

# Entwurf MatFlow Workflowanwendung für Machine Learning Experimente

Florian Küfner, Soeren Raymond, Alessandro Santospirito,  
Lukas Wilhelm, Nils Wolters

Dezember 2021

# Inhaltsverzeichnis

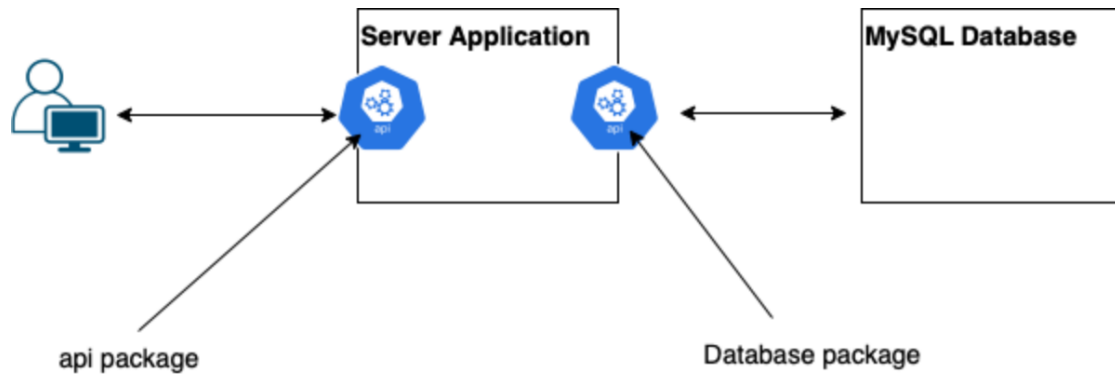
<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Systemüberblick</b>	<b>4</b>
<b>3</b>	<b>Kommunikation</b>	<b>5</b>
3.1	Client <-> Server . . . . .	5
3.2	Server <-> Airflow API . . . . .	5
<b>4</b>	<b>MatFlow</b>	<b>6</b>
<b>5</b>	<b>API Package</b>	<b>6</b>
<b>6</b>	<b>Komponentenbeschreibung</b>	<b>7</b>
6.1	Client . . . . .	7
6.2	Server . . . . .	8
<b>7</b>	<b>Paketbeschreibung</b>	<b>9</b>
7.1	Client Application . . . . .	9
7.2	Server Application . . . . .	9
7.3	Database . . . . .	21
<b>8</b>	<b>Abläufe</b>	<b>27</b>
<b>9</b>	<b>Datenbank</b>	<b>28</b>
9.1	Datenbankaufbau . . . . .	28

## 1 Einleitung

## 2 Systemüberblick

## 3 Kommunikation

### 3.1 Client <-> Server



Die Kommunikation zwischen Client und Server erfolgt über das API Package, bei der die Klasse FrontendAPI als RESTful API über JSON mit dem Client kommuniziert und das Database Package die Kommunikation zwischen der Datenbank und der Server Application regelt. Beide Schnittstellen sind zugleich Fassaden, die die Kommunikation zwischen den einzelnen Subsystemen regeln. Hier ist also eine horizontale Schichtenarchitektur vorhanden.

#### 3.1.1 Frontend API

Hinweis: Json Objekte werden in Python nativ als String interpretiert. Das ist dementsprechend im Entwurf aber durch die Benennung der Parameter klar, da json Objekte immer mit json\_details o.Ä. beschrieben werden. Mithilfe von Flask läuft auf dem Host-server eine RESTful API, die

#### 3.1.2 Database package

### 3.2 Server <-> Airflow API

## 4 MatFlow

MatFlow, wie schon im Kommunikationskapitel gesehen, ist eine Schichtenarchitektur, die über Fassaden miteinander verbunden ist (API Package und Database package)

## 5 API Package

Hier ist die Klasse FrontendAPI die eigentliche Fassade. Der Grund dafür ist womögliche Abhängigkeiten zwischen der Client Applikation und der Serverapplikation zu entkoppeln. Sie wurde als Singleton implementiert, da die API nur auf einem Port auf dem Server laufen soll und global ansprechbar sein soll. Mehrere essentiell gleiche APIs auf verschiedenen Ports wären uneindeutig.

## **6 Komponentenbeschreibung**

### **6.1 Client**

## 6.2 Server



## 7 Paketbeschreibung

### 7.1 Client Application

### 7.2 Server Application

#### 7.2.1 API Package

**Package: api** This package is the interface between the client's application and the server application. All requests are sent to this API(package) in JSON format.

**Class: FrontendAPI** This class is the main interface of the whole application. The client application can get the api status code to obtain information regarding the execution of api calls (e.g. status code changes when an exception has been thrown)

**static get\_FrontendAPI(): FrontendAPI** returns the FrontendAPI in singleton design fashion, meaning there is only one instance of FrontendAPI in circulation at all times.

**get\_status\_code(): int** return status code

**set\_status\_code(status\_code: int)** sets the status code, only performed by the **ExceptionHandler**

- **status\_code** the status code showing the state of the latest API call

**get\_server\_details(): String** gets all server details (container limit, cpu resources, gpu resources, servername, ip address, executing server (yes/no)) and returns them in a json format

**set\_server\_details(json\_details: String)** sets all server details (container limit, cpu resources, gpu resources, servername, ip address, executing server (yes/no)) in a json format. Only one bulk update as a whole due to long delay with server communication from client's perspective

- **server\_details** all server details in json format

**get\_all\_users\_and\_details(): String** gets all users and their details (user names, privileges, statuses) in a json format. Only one bulk update as a whole due to long delay with server communication from client's perspective

**set\_all\_users\_and\_details(json\_details: String)** sets all user details (user name, privilege, stature) for a specific user in a json format. Only one bulk update as a whole due to long delay with server communication from client's perspective. If user does not already exist, one will be created.

**delete\_user(json\_details: String)** deletes a user by username

- **json\_details** json object which only contains the username key and value

**get\_wf\_instance\_version\_number (json\_details: String): String** returns workflow instance's version number

- **json\_details** json object which only contains the workflow instance name and version number

**replace\_wf\_instance\_version\_number (json\_details: String)** replaces the specified workflow instance's version number

- **json\_details** json object which only contains the workflow instance name and version number

**set\_config\_file(json\_details)** changes multiple config files based on input/ key value pair changes in client's applications during workflow instance configuration

- **json\_details** json object which contains the changed config files name, all key value pairs per config file, contains workflow instance name

**get\_config\_from\_wf\_instance(json\_details): String** gets all config file names related to specific workflow instance (version)

- **json\_details** json object which contains the wanted workflow instance name and its respective version and config file name

**get\_all\_wf\_instances\_names\_and\_config\_files\_names(json\_details: String)** gets all config file and all workflow instances **names**

**verify\_login(json\_details:String)** verifies username with associated password

- **json\_details** contains username and password

**register(json\_details:String)** registers a new user

- **json\_details** contains username, password and repeated password

**Class : JSONToPython** This class converts all json data into the wanted object by extracting certain keys and values and instantiating a new (temporary) object which executes the wanted methods (e.g. extract server and then execute set new container limit). Instantiated objects will be deleted by the garbage collection after they are finished writing back / getting data from the database.

**static extract\_user(json\_details:String): User** extracts json details and builds a new User based off of these json details

- **json\_details** contains username, privilege and status

**static extract\_server(json\_details:String): Server** extracts json details and builds a new Server based off of these json details

- **json\_details** contains servername, container limit, cpu and gpu resources, executing server (yes/no) and ip address

**static extract\_template(json\_details:String): Template** extracts json details and builds a new Template based off of these json details

- **json\_details** contains dag definition file name and template name

**static extract\_configs(json\_details:String): ReducedConfigFile[]** extracts json details and builds a new ReducedConfigFile array based off of these json details

- **json\_details** contains config files names which in themselves contain config file name, workflow instance name, and key value pairs

**Class : PythonToJSON** This class converts all python objects into json data by extracting certain keys and values and dumping them into a json object.

**static encode\_user(user: User): String** extracts all user attributes and dumps them into json object

- **user** user whose attributes are to be encoded

**static encode\_server(server: Server): String** extracts all server attributes and dumps them into json object

- **user** server whose attributes are to be encoded

**static encode\_template(template: Template): String** extracts template attributes and dumps them into json object

- **user** template whose attributes are to be encoded

**static encode\_wf\_instance(wf\_instance: WorkflowInstance): String** extracts workflow instance attributes and dumps them into json object

- **user** workflow instance whose attributes are to be encoded

**static encode\_config(config: ReducedConfigFile): String** turns key value pairs into json object

- **user** config file whose attributes are to be encoded

**Class: Exception Handler** This class handles all MatFlowExceptions and those who inherit from MatFlowException. It is responsible for changing the API's status code.

**handle\_exception(exception: MatFlowException)** sets the API's status code to the specific exception's status code by calling the private method `set_status_code`.

- **exception** specific MatFlowException that was thrown during request

**success()** sets status code to success code

### 7.2.2 Workflow Package

This package defines the logic behind the creation and the management of workflow templates, workflow instances and versions of workflow instances. It receives requests in the singleton object of WorkflowManager from the client application through the api package of the server application. The package also communicates with the database interface to save and recall data as well as with the API of Apache Airflow.

## Class: WorkflowManager

This class provides a singleton object that receives all requests addressed to this package. This class is also communicating with the API of Airflow as well as with the Database package.

### Methods

- **getInstance():WorkflowManager** Returns the singleton object if already existing otherwise it calls the private constructor
- **createTemplate(template:Template):void** Causes the creation of a new template entry in the database. The new template can be used for creating workflow instances afterwards.

### Parameters

- **template:** A Template object that provides all the necessary components for creating a new workflow template

### Exceptions

- `DoubleTemplateNameException` Gets thrown if the user tries to use an already existing template name
- `InvalidDagFileException` Gets thrown if the user hands over an invalid dag-definition-file
- **`createWorkflowInstanceFromTemplate(templateName:String, workflowInstanceName:String, configFiles:File[]):void`** Causes the instantiation of a workflow instance under the use of a workflow template.

### Parameters

- `templateName`: The identifier of the template that is used for instantiation
- `workflowInstanceName`: The name of the new workflow instance
- `configFiles`: Contains all the files needed for the execution of the workflow

### Exceptions

- `DoubleWorkflowInstanceNameException` Gets thrown if the user tries to use an already existing workflow instance name
- `EmptyConfigFolderException` Gets thrown if the user hands over an empty config-folder
- **`getTemplateNames():String[]`** Makes database request and returns the names of all templates in the system
- **`getTemplateFromName(templateName:String):Template`** Forwards the request for the named template to the database and returns result

### Parameters

- `templateName`: The identifier of the desired template
- **`getNamesOfWorkflowsAndConfigFiles():String[][]`** Forwards the request to the database and returns the names of all workflow instances in combination with the associated config-files
- **`getKeyValuePairsFromConfigFile(workflowInstanceName:String, configFileName:String):String[][]`** Requests the desired config-file from the current version of the named workflow instance from the database. Afterwards changes the format from a file into an array of key-value-pairs and returns the result

#### Parameters

- workflowInstanceName: The identifier of the workflow instance
- configFileName: The name of the desired config file
- **createNewVersionOfWorkflowInstance(workflowInstanceName:String, changedFiles:ReducedConfigFile[], versionNote:String):void** Causes the creation of a new version of the workflow instance in the database. The predecessor of the new version is the active version of the instance.

#### Parameters

- workflowInstanceName: The identifier of the workflow instance of which a new version is created
- changedFiles: The config-files that were changed in comparison to the predecessor version in key-value-pair representation
- versionNote Note about the new version given by the user
- **getVersionsFromWorkflowInstance(workflowInstanceName:string):FrontendVersion[]** Requests information about all the versions of the given workflow instance from the database. Afterwards calculates the difference to the predecessor for every version and returns that information in combination with the version numbers and version notes

#### Parameters

- workflowInstanceName: The identifier of the given workflow instance
- **setActiveVersionThroughNumber(workflowInstanceName:String, versionNumber:String):void** Changes the active version of a workflow instance in the database

#### Parameters

- workflowInstanceName: The name of the workflow instance of which the active version shall be changed
- versionNumber: The number of the new active version

#### Exceptions

- WorkflowInstanceRunningException Gets thrown if the workflow instance is running. In that case the active version can not be changed

## Class: Template

This class represents a workflow template. It contains the identifying name of the template as well as a dag-definition-file.

## Constructor

- **Template(name:String, dagDefinitionFile:File):Template**

### Parameters

- name: The name of the new template
- dagDefinitionFile: The file which defines the behavior of workflows instantiated of this template

### Exceptions

- DoubleTemplateNameException Gets thrown if the database already contains a template with the given name
- InvalidDagFileException Gets thrown if the handed dag-definition-file is invalid

## Class: WorkflowInstance

This class represents a workflow template. It contains the identifying name of the template as well as a dag-definition-file.

## Constructor

- **WorkflowInstance(name:String, dagDefinitionFile:File, configFolder:File[]):WorkflowInstance**

### Parameters

- name: The identifying name of the workflow instance
- dagDefinitionFile: The file which defines the behavior of the workflow instance
- configFolder: An array of files that is needed for executing the workflow

### Exceptions

- DoubleWorkflowInstanceNameException Gets thrown if the workflow instance name is already given away to another workflow instance
- EmptyConfigFolderException Gets thrown if the handed configFolder-array is empty

## Class: ParameterChange

This class represents the change of a key-value pair. It contains the old aswell as the new version of both the key aswell as the value. Furthermore it holds the name of the file the change was made in.

### Constructor

- **ParameterChange(oldKey:String, newKey:String, oldValue:String, newValue:String, configFileName:String):ParameterChange**

## Class: ReducedConfigFile

This class holds all information to represent a config-file in the frontend.

### Constructor

- **ReducedConfigFile(fileName:String, keyValuePairs:(String,String][]):ReducedConfigFile**

#### Parameters

- **fileName:** The name of the config-file through which the config-file can be identified among other config files of the workflow instance
- **keyValuePairs:** The contents of the file in key-value-pair representation

## Class: ConfigFile

This is a subclass of ReducedConfigFile and additionally holds the config-file itself as well as the name of the associated workflow instance.

### Constructor

- **ConfigFile(workflowInstanceName:String, configFileName:String, file:File):ConfigFile**  
It isn't necessary to hand the key-value-pairs because they can be derived from the file itself.

#### Parameters

- **workflowInstanceName:** The name of the associated workflow instance
- **configFileName:** The name of the config-file through which the config-file can be identified among other config files of the workflow instance
- **file:** The actual config-file

### Methods

- **applyChanges(changes:ReducedConfigFile):void** The method compares the key-value-pairs given through the ReducedConfigFile to the pairs of the object itself. Changes detected are applied to the actual config file that is located in the database.



#### **Parameters**

- changes: Contains the key-value-pairs of the file after the changes made by the user in the frontend

### **Class: Version**

This class represents a version of a workflow instance. It's main task is to switch between the two different representations of versions one in the frontend, one in the database.

#### **Constructor**

- **Version(versionNumber:VersionNumber, note:String):Version**

#### **Parameters**

- versionNumber: Number that identifies the new Version
- note: Note from user that can be used for documenting the version

### **Class: FrontendVersion**

This class inherits from Version and is specialized to satisfy the need for information of the client application

#### **Constructor**

- **FrontendVersion(versionNumber:VersionNumber, note:String, changes:ParameterChange)**

#### **Parameters**

- versionNumber: Number that identifies the new Version
- note: Note from user that can be used for documenting the version
- changes: Contains the difference to the predecessor version in form of the changed key-value-pairs

### **Class: DatabaseVersion**

This class inherits from Version and is specialized to fit the way versions are managed in the database

#### **Constructor**

- **DatabaseVersion(versionNumber:VersionNumber, note:String, changed-ConfigFiles:File[]):DatabaseVersion**

#### Parameters

- versionNumber: Number that identifies the new Version
- note: Note from user that can be used for documenting the version
- changedConfigFiles: Contains all the config-files that were changed in this version

## Class: VersionNumber

This class represents string expressions that are valid version numbers

#### Methods

- **static fromString(number:String):VersionNumber** This static method converts a string of correct syntax into a VersionNumber by calling the private constructor.

#### Parameters

- number: String that has to fit the regular expression of a version number
- **getSuccessor(workflowInstance:String):VersionNumber** This method calculates the smallest free successor of the VersionNumber for the given workflow instance

#### Parameters

- workflowInstance: Identifier of the workflow instance to which the new number corresponds to

### 7.2.3 User Administration

**Class: Registration** This class is used to sign up new Users for our application and also for airflow.

#### Methods

**signUser(username: String, password: password): boolean** Method uses username and password to sign up new user and returns a boolean to confirm the registration

#### Parameters

- username: The preferred username
- password: The preferred password

## Exceptions

- `UserExistsException` Gets thrown if the user tries to register with an already used username

**Class: User** This class is used to construct new Users.

## Constructor

**`user(username: string, status: string, privilege: string)`**

## Parameters

- `username`: The users username
- `status`: The users status. Can be active, inactive or banned
- `privilege`: The users privilege. Can be Reviewer, Developer or Admin

**Class: Login** This class is used to log in existing Users.

## Methods

**`loginUser(username: String, password: password): boolean`** Method uses username and password to log in user and returns a boolean to confirm the login.

## Parameters

- `username`: The username to log in
- `password`: The password to log in

**Class: UserController** This class is used to change the parameters of existing users.

## Constructor

**`UserController(user: user)`**

## Parameters

- `user`: The user that needs to be edited

## Methods

**deleteUser(user: user): void** Method gets the wanted User and deletes him. Afterwards the user is not able to log in anymore.

#### Parameters

- user: The user that is supposed to get deleted

**resetPassword(oldPassword: password, newPassword: password): void** Method resets the user's password and sets the new password.

#### Parameters

- oldPassword: The user's old password that gets deleted
- newPassword: The user's new password that is getting set

**setPrivilege(privilege: string): void** Method resets the user's old privilege and updates to the new privilege.

#### Parameters

- privilege: The privilege the user gets updated to

**activateUser(): void** Method activates the User. The User can now log in to the application.

### 7.2.4 Hardware Administration

**Class: HardwareController** This class is used to surveillance and edit the Hardware of the Server

#### Methods

**addServer(server: server): void** Method adds a Server to the HardwareAdministration

#### Parameters

- server: The server that is getting added

**changeContainerLimit(server: server, containerLimit: number): void** Method changes the container limit of the chosen server to the preferred number.

### Parameters

- server: The server that is getting edited
- containerLimit: the number to which the containerlimit is getting updated to

**selectServer(server: server): void** Method selects the server that is getting edited

### Parameters

- server: The server that is getting selected

**Class: Server** This class is used to create a Server object

### Constructor

**server(name: string, address: string, status: string, containerLimit: number, selectedForExecution: boolean, resources: [(string, string)])**

### Parameters

- name: The name of the server
- address: The serveraddress
- status: The server status
- containerLimit: The maximum of containers on the server
- selectedForExecution: Defines if the server is getting used or not
- resources: The servers resources

## 7.3 Database

### 7.3.1 WorkflowData

Diese Klasse ist die Schnittstelle für alle Klassen die auf den Datenbankeinträgen für Workflows arbeiten wollen. Sie erstellt für jeden Zugriff den passenden Datenbankbefehl und konvertiert die Ergebnisse in das gewünschte Format des Aufrufers. Sie wirft bei fehlerhaften Werten eine **MatFlowException**.

**createWorkflowInstanceFromTemplate(tName: String, wfName: String, confFold: Folder):void** Create a new instance of a workflow by using the dag-file of a Template, with the Version set to 1. Search confDirect for every .conf-file and add them into the Database. Set this Version as active Version.

- **tName** name of Template of which the dag should be taken from
- **wfName** name of the new Workflow
- **confFold** the folder where all data of the Workflow is saved into

**getNamesOfWorkflowsAndConfigFiles():String[][]** Return all Workflow names and the names of their corresponding config files. The returning value is a list of lists of Strings where an inner list has the form [<Workflow Name>, <config File1 Name>, <config File2 Name>, ...]

**createNewVersionOfWorkflowInstance(wfName:String, newVersion:DatabaseVersion, oldVersionNr:String):void** Create a new Version of an existing Workflow with changed config Files. Search which config Files changed and set those in the new Version.

- **wfName** name of the Workflow
- **newVersion** new Version identifier
- **oldVersionNr** identifier of Version the new one is based on

**getDatabaseVersionsOfWorkflowInstance(wfName:String):DatabaseVersion[]** Return all Versions of a Workflow Instance as DatabaseVersion Objects in a list. Asks the Database for the data and constructs every Object.

- **wfName** name of the Workflow

**setActiveVersionThroughNumber(wfName:String, version:String):void** Set the previous Version as inactive and a new Version as active

- **wfName** name of the Workflow
- **version** version to be set active

**getVersionNumbersOfWorkflowInstance(wfName:String):String[]** Return a sorted list of Strings of all Versions of a Workflow.

- **wfName** name of Workflow which versions should be listed

### 7.3.2 TemplateData

Diese Klasse ist die Schnittstelle für alle Klassen die auf den Datenbankeinträgen für Workflow\_Template arbeiten wollen. Sie erstellt für jeden Zugriff den passenden Datenbankbefehl und konvertiert die Ergebnisse in das gewünschte Format des Aufrufers. Sie wirft bei fehlerhaften Werten eine **MatFlowException**.

**createTemplate(template:Template):void** Read Values from template and convert them into a sql query. Throw error if name already exists.

- **template** Template object to convert

**getTemplateNames():String[]** Return a list of all existing Template names. List is empty when no Template exists.

**getTemplateByNames(name:String):Template** Return Template that is asked for. Throw error if Template doesn't exist.

- **name** name/identifier of the wanted Template

### 7.3.3 DatabaseTable

DatabaseTable ist die einzige Klasse die direkt mit der Datenbank kommuniziert. Ihre Funktion ist hauptsächlich MySQL Befehle entgegennimmt, diese an die Datenbank weiterzugeben und die dadurch erhaltenen Antworten zurückgibt. Im Fall eines Fehlers, wie einem leeren Ergebnis oder eines ungültigen Befehls wird eine **MatFlowException** geworfen.

**set(create: String):String** Create new entry on Database. Try to execute create on Database and return value.

- **create** a sql query to create an entry

**delete(del: String):String** Delete an existing entry. Try to execute del on Database and return value.

- **del** a sql query to delete an entry

**modify(change: String):String** Modify an existing entry. Try to execute change on Database and return value.

- **change** a sql query to change values

### 7.3.4 Exception package

**Class: Exception** This is the native Python exception package. See <https://docs.python.org/3/tutorial/errors.html> for more information on usage and best practices.

**Class: MatFlowException** This is the MatFlow parent exception from which all the specified exceptions inherit. It adds more functionality to a standard exception through the exception's status code and function which gets the status code. This is important for our API Package.

#### Constructor

**MatFlowException(message: String, status\_code: int): MatFlowException**

#### Parameters

- message: The message that is displayed when the exception is thrown. Native to the parent class but not relevant to application, nevertheless intriguing when extending this application
- status\_code: The status code representing the thrown exception

#### Methods

**get\_status\_code(): int** gets the status code

**Class: UserExistsException** This is an exception that is thrown when a user does not exist.

#### Constructor

**UserExistsException(message : String)**

#### Parameters

- message: The message that is displayed when this exception is thrown.

**Class: DoubleTemplateNameException** This is an exception that is thrown when the desired template name already exists.

#### Constructor

**DoubleTemplateNameException(message : String)**



### Parameters

- message: The message that is displayed when this exception is thrown.

**Class: InvalidDagFileException** This is an exception that is thrown when the dag definition file is not correct.

### Constructor

**InvalidDagFileException(message : String)**

### Parameters

- message: The message that is displayed when this exception is thrown.

**Class: DoubleWorkflowInstanceNameException** This is an exception that is thrown when the desired workflow instance name already exists.

### Constructor

**DoubleWorkflowInstanceNameException(message : String)**

### Parameters

- message: The message that is displayed when this exception is thrown.

**Class: EmptyConfigFolderException** This is an exception that is thrown when the config folder is empty, meaning that no config can be selected.

### Constructor

**EmptyConfigFolderException(message : String)**

### Parameters

- message: The message that is displayed when this exception is thrown.

**Class: WorkflowInstanceRunningException** This is an exception that is thrown when the workflow instance is already running.

### Constructor

**WorkflowInstanceRunningException(message : String)**

### Parameters

- message: The message that is displayed when this exception is thrown.

### 7.3.5 TGDS-Operator

**Class: AlternatingOperator** This class is a subclass of `airflow.models.baseoperator.BaseOperator` and is inserted in the plugins folder of the Airflow installation. Then it can be used in the creation of dag-definition-files. The operator takes two tasks and alternates between them. The tasks shall be able to give each other parameters dynamically.

### Methods

**execute(self, context: Any): void** Overwrites the method `execute` from it's parent class. This method is called when the operator is used in a workflow.

### Parameters

- self: The object itself
- context: The airflow context that can be used to read config values

## 8 Abläufe

## 9 Datenbank

### 9.1 Datenbankaufbau

Die Datenbank beinhaltet folgende Tabellen:

<b>Workflow_Template</b>		
name	String	<i>key; notNull</i>
dag	.py -File	<i>notNull</i>

<b>Workflow</b>		
name	String	<i>key; notNull</i>
files	String	<i>notNull</i>
dag	.py -File	<i>notNull</i>

<b>Version</b>		
wfName	String	<i>key; notNull; name from Workflow</i>
version	String	<i>key; notNull</i>
note	String	

<b>ActiveVersion</b>		
wfName	String	<i>key; notNull; name from Workflow</i>
version	String	<i>notNull; from Version</i>

<b>VersionFile</b>		
wfName	String	<i>key; notNull; name from Workflow</i>
version	String	<i>key; notNull; from Version</i>
confKey	String	<i>key; notNull; from ConfFiles</i>

<b>ConfFile</b>		
confKey	String	<i>key; notNull</i>
file	.conf -File	<i>notNull</i>

**Notiz** Die dag Datei hätte als Schlüssel zwischen Workflow\_Template und Workflow fungieren können, ist aber redundant für alle Operationen auf einem Workflow. Deshalb wurde entschieden die Datei bei einer Workflowerstellung direkt zu kopieren.