The following paragraphs provide a brief introduction to the primary files and folders in this repository.

1. Project.zip: A SWAT Model Calibration Example

The project.zip file serves as an example of how to calibrate a SWAT model using PASS4SWAT. Users can unzip this file and run the following command:

python pass4swat.py number\_parallel\_simulation

Replace number\_parallel\_simulation with the desired number of parallel worker pods to use during calibration and uncertainty analysis.

Once the process finishes, you can inspect the calibration results by examining the generated files. Some of these files include:

par\_val.txt: Located in the SUFI2.IN folder.

95ppu.txt, goal.txt, and best\_par.txt: Located in the SUFI2.OUT folder.

Alternatively, you can open the project with SWAT-CUP to view the calibration results.

1. The template folder contains all the necessary files to use PASS4SWAT. Here's how to get started:

Duplicate the Template Folder: Begin by making a copy of the "template" folder.

Copy Your Model Input Files: Place your model input files into the root directory and a subfolder named "Backup" within the copied folder.

Configure Calibration:

The configuration files SUFI2\_swEdit.def, SUFI2\_extract\_rch.def (in the root folder), and par\_inf.txt (in the SUFI2.IN folder) can be edited to customize your calibration process (refer to the documentation for details).

Run Calibration: Once you've prepared the files, initiate the calibration process by running the following command in your terminal:

python pass4swat.py number\_parallel\_simulation

Replace number\_parallel\_simulation with the desired number of parallel worker pods to use.

Note that you can also open the project with SWAT-CUP for configuration and to view the calibration results.

1. The declare\_queue.sh file is used to initialize the task queue. It's mounted into a Docker container and then invoked to start the initialization process.
2. The model\_simulation.sh file is responsible for running model simulations. It's mounted into a Docker container and then invoked to perform the following actions:

Retrieve a task: The script retrieves a task from the task queue.

Perform simulation: Based on the retrieved task, the script executes the model simulation.

1. The pass4swat.py is a Python script that automates the entire calibration process, including preprocessing, model simulation, and post-processing tasks.
2. YAML and TPL files

rabbitmq-controller.yaml defines a ReplicationController in YAML format, which is used to manage a set of pods in Kubernetes. Specifically, this configuration creates a ReplicationController named "rabbitmq-controller" that ensures at least one pod is always running the "rabbitmq:3.12" image. This pod runs a RabbitMQ server used as a task queue.

rabbitmq-service.yaml creates a Service named "rabbitmq-service". This service acts like a virtual endpoint for clients looking for the RabbitMQ server. Any request sent to the port defined in the service (5672) will be routed by Kubernetes to one of the available pods with matching labels (app: taskQueue, component: rabbitmq). This allows clients to connect to the RabbitMQ server without needing to know the specific pod names or IP addresses.

rabbitmq-initializer.yaml create a pod that executes a script to initialize a task queue. The script is mounted from your local machine.

model-paralllel-simulation-job.yaml defines a Job that will run multiple pods concurrently, each executing a script named /model/model\_simulation.sh for model simulation. The job will end when the task queue is empty.

test.yaml creates a long-running container that you can use to interact with the project environment. By connecting to this container's shell, you can execute commands, inspect files, or perform other actions within the container's file system that has access to your project files. This can be useful for debugging or troubleshooting.

model-parallel-simulation-job.yaml.tpl and rabbitmq-initializer.yaml.tpl are template files used by pass4swat.py to generate the corresponding YAML files, model-parallel-simulation-job.yaml and rabbitmq-initializer.yaml. During this process, information such as the desired number of simulations, number of concurrent pods, and the path to be mounted into the containers is automatically filled in the generated YAML files.

1. Docker file

Dockerfile defines a Docker image based on the official ".NET" image, "mono:latest". It uses the "mono:latest" image as the starting point for this new image. Mono is an open-source implementation of Microsoft's .NET Framework, allowing you to run .NET applications within the container. It updates the list of available packages within the base image and installs several tools using the apt-get package manager, including url, ca-certificates, amqp-tools (Tools for interacting with AMQP messaging protocols), python and dnsutils. And finally it create workspace within the container's file system to store project files.