Type of the Paper: Article

Environmental Decision Support Systems as a Service: Demonstration on CE-QUAL-W2 model

Yoav Bornstein[1], Ben Dayan[2], Scott Wells[3], and Mashor Housh[1]

[1] Department of Natural Resources and Environmental Management, University of Haifa, Israel; yoavborenst@gmail.com, mhoush@univ.haifa.ac.il
[2] ben.dayan@gmail.com
[3] Department of Civil and Environmental Engineering, Portland State University, Oregon, United States; wellss@pdx.edu

Abstract: An Environmental Decision Support System (EDSS) can be used as an important tool for rehabilitation and preservation of ecosystems. Nonetheless, high assimilation costs (both money and time) are one of the main reasons that these tools are not widely adapted in practice. This work presents a low-cost paradigm of "*EDSS as a Service*", this paradigm is demonstrated for developing Water Quality EDSS as a Service that utilizes the well-known CE-QUAL-W2 model as a kernel for deriving optimized decisions. The paradigm is leveraging new open-source technologies in software development (e.g. Docker, Kubernetes, and Helm) with cloud computing in order to significantly reduce assimilation costs of the EDSS for organizations and researchers working on rehabilitation and preservation of water bodies.

Keywords: CE-QUAL-W2, EDSS, SaaS

1. Introduction

Rehabilitation and preservation of ecosystems is an important goal to achieve globally. There are multiple incentives for meeting this goal, such as preserving biodiversity, mitigating climate change, and assuring future generations can enjoy clean air, land and water. Among the ecosystems that need preservation or recovery efforts are wetlands. Wetlands include a wide variety of habitats such as marshes, peatlands, floodplains, rivers, and lakes. Human-made wetlands can also be found in the form of reservoirs or wastewater treatment ponds [1]. Management of water quantity and quality play an important part in these conservation efforts. Environmental agencies usually include in their guidelines the need for stakeholder involvement in the decision-making process [1,2]. A common tool that can be used to engage stakeholders in the decision process is a Decision Support System (DSS). Specifically, in the case of environmental usage, the tool is often called an Environmental Decision Support System (EDSS). In water quality related issues, a water quality model can be used as a kernel in the EDSS that can guide recommendations to stakeholders. The recommendations may include impacts of changes in flow quantity and water quality, managing day-to-day operations of hydropower dams under environmental restrictions, or responding to an unexpected pollution event.

To meet conservation efforts in wetlands, environmental agencies often recommend developing EDSSs to guide the decision making process and facilitate the stakeholders' engagement [1]. These EDSSs are part of a holistic system for the allocation and management of water that aims at maintaining wetland ecosystem functions. Nevertheless, such tools are still not widely adopted in practice. The following factors make an EDSS expensive (in resources and time) to implement: (1) Model assimilation and calibration; (2) Software development for the implementation and maintenance of the EDSS; (3) Computer resources needed for the model computation and hosting the EDSS application (the installation of the software on a computer or a server).

 doi:10.20944/preprints202108.0274.v1

This study shows how the last two challenges could be addressed using a low-cost implementation that leverages new open-source technologies in software development (e.g. Docker, Kubernetes, and Helm) and cloud computing. We demonstrate a Water Quality EDSS that uses the CE-QUAL-W2 model [3] as a kernel to explore water quality changes as a result of different management decisions.

The choice of using the CE-QUAL-W2 model was based on a simple approach for selecting a water quality model from Mateus [4]. This approach was based on a systematic review of the main available models. The review consisted of the model abilities, dissemination and publications, and the usage experience. In all categories, CE-QUAL-W2 model was ranked first. CE-QUAL-W2 can simulate rivers, lakes, reservoirs, and estuaries [3], thus when used as a kernel in an EDSS it allows decision support for multiple types of wetlands. This model is open source; this means that there is no need to invest money in buying licenses to use the model, and users that are familiar with software programming can add features. Mateus [4] also notes that CE-QUAL-W2 simulations are relatively fast and require low computational power compared to other models. The model input and output are all text file based. The model itself is an executable that can be run without a need to interact with a Graphical User Interface (GUI). This design of the model acts as a simple external Application Programing Interface (API) that allows another software program (e.g. EDSS) to change the inputs, execute the model, and analyze the results. CE-QUAL-W2 was implemented in over 2,000 sites in 116 countries [5]. The source code is actively maintained with bug fixes and new features by Portland State University.

As a result, we choose the CE-QUAL-W2 model as the kernel of our EDSS. The main objective of a water quality EDSS is to provide a simple interface for stakeholders to better plan future projects in the wetland or handle the day-to-day operations of the wetland. Using CE-QUAL-W2 for decision making is not new, attempts have been made over the years to use the model in decision making contexts.

For example, Eturak [6] used the model as an EDSS to understand the impact of the planned "Buyuk Melen" reservoir and its watershed in Turkey. As the reservoir was still in the planning stages, there was no option to calibrate the model. Thus, the setup of the model was done according to the best knowledge that was available at the time. Next, a few scenarios with different flow volumes to the reservoir from domestic and industrial wastewater were chosen and their simulations executed using the model. Later, the results were compared and graphed for the stakeholders to engage a discussion around the implications of different scenarios.

The manual approach of Eturak [6], where a modeler familiar with CE-QUAL-W2 can provide the needed analysis, highlights several drawbacks: (1) Execution of the different scenarios needs continuous involvement of the modeler in the process: setting up the different inputs, executing model runs, and then comparing the results. Thus, the modeler must be involved in each scenario proposed by the stakeholders. This increases the cost of using the EDSS and makes the discussion/involvement of the stakeholders more difficult; (2) The manual process is time-consuming. The stakeholders will need to wait for a report from the modeler for each of their scenario requests. This does not allow for an active discussion in which scenarios are refined rapidly. In practice, it is hard to define the scenarios in advance; usually, the scenarios are refined during an active discussion between the stakeholders (partly by seeing how the model reacts). As such, active discussion is very important, and it could be done only if the EDSS can be used in real-time without the need to manually perform time-consuming analyses.

Kumar [7] developed a user-friendly web-based EDSS to interact with an existing calibrated CE-QUAL-W2 model in Occoquan Reservoir in northern Virginia, USA. The purpose was to enhance stakeholder's interaction with the modeling software. The implementation included a multi-part system that is controlled by the user from a web server. The web server connects to a bridge module that sends the requests for model execution and mines the results. The final part utilizes other computers and servers on the local network in order to execute the different model runs in parallel. The results are then returned to the user for analysis.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

In the experiment of Kumar [7], the downsides of the manual approach in Erturk [6] were addressed. Thus, the modeler is no longer involved in the process and the parallel execution on multiple computers facilitates fast computation. Still, the work of Kumar [7] has some other downsides: (1) Not all the parts of the EDSS were reusable. The code was written specifically for the subject reservoir. That is, any other user that wants to use this infrastructure will need to change the source code to match his system; (2) On each computer in the network that is intended to be used for model execution, a piece of supporting software must be installed. This requires Information Technology (IT) to set up and maintain the software; and (3) Part of the implementation uses ArcGIS, which is a licensed program.

Shaw [8] implemented a very different approach for the Cumberland River system. This study described a method for computing hourly power generation schemes for a hydropower reservoir using high-fidelity models, surrogate modeling techniques, and optimization methods. The predictive power of the high-fidelity hydrodynamic and water quality model CE-QUAL-W2 was emulated by an Artificial Neural Network (ANN), then integrated into a Genetic Algorithm (GA) optimization approach to maximize hydropower generation subject to constraints on dam operations and water quality. By using the ANN as a surrogate model, Shaw [8] demonstrated a way to address the drawbacks of both Eturak [6] and Kumar [7]. The surrogate model runs within 2 seconds versus a 6 minute runtime in the CE-QUAL-W2 model of the considered system. This allowed running the EDSS on a single 6 core computer in a reasonable time frame. Nevertheless, the surrogate approach still has some drawbacks: (1) There is a need to implement and train a surrogate model in addition to the CE-QUAL-W2 model. For the training itself, many CE-QUAL-W2 runs were needed. In this case, 729 runs were made; (2) The solution is implemented using MATLAB and its "Neural Network" and "Optimization" Toolbox's which are licensed software.

Given the examples above, there is still a potential to develop a reusable EDSS that will be solely based on open source tools without the need to invest in expensive computation hardware. This could be achieved by using the Software as a Service (SaaS) paradigm combined with cloud computing technology. According to market analysts, such as the International Data Corporation (IDC), cloud computing has become more common and accessible over the last decade. They also show there is a trend of reduced usage costs for the users. They indicate these factors made the usage of SaaS more popular in the last few years. This conclusion is derived from the large growth in both revenues and market share for SaaS in the public cloud [9]. This can be explained by the benefits of this paradigm both for the customers and for the service providers. Some of these benefits are: (1) Customers do not need to have any computer infrastructure or install software on computers; (2) The company does not need to pay for computers and servers that are not in use or the "pay as you use" model; (3) The virtually "infinite" parallelization option for on demand compute power allows the companies to offer efficient solutions for any number of customers, termed scalability; (4) The company eliminates the need for local IT personnel to maintain the computation infrastructure and server rooms; (5) Updates are deployed quickly to all users. Similar to our paradigm other studies discussed leveraging cloud computing and SaaS for environmental software development. Swain [10] presented an open source platform for interacting and developing environmental applications. The platform was designed to simplify modern web-based software development over cloud infrastructure for scientists. The study focused on simplifying the development, but did not consider computational aspects such as parallel execution of the models. Ercan [11] developed a cloud-based SaaS to calibrate the Soil and Water Assessment Tool [12]. The proposed solution is based on a single algorithm that utilizes multiple (up to 256) Central Processing Units (CPU) for parallel executions of the model [11]. Recently, Li [13] implemented a Docker [14] based framework for developing EDSS that can be used on cloud infrastructure [13]. Considering the examples above, they all support the need to simplify environmental software development in order to make it more accessible to environmental scientists and hydrologists. This study continues these efforts, while focusing

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

on a water quality EDSS that requires intensive computational power. We propose a generic EDSS, which is based on the popular CE-QUAL-W2 model for: (1) Simplifying and reducing the needed amount of software development by requiring only a decision algorithm development instead of a full computational infrastructure; (2) Supporting flexible scaling of computer resources of parallel model runs, which are expected in decision problems that involve water quality simulations. To achieve the above goals, we explore new technologies that will simplify and reduce the cost of assimilating a water quality EDSS. Thus, offering a new paradigm of "Water Quality EDSS as a Service" which is an open-source computationally efficient platform that can support any EDSS algorithm application that utilizes the CE-QUAL-W2 model. This paradigm can make these tools more accessible and approachable for usage by environmental agencies and organizations and will enable advanced decision making and increase stakeholders' engagement.

2. Methods

The main principles that guide us in designing and constructing the EDSS as a Service are: (1) Supplying a low-cost solution and keeping the service as open source that can be changed and enhanced by future users; (2) Allowing the use of any cloud computing provider such as AWS, Google, Azure, or Alibaba.

2.1. Solution Architecture

In order to meet the goals above for developing an EDSS as a Service, an architectural design based on the latest available technology for SaaS is proposed. Figure 1 shows the encapsulation of the different core layers, followed by a detailed description of each of the core layers.

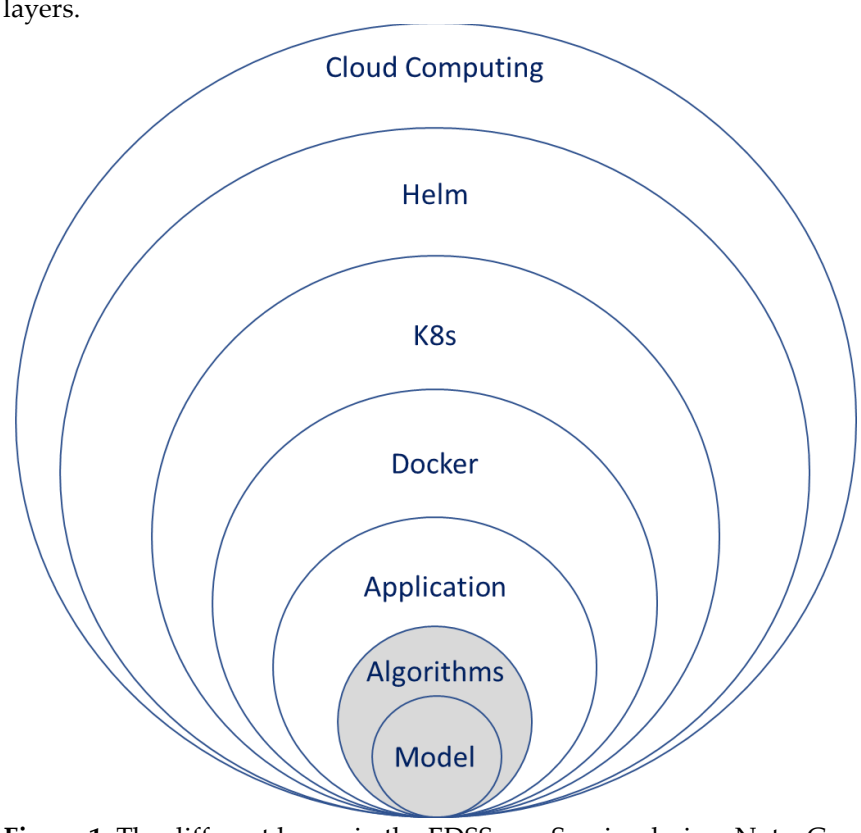

Figure 1. The different layers in the EDSS as a Service design. Note: Grey represents layers that will require changes between implementations of the EDSS. White represents the infrastructure layers that do not require changes between different implementations.

Model – The CE-QUAL-W2 model is the heart of the EDSS as it is used as a kernel for the decision making. In the last few years, the model was released only for Windows operating systems. Although it is possible to develop this kind of EDSS as a Service in a Windows environment [15], we prefer the Linux environment since it is both more cloud environment compatible and license-free. For that need, we created an open source

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

GithHub project. This project holds the needed files and instructions in order to compile the CE-QUAL-W2 source code in a way it can be executed in a Linux environment [16]. Beside the model executable, this layer also includes the user specific input files of a calibrated model ready for simulations. These input files are the template that the application layer is changing according to the algorithm requirements.

Algorithm – The layer of the algorithm is responsible for two main operations: (1) Deciding on the needed permutation for the model simulations and supplying the different parameters needed for the model input files for each of these simulations; (2) Analyzing the results of the simulations according to the developed algorithm. A single EDSS can hold multiple algorithms for the user to choose from. For example, an algorithm can do a grid search for the best matching simulation output according to the user input targets, while another algorithm can plot a specific model output parameter as recorded in all the different simulations.

Application – This layer has five different responsibilities. The interactions of these responsibilities are described in Figure 2: (1) Supply a simple web user interface for the EDSS, allowing users to send requests to the service (Figure 3). An example of a user request is shown later in section 2.2.1; (2) Pass the user request to the algorithm layer and receive a response from the algorithm to specify the needed simulations permutations list; (3) Initiate parallel model execution requests according to the permutations; (4) Collect all the model simulation results and send them back to the algorithm once model simulations are completed. (5) Get the analyzed results from the algorithm and display it back in the web user interface.

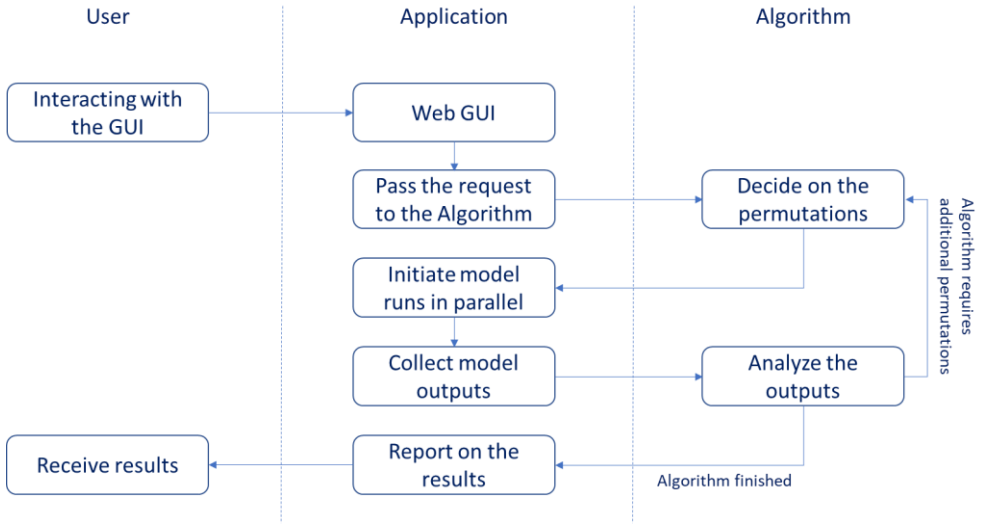

Figure 2. Application layer interactions.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

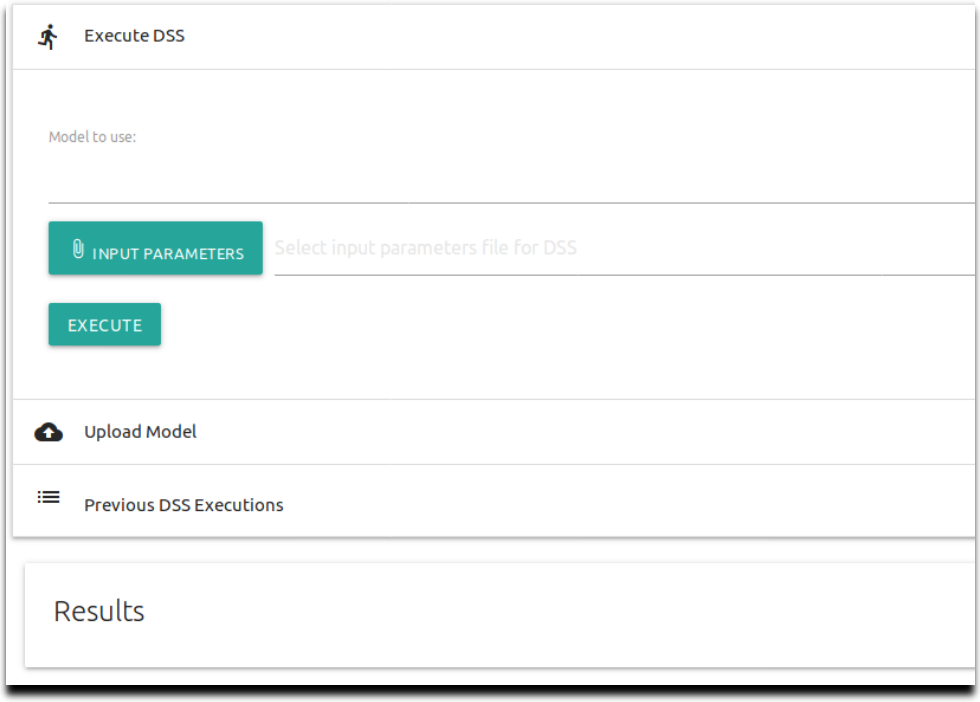

Figure 3. The web user interface for the EDSS.

Docker - The Docker Container layer was developed to allow isolation and packaging of software together with all of its dependencies. A container is an executable that can be run on any computing environment without needing to worry about the operating system or the hardware infrastructure [14]. The Docker infrastructure is free for use and holds the following benefits for the EDSS implementation: (1) No need to setup the infrastructure or the operating system; (2) It can run on cloud resources as well as on a single computer. The Docker engine can be run on Windows, Linux, or Mac operating systems; (3) Cloud providers supply a simple and cost-effective interface for setting up your application using Docker. In this EDSS as a Service implementation, a single Docker container was created. This container can be used either for running the application or running individual model simulations in parallel across multiple model executions.

Kubernetes (K8s) – Is a layer that was developed after organizations started to adopt the Docker solution and were looking for a way to streamline the scaling process and coordinate multiple services encapsulated as Docker containers. Scaling involves initiating more and more Docker containers according to the demand [17]. K8s is also open-source. In this EDSS, it is leveraged for the purpose of managing the runs of multiple Docker containers in parallel, each running a different model simulation. This allows for the automatic scaling of the cloud computing power when needed, allowing the user to get the benefit of the "pay as you use" cloud computing model while all the simulations are performed in parallel.

Helm – This is a packaging layer that was developed in order to simplify the deployment of K8s applications according to a predefined configuration file [18]. This layer is free for usage and allows easy and repeatable deployment of this EDSS to the cloud computing environment. Without using Helm, deploying the EDSS to a cloud provider would require many manual configurations.

Cloud computing – This final layer, is the base that allows the EDSS as a Service to hold minimal compute units that run only the user interface during the idle times and scale to multiple compute units when model simulations are needed. In order to be able to use any cloud computing provider, the EDSS was built to rely on basic computing building blocks which are in use by all cloud computing providers although it could have been easier to design the system for a specific cloud computing provider. Where applicable, current industry standards and best practices were applied to ensure that the APIs

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

and interfaces between the layers will be supported by various 3rd party tools to simplify the deployment, scaling, and management of the infrastructure layers.

2.2. *Interfaces*

There are two different personas for this EDSS, dedicated interfaces were created to match their distinct needs. The first interface is for the user who wants to consult the EDSS about a water quality issue. The second persona is the EDSS developer (algorithm developer) who wants to introduce a new algorithm or to enhance an existing algorithm.

2.2.1. User interface

The user interface, as displayed in Figure 3, is divided into three sections: (1) Initiate the model – the model user sets the model that will be run and the input parameters file for the algorithm. According to this input, the algorithm both decides on the needed permutations and analyses the runs. The input file can be in any format that the algorithm code can read. For the case study, we developed an algorithm that reads JSON format files due to its ease of processing using any computer programming language [19], but CSV or TXT could have been used as well. Once the input file is selected, the user can start the run by pressing the "Execute" button; (2) Upload model – the model user uploads their CE-QUAL-W2 model input files (not the executable itself) based on their own calibration. This allows the user flexibility of usage without needing their CE-QUAL-W2 input files to be part of the source code that is wrapped with the Docker image; (3) Results – These are the results reported by the algorithm once all the model runs are finished and saved to a downloadable zip file. There is also an option to view results from previous runs as shown in Figure 3. In the EDSS GitHub repository, we provide a demo video tutorial with instructions on using the EDSS.

2.2.2. Algorithm developer interface

An EDSS can use numerous different algorithms to support decision making, depending on the issue in hand. This current design of the EDSS as a Service allows developing different algorithms and embedding them in the infrastructure for different use cases. The algorithm developer needs basic knowledge in Python programing language, working with Git source code control, and a basic understanding in cloud compute infrastructure in order to test the code change. The logic for each of the core modules is well isolated, and the interfaces between the layers (Application -> Docker -> K8s -> Helm -> Cloud Compute) are clearly defined. The developer does not need to change the infrastructure layers in order to support a new algorithm.

For developing a new algorithm, the following steps need to be taken: (1) Fork of the current GitHub project [20] to a new one; (2) Creating a Travis CI account [21]. The current GitHub project is configured to use Travis CI for running the unit tests and pushing the created Docker images to the Docker repository. The new Travis CI account needs to be configured in the GitHub project; (3) Creating a Docker account [14] to hold the Docker images. All the mentioned accounts (GitHub, Travis CI, and Docker) are free for open source projects; (4) The "*dss/scripts/build_and_push_to_docker_hub.sh*" script needs to be updated to reference the new Docker repository. The Helm "*dss/chart/wqdss/values.yaml*" file needs to be updates as well, to retrieve the Docker images from the correct repository; (5) As described in Figure 4, the entry function of the algorithm is "*execute*" under the "*dss/src/wqdss/processing.py*" Python module in the "*Execution*" class (see EDSS GitHub repository [20]). This is where the code should be changed for a new algorithm. In a case of several algorithms, the input file should include the algorithm name and the applicable function according to the selected algorithm that will be called from the "*execute*" function. In order to request multiple model simulations, the "*execute_run_async*" function needs to be called. Once this function returns, the results can be analyzed. This function can be called several times if needed. In order to publish the results, the class member "*self.result*" should be updated before exiting the "*execute*" function. (6) The changed code then needs to be pushed to the main GitHub branch in order for the Travis CI to trigger the unit tests and publish the Docker image to the Docker repository; (7) Deploying the code using Helm and verifying the EDSS behavior.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

```python
async def execute(self, params):
    # Start your algorithm here.
    # "params" is the file passed by the user.
    # It can be further processed or changed before passed to "execute_run_async" function.
    # This is an example where "permutations" and "iteration" are set by the algorithm.
    awaitables = [self.execute_run_async(self.model_name, params, p, iteration) for p in permutations]
    # The requests for model execution are passed, and now we await for the results.
    await asyncio.gather(*awaitables)
    # "self.runs" is updated with the results and can now be processed.
    # The processing is either for the final results or for another execution round.
    # Write your processing code here.
    self.result.append({"The info you want to return to the user"})
    # Once the function is exited, the info is returned to the user.
```

Figure 4. Code snippet for creating a new algorithm.

2.3. Cloud computing resources scaling

When using cloud resources with this EDSS, the scaling is performed in two levels: (1) The Horizontal Pod Autoscaling (HPA), which determines how many different Docker pods (images) are brought up in order to receive a request for model execution. The Docker images are configured to run a single model each. The Helm "values.yaml" file has an "hpa" section with an option to configure the minimum and maximum number of pods that are brought up. In addition, the CPU target utilization percentage is also defined in this section. Once the CPU usage crosses this limit a new pod is created; (2) The cloud provider K8s cluster auto scaling. Once there is more CPU usage requests than the cloud computing is already providing, additional compute resources to run the pods will be automatically brought up by the cluster according to the limits defined in the cluster settings.

There are multiple factors that will affect the scaling efficiency. Starting from the chosen type of compute node that brings with it a different type of CPU generation. Run time of a single model can be cut in halve between the newest CPU generation and old CPU generations. As mentioned above, different configurations of the K8s cluster and HPA will also have a significant effect on optimizing the scaling, where optimized scaling is defined as running all the model permutations in parallel. However, having optimized scaling comes with a cost of having more cloud computing nodes ready for execution, and from there a higher idle time cost, as you need to pay for the running computing nodes even if almost no CPU is utilized. As such, having optimized scaling is not necessarily a primary goal. Nonetheless, we want to benchmark an optimal scaling performance for reference. To demonstrate the scaling, the "Spokane river example" which is shipped with the CE-QUAL-W2 version 4.1 package was employed. We changed the simulation days from 200-205 to 200-300 in order to create a longer run and highlight the benefit of the scaling. The Google Cloud provider was chosen to deploy the EDSS. For the computing nodes, the e2-standard-2 (2 vCPUs, 8GB memory) were used. To perform 10 parallel simulations, both the HPA and K8s cluster configurations were made such that all the needed computing nodes and pods were already up and running at the beginning of the execution. That is, 10 Pods and 5 computing nodes. This run of 10 model permutations took 26 min. Using the same configuration, only changing the HPA to allow a single pod (i.e., 10 serial runs without parallelization) took three hours and 51 minutes (23*10=230 minutes).

3. Results

The state of Israel has thirteen different drainage and river authorities, each responsible for streams and rivers in a different part of the country. Among their many duties, these authorities are also responsible for rehabilitating the rivers and streams and adapting them for leisure and recreational purposes [22]. Each authority is independent and is relatively a small organization (staffed with less than 10 people). One of these authorities is the Yarqon River Authority (YRA), that is responsible for the Yarqon stream [23]. The Yarqon is a lowland coastal stream, about 28 km long, in central Israel that flows between a mix of agricultural fields and urban areas, ending in an estuary connected to the Mediterranean Sea as shown in Figure 5. While the historic flow of the stream originated in natural springs, over the years, due to over-exploitation of the aquifer, the natural

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

springs dried out [24]. Today, there are two different water sources for the stream: (1) Pumps supplying water from lower depths in the aquifer for the upstream part instead of the springs that dried out; (2) Three different Waste Water Treatment Plants (WWTP) supplying water in a tertiary quality in two locations in the middle section of the stream [24]. In order to promote preservation and recovery efforts, the YRA works with 18 different stakeholders [23], which poses a significant obstacle to reach consensus on rehabilitation efforts. We set out to show how a Water Quality EDSS as a Service can serve as a solution for the YRA in its efforts to rehabilitate the stream.

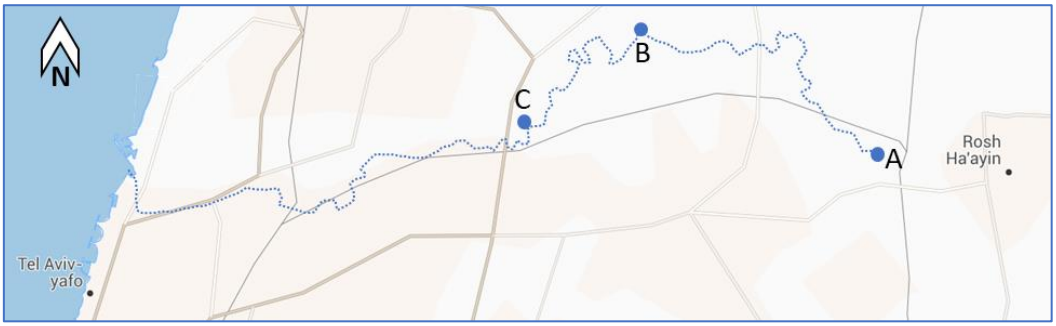

Figure 5. Study area map. (A) Source of the stream, from aquifer water pumps. (B) entrance of two WWTP. (C) entrance of a third WWTP.

Note: background map is adapted from "Michelin maps"

3.1 Applying the CE-QUAL-W2

Water quality models are rarely used in Israel's rivers and streams. In order to show the benefits of the EDSS, a CE-QUAL-W2 model had to be applied for the Yarqon stream. However, the available data on flows, temperature and water quality in the stream was not sufficient to perform a calibration of the model. For EDDS demonstration purposes, we applied a non-calibrated model for the stream according to the best data that was available. Although, under these conditions the EDSS cannot supply real outputs for decision making, the YRA could still get the demonstration on how the EDSS will be able to assist them if they invest in calibrating a model for the stream.

3.2 The EDSS algorithm

For demonstrating purposes, the example implemented algorithm was a simple recursive grid search, which uses two types of inputs as shown in Figure 6: (1) *model_run* that specify the parameters needed for the algorithm to set the different run permutations; (2) *model_analysis* that specify the parameters for finding the best run. JSON format for the input file was chosen over other formats like CSV or TXT due to its flexibility. Under the *model_run* section, there is an option to define any number of model input files that need to be changed in the *input_files* section. For each input file, the user needs to define the following: (1) *name* – Name of the input file; (2) *col_name* – Name of the CSV column that needs to be changed (the parameter that is changed); (3) *min_val* – The minimum value of the parameter that is being changed; (4) *max_val* - The maximum value of the parameter that is being changed; (5) *steps* – The increase interval in the parameter value between the minimum and maximum definitions. This parameter can hold a list of values. If more than one value is defined, a recursive run is done in smaller intervals for further rounds of model simulations around the previous simulation result that best matched the target. Although it is possible for the user to define the smallest interval in the first run and get the same results faster, the recursive option was added to allow cloud com-

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

puting cost reduction by reducing the overall number of permutations, without significantly impacting the accuracy of the result. In the example shown in Figure 6, in the first pass of model execution, 9 different permutations will be executed as there are (2-1)/0.5+1=(34-30)/2+1=3 different parameters values to set in each of the 2 input files. In the second pass of model execution, the minimum and maximum range is set from the previous round best run parameter value +/- previous step value divide by two. In this case there are another 0.5/0.05+1=2/0.2+1=11 different parameters values to set in each of the 2 input files.

Under the *model_analysis* section, the model output file that needs to be analyzed is defined in the *output_file* field. Under the section of *parameters*, any number of parameters can be defined. Each parameter has the following definitions: (1) *name* – Column name in the output csv file; (2) *target* – The target value of the parameter we want to reach; (3) *weight* – This is a relative parameter that allows to set a priority between the different defined parameters; (4) *score_step* – is defined in order to unify the units of different parameters. It is defined as the deviation from target per unit score. For example, in Figure 6, a deviation of 0.1 g/m^3 in Total Nitrogen (TN) is considered one score, while a deviation of 0.5 g/m^3 in Dissolved Oxygen (DO) is considered one score. As such, division of the parameter's deviations by the corresponding *score_step*, will facilitate summing the deviations of different parameters in score units as shown in Equation 1. Equation 1 defines what score each model simulation will get according to the distance from the target.

$$Score = \sum_{i=1}^{n} \left(\frac{| target_i - actual\ value | / score\ step}{weight} \right) \quad (1)$$

Since the score represents deviation from targets, we seek the model simulation which minimizes the score. For example, for the run in Figure 6, we seek the model simulation which minimized the score as defined in Equation 2.

$$Score = \left(\frac{| 0.6 - Simulated\ TN | / 0.1}{4} \right) + \left(\frac{| 11 - Simulated\ DO | / 0.6}{2} \right) \quad (2)$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

```
{
  "model_run": {
    "input_files": [
      {
        "name": "hangq01.csv",
        "col_name": "Q",
        "min_val": "1",
        "max_val": "2",
        "steps": ["0.5", "0.05"]
      },
      {
        "name": "qin_br8.csv",
        "col_name": "QWD",
        "min_val": "30",
        "max_val": "34",
        "steps": ["2", "0.2"]
      }
    ]
  },
  "model_analysis": {
    "output_file": "tsr_2_seg7.csv",
    "parameters": [
      {
        "name": "TN",
        "target": "0.6",
        "weight": "4",
        "score_step": "0.1"
      },
      {
        "name": "DO",
        "target": "11",
        "weight": "2",
        "score_step": "0.5"
      }
    ]
  }
}
```

Figure 6. Example input file. Note: For the permutations, two different model files are changed, and a single recursive step is defined. For the algorithm analysis, two output parameters are considered with different "weight" and "score_step".

3.3 Case study results

In order to demonstrate the abilities of the EDSS, we choose a decision making problem in which the decision maker needs to decide on the pump flow rate that supplies the water from the aquifer in the most upstream section (point A in Figure 5) to achieve a desired goal of water quality. More specifically, the algorithm needs to quantify the impact of changes in the pump flow rate on the downstream section water quality, given the additional inputs from WWTPs in the middle of the stream, and the natural processes occurring along the stream's 28 km path (Figure 5). This decision-making problem represents a constant debate among stakeholders on the amount of freshwater that should be allocated to the stream in order to meet water quality targets.

A scenario of excess Ammonia (NH4-N) concentration downstream was simulated. As shown in Figure 7, the EDSS input file defines possible range of pump flows (*col_name* =

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

q) between 0.2 m^3/sec (*min_val*) and 0.8 m^3/sec (*max_val*). The flow changes as defined in the steps were set for 2 recursive runs, one with 0.1 m^3/sec and the second with 0.01 m^3/sec. The "qin_br1.csv" (*name*) is the input file that needs to be changed. The NH4-N concentration target of 0.57 g/m^3 (*target*) was set. The weight is not relevant in this case, as only a single target parameter was used. The simulated NH4-N concentration is extracted from the output file of "tsr_1_seg42.csv" (*name*). The defined *score_step* was set to 0.01 g/m^3. The pump flow that minimizes the score was found by the EDSS as shown in Figure 8: (1) id - Each EDSS execution has a unique string; (2) status - this field is changed from "RUNNING" to "COMPLETED" once the EDSS publishes the results; (3) result – shows the results of the best run. In this case, as two recursive runs were defined, we can see two sets of results with different identification strings (*best_run*). The first is for the search between 0.2 m^3/sec and 0.8 m^3/sec with 0.1 m^3/sec interval. In which, a flow of 0.6 m^3/sec (*params*) had the best relative score of 0.525 out of the 7 runs. The second results are for the run between 0.55 m^3/sec and 0.65 m^3/sec with 0.01 m^3/sec interval. In this second search, the best score is 0.025 which corresponds to a flow of 0.55 m^3/sec; (4) A link to download the zip output files from the model executions is available as shown in Figure 8.

```
{
    "model_run": {
        "input_files": [
            {
              "name": "qin_br1.csv",
              "col_name": "q",
              "min_val": "0.2",
              "max_val": "0.8",
              "steps": ["0.1", "0.01"]
            }
        ]
    },
    "model_analysis": {
        "output_file": "tsr_1_seg42.csv",
        "parameters": [
            {
              "name": "NH4",
              "target": "0.57",
              "weight": "4",
              "score_step": "0.01"
            }
        ]
    }
}
```

Figure 7. Case study input file

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0274.v1

Figure 8. Output of the EDSS. Note: In the first round of simulations (i.e. steps of 0.1 m^3/sec), 0.6 m^3/sec flow had the best score. In the second round of simulations (i.e. steps of 0.01 m^3/sec around 0.6 m^3/sec) 0.55 m^3/sec flow had the best score.

In this example seven parallel simulations were done on cloud computing resources in the first round. In the second round eleven simulations were done in parallel. All together eighteen simulations were done in order to reach the result. A more cloud computing expensive path could have been taken if a "step" of 0.01 m^3/sec was solely defined with no recursive rounds. In that case 61 simulations would have been done, resulting in a shorter run time (in case these runs are performed in parallel), but with more cloud computing charges.

4. Discussion

The developed paradigm of a "Water Quality EDSS as a Service" holds multiple benefits for environmental organizations who are working on wetlands preservation and recovery. This specific study implementation is highly beneficial to organizations or researchers who already have a calibrated CE-QUAL-W2 model for the wetland they are managing, as this remains the highest cost in the EDSS. We can divide the direct users' benefits into two categories: (1) Benefits to the end user who consults the EDSS; (2) Benefits to the developer, looking to introduce a new algorithm to the user.

4.1. User benefits

1. Ease of use – The simple web interface allows interaction with the system without a need to understand the format of the CE-QUAL-W2 input files. There is also no need to install any software on the user's computer.

2. Reduce simulation time - As the different model simulations are in parallel, the user can get the result within the time frame of a single model simulation.

3. Extensibility - Any kind of algorithm can be embedded by a software developer in the service.

4. Low cost – All the software that is in use in this solution is open source. The only needed payments are for the cloud compute "pay as you use" and for a software developer to adjust the needed algorithm to the organization needs and then bring up the service for use. Next, we review the developer benefits which also help in maintaining a low development cost. An additional aspect is that the developed infrastructure is generic and can support any cloud computing provider (such as AWS, Google, Azure, or Alibaba), where some environmental organizations or researchers can get grants for usage from one of the providers. An additional cost saving can also be achieved if a large environmental organization maintains the service and algorithms for smaller organizations. For example, in the YRA case, if the Israeli water authority takes responsibility for developing the algorithms and maintaining the service, all thirteen drainage and river authorities will be able to benefit from it without the need for each of the thirteen to develop their own algorithms.

4.2. Developer benefits

1. Ease of new algorithm implementation – The architecture composed of different software layers as described in Figure 1, might look overwhelming, but it is meant to isolate the different core components from the infrastructure layers. The algorithm developer would only need to understand the interfaces between the application and the algorithm, be familiar with Python programming language, and have basic knowledge on how to deploy the service to the cloud provider.

2. Flexible user interface – The flexible design for the user input does not require having any web development knowledge. It also allows creation of an input file that is clear and can be easily produced by the user.

In addition to the direct user's benefits, the "EDSS as a Service" paradigm, also enabled the usage of a legacy software in a modern service. Specifically, the CE-QUAL-W2 model has been developed, fine-tuned, and stabilized over the past 40 years. As an open source project, it will be hard to finance the model redesign into a modern software design. This "EDSS as a Service" allows users to leverage the benefits of a modern web service along with its kernel of a well-established model. The developed infrastructure can be also used for the purpose of fine tuning the model calibration. Today, the modeler calibrating the CE-QUAL-W2 model needs to run the model multiple times while changing different tuning parameters. This is done in order to find the best match to the historic observed results in the field. This process is tedious and time consuming, especially for long running simulations. An algorithm, similar to the grid search algorithm that was implemented for the YRA, can be utilized to perform parallel search for the best tuning parameters that maximize the goodness of fit between observed and simulated results.

Author Contributions: Conceptualization, Y.B. and M.H.; methodology, Y.B.; software, B.D.; validation, Y.B. and S.W.; investigation, Y.B.; data curation, Y.B.; writing—original draft preparation, Y.B.; writing—review and editing, M.H., S.W., and B.D.; visualization, Y.B.; supervision, M.H.; project administration, Y.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement:

EDSS GitHub repository: https://github.com/WQDSS/Evaluna

CE-QUAL-W2 for Linux environment compilation, GitHub repository - https://github.com/WQDSS/CE-QUAL-W2-Linux

Acknowledgments: This work was supported by the Yarqon River Authority.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ramsar Convention Secretariat. Water allocation and management: Guidelines for the allocation and management of water for maintaining the ecological functions of wetlands. Ramsar handbooks for the wise use of wetlands, 4th edition, vol. 10. Ramsar Convention Secretariat, Gland, Switzerland **2010**
2. U.S. Environmental Protection Agency. 2001. Stakeholder involvement & public participation at the U.S. EPA: Lessons learned, barriers, & innovative approaches. Available online: https://www.epa.gov/sites/production/files/2015-09/documents/stakeholder-involvement-public-participation-at-epa.pdf (accessed May 22, 2021)
3. Cole, T.M., and Wells, S. A. (2018) "CE-QUAL-W2: A two-dimensional, laterally averaged, hydrodynamic and water quality model, version 4.1," Department of Civil and Environmental Engineering, Portland State University, Portland, OR.
4. Mateus, M., da Silva, R., Almeida, C., Silva, M., Reis, F. ScoRE—A Simple Approach to Select a Water Quality Model. *Water*, **2018**, *10* , https://dx.doi.org/10.3390/w10121811
5. CE_QUAL-W2 application by Country. Available online: http://cee.pdx.edu/w2/ (accessed May 22, 2021)
6. Erturk, A., Ekdal, A., Gurel, M., Zorlutuna, Y., Tavsan, C., Seker, D., Tanik, A., Ozturk, I.. Application of Water Quality Modelling as a Decision Support System Tool for Planned Buyuk Melen Reservoir and Its Watershed. Sustainable Use and Development of Watersheds. NATO Science for Peace and Security Series, (Series C: Environmental Security), **2008**, 227-242. https://doi.org/10.1007/978-1-4020-8558-1_14
7. Kumar, S., Godrej, A., Grizzard, T.. A web-based environmental decision support system for legacy models. *J. Hydroinformatics*. **2015**, *17*, http://dx.doi.org/10.2166/hydro.2015.007
8. Shaw, A., Sawyer, H., LeBoeuf, E., McDonald, M., Hadjerioua, B. Hydropower Optimization Using Artificial Neural Network Surrogate Models of a High-Fidelity Hydrodynamics and Water Quality Model. *Water Resour. Res.*, **2017**, *53*, http://dx.doi.org/10.1002/2017WR021039
9. Martins, R., Oliveira, T., Thomas, M. An empirical analysis to assess the determinants of SaaS diffusion in firms, *Comput. Hum. Behav.*, **2016**, *62*, 19-33, https://doi.org/10.1016/j.chb.2016.03.049
10. Swain, N., Christensen, S., Snow, A., Dolder, H., Espinoza-Dávalos, G., Goharian, E., Jones, N., Nelson, J., Ames, D., Burian, S. A new open source platform for lowering the barrier for environmental web app development, *Environ. Model. Softw*, **2016**, *85*, 11-26, https://doi.org/10.1016/j.envsoft.2016.08.003
11. Ercan, M., Goodall, J., Castronova, A., Humphrey, M., Beekwilder, N., Calibration of SWAT models using the cloud, *Environ. Model. Softw*., **2014**, *62*, 188-196. https://doi.org/10.1016/j.envsoft.2014.09.002
12. Gassman, P. W., M. R. Reyes, C. H. Green, and J. G. Arnold. 2007.The Soil and Water Assessment Tool: Historical development,applications, and future research directions. Trans. ASABE50(4): 1211-1240
13. Li, Y., Towards fast prototyping of cloud-based environmental decision support systems for environmental scientists using R Shiny and Docker, *Environ. Model. Softw*, **2020**, *132*, https://doi.org/10.1016/j.envsoft.2020.104797
14. Docker. Available online: https://www.docker.com/ (accessed May 22, 2021)
15. Docker Windows Containers. Available online: https://www.docker.com/products/windows-containers (accessed May 22, 2021)
16. CE-QUAL-W2-Linux, port of the CE-QUAL-W2 model to Linux. Available online: https://github.com/WQDSS/CE-QUAL-W2-Linux (accessed May 22, 2021)
17. Kubernetes. Available online: https://kubernetes.io/ (accessed May 22, 2021)
18. Helm. Available online: https://helm.sh/ (accessed May 22, 2021)
19. JSON. Available online: https://www.json.org/json-en.html (accessed May 22, 2021)
20. EDSS, Available online: https://github.com/WQDSS/Evaluna (accessed May 22, 2021)
21. TravisCI. Available online: https://travis-ci.org/ (accessed May 22, 2021)
22. Israel drainage authorities (Hebrew). Retrieved from Kishon Drainage Authority: https://www.rnkishon.co.il/%D7%A8%D7%A9%D7%95%D7%99%D7%95%D7%AA-%D7%94%D7%A0%D7%99%D7%A7%D7%95%D7%96-%D7%91%D7%99%D7%A9%D7%A8%D7%90%D7%9C/ (2021, May 22).
23. Yarqon River Authority. Available online: https://www.yarqon.org.il/en/ (accessed May 22, 2021)
24. Arnon, S., Avni, N. & Gafny, S. Nutrient uptake and macroinvertebrate community structure in a highly regulated Mediterranean stream receiving treated wastewater. *Aquat Sci*, ***2015***, 77, 623–637. https://doi.org/10.1007/s00027-015-0407-6