Integer temperature_hotend,printer=31,extruder=2,nozzleType=string temperature=Integer
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	temperature_hotend,printer=31,extruder=1,nozzleType=AA_0.4 temperature=204.8 temperature_hotend,printer=31,extruder=2,nozzleType=AA_0.4 temperature=48.8
Kommentar	Erfolgreich
Test 2	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss offline sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/hotend-temperatures
Erwartete Ausgabe	(null)
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	null
Kommentar	Erfolgreich

Testfallbeschreibung	Printer status
3D-Drucker	10.6.0.31
Anmerkung	Einige Ausgabewerte können nicht als erwartet bestimmt werden, da diese sich dynamisch verändern und von verschiedenen Umgebungsbedingungen abhängen. Daher sind teilweise nur Datentypen angeben, die erwartet werden. Wichtig ist, dass die Syntax der erwarteten Ausgabe eingehalten wird, da diese von dem Logging Agenten Telegraf erwartet wird.
Test 1	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss online sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/status
Erwartete Ausgabe	printer_status,printer=31,weekday= <i>String</i> ,month= <i>String</i> ,year= <i>String</i> status= <i>String</i>
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	printer_status,printer=31,weekday=Saturday ,month=January,year=2020 status=printing
Kommentar	Erfolgreich
Test 2	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss offline sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/status
Erwartete Ausgabe	printer_status,printer=31,weekday= <i>String</i> ,month= <i>String</i> ,year= <i>String</i> status=unknown
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	printer_status,printer=31,weekday=Saturday ,month=January,year=2020 status=unknown
Kommentar	Erfolgreich

Testfallbeschreibung	Material extruded
3D-Drucker	10.6.0.31
Anmerkung	Einige Ausgabewerte können nicht als erwartet bestimmt werden, da diese sich dynamisch verändern und von verschiedenen Umgebungsbedingungen abhängen. Daher sind teilweise nur Datentypen angeben, die erwartet werden. Wichtig ist, dass die Syntax der erwarteten Ausgabe eingehalten wird, da diese von dem Logging Agenten Telegraf erwartet wird.
Test 1	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss online sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/material-extruded
Erwartete Ausgabe	material_extruded,printer=31,extruder=1,nozzleType= <i>String</i> ,weekday= <i>string</i> , month= <i>string</i> ,year= <i>string</i> material-extruded= <i>integer</i> material_extruded,printer=31,extruder=2,nozzleType= <i>String</i> ,weekday= <i>string</i> , month= <i>string</i> ,year= <i>string</i> material-extruded= <i>integer</i>
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	material_extruded,printer=31,extruder=1,nozzleType=AA_0.4,weekday=Saturday, month=January,year=2020 material-extruded=1933890 material_extruded,printer=31,extruder=2,nozzleType=AA_0.4,weekday=Saturday, month=January,year=2020 material-extruded=832010
Kommentar	Erfolgreich
Test 2	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss offline sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/material-extruded
Erwartete Ausgabe	(null)

Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	null
Kommentar	Erfolgreich

Testfallbeschreibung	Printjob progress
3D-Drucker	10.6.0.31
Anmerkung	Einige Ausgabewerte können nicht als erwartet bestimmt werden, da diese sich dynamisch verändern und von verschiedenen Umgebungsbedingungen abhängen. Daher sind teilweise nur Datentypen angeben, die erwartet werden. Wichtig ist, dass die Syntax der erwarteten Ausgabe eingehalten wird, da diese von dem Logging Agenten Telegraf erwartet wird.
Test 1	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss online sein und gerade den Status „printing“ besitzen
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/printjob-progress
Erwartete Ausgabe	printer_status,printer=31,jobname=String progress=integer
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	printer_status,printer=31,jobname=UM3_reifen progress=56
Kommentar	Erfolgreich
Test 2	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss offline sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/printjob-progress
Erwartete Ausgabe	(null)
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	null
Kommentar	Erfolgreich

Testfallbeschreibung	Printjob history
3D-Drucker	10.6.0.31
Anmerkung	Einige Ausgabewerte können nicht als erwartet bestimmt werden, da diese sich dynamisch verändern und von verschiedenen Umgebungsbedingungen abhängen. Daher sind teilweise nur Datentypen angeben, die erwartet werden. Wichtig ist, dass die Syntax der erwarteten Ausgabe eingehalten wird, da diese von dem Logging Agenten Telegraf erwartet wird.
Test 1	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss online sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/printjob-history
Erwartete Ausgabe	printjob_history,result=String,printer=31,weekday=String,month=String,year=String uuid=String
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	printjob_history,result=Finished,printer=31,weekday=Saturday,month=January,year=2020 uuid=95dfda23-bbb1-46a9-abeb-1ad5f2c600d5
Kommentar	Erfolgreich
Test 2	
Voraussetzung	Datenservice muss auf Tomcatserver laufen und 3D-Drucker muss offline sein
Aktion	Webbrowser öffnen
Eingabe	http://localhost:8080/api/10.6.0.31/printjob-history

Erwartete Ausgabe	(null)
Test Browser	Google Chrome Version 78.0.3904.108
Test Resultat	null
Kommentar	Erfolgreich

3.2.3. Telegraf

Telegraf ist ein Plugin-driven Server-Agent zum Sammeln und Reporting von Metriken und der erste Teil des sogenannten TICK-Stacks (Telegraf, InfluxDB, Chronograf, Kapacitor). Telegraf verfügt über Plugins, mit denen eine Vielzahl von Metriken direkt von dem System, auf dem es ausgeführt wird oder Metriken von APIs von Drittanbietern abgerufen werden können. Es enthält auch Ausgabe-Plugins zum Senden von Metriken an eine Vielzahl anderer Datenspeicher und Diensten, darunter InfluxDB, Graphite, OpenTSDB, Datadog, Librato, Kafka, MQTT, NSQ und viele andere.

3.2.3.1. Konfiguration

Telegraf wird standardmäßig mit einer Defaultkonfiguration ausgeliefert. Die Einstellungen sind gebündelt in der Datei „telegraf.config“ zu finden. Es wird in drei Abschnitte unterteilt.

- Globale Konfigurationen
- Output Plugins
- Input Plugins

Um Metriken von dem Datenaufbereitungsservice abzurufen und an die InfluxDB weiterzugeben, wurde diese Konfiguration erweitert. Es werden Input-Plugins benötigt, die über HTTP GET-Requests periodisch auf die APIs des Datenaufbereitungsservices zugreifen um die Metriken zu sammeln. In der folgenden Abbildung ist die Konfiguration der Metrik „printer-status“ für den Drucker 31 dargestellt. Anhand der URL ist zu erkennen, dass der Datenaufbereitungsservice konfigurationslos entwickelt wurde, da erst bei dem GET-Request die IP-Adresse des Druckers mitgegeben wird. Würde sich die IP-Adresse eines Druckers ändern, müsste man in der Telegraf Konfiguration nur die URL anpassen.

```
[[inputs.http]]
  ## One or more URLs from which to read formatted metrics
  urls = [
    "http://api:8080/api/10.6.0.31/status"
  ]

  ## HTTP method
  method = "GET"

  ## Amount of time allowed to complete the HTTP request
  timeout = "20s"

  ## Data format to consume.
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_INPUT.md
  data_format = "influx"
```

Um die Metriken an die InfluxDB zu senden, wurde ein InfluxDB Output Plugin konfiguriert. In diesem wird zum Beispiel die URL und Name der Datenbank festgelegt. In der folgenden Abbildung sehen Sie einen Ausschnitt aus dieser Konfiguration.

```
# Configuration for sending metrics to InfluxDB
[[outputs.influxdb]]
## The full HTTP or UDP URL for your InfluxDB instance.
##
## Multiple URLs can be specified for a single cluster, only ONE of the
## urls will be written to each interval.
# urls = ["unix:///var/run/influxdb.sock"]
# urls = ["udp://127.0.0.1:8089"]
urls = ["http://influxdb:8086"]

## The target database for metrics; will be created as needed.
database = "ultimaker"

## If true, no CREATE DATABASE queries will be sent. Set to true when using
## Telegraf with a user without permissions to create databases or when the
## database already exists.
# skip_database_creation = false

## Name of existing retention policy to write to. Empty string writes to
## the default retention policy. Only takes effect when using HTTP.
# retention_policy = ""

## Write consistency (clusters only), can be: "any", "one", "quorum", "all".
## Only takes effect when using HTTP.
# write_consistency = "any"

## Timeout for HTTP messages.
# timeout = "5s"

## HTTP Basic Auth
# username = "telegraf"
# password = "metricsmetricsmetricsmetrics"

## HTTP User-Agent
# user_agent = "telegraf"

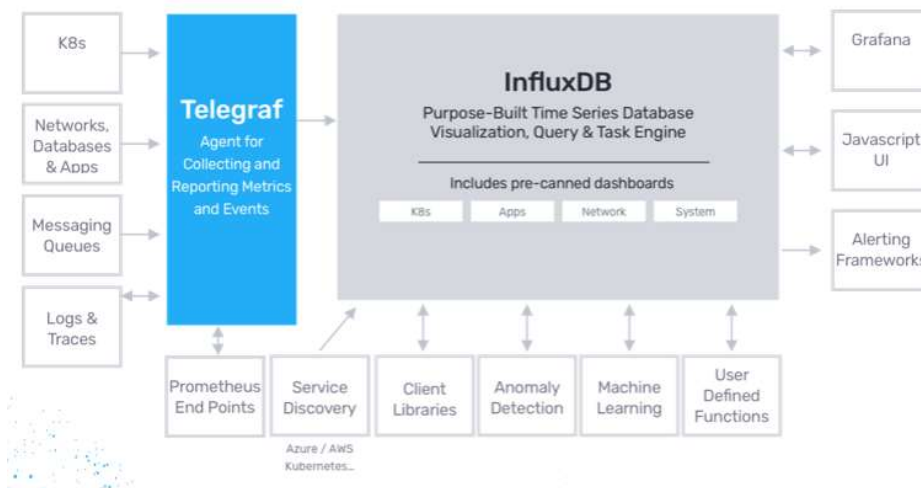
## UDP payload size is the maximum packet size to send.
# udp_payload = "512B"

## Optional TLS Config for use on HTTP connections.
# tls_ca = "/etc/telegraf/ca.pem"
# tls_cert = "/etc/telegraf/cert.pem"
# tls_key = "/etc/telegraf/key.pem"
## Use TLS but skip chain & host verification
# insecure_skip_verify = false
```

Eine detaillierte Beschreibung der Telegraf Konfigurationsmöglichkeiten finden Sie unter <https://github.com/influxdata/telegraf/blob/master/docs/CONFIGURATION.md>.

3.2.4. InfluxDB

InfluxDB ist ein hochleistungsfähiges Open Source Datenbankmanagementsystem, welches speziell für Zeitreihendaten entwickelt wurde und ist Teil des TICK (Telegraf, InfluxDB, Chronograf, Kapacitor) Stacks. Es ermöglicht Datenerfassung mit hohem Durchsatz sowie Komprimierung und Echtzeitabfragen der erfassten Daten. InfluxDB kommt in allen Anwendungsfällen als Datenspeicher zum Einsatz, bei denen große Mengen von Daten mit Zeitstempel verarbeitet werden, beispielsweise DevOps-Überwachungsdaten, Protokolldaten, Anwendungsmetriken, Daten von IoT-Sensoren oder Echtzeitanalysedaten. Als Abfragesprache wird InfluxQL verwendet, welche sehr nahe an SQL angelehnt ist. InfluxDB stellt diverse APIs zur Verfügung, welche nachfolgend dargestellt sind.



3.2.4.1. InfluxDB line-protocol

Das InfluxDB line-protocol ist ein textbasiertes Format um Datenpunkte in die Datenbank zu schreiben. Die Datenpunkte müssen exakt der Syntax des line-protocol entsprechen damit InfluxDB diese speichern kann. Das Protokoll besteht aus einer einfachen Textzeile mit den Komponenten Messung, Tag-Set, Feld-Set und Zeitstempel. Folgendes Beispiel zeigt einen Datenpunkt einer Temperaturmessung.

```
weather,location=us-midwest temperature=82 1465839830100400200
```

measurement	tag_set	field_set	timestamp
weather	location=us-midwest	temperature=82	1465839830100400200

An erster Stelle des Protokolls steht der Name der Messung, welcher verpflichtend anzugeben ist. Im obigen Beispiel „weather“.

Das Tag-Set ist optional und enthält alle Tags, die man der Messung mitgeben möchte. Tags können bei einer Analyse der Daten sehr nützlich sein um zum Beispiel die Daten zu gruppieren. Ein Tag besteht aus einem key-value pair, welches durch eine „=-“ Zeichen verbunden ist. Innerhalb eines Tag-Sets werden die Tags durch ein einfaches Komma ohne Leerzeichen verbunden. In unserem Beispiel besteht das Tag-Set aus einem Tag mit dem key-value pair „location=us-midwest“.

Tag-Set und Feld-Set sind mit einem Leerzeichen getrennt. Falls es kein Tag-Set gibt, werden Messung und Feld-Set mit einem Leerzeichen getrennt wie folgender Code zeigt.

```
weather temperature=82 1465839830100400200
```

Das Feld-Set enthält die Werte der Felder zu einem Datenpunkt. Ein Datenpunkt fordert zumindest ein Feld-Set mit einem Feld. Ein Feld besteht wiederum aus einem key-value pair, welches durch eine „=-“ Zeichen verbunden ist. Innerhalb eines Feld-Sets werden die Tags durch

ein einfaches Komma ohne Leerzeichen verbunden. In unserem Beispiel besteht das Feld-Set aus einem Feld mit dem key-value pair „temperature=82“.

Feld-Set und Zeitstempel, falls einer mitgegeben wird, sind wiederum mit einem Leerzeichen getrennt.

Am Ende des line-protocol kann optional ein Zeitstempel mitgegeben werden, der das Format „*Nanosekunden Unix Zeit*“ haben muss. Wird kein Zeitstempel mitgegeben, verwendet InfluxDB den lokalen Server-Zeitstempel.

Weitere Informationen bezüglich InfluxDB line-protocol finden Sie unter https://docs.influxdata.com/influxdb/v1.7/write_protocols/line_protocol_tutorial/. Die in diesem Dokument dargelegten Information sollen nur zu einem besseren Verständnis des gesamten Projektes dienen.

3.2.4.2. Konfiguration

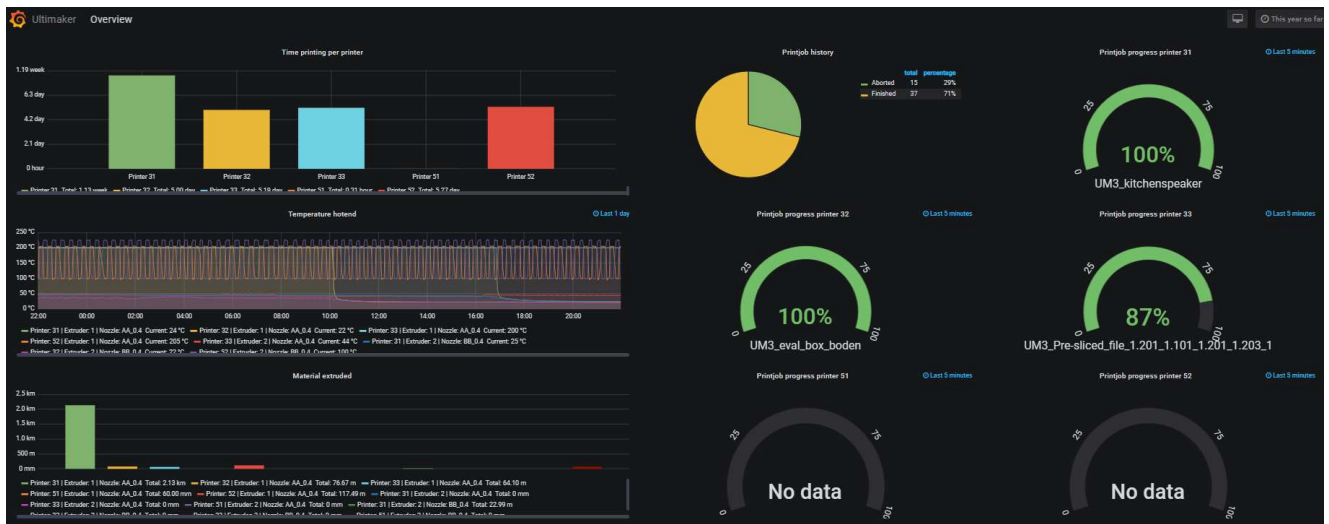
Die Datenbank muss in unserem Fall nicht zusätzlich konfiguriert werden. Der Name der Datenbank in der die Metriken gespeichert werden, wird bereits in der Telegraf Konfiguration festgelegt und automatisch in der InfluxDB erstellt.

3.2.5. Grafana

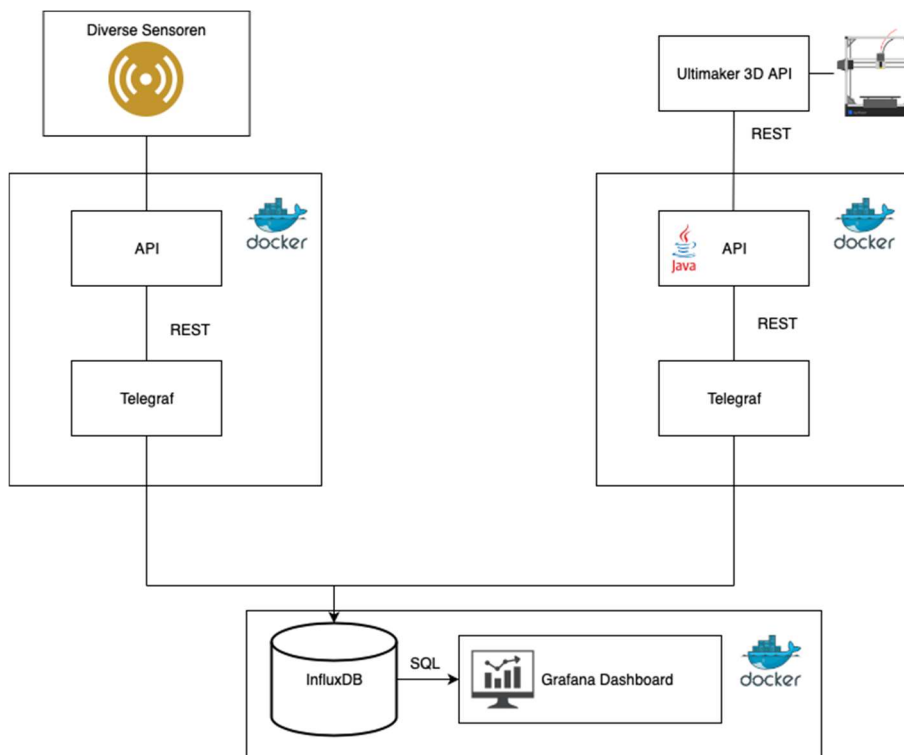
Grafana ist ein Open-Source-Dashboard, welches als Webanwendung ausgeführt wird. Mit Grafana können Metriken abgefragt, angezeigt und Alarme erstellt werden. Es kann aus über 30 verschiedenen Datenquellen, unter anderem InfluxDB, ausgewählt werden. Es bietet eine Vielzahl von Darstellungsmöglichkeiten von Graphen und Histogrammen bis Heatmaps und Geomaps, die es ermöglichen, Daten ansprechend analysieren zu können. Dabei beschränkt sich das Dashboard nicht nur auf die graphische Darstellung, sondern unterstützt auch unterschiedlichen Alarmierungen. Nach Definition der entsprechenden Schwellwerte alarmiert Grafana sofort und schickt den aktuellen Graphen auch gleich mit.

3.2.5.1. Erstellung Dashboards

Eine Beschreibung der Funktionen zur Erstellung von Dashboards finden Sie im Benutzerhandbuch. Es wurde für jede Metrik ein eigenes Dashboards erstellt und zusätzlich ein Übersichtsdashboard welches unterhalb dargestellt wird.



3.3. Einbindung in die Netzwerkinfrastruktur der Grand Garage



Die Grand Garage nutzt eine Vielzahl an Sensoren und Geräten. Es wurde dabei eine Infrastruktur geschaffen, mit der es möglich ist, eben diese Sensoren und Geräte über eine einheitliche Plattform überwachen und deren Daten analysieren zu können. Weiters ist es in dieser Infrastruktur möglich, zusätzliche Geräte an diese Plattform anzuschließen. Die Daten einzelner Geräte werden hierbei von Services aufgegriffen und über den Telegraf Agent an eine einheitliche Datenbank (InfluxDB) weitergegeben. Mit dem Dashboard-tool Grafana werden die Datenreihen aus der Datenbank abgefragt und grafisch je nach Einstellung visualisiert. Eine große Rolle innerhalb dieser Infrastruktur spielt dabei Docker, eine Software zur Containervisualisierung) mit der es möglich ist, die einzelnen Komponenten dieser Infrastruktur in isolierten Containern bereitzustellen.

3.3.1. Docker

Docker ist eine freie Software zur Container Virtualisierung. Jeder Container beinhaltet ein eigenes Linux Betriebssystem, die Applikation an sich und alle für die Ausführung benötigten Abhängigkeiten. Container werden aus sogenannten „Images“ hochgefahren und haben den Vorteil, dass diese effizienter sind als virtuelle Maschinen, da nur die notwendige Software darauf installiert ist. Mehrere Container können, um einfacher verwaltet (hoch und runter fahren) zu werden, über ein Docker-compose File gemeinsam isoliert ausgeführt werden. Wie aus der Grafik ersichtlich, besteht die Infrastruktur aus verschiedenen Docker Images, welche auf einem Host-Server laufen.

In unserem Fall besteht das Docker-compose File aus 2 Docker-Images:

- Java Webservice als API-Client für die API der Drucker, sowie als API Provider für Telegraf
- Telegraf Agent

Die InfluxDB, sowie das Grafana Dashboard befinden sich in bereits vorhandenen Containern und werden nur benutzt. Die InfluxDB wird durch die benötigten Measurements erweitert und in Grafana werden Dashboards angelegt. Für Testzwecke kann man jedoch auch eigene Container Instanzen starten, um Konflikte oder Fehler auf dem Produktivsystem zu vermeiden.

4. Installationsanleitung

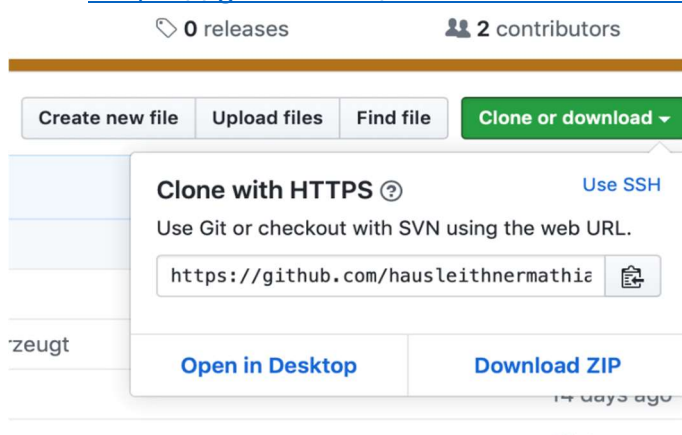
4.1. Herunterladen des Projekts

Das Projekt befindet sich auf GitHub und ist öffentlich und kann unter folgender URL geforked werden:

<https://github.com/hausleithnermathias/it-project.git>

Wenn Sie Git nicht auf Ihrem Rechner installiert haben, können Sie das Projekt auch einfach als Zip-Datei herunterladen.

<https://github.com/hausleithnermathias/it-project>



4.2. Docker Images erstellen und starten

Auf dem Server, an dem die Befehle ausgeführt werden und das Service gestartet wird, muss Docker installiert sein. Befindet sich das Projekt auf dem Host-Server, auf dem die Container laufen, kann man ein neues Docker-image erstellen. Das dazu benötigte Dockerfile liegt im Repository und ist bereits voreingestellt, Änderungen sind ohne zusätzliche Erweiterungen nicht notwendig. Da in der Produktivumgebung die Java Applikation gemeinsam mit Telegraf läuft, reicht es nicht aus ein einziges Image zu erstellen. Man benötigt eine Komposition aus diesen beiden Technologien. Abhilfe schafft hierbei Docker-compose. Hierzu werden alle benötigten Images in einem File angegeben. Da in unserem Fall in der Grand Garage bereits InfluxDB und Grafana Instanzen laufen, müssen diese nicht mehr von uns hochgefahren werden.

```
version: "3.7"

services:
  api:
    build: .
    expose:
      - 8080
    restart: always

  telegraf:
    image: telegraf:1.12.5
    networks:
      - default
      - influxdb
    volumes:
      - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf:ro
    restart: always

networks:
  influxdb:
    external:
      name: influxdb
```

```
version: "3.7"

services:
  api:
    build: .
    expose:
      - 8080
    restart: always

  telegraf:
    image: telegraf:1.12.5
    networks:
      - default
      - influxdb
    volumes:
      - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf:ro
    restart: always

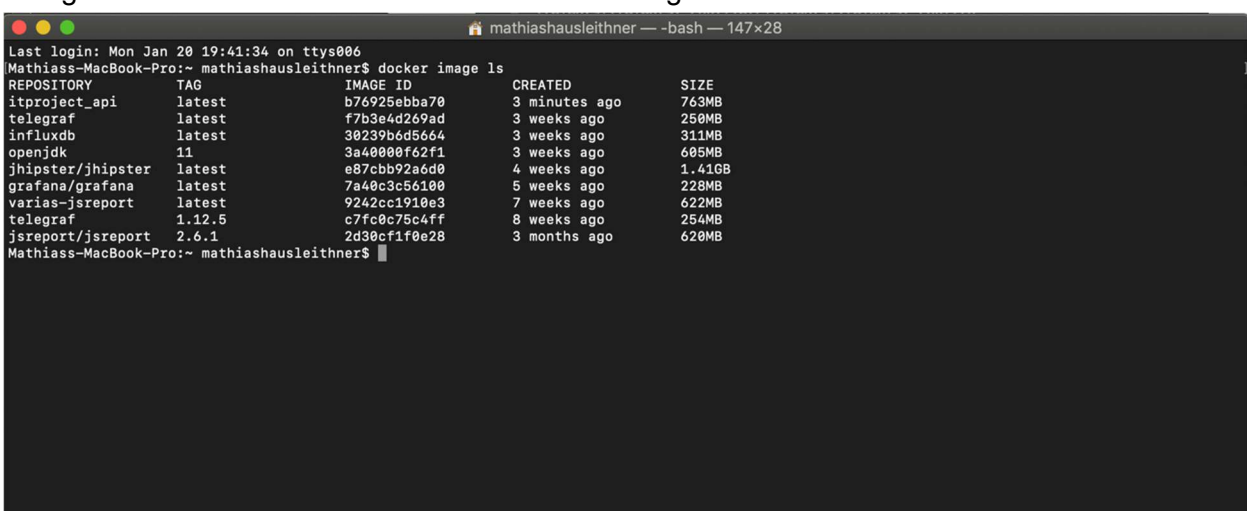
  influxdb:
    image: influxdb:latest
    environment:
      - INFLUXDB_DB=ultimaker
    ports:
      - "8083:8083"
      - "8086:8086"

  grafana:
    image: grafana/grafana:latest
    ports:
      - "5000:3000"
    links:
      - influxdb
networks:
  influxdb:
    external:
      name: influxdb
```

Falls zusätzlich eine InfluxDB und Grafana Instanz benötigt wird, müssen diese im docker-compose file angegeben werden. Über DockerHub, einer Datenbank von Docker-images, können diese ganz einfach geladen werden. Das docker-compose file muss dann folgendermaßen aussehen:

Über den Terminalbefehl „docker-compose up“ im äußersten Programmverzeichnis werden alle Images erstellt und hochgefahren.

Über den Terminalbefehl „docker image ls“ werden alle vorhandenen Images aufgelistet. Bei erfolgreichen Build Prozess müssen die neuen Images nun aufscheinen.



```
mathiashausleithner ~ -bash — 147x28
Last login: Mon Jan 20 19:41:34 on ttys006
Mathias-MacBook-Pro:~ mathiashausleithner$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
itproject_api        latest             b76925ebba70       3 minutes ago      763MB
telegraf             latest            f7b3e4d269ad       3 weeks ago        250MB
influxdb             latest            30239b6d5664       3 weeks ago        311MB
openjdk              11                3a40000f62f1       3 weeks ago        605MB
jhipster/jhipster    latest            e87cbb92a6d0       4 weeks ago        1.41GB
grafana/grafana      latest            7a40c3c56100       5 weeks ago        228MB
varias-jsreport       latest            9242cc1910e3       7 weeks ago        622MB
telegraf             1.12.5            c7fc0c75c4ff       8 weeks ago        254MB
jsreport/jsreport    2.6.1             2d30cf1f0e28       3 months ago       620MB
Mathias-MacBook-Pro:~ mathiashausleithner$
```

```

Last login: Mon Jan 20 19:43:05 on ttys006
Mathias-MacBook-Pro:~ mathiashausleithner$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED            STATUS              PORTS                               NAMES
f2104f9e7ee        telegraf:1.2.5     "/entrypoint.sh tele-"  12 minutes ago    Up 12 minutes      8092/udp, 8125/udp, 8094/tcp         itproject_telegraf_1
2ae492f52421        grafana/grafana:latest "/run.sh"              12 minutes ago    Up 12 minutes      0.0.0.0:3000->3000/tcp              itproject_grafana_1
f78e9c63bbcd        itproject_api       "java -jar app/target-" 12 minutes ago    Up 12 minutes      0.0.0.0:8080->8080/tcp              itproject_api_1
a0ba84eb58f7        influxdb:latest     "/entrypoint.sh infl-"  12 minutes ago    Up 12 minutes      0.0.0.0:8083->8083/tcp, 0.0.0.0:8086->8086/tcp itproject_influxdb_1
Mathias-MacBook-Pro:~ mathiashausleithner$

```

Gleichzeitig müssen von diesen Images Container verfügbar sein. Überprüfen kann man das durch den Befehl „docker ps“

Die Applikation wird nun ausgeführt und speichert Daten in die Datenbank. Über „http://localhost:3000“ kann auf Grafana zugegriffen werden.

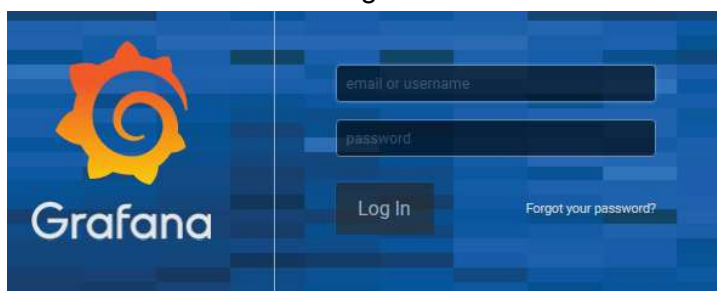
Weitere wichtige Docker-Befehle

- docker ps → listet alle laufenden Container
- docker rmi IMAGENAME → löscht das gewählte Image
- docker stop CONTAINERNAME → stoppt den angegeben Container
- docker-compose down → im Programmverzeichnis ausführen, um die Container herunterzufahren

4.3. Konfiguration der Dashboards

4.3.1. Aufruf Grafana

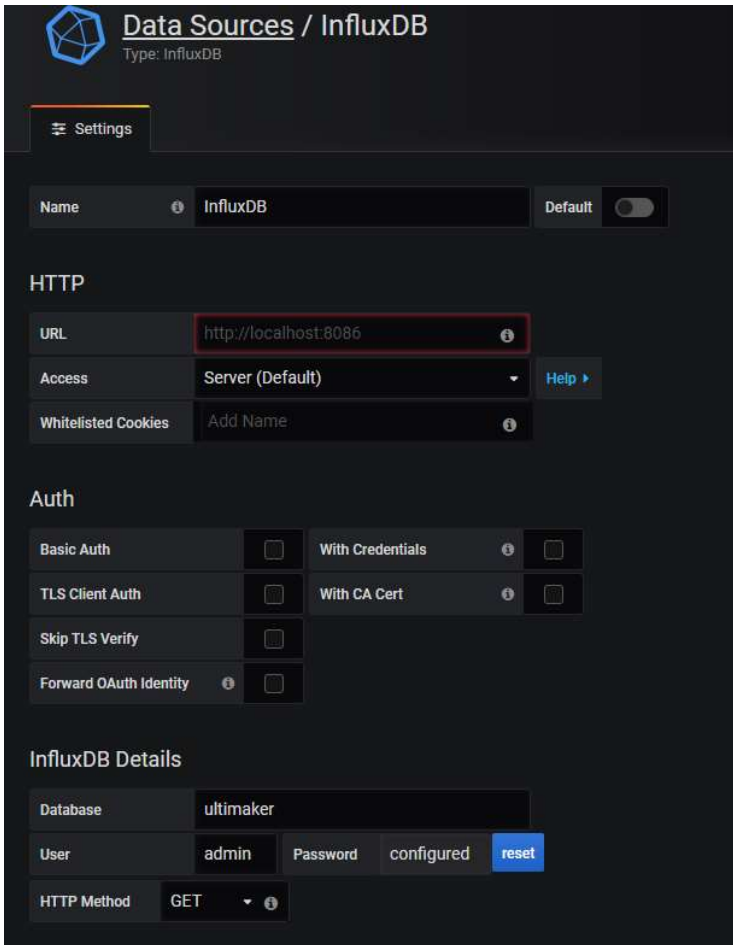
Um das Grafana Dashboard zu öffnen, öffnen Sie den Webbrowser und geben Sie in der Adressleiste die URL <https://reports.grandgarage.eu/> ein. Sie müssen sich dabei nicht im Netzwerk der Grand Garage befinden. Sollte das Dashboard nicht erreichbar sein, wenden Sie sich bitte an die IT-Abteilung.



Sie werden nun aufgefordert, Zugangsdaten einzugeben. Um neue Dashboards zu konfigurieren benötigen Sie einen Editor-Zugang. Bitte wenden Sie sich dazu an die IT-Abteilung.

4.3.2. Konfiguration der Datenquelle

Bevor Druckerdaten abgerufen werden können, muss die Datenbank „ultimaker“ von InfluxDB konfiguriert werden. Für die Konfiguration benötigt man jedoch Administratorrechte und kann daher nur von einem IT Personal der Grand Garage durchgeführt werden.



Data Sources / InfluxDB
Type: InfluxDB

Settings

Name: InfluxDB Default ☐

HTTP

URL: ?

Access: Server (Default) Help

Whitelisted Cookies: ?

Auth

Basic Auth: ☐ With Credentials: ? ☐

TLS Client Auth: ☐ With CA Cert: ? ☐

Skip TLS Verify: ☐

Forward OAuth Identity: ? ☐

InfluxDB Details

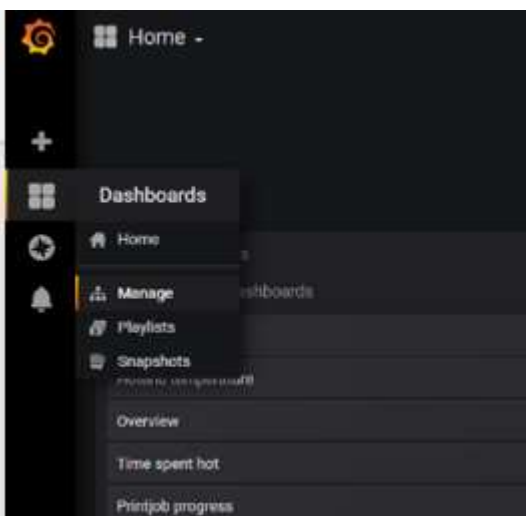
Database:

User: Password: configured reset

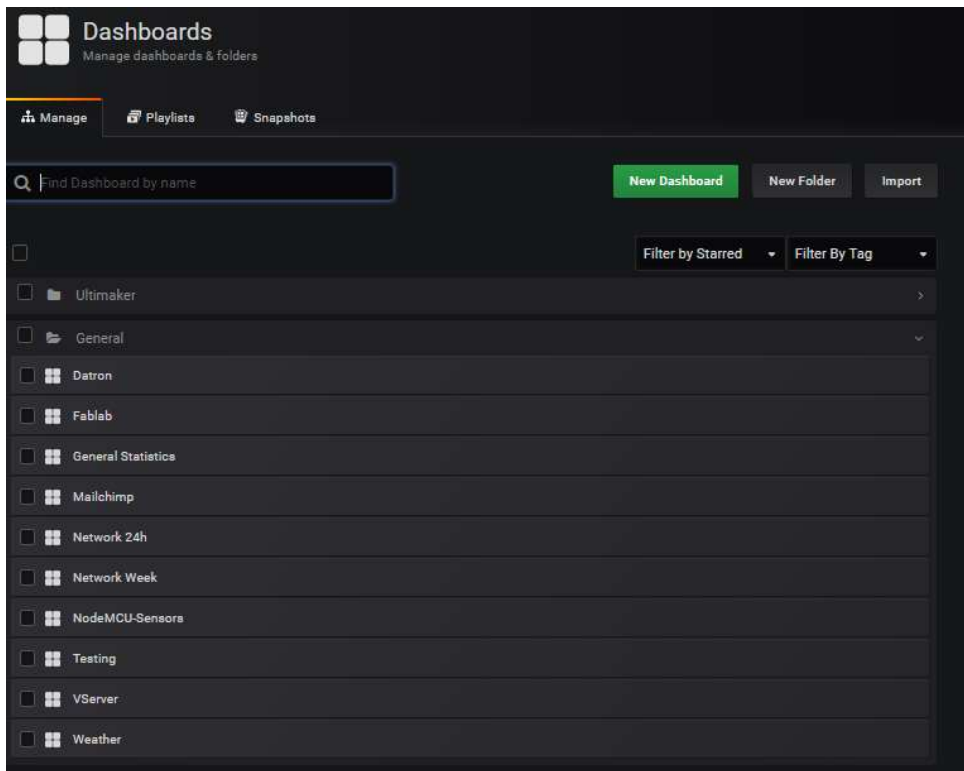
HTTP Method: GET ?

4.3.3. Import der bestehenden Dashboards

In dem Repository finden sie einen Ordner „grafana-dashboards“, in dem sich die bereits erstellten Dashboards als JSON-Files befinden. Um diese in Grafana zu importieren, klicken Sie in der linken Taskleiste auf den „Dashboards“ Button und anschließend auf „Manage“.



Im folgenden Fenster sind die bereits erstellten Dashboards ersichtlich. Um ein neues Dashboard zu importieren, klicken Sie auf „Import“ und anschließend „Upload .json File“. Wählen Sie nun das JSON-File des gewünschten Dashboards aus und importieren Sie es.



4.4. Wartung der Komponenten

4.4.1. Datenaufbereitungsservice

Folgende Softwarepakete müssen auf dem Rechner vorinstalliert sein, um das Programm ausführen zu können:

- Maven 3.6.3 (Standpunkt 2.1.2020, eine neuere Version kann problemlos verwendet werden)
- Java 11 (Eine neuere oder ältere Version kann unter Umständen zu Problemen führen)
- Docker Daemon (Dockerimages erstellen und lokal testen)

Es ist wichtig, dass zu diesen beiden Tools eine Systemvariable im Betriebssystem angelegt wird, da sonst die Terminal Befehle (z. B. Builden) nicht vom Betriebssystem erkannt werden. Wie man Maven oder Java als Systemvariable hinzufügt hängt vom Betriebssystem ab und kann einfach im Internet recherchiert werden.

Wenn die beiden Terminalbefehle `mvn --version` und `java --version` gefunden werden und die Versionsnummer anzeigen wurde die Installation ordnungsgemäß durchgeführt.

Das Java Projekt liegt auf einem Git-Repository und kann über folgende URL geforked werden:

<https://github.com/hausleithnermathias/it-project.git>

Zur Bearbeitung des Source-codes empfiehlt sich eine Entwicklungsumgebung wie Eclipse oder IntelliJ IDEA.

Wurde der Programmcode geändert, muss nach dem Speichern die Applikation neu gebuildet werden. Diesen Vorgang übernimmt Maven und kann mit folgenden Befehl im Wurzelverzeichnis des Projekts ausgeführt werden:

```
mvn clean package -DskipTests
```

Fehlerfreier Build Prozess:

```
[INFO] Reactor Summary:
[INFO]
[INFO] swagger-java-client 1.0.0 ..... SUCCESS [ 7.560 s]
[INFO] api-service 0.0.1-SNAPSHOT ..... SUCCESS [ 0.039 s]
[INFO] api 0.0.1-SNAPSHOT ..... SUCCESS [ 1.701 s]
[INFO] app 0.0.1-SNAPSHOT ..... SUCCESS [ 0.763 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.298 s
```

Nun kann man nach der Installationsanleitung vorgehen und die Docker-Images erstellen.

4.4.2. Telegraf

Alle Einstellungen von Telegraf befinden sich in der Datei „telegraf.config“, die im Repository im Ordner „telegraf“ zu finden ist. Für eine detaillierte Beschreibung der Konfiguration wird auf den Abschnitt „Telegraf“ in der technische Dokumentation verwiesen. Nach Änderungen in der Konfiguration müssen Sie das Dockerimage neu erstellen und hochfahren, damit die Änderungen wirksam werden.

Wichtig: Falls Sie das globale data collection intervall ändern (Standard: 20s), müssen Sie die Queries des „Printer status“ Dashboards in Grafana anpassen, da die Ergebnisse der Query direkt mit dem Intervall zusammenhängen. In der Query wird berechnet wie oft ein bestimmter Druckerstatus (z.B. printing, idle,..) in 1 Stunde vorkommt. Die Berechnung lautet im jetzigen Fall daher (Anzahl Status / 180), da 20s in einer Stunde 180 mal vorkommen. Möchte man zum Beispiel das Intervall auf 10s verringern, so muss man die Berechnung auf (Anzahl Status / 360) ändern. Wie man Queries in Grafana verändert, entnehmen Sie bitte dem Benutzerhandbuch.

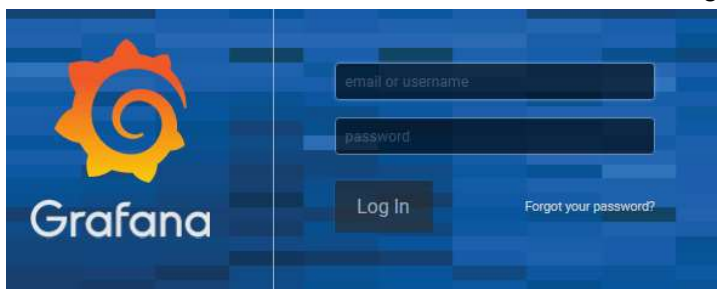
4.4.3. Grafana

Eine Anleitung für die Bearbeitung von den Grafana Dashboards finden Sie im beigelegten Benutzerhandbuch.

5. Benutzerhandbuch

5.1. Einstieg

Nach erfolgreicher Installation, Konfiguration und Start aller benötigten Softwarekomponenten, können die Metriken der Ultimaker3 und Ultimaker5 Drucker in vorkonfigurierten Dashboards der Webanwendung Grafana eingesehen werden. Um Grafana zu öffnen, öffnen Sie den Webbrowser und geben Sie in der Adressleiste die URL <https://reports.grandgarage.eu/> ein. Sie müssen sich dabei nicht im Netzwerk der Grand Garage befinden.

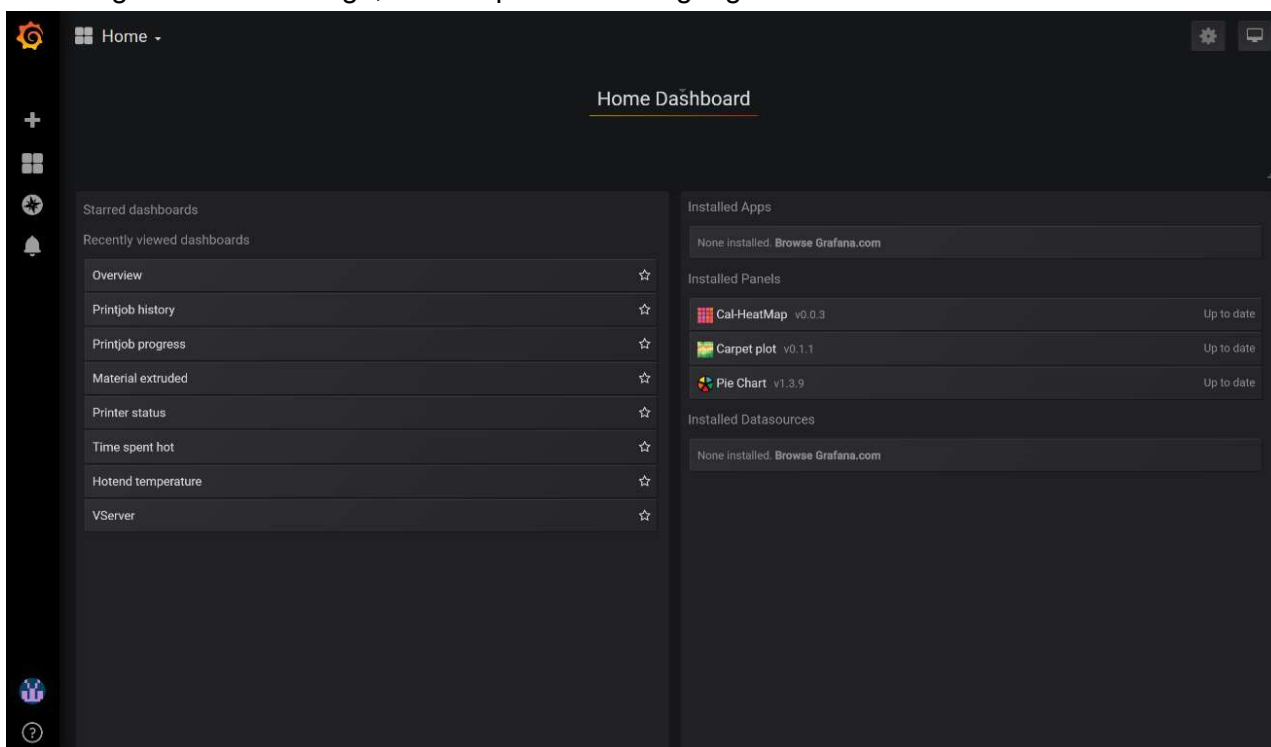


Sie werden nun aufgefordert, Zugangsdaten einzugeben. Mit folgenden Zugangsdaten erhalten Sie Zugang als Rolle „Viewer“. Das heißt Sie können keine Dashboards bearbeiten.

Username: user

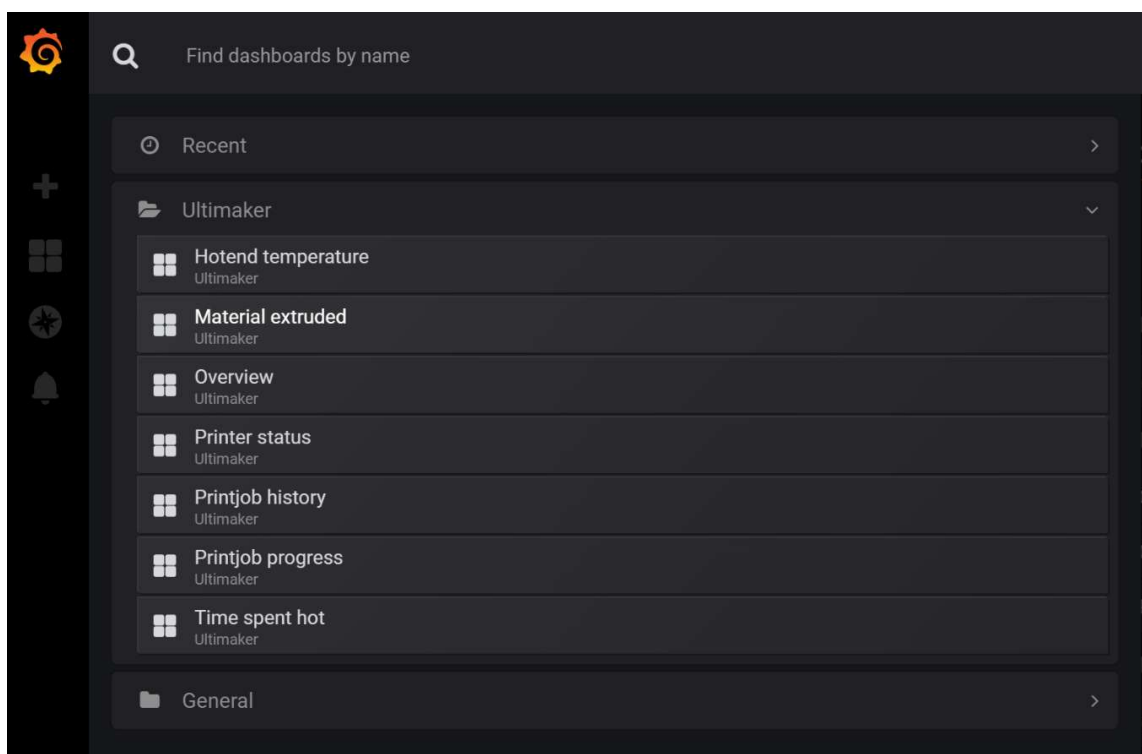
Passwort: wifi4members

Möchte Sie Änderungen an den Dashboards vornehmen, wenden Sie sich bitte an die IT-Abteilung der Grand Garage, um entsprechende Zugangsdaten mit Editor-Rechten zu erhalten.



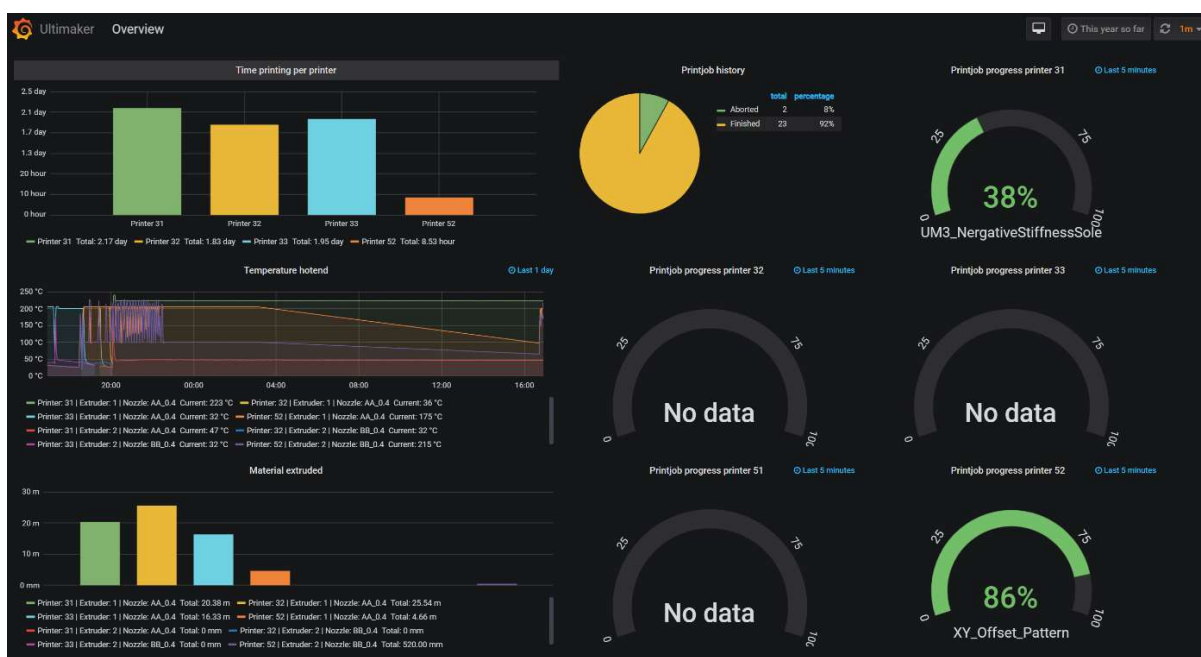
5.2. Dashboards

Um zu den Dashboards der Ultimaker Drucker zu gelangen, klicken Sie auf Home und es öffnet sich ein Dropdownmenü mit den eingerichteten Dashboards. Unter dem Ordner „Ultimaker“ finden Sie alle Dashboards der Drucker.



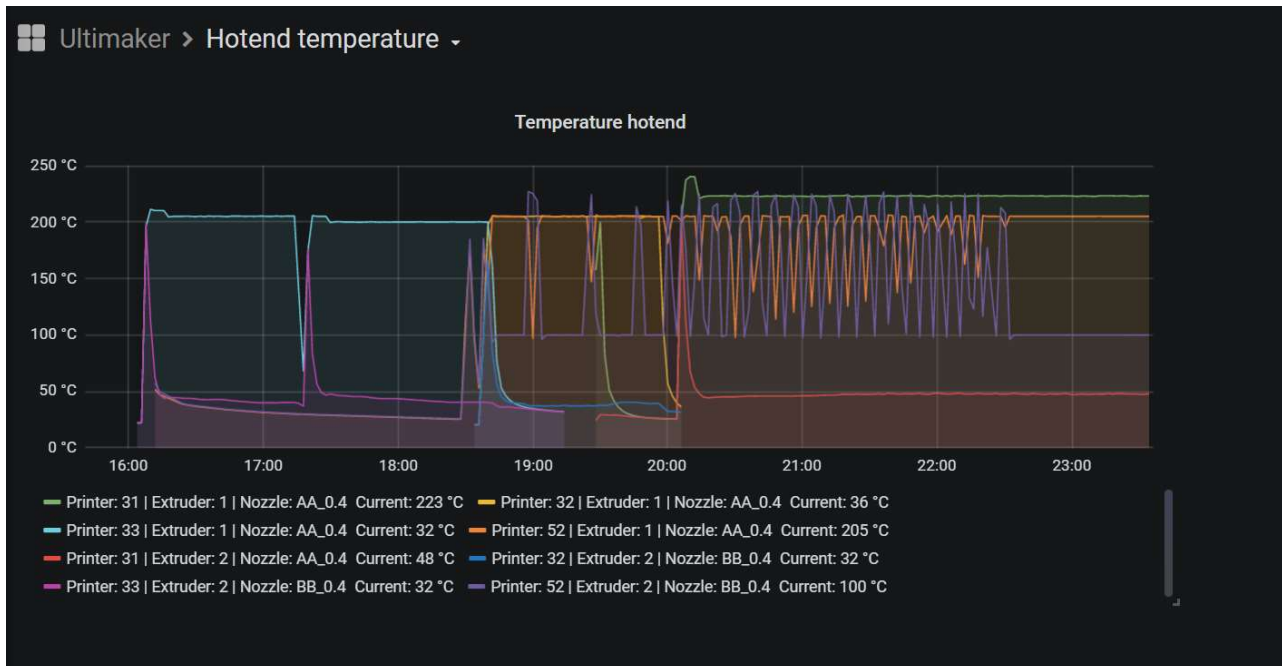
Die Dashboards wurden grundsätzlich nach der Art der aufgezeichneten Metriken kategorisiert. Zusätzlich wurde ein Dashboard „Overview“ erstellt, welches als gesamtheitliche Übersicht der einzelnen Metriken dienen soll. Ein Dashboard besteht aus mindestens einem Panel.

5.2.1. Overview



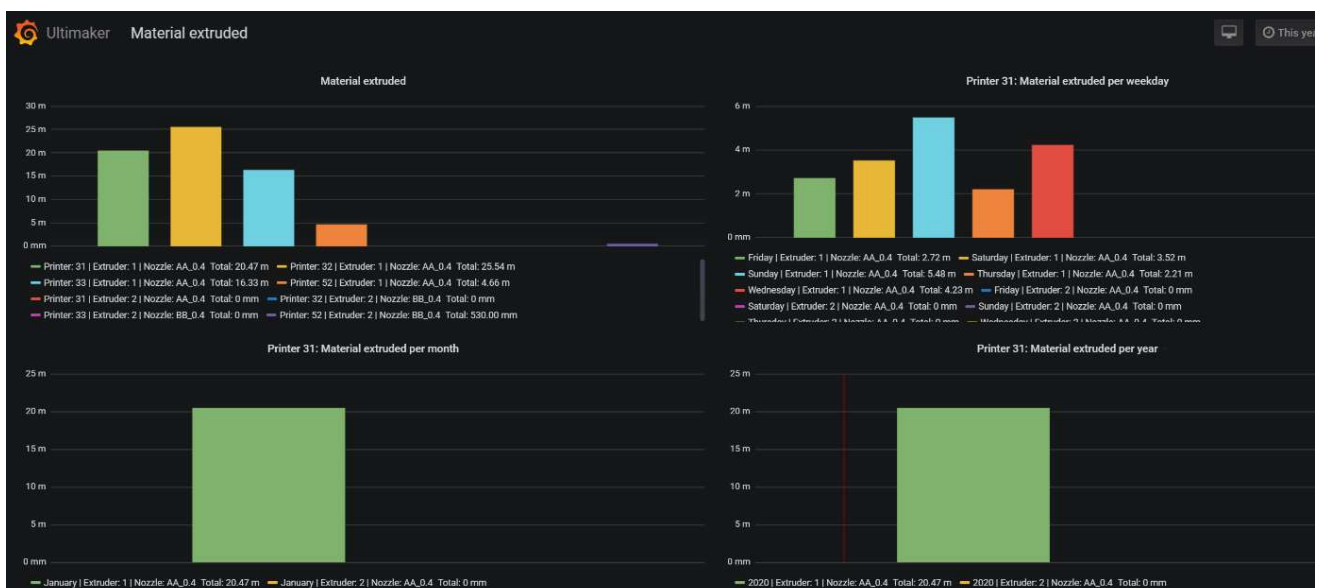
Dieses Dashboard enthält verschiedene Panels aus den einzelnen Dashboards der Metriken und soll einen grundsätzlichen Überblick geben. Es wird auch in einer Dashboard-Playlist verwendet, die auf Flatscreens in der Grand Garage läuft.

5.2.2. Hotend temperature



In diesem Dashboard wird der Temperaturverlauf der Hotends der Drucker angezeigt. Sowohl die Ultimaker3 als auch die Ultimaker5 besitzen zwei Extruder mit jeweils einem dazugehörigen Hotend inklusive Düse. Es gibt daher je Drucker zwei Temperaturverläufe. Zusätzlich wird je Extruder noch der Düsentyp aufgezeichnet, der aktuell im Hotend verwendet wird.

5.2.3. Material extruded



In diesem Dashboard wird die Menge des extrudierten Materials der Drucker angezeigt. Wie bei der „Hotend temperature“ wird hier bei einem Drucker wieder zwischen zwei Extrudern inklusive Hotend und Düse unterschieden. Zusätzlich zu dem Hauptpanel wird, gibt es noch je Drucker drei Panels, in denen die extrudierte Materialmenge je Tag, je Monat und je Jahr angezeigt wird.

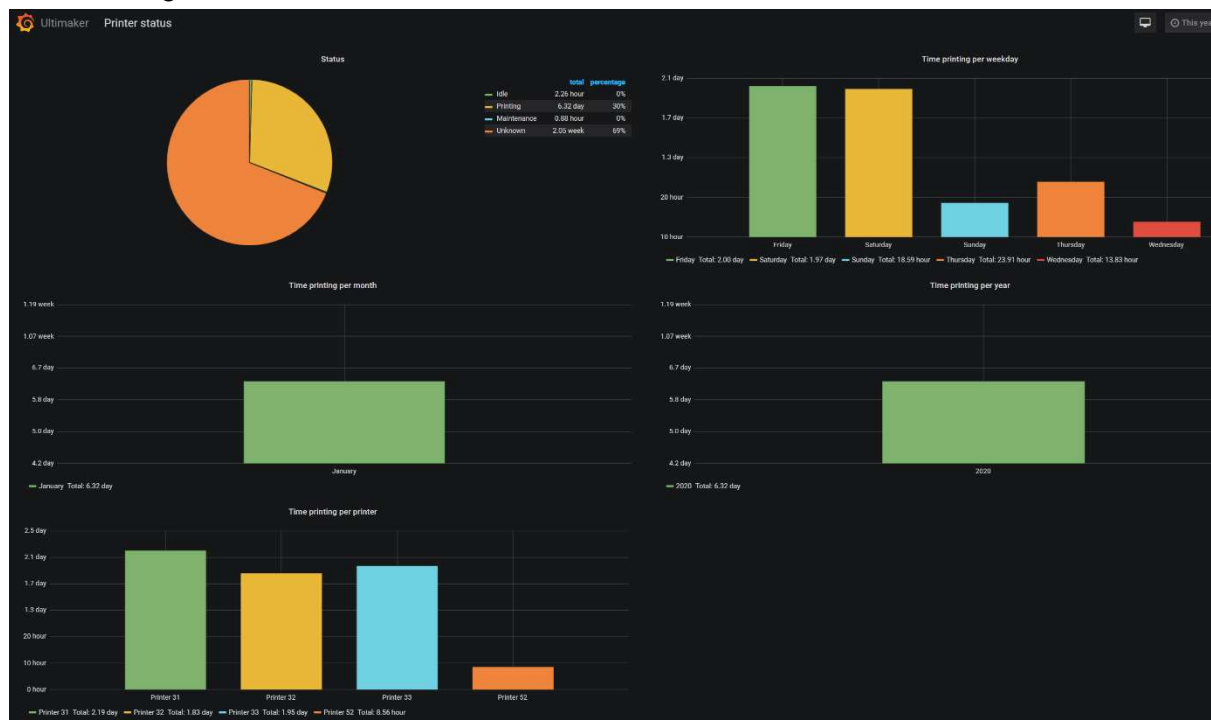
5.2.4. Printer status

In diesem Dashboard wird der Status der Drucker im zeitlichen Verlauf ausgewertet. Es gibt folgende Status in denen sich ein Drucker befinden kann:

- Idle
- Printing
- Maintenance
- Error
- Booting

Ist ein Drucker ausgeschaltet oder nicht erreichbar wird in diesem Zeitraum der Status „Unknown“ angezeigt.

Neben einer Statistik über den gesamten Zeitraum gibt es noch Panels die den Status nach den Kriterien Tag, Monat, Jahr und Drucker auswerten.



5.2.5. Printjob history

In diesem Dashboard wird die Anzahl der Printjobs statistisch ausgewertet. Es gibt folgende Ergebnisse, die ein Printjob haben kann:

- Finished
- Aborted
- Failed

Neben einer Statistik über den gesamten Zeitraum, gibt es noch Panels die die Druckerhistorie nach den Kriterien Tag, Monat und Jahr auswerten.



5.2.6. Printjob progress

In diesem Dashboard wird der Druckfortschritt in % der einzelnen Drucker angezeigt. Natürlich sind in den Panels nur dann Daten ersichtlich, wenn ein Drucker gerade einen Druckjob ausführt.



5.2.7. Time spent hot



In diesem Dashboard wird die Menge die Dauer angezeigt, in der die Hotends der Drucker sich in einem „heißen“ Zustand befinden. Wie bei den Metriken „Hotend temperature“ und „Material extruded“ wird hier bei einem Drucker wieder zwischen zwei Extrudern inklusive Hotend und Düse unterschieden. Zusätzlich zu dem Hauptpanel wird, gibt es noch je Drucker drei Panels, in denen die „Time spent hot“ je Tag, je Monat und je Jahr angezeigt wird.

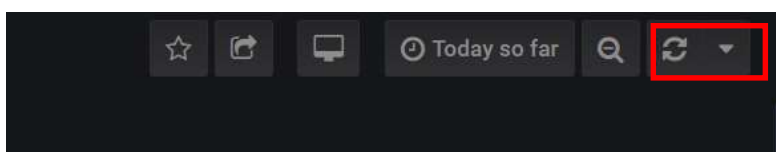
5.3. Dashboard Funktionen

In diesem Abschnitt werden Funktionen beschrieben, die einem User im Dashboard zur Verfügung stehen. Grundsätzlich stehen User mit der Rolle „Viewer“ nur Funktionen zur Verfügung mit denen dieser nichts am Dashboard bearbeiten kann. Editor-User haben zusätzlich zu den Funktionen der Rolle „Viewer“ auch die Möglichkeiten ein Dashboard zu bearbeiten. Bitte beachten Sie, dass nur einige wichtige Funktionen beschrieben werden. Für eine detaillierte Beschreibung besuchen Sie bitte die offizielle Grafana Dokumentationsplattform unter <https://grafana.com/docs/grafana/latest/>.

5.3.1. Rolle „Viewer“

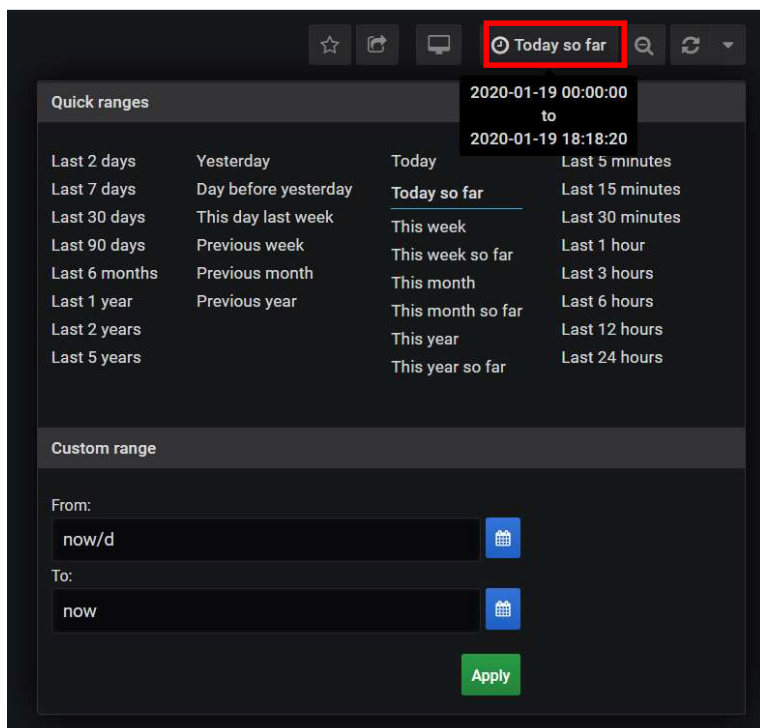
5.3.1.1. Refresh Dashboard

Um die angezeigten Daten zu aktualisieren klicken Sie einfach oben rechts in der Taskleiste auf den Refreshbutton. Möchten Sie periodisch aktuelle Daten erhalten, können Sie neben dem Refreshbutton auf den Pfeil klicken und die Zeitperiode einstellen, in der die Daten des Dashboards aktualisiert werden sollen.



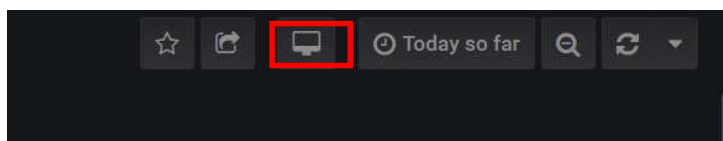
5.3.1.2. Timerange

Möchten sie die Timerange ändern, in der Daten angezeigt werden sollen, so können Sie diese ebenfalls in der oberen Taskleiste des Dashboards ändern.



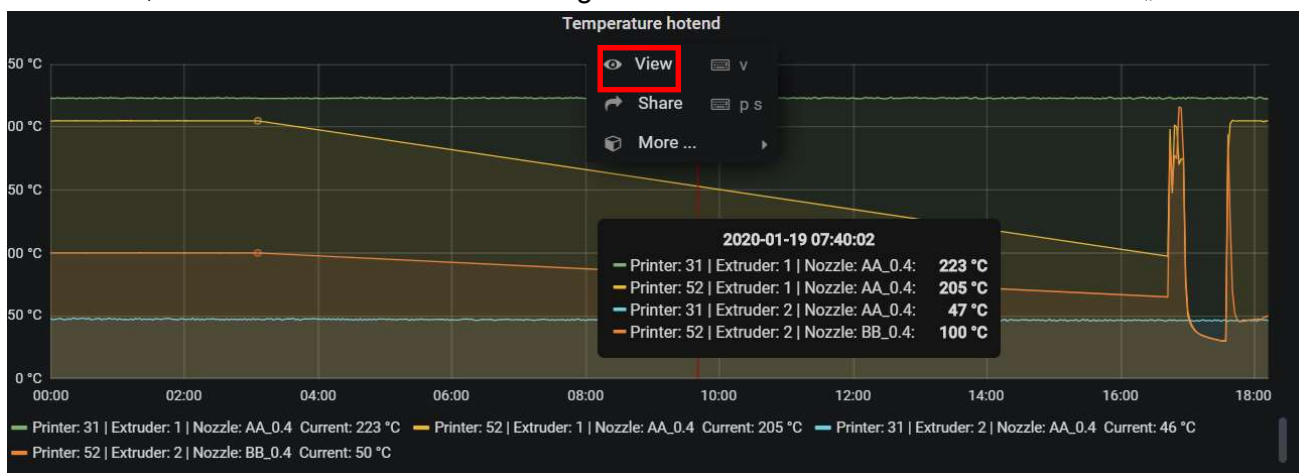
5.3.1.3. Cycle view mode

Mit dieser Funktion können Sie das Dashboard in verschiedene View Modes versetzen.



5.3.1.4. View Panel

Möchten Sie in einem Dashboard mit mehreren Panels ein einzelnes Panel im Vollbildmodus betrachten, klicken Sie auf den Header des gewünschten Panels und anschließend auf „view“.



5.3.2. Rolle „Editor“

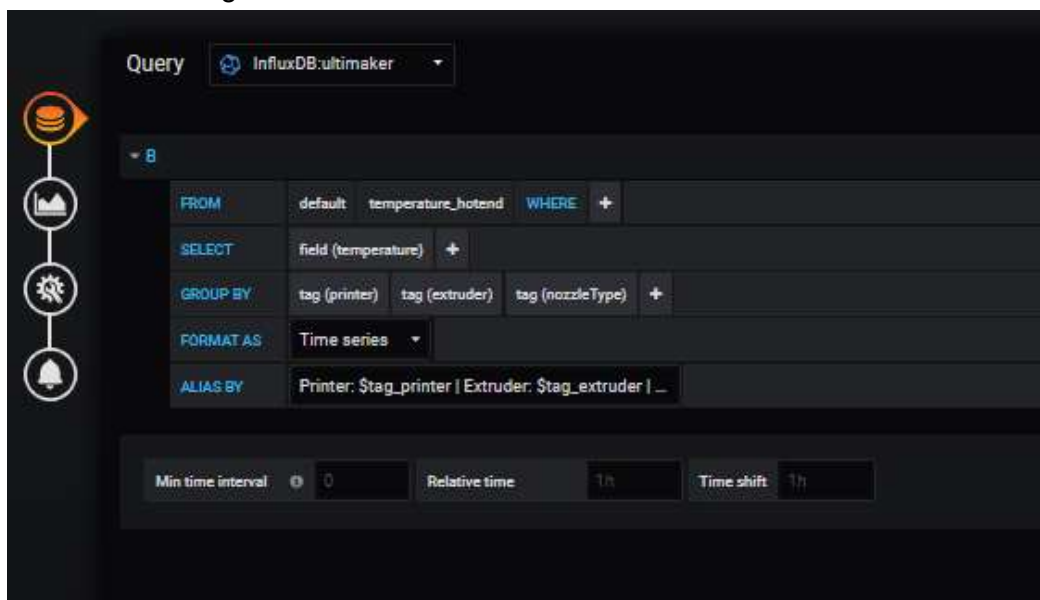
Mit diesen Funktionen können Sie Veränderungen an den verschiedenen Dashboards vornehmen. Bitte klären Sie sämtliche Änderungen mit der IT-Abteilung der Grand Garage ab.

5.3.2.1. Edit Panel

Um Veränderungen an einem Panel vorzunehmen, klicken Sie auf den Header des gewünschten Panels und anschließend auf „edit“. Der Bearbeitungsbereich ist in folgende Abschnitte unterteilt.

Queries

Hier wird die Datenquelle ausgewählt und die Queries definiert. Die Abfragesprache ist sehr nahe an SQL angelehnt.



Visualization

In diesem Abschnitt wird die gewünschte Visualisierungsform ausgewählt und konfiguriert.



General

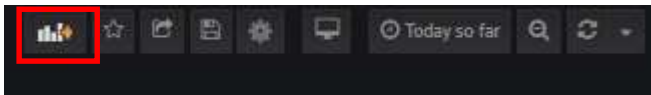
Im Abschnitt General können noch allgemeine Information wie Panel Titel oder eine Beschreibung hinzugefügt werden.

Alert

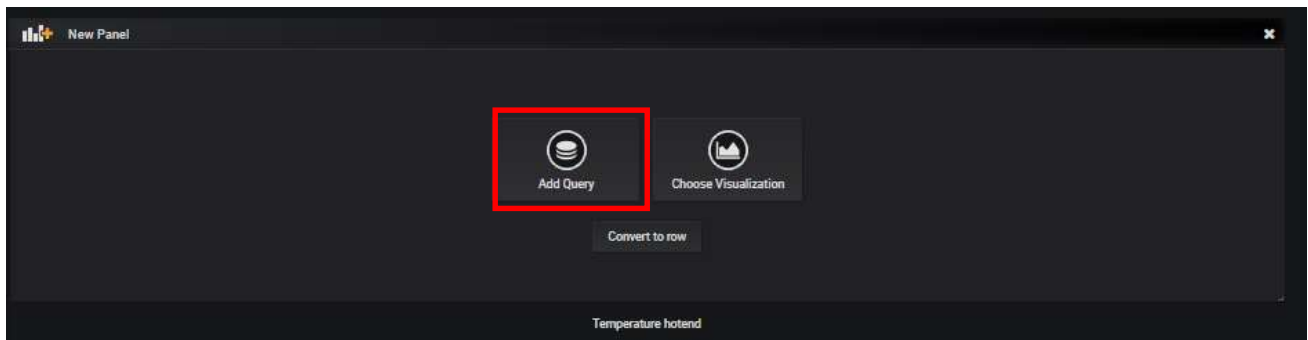
Es besteht auch die Möglichkeit bei bestimmten Bedingungen einen Alarm auszulösen und diesen als Nachricht an jemanden zu senden. Zum Beispiel könnte man bei der Überschreitung einer kritischen Temperatur eines Extruders einen Alarm setzen.

5.3.2.2. Add Panel

Um ein neues Panel hinzuzufügen, klicken Sie in der oberen Taskleiste auf folgenden Button.



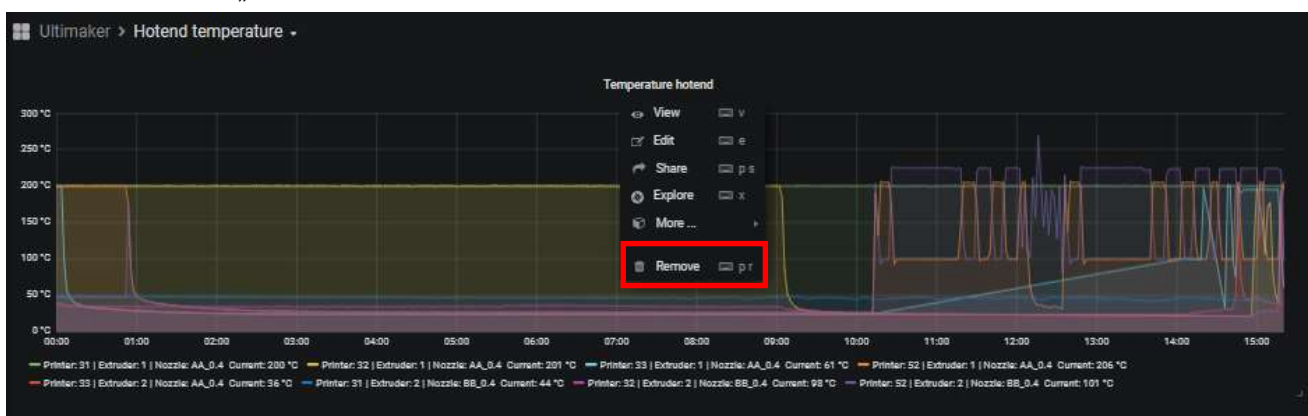
Um die Konfiguration des neuen Panels zu starten, klicken sie auf „Add Query“.



Für eine nähere Beschreibung der einzelnen Abschnitte im Editor-Bereich siehe Funktion „Edit Panel“ des Benutzerhandbuches.

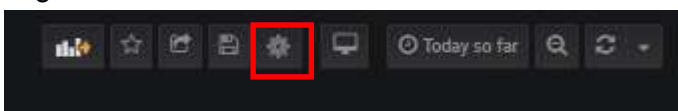
5.3.2.3. Remove Panel

Um ein Panel zu löschen, klicken Sie auf den Header des gewünschten Panels und anschließend auf „remove“.



5.3.2.4. Dashboard Settings

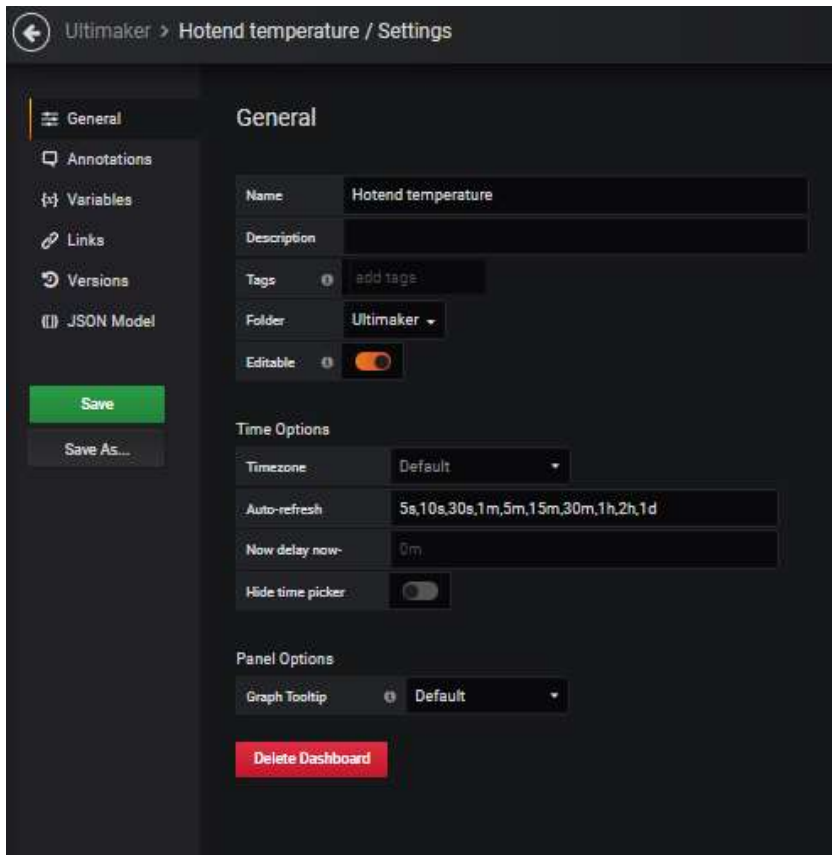
Um zu den Einstellungen des Dashboards zu gelangen, klicken Sie in der oberen Taskleiste auf folgenden Button.



In den Dashboard Settings können unter Anderen folgende Einstellungen vorgenommen bzw. Informationen angesehen werden:

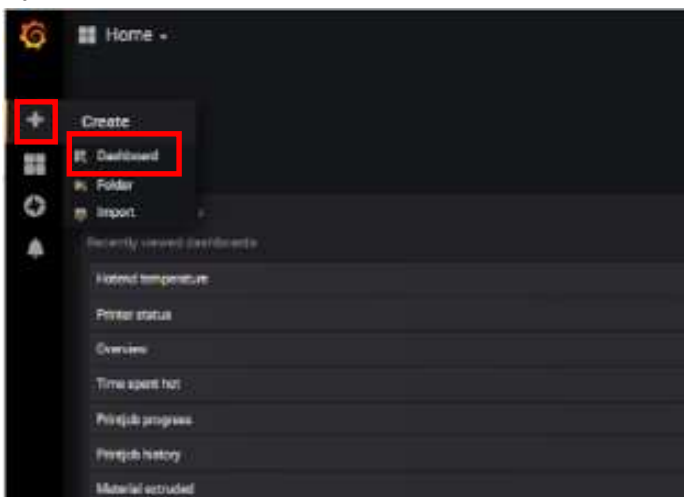
- Allgemeine Einstellungen: Name, Beschreibung, Ordner,...
- Zeit Einstellungen
- Versionsverlauf des Dashboards

- JSON Model: kann verwendet werden, um Dashboard in einer anderen Grafana Instanz zu importieren



5.3.2.5. Create Dashboard

Um ein neues Dashboard zu erzeugen, klicken Sie auf der linken Taskleiste auf das „Create Symbol“ und anschließend auf „Dashboard“.



Nun können Sie beginnen, Panels zum Dashboard hinzuzufügen.