**FH JOANNEUM - University of Applied Sciences**

**Smart Application Platform Graz**

**Dokumentation**

**Integriertes Praxisprojekt**

**Verfasser:**

**Buchsbaum Julia**

**Janusch Michael**

**Betreuer:**

**FH-Prof. Dipl.-Ing. Dr.techn. Peter Salhofer**

**Graz, Feber 2018**



Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc506925120)

[Abbildungsverzeichnis 3](#_Toc506925121)

[1 Einleitung 4](#_Toc506925122)

[1.1 Problemstellung 4](#_Toc506925123)

[1.2 Zielsetzung 4](#_Toc506925124)

[2 Infrastruktur 5](#_Toc506925125)

[2.1 FIWARE Komponenten 5](#_Toc506925126)

[2.1.1 Orion 5](#_Toc506925127)

[2.1.2 IDAS 5](#_Toc506925128)

[2.1.3 Cygnus 5](#_Toc506925129)

[2.1.4 Comet 5](#_Toc506925130)

[2.1.5 Mongo DB 5](#_Toc506925131)

[2.1.6 Wirecloud 5](#_Toc506925132)

[2.1.7 Postgres DB 5](#_Toc506925133)

[2.2 Docker Infrastruktur 5](#_Toc506925134)

[2.3 FIWARE Lab Infrastruktur 5](#_Toc506925135)

[3 Dashboard 6](#_Toc506925136)

[3.1 Wirecloud 6](#_Toc506925137)

[3.1.1 Widgets 6](#_Toc506925138)

[3.1.2 Operatoren 7](#_Toc506925139)

[3.1.3 Themes 8](#_Toc506925140)

[3.2 Vorhandene Widgets und Operatoren 10](#_Toc506925141)

[3.3 Custom Widgets und Operatoren 10](#_Toc506925142)

[3.4 Beispiele 10](#_Toc506925143)

[4 Security 12](#_Toc506925144)

[5 Fazit 13](#_Toc506925145)

[6 Linksammlung 14](#_Toc506925146)

Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

1. Einleitung
   1. Problemstellung
   2. Zielsetzung
2. Infrastruktur
   1. FIWARE Komponenten
      1. Orion
      2. IDAS
      3. Cygnus
      4. Comet
      5. Mongo DB
      6. Wirecloud
      7. Postgres DB
   2. Docker Infrastruktur
   3. FIWARE Lab Infrastruktur
3. Dashboard
   1. Wirecloud

Wirecloud ist ein Teil der FIWARE Platform und ermöglicht die Erstellung von Dashboards in Form einer Web Applikation. Somit stelle Wirecloud die Frontend Komponente einer FIWARE Infrastruktur dar. Die Dashboards werden im Wirecloud Kontext als Mashups bezeichnet und werden mit sogenannten Widgets und Operatoren aufgebaut. Diese Bausteine verfügen über Input sowie auch Output Endpunkte und können über diese miteinander verbunden werden. Im Folgenden werden Widgets und Operatoren, sowie Wirecloud Themes näher beschrieben. Diese Beschreibung fasst die wichtigsten Punkte der offiziellen Wirecloud Dokumentation [1] zusammen.

* + 1. Widgets

**Basis Konzepte**

Folgende Basis Konzepte sollten bei dem Design und der Entwicklung von Widgets beachtet werden:

* Widgets sollten kleine und wiederverwendbare Web Applikationen darstellen.
* Widgets sollten so generisch wie möglich sein, können aber in Ausnahmefällen auch spezifische, ad-hoc Lösungen darstellen.
* Widgets sollten an reale Probleme angepasst werden.
* Widgets sind hauptsächlich Elemente des Front Ends. Widgets können zwar direkt auf Backend Services zugreifen, diese Aufgabe sollte aber wenn möglich von Operatoren übernommen werden.
* Während der Entwicklung von Widgets kann jede Technologie, welche von Web Browsern akzeptiert wird verwendet werden.

**Aufbau von Widgets**

Widgets bestehen grundsätzlich aus den folgenden Komponenten:

*Beschreibung (config.xml)*

Das config.xml beinhaltet eine deklarative Beschreibung des Widgets. Unter anderen beinhaltet dies Beschreibung folgende Punkte:

* Name und Version
* Details (z. B. Name des Autors, E-Mail-Adresse, Referenzen zu Read Me Files, etc.)
* Definition von Input sowie Output Endpunkten
* Optionen für unterschiedliche Einstellungen
* Referenz zum index.html
* Rendering Einstellungen

Das File wir mithilfe der sogenannten Mashable Application Component Description Language (MACDL) geschrieben. Beispiel für config.xml Dateien können im Github Repository gefunden werden.

*Code*

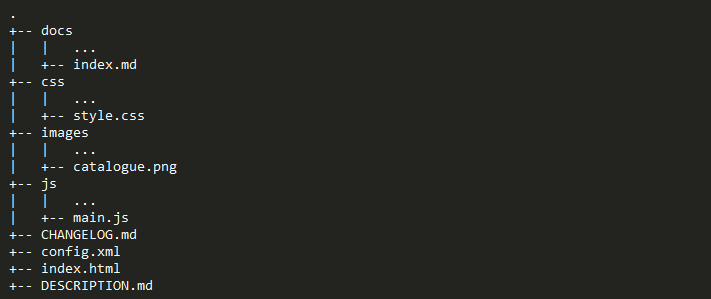
Die zweite Komponente eines Widgets ist der eigentliche Code, welcher zum Beispiel aus HTML, JavaScript und CSS Dateien besteht. Diese Dateien beinhalten die Implementierung eines Widgets und können auch die sogenannte Widget API [2] verwenden.

*Statische Ressourcen*

Zu den statischen Ressourcen zählen z. B. Bilder, Dokumentation oder ähnliches.

**Packaging**

Alle Komponenten von einem Widget werden in ein zip File verpackt. Die Dateiendung wird jedoch von zip auf wgt geändert. Folgende Abbildung zeigt eine mögliche Struktur eines Widgets.



Im Zuge des Projektes wurden außerdem npm und Bower als Package Manager sowie auch der Taskrunner Grunt verwendet um bestimmte Aufgaben wie das Laden von Komponenten, das Kompilieren und das Packaging zu vereinfachen. Beispiele für package.json, bower.json und gruntfile.js Dateien können im Github Repository gefunden werden.

* + 1. Operatoren

Operatoren verfügen über einen ähnlichen Aufbau wie Widgets. Der größte Unterschied zu Widgets liegt darin, dass Operatoren über keine visuellen Komponenten verfügen. Aus diesem Grund referenziert die config.xml Datei nicht auf ein index.html sondern auf die jeweiligen Scripts, welche den Code beinhalten.

Grundsätzlich werden Operatoren in drei Kategorien eingeteilt:

* **Data Source Operators** 🡪 stellen Informationen zur Verfügung, welche von anderen Widgets oder Operatoren verwendet werden.
* **Data Target Operators** 🡪 stellen Informationen zur Verfügung und verwenden diese um bestimmte Aufgaben durchzuführen.
* **Data Transformation Operators** 🡪 transformieren Daten, um sie für die Verwendung von Widgets und Operatoren aufzubereiten.
  + 1. Themes

**Allgemeines**

Wirecloud bietet auch die Möglichkeit neue Themes für das Dashboard zu erstellen oder Teile von bereits vorhanden Themes zu überschreiben. Dafür muss zuerst ein neuer Ordner mit folgenden Struktur erstellt werden:



Laut der offiziellen WireCloud Dokumentation wird die \_\_init\_\_.py Datei für die Plug-in Architektur, welche von WireCloud verwendet wird benötigt. Die Datei leer sein, muss jedoch vorhanden sein.

Grundsätzlich leiten alle Themes vom Default WireCloud Theme ab, wenn ein anderes Theme als Basis verwendet werden sollte muss die folgende Zeile in der \_\_init\_\_.py Datei hinzugefügt werden:



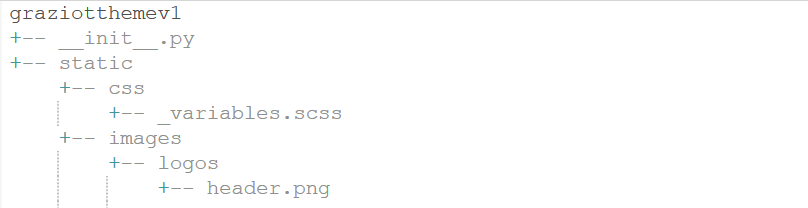
Weitere Informationen welche Templates zur Verfügung stellen und welche Variablen überschrieben werden können, können in der WireCloud Themes Dokumenation [3] entnommen werden. Im folgenden Abschnitt liegt der Fokus auf den Einstellungen, welche getroffen werden müssen um ein angepasstes Theme zu verwenden.

**Ändern eines WireCloud Themes**

Folgende Anleitung beschreibt die Schritte, welche durchgeführt werden müssen um ein WireCloud Theme von einer lokalen Docker Umgebung zu ändern. Mit geringen Anpassungen können dieselben Schritte für eine Docker Umgebung auf einer FIWARE Lab Instanz verwendet.

*Ordner Struktur*

Wie bereits erwähnt muss zu Beginn die passende Ordner Struktur erstellt werden. Danach kann im defaultheme von WireCloud nach dem Teil, welcher überschrieben werden sollte gesucht werden. Das WireCloud Projekt kann von Github [4] geklont werden. In diesem Beispiel wird das Header Logo und einige Variablen überschrieben, wofür folgende Ordner Struktur erstellt wurde:



Wichtig dabei ist, dass die Datei oder das Bild, welches überschrieben werden sollte, den gleichen Dateinamen und Pfad wie im Default Theme hat.

Im nächsten Schritt wird der gesamte Ordner (graziotthemev1) auf die WireCloud Instanz in den Ordner /opt/wirecloud\_instance kopiert. Dies kann mit folgenden Docker Befehl durchgeführt werden:

docker cp graziotthemev1 wirecloud:/opt/wirecloud\_instance/graziotthemev1

Nach jeder neuen Änderung im Theme Ordner, muss der Ordner umbenannt werden, da WireCloud sonst keine Änderungen erkennt.

*Settings.py*

Danach muss die settings.py Datei, welche sich unter /opt/wirecloud\_instance/wirecloud\_instance befindet bearbeitet werden, so dass das neue Theme referenziert wird. Dafür muss die folgende Zeile geändert werden:

THEME\_ACTIVE = "graziotthemev1"

*Laden des neues Themes*

Um das neue Thema zu laden, müssen alle Docker Container, welche für WireCloud relevant sind neugestartet werden. Am sichersten ist es jedoch gleich alle FIWARE Container zu stoppen und wieder zu starten. Dies kann entweder mit den folgenden Docker Befehlen durchgeführt werden:

docker stop @(docker ps -aq)

docker start @(docker ps -aq)

Alternativ können auch Docker Compose Befehle durchgeführt werden, dafür muss das Terminal im selben Pfad geöffnet werden, in welchen sich auch die docker-compose.yml Datei befindet:

docker-compose stop

docker-compose start

Nach dem Restart sollte überprüft werden ob alle Container erfolgreich gestartet werden konnten. Dies geschieht mit dem Befehl:

docker container ls

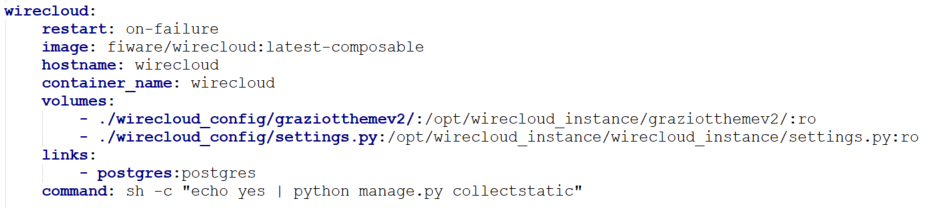
Wenn alle Container laufen, muss man sich auf die WireCloud Instanz verbinden und alle statischen Ressourcen neu laden. Dies geschieht mithilfe von folgenden Befehlen:

docker exec -it wirecloud bash

python manage.py collectstatic

*Ordner mounten*

Werden häufig Änderungen am Theme vorgenommen, ist es sinnvoll die betroffenen Ordner mithilfe der docker-compose.yml Datei zu mounten. Die entsprechende Konfiguration dafür in der docker-compose.yml Datei sieht folgendermaßen aus:



Es wurde nicht getestet ob für diese Einstellung nach einer Änderung des Themes es ausreichend ist, die Container neu zu starten oder ob bei jeder Änderung ein docker-compose up ausgeführt werden muss.

* 1. Vorhandene Widgets und Operatoren
  2. Custom Widgets und Operatoren
  3. Beispiele

1. Security
2. Fazit
3. Linksammlung

|  |  |
| --- | --- |
| [1] | „WireCloud Dokumentation,“ [Online]. Available: https://wirecloud.readthedocs.io/en/latest/. |
| [2] | „Widget API,“ [Online]. Available: https://wirecloud.readthedocs.io/en/latest/widgetapi/widgetapi/. |
| [3] | „WireCloud Themes,“ [Online]. Available: https://wirecloud.readthedocs.io/en/latest/development/platform/themes/. |
| [4] | „Github WireCloud,“ [Online]. Available: https://github.com/Wirecloud/wirecloud. |