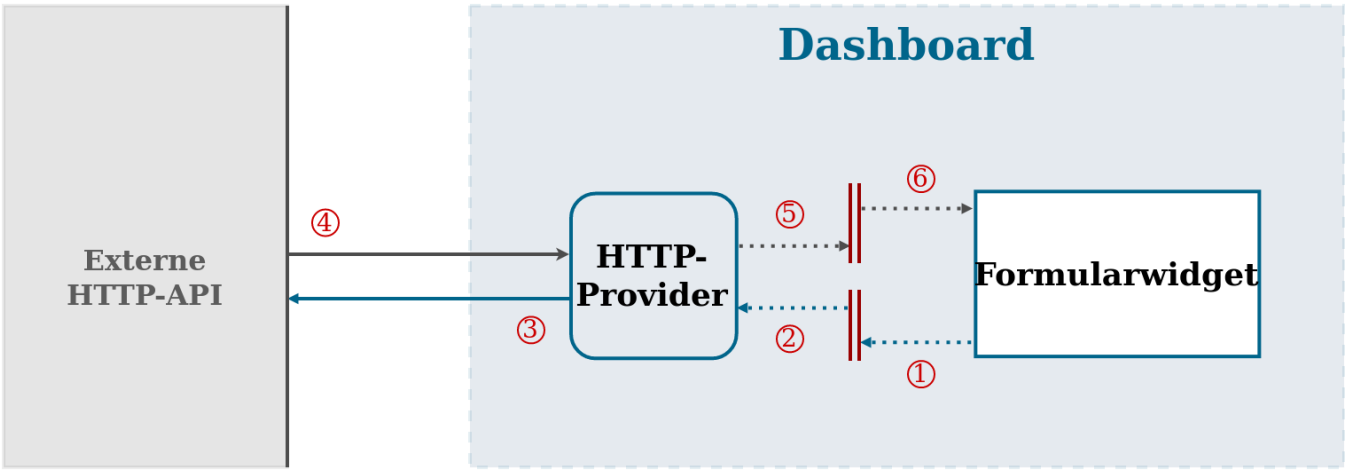
HTTP-Provider

Der HTTP-Provider ermöglicht es HTTP-Schnittstellen in YUNA-Dashboards zu integrieren und somit Daten von externen Services in Dashboards einzubinden oder umgekehrt die Daten aus einem Dashboard an diese zu senden.
Mit Hilfe von Handlebar-Templates können die HTTP-Anfragen dynamisch konfiguriert werden.

Beispielgrafik: Versenden von Formulardaten an eine HTTP-Schnittstelle mit Hilfe des HTTP-Provider:



1. Bei Betätigung des Absenden-Buttons im Formularwidget werden die eingegebenen Daten an den Output-Channel des Formularwidgets übergeben.
2. Die durch den IO-Channel bereitgestellten Daten werden durch den HTTP-Provider konsumiert, da dieser IO-Channel als Input konfiguriert ist.
3. Der HTTP-Provider führt ein mit den bereitgestellten Daten angereichertes Request durch.
4. Die angefragte HTTP-Schnittstelle liefert eine Antwort an den HTTP-Provider.
5. Die Antwort wird in den Response-Output-Channel des HTTP-Providers übergeben. Der Body der Antwort wird entsprechend der Konfiguration des HTTP-Providers ausgelesen.
6. Das Formularwidget erhält die neuen Daten aus dem IO-Channel und kann diese darstellen.

YUNAML Beispielkonfigurationen für HTTP-Provider:

Durch klick auf die verschiedenen YUNAML-Elemente gelangen sie zu den jeweiligen detaillierten Informationen in den folgenden Tabelle.

Einfaches GET-Request mit minimaler Konfiguration

```
<io-provider>
    <type>HTTP</type>
    <config>
        <outputs>
            <response>OutputChannelName2</response>
        </outputs>
        <inputs>
            <optional>OptionalChannel</optional>
        </inputs>
        <method>GET</method>
        <url>https://qa.yuna.dev/</url>
        <unsafeCredentialDelegation>true</unsafeCredentialDelegation>
    </config>
</io-provider>
```

Komplexes POST-Request mit allen verfügbaren Konfigurationen

```
<io-provider>
    <type>HTTP</type>
    <config>
        <outputs>
            <request>OutputChannelName1</request>
            <response>OutputChannelName2</response>
        </outputs>
        <inputs>
            <trigger>formWidgetDataWasSentChannel</trigger>
            <mandatory>
                <list>dataChannel</list>
            </mandatory>
```

```
            <optional>
                <list>OptionalChannel2</list>
                <list>InputChannel2</list>
            </optional>
        </inputs>
        <requestBodyAs>json</requestBodyAs>
        <responseBodyAs>json</responseBodyAs>
        <method>POST</method>
        <url>https://qa.yuna.dev/</url>
        <unsafeCredentialDelegation>true</unsafeCredentialDelegation>
        <urlParameters>
            <parameterName>parameterValue</parameterName>
        </urlParameters>
        <body>
            { "name": "{{dataChannel.name}}",
            "password": "{{dataChannel.password}}" }
        </body>
        <headers>
            <keep-alive>{{paramChannel.keepAlive}}</keep-alive>
        </headers>
    </config>
</io-provider>
```

Konfiguration des HTTP-Providers

YUNAML-Tag	Beschreibung	Konfiguration erforderlich?	Beispiel
type	Der Typ des IO-Providers. Für den HTTP-Provider muss dieser Parameter auf "HTTP" gesetzt werden. Weitere mögliche Werte sind "UrlParams" und "DataID".	✅	`<type>HTTP<http>`

config	Die verschiedenen Konfigurationsparameter des HTTP-Providers. Die Konfiguration lässt sich für das leichtere Verständnis in zwei Teile zerlegen: 1. Die Konfiguration der In- und Outputchannels, über die der Provider an die anderen Inhalte eines Dashboards angebunden wird. 2. Die Konfiguration der Anfrage, die gegen eine HTTP-Schnittstelle ausgeführt werden soll.	✓	

Konfiguration der In- und Output-Channels

YUNAML-Tag	Beschreibung	Konfiguration erforderlich?	Beispiel
config > **inputs**	Die Input-Channel des HTTP-Providers. Die konfigurierten Channel werden für zwei Zwecke verwendet: 1. Das Befülllen der Templates in der Konfiguration der HTTP-Anfrage. 2. Die Bestimmung, wann eine Anfrage durchgeführt werden soll. Um den Zeitpunkt für die Anfrage zu konfigurieren, werden die Input-Channel in drei YUNAML-Tags definiert: **<trigger>**, **<mandatory>** und **<optional>**. Bei jeder Änderung eines Channels, der in mindestens einem dieser Tags definiert ist, wird geprüft, ob eine Anfrage durchgeführt werden soll. Dabei werden folgende Bedingungen geprüft: 1. Nur wenn Channel in **<trigger>** definiert sind: Wurde ein **<trigger>**-Channel aktualisiert? 2. Nur wenn Channel in **<mandatory>** definiert sind: Wurden alle **<mandatory>**-Channel mit Daten befüllt? Sind weder **<trigger>**- noch **<mandatory>**-Channel definiert, wird bei jeder Änderung eines in **<optional>** angegebenen Kanals eine Anfrage ausgeführt. Ein Channel kann sowohl **<trigger>** als auch **<mandatory>** sein. Sollen mehrere Channel in einem der drei Tags definiert werden, müssen diese in <list>-Tags angegeben werden.	✗	**<inputs>** **<trigger>***someChannel*<**/trigger>** **<mandatory>** **<list>***dataChannel*<**/list>** <**/mandatory>** **<optional>** **<list>***OptionalChannel2*<**/list>** **<list>***InputChannel2*<**/list>** **</optional>** **</inputs>**
config > inputs > **trigger**		✗	
config > inputs > **mandatory**		✗	
config > inputs > **optional**		✗	
config > **outputs**	Unter **<outputs>** wird konfiguriert, in welche IO-Channel die gesendete Anfrage und die Antwort der abgefragten Schnittstelle veröffentlicht werden. Das Format, in dem die Daten veröffentlicht werden, kann über **<requestBodyAs>** und **<responseBodyAs>** konfiguriert werden.	✗	**<outputs>** **<request>***OutputChannelName1*<**/request>** **<response>***OutputChannelName2*<**/response>** **</outputs>**
config > outputs > **request**	Der **<request>**-Channel wird genutzt um den abgesendeten Request zu veröffentlichen. Dabei wird der Body der Anfrage entsprechend der Konfiguration in **<requestBodyAs>** konvertiert. Das in den angegebenen Channel veröffentlichte Objekt hat unter anderem folgende Eigenschaften: • `method`: Die verwendete HTTP-Methode • `url`: Die Ziel-URL der Anfrage • `headers`: Die Header der Anfrage • `body`: Der Body der Anfrage (Für Details zur Formatierung siehe **<requestBodyAs>**)	✗	

<table>
<tr><td>config > outputs > response</td><td>Der <response>-Channel wird mit der Antwort der angefragten HTTP-Schnittstelle befüllt. Dabei wird der Body der Anfrage entsprechend der Konfiguration in <responseBodyAs> konvertiert.

Das in den angegebenen Channel veröffentlichte Objekt hat unter anderem folgende Eigenschaften:
<code>ok</code>: Ob die Anfrage erfolgreich war.
<code>status</code>: Der HTTP-Status-Code der Anfrage. Einen allgemeinen Überblick über HTTP-Status-Codes bietet die MDN-Dokumentation. Die Implementierung in einzelnen HTTP-Services kann von dieser allgemeinen Definition abweichen.
<code>statusText</code>: Die zu dem Status-Code gehörende Status-Nachricht.
<code>url</code>: Die URL der Antwort.
<code>body</code>: Der Body der Antwort (Für Details zur Formatierung siehe <responseBodyAs>)
<code>headers</code>: Die Header der Antwort.</td><td>✗</td><td></td></tr>
<tr><td>config > requestBodyAs</td><td rowspan="2">Der Datentyp, in dem der Body des Requests bzw. der Response erwartet wird.

Der HTTP-Provider versucht den Body entsprechend der Konfiguration zu konvertieren.
Schlägt dies Fehl, werden keine Daten in den <request>-Channel veröffentlicht.

Unterstützte Datentypen:<table><tr><th>Datentyp</th><th>Erläuterung</th><th>mögl. Anwendung</th></tr><tr><td>text</td><td>Text als UTF-8 Zeichenkette</td><td>Anzeige in einem Widget</td></tr><tr><td>json</td><td>Objektstruktur im JSON Textformat</td><td>Weiterleitung an ein Widget /Channel

zur Weiterverarbeitung</td></tr><tr><td>formData</td><td>Schlüssel-/Wert-Paare als Representation von Formularfeldern und ihren Werten</td><td>Senden an/Empfangen von einem Formular-Endpunkt</td></tr><tr><td>blob</td><td>Datei ähnliche, rohe Daten</td><td>Down-/Upload zur Weiterverarbeitung</td></tr></table>Für weitere Details zu den verschiedenen Datentypen und ihren Verwendungszwecken, finden die in der MDN-Dokumentation des Body mixin.</td><td>✗</td><td><requestBodyAs>json</requestBodyAs></td></tr>
<tr><td>config > responseBodyAs</td><td>✗</td><td><responseBodyAs>json</responseBodyAs></td></tr>
</table>

Konfiguration der HTTP-Anfrage

Die über den HTTP-Provider zu sendende Anfrage kann über mehrere YUNAML-Tags konfiguriert werden. Diese sind in untenstehender Tabelle erläutert.

Die Inhalte aller YUNAML-Tags in diesem Block können mithilfe von Handlebar-Templates dynamisch gesetzt werden. Dabei werden die Daten aus den **<input>-Channels** verwendet, um die Platzhalter in den Handlebar-Templates zu befüllen.

Beispiel für die Ersetzung in Handlebar-Templates:

Der YUNAML-Tag **<url>** in der Konfiguration der HTTP-Anfrage enthält folgendes Template:

```
<url>https://jsonplaceholder.typicode.com/posts/{{formChannel.[0].number}}</url>
```

Als Beispiel werden hier Daten des Formularwidget Output-Channel verwendet, welche als Liste repräsentiert werden.
Mit dem Template wird auf das erste Element der Liste und dessen Feld "number" referneziert.

In der Konfiguration der Input-Channel des HTTP-Providers findet sich folgender Eintrag:

```
<inputs>

    <optional>formChannel</optional>

</inputs>
```

Wird durch ein Formularwidget mit dem Output-Channel "`formChannel`" nun ein Formular mit dem Feld "number" abgeschickt, welches den Wert 99 enthält, wird das Template aus dem **<url>**-Tag in die folgende URL umgewandelt:

```
<url>https://jsonplaceholder.typicode.com/posts/99</url>
```

Erläuterung zur Deserialisierung von Input-Daten mit mehreren Eigenschaften in Handlebar-Templates

Sollen in einem Handlebar-Template Objekte mit mehren Eigenschaften dargestellt werden, ohne die Eigenschaften einzeln anzugeben, muss ein 'json'-Helper verwendet werden.

Der YUNAML-Tag **<body>** in der Konfiguration der HTTP-Anfrage enthält folgendes Template:

```
<body>{{formChannel}}</body>
```

Wird durch ein Formularwidget mit dem Output-Channel "`formChannel`" nun ein Formular mit mehreren Eingabe-Feldern abgeschickt, enthält der Body des Requests folgende Daten:

```
Object [object]
```

Um dies zu vermeiden, muss der 'json'-Helper in dem Handlebar-Template verwendet werden:

```
<body>{{json formChannel}}</body>
```

Dadurch werden die durch das Formularwidget veröffentlichten Daten korrekt zu folgendem Ergebnis deserialisiert:

```
{
        "FormularFeld_1":"Wert von FormularFeld 1",
        "FormularFeld_2":"Wert von FormularFeld 2"
}
```

YUNAML-Tag	Beschreibung	Konfiguration erforderlich?	Beispiel
config > **method**	Die HTTP-Methode, die für die Anfrage verwendet werden soll. Häufig genutzte Methoden: GET, POST, PUT, DELETE	✅	**<method>***GET***</method>**
config > **url**	Die Adresse, an die die Anfrage gestellt werden soll. Dies kann entweder eine absolute URL sein, also zum Beispiel mit "https://" beginnen, oder relativ, wenn eine YUNA-REST-Schnittstelle abgefragt werden soll. URL-Parameter können entweder direkt in dem Template angegebem werden, oder über die Einträge im Tag <urlParameters> hinzugefügt werden.	✅	**<url>**`https://jsonplaceholder.typicode.com/posts/{{formChannel.[1].number}}`**</url>**
config > **urlParameters**	URL-Parameter, die an die URL angehängt werden sollen. Die Angabe erfolgt in XML-Notation. Der Tag definiert den Namen des Headers, der Inhalt den Wert.	❌	**<urlParameters>** ***<parameterName>****parameterValue****</parameterName>*** **</urlParameters>**

config > **body**	Der Body, der mit der Anfrage versendet werden soll.
	
	⚠️ Wird ein Body definiert, dürfen die HTTP-Methoden 'GET' und 'HEAD' nicht verwendet werden.
	
	Der Body kann mit Handlebar-Templates dynamisch konfiguriert werden.
	Es können dabei Objekte aus dem Input-Channel des HTTP-Providers verwendet werden.
	Beispiel: {{inputChannel.name}} wird durch das Objekt/den Wert aus "name" des Inputchannel ersetzt.
		❌	**<body>**
			
			{
			
			"staticTextField": "Hello ",
			
			"fieldFromTemplate": "{{inputChannel.name}}"
			
			}
			
			</body>

config > **headers**	Header, die für die Anfrage verwendet werden sollen. Im **<headers>**-Tag werden die Namen der Header in Tags angegeben und die jeweiligen Werte als Inhalt des Tags. Schematisch: `<header-key>header-value</header-key>`	❌	**<headers>** ***<keep-alive>****{{paramChannel.keepAlive}}****</keep-alive>*** **</headers>**

config > **unsafeCredentialDelegation**	Erlaubte Werte: **true** | **false** (Standard = Abgeschaltet) Aktiviert bei einem CORS-Request die automatische Anmeldung des Browsers bei dem Zielserver des HTTP-Request, falls möglich. Standardmäßig verbietet eine Browserrichtlinie die automatische Anmeldung bei einem CORS-Request. Weitere Browser-, Netzwerk-, Sicherheits- oder Systemrichtlinien können den CORS-Request sowie die automatische Anmeldung bei dem Zielserver verhindern. Der verwendete Browser muss möglicherweise für die automatische Anmeldung konfiguriert werden. Die Adresse des Zielservers muss möglicherweise in einer Liste (Whitelist) für die automatische Anmeldung hinterlegt sein. Ist diese Option aktiviert werden Informationen für eine Benutzeranmeldung (Cookies, Nutzername, Passwort, usw.) an den Zielserver gesendet. **Verwenden Sie diese Option nur wenn Sie dem Zielserver vertrauen (z.B. Server innerhalb des Firmennetzwerks).** Der Zielserver muss folgende Header setzen, damit diese Option genutzt werden kann: 1. **Access-Control-Allow-Origin: https://mein-yuna-portal.dev** Der Wert des Header muss der Quelle entsprechen (z.B. https://mein-yuna-portal.de/) und darf nicht auf "*" gesetzt sein. 2. **Access-Control-Allow-Credentials: true**	✗	**<unsafeCredentialDelegation>**false**</unsafeCredentialDelegation>**

Same-Origin-Policy (SOP) der Browser und *Cross-Origin Resource Sharing (CORS)* bei der Nutzung des HTTP-Provider

⚠ **Das Abrufen von Ressourcen von anderen Servern wird durch die Same-Origin-Policy der Browser standardmäßig unterbunden.**

Wenn Yuna z.B. unter der URL **https://yuna.demo.dev** verfügbar ist und über den HTTP-Provider ein Request an **https://weather.example.dev/api/weather?q=Kassel** gesendet werden soll kann dieser Request vom Browser unterbunden werden, da es sich nicht um den Selben Server handelt der Yuna bereitstellt. Dies ist ein Sicherheitskonzept (**Same-Origin-Policy**) bei der Verwendung von JavaScript im Browser. Externe APIs sind üblicherweise für den Zugriff von beliebigen anderen Servern über **Cross-Origin Resource Sharing** konfiguriert.

Durch die Konfiguration von **Cross-Origin Resource Sharing** auf dem externen Servers kann dieser den Request aus Yuna heraus erlauben. Um den Request an den Server (hier: **weather.example.dev**) zu erlauben muss auf dem Server (hier: **weather.example.dev**) der CORS Header konfiguriert werden.

Weitere Informationen finden Sie auf der Seite **enable cross-origin resource sharing**.
Wie der Zielserver des Request konfigriert wird finden Sie für die entsprechende Webserver-Software (Apache, nginX, usw.) unter enable CORS.

YUNA ML Beispiel

In diesem Beispiel wird über den IO-Provider ein http POST-Request mit Daten aus einem Tabellen-Widget an die Adresse https://postman-echo.com/post gesendet. Die Response (Server-Antwort) wird in einem Formular-Widget dargestellt.

```
<xml>
    <!-- Copyright (c) 2019 eoda GmbH -->
    <view name="dpe-973" roles="System_Admin, AdHoc_Full_Issue">
        <caption>DPE-973: SIS-Service-Call</caption>
        <description>Dashboard, das ein dem SIS-Service-Call ähnliches Beispiel abbildet</description>
        <userdocu>https://jira.eoda.de/browse/dpe-973</userdocu>
        <widget>
            <position>
                <x>0</x>
                <y>0</y>
            </position>
            <size>
                <x>12</x>
                <y>8</y>
            </size>
            <caption>
                <show>true</show>
                <label>form-widget</label>
```

```
            </caption>
            <widgettype>formwidget</widgettype>
            <inputs>
                <response>responseChannel</response>
            </inputs>
            <outputs>
                <submit>formWidgetDataWasSentChannel</submit>
            </outputs>
            <formTemplate>
                <![CDATA[
<div>
<h2>Response:</h2>
<p>selected: {{inputs.response}}</p>
</div>
<br>
            ]]>
            </formTemplate>
            <submitButton>
                <dataID>qy_dpe-1387</dataID>
                <label>GO!</label>
            </submitButton>
        </widget>

        <!-- Copyright (c) 2019 eoda GmbH -->
        <widget name="MultiChoiceSelection_Table_dpe_1387">
            <position>
                <x>0</x>
                <y>9</y>
            </position>
            <size>
                <x>16</x>
                <y>7</y>
            </size>
            <widgettype>tabledirective</widgettype>
            <dataID>qy_dpe-1387</dataID>
            <sorting>ProductGroup</sorting>
            <sortingorder>asc</sortingorder>
            <outputs>
                <selected>selectedData</selected>
            </outputs>
            <cols>
                <list>
                    <field>ProductGroup</field>
                    <title>ProductGroup</title>
                    <sortable>ProductGroup</sortable>
                    <filter>
                        <ProductGroup>text</ProductGroup>
                    </filter>
                    <show>true</show>
                    <width>50</width>
                    <style></style>
                </list>
                <list>
                    <field>EquipmentNo</field>
                    <title>EquipmentNo</title>
                    <sortable>EquipmentNo</sortable>
                    <filter>
                        <EquipmentNo>text</EquipmentNo>
                    </filter>
                    <show>true</show>
                    <width>50</width>
                    <style></style>
                </list>
                <list>
                    <field>ProductionCounter</field>
                    <title>ProductionCounter</title>
                    <type>number</type>
                    <width>50</width>
                    <filter>
                        <ProductionCounter>number-custom</ProductionCounter>
                    </filter>
```

```
                    <show>true</show>
                </list>
                <list>
                    <field>ParentEquipmentNo</field>
                    <title>ParentEquipmentNo</title>
                    <width>50</width>
                </list>
                <list>
                    <field>EndCustomer</field>
                    <title>EndCustomer</title>
                    <width>50</width>
                </list>
                <list>
                    <field>Location</field>
                    <title>Location</title>
                    <width>50</width>
                </list>
                <list>
                    <field>LocationCountry</field>
                    <title>LocationCountry</title>
                    <width>50</width>
                </list>
                <list>
                    <field>EquipmentFamily</field>
                    <title>EquipmentFamily</title>
                    <width>50</width>
                </list>
                <list>
                    <field>MachineType</field>
                    <title>MachineType</title>
                    <width>50</width>
                </list>
                <list>
                    <field>LastServiceMission</field>
                    <title>LastServiceMission</title>
                    <type>genDate</type>
                    <width>50</width>
                    <filter>
                        <LastServiceMission>date-custom</LastServiceMission>
                    </filter>
                    <show>true</show>
                </list>
            </cols>
            <generalOptions>
                <selectable>true</selectable>
            </generalOptions>
        </widget>

        <io-provider>
            <type>HTTP</type>
            <config>
                <inputs>
                    <mandatory>selectedData</mandatory>
                    <trigger>formWidgetDataWasSentChannel</trigger>
                </inputs>
                <outputs>
                    <response>responseChannel</response>
                </outputs>
                <method>POST</method>
                <url>https://cors-anywhere.herokuapp.com/https://postman-echo.com/post</url>
                                <unsafeCredentialDelegation>false</unsafeCredentialDelegation>
                <body>{{json selectedData}}</body>
                <requestBodyAs>json</requestBodyAs>
                <responseBodyAs>text</responseBodyAs>
                <headers>
                                        <Cache-Control>no-cache</Cache-Control>
                </headers>
            </config>
        </io-provider>
    </view>
</xml>
```