Edge-to-cloud Virtualized Cyberinfrastructure for Near Real-time Water Quality Forecasting in Lakes and Reservoirs

Vahid Daneshmand
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida, USA
vdaneshmand@ufl.edu

Adrienne Breef-Pilz
Department of Biological Sciences
Virginia Tech
Blacksburg, Virginia, USA
abreefpilz@vt.edu

Cayelan C. Carey
Department of Biological Sciences
Virginia Tech
Blacksburg, Virginia, USA
cayelan@vt.edu

Yuqi Jin
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida, USA
jinyuqi@ufl.edu

Yun-Jung Ku
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida, USA
y.ku@ufl.edu

Kensworth C. Subratie
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida, USA
kcratie@ufl.edu

R. Quinn Thomas
Department of Forest Resources and Environmental Conservation
Virginia Tech
Blacksburg, Virginia, USA
rqthomas@vt.edu

Renato J. Figueiredo
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida, USA
renatof@ufl.edu

***Abstract*—The management of drinking water quality is critical to public health and can benefit from techniques and technologies that support near real-time forecasting of lake and reservoir conditions. The cyberinfrastructure (CI) needed to support forecasting has to overcome multiple challenges, which include: 1) deploying sensors at the reservoir requires the CI to extend to the network's edge and accommodate devices with constrained network and power; 2) different lakes need different sensor modalities, deployments, and calibrations; hence, the CI needs to be flexible and customizable to accommodate various deployments; and 3) the CI requires to be accessible and usable to various stakeholders (water managers, reservoir operators, and researchers) without barriers to entry. This paper describes the CI underlying FLARE (Forecasting Lake And Reservoir Ecosystems), a novel system co-designed in an interdisciplinary manner between CI and domain scientists to address the above challenges. FLARE integrates R packages that implement the core numerical forecasting (including lake process modeling and data assimilation) with containers, overlay virtual networks, object storage, versioned storage, and event-driven Function-as-a-Service (FaaS) serverless execution. It is a flexible forecasting system that can be deployed in different modalities, including the Manual Mode suitable for end-users' personal computers and the Workflow Mode ideal for cloud deployment. The paper reports on experimental data and lessons learned from the operational deployment of FLARE in a drinking water supply (Falling Creek Reservoir in Vinton, Virginia, USA). Experiments with a FLARE deployment quantify its edge-to-cloud virtual network performance and serverless execution in OpenWhisk deployments on both XSEDE-Jetstream and the IBM Cloud Functions FaaS system.**

This work is supported by the US National Science Foundation as part of awards CNS-1737424, DBI-1933102, DBI-1933016, OAC-2004441, and OAC-2004323. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

***Index Terms*—edge, cloud, cyberinfrastructure, Function-as-a-Service, ecological forecasting, water quality**

I. Introduction

Surface water in lakes, reservoirs, and rivers provides 64 percent of public freshwater systems [9]. Consequently, managing the water quality of these ecosystems is critical for protecting public health. Due to increased human activities, water quality in lakes and reservoirs around the globe is increasingly exhibiting variability outside the boundaries of historical conditions [20], which makes it challenging to manage water quality in the present, as well as to anticipate changes in the future [2]. Thus, there is significant pressure on managers responsible for providing day-to-day critical lake and reservoir ecosystem services [4]. In response to this challenge, near-term forecasts of water quality are increasingly needed to inform real-time management of drinking water supplies. Ecological forecasts provide managers with probabilistic estimates of future water quality conditions in their target lake or reservoir, allowing them to take preemptive management steps to preempt water quality threats [4].

Water quality forecasting workflows require a lake model, observational data, data assimilation algorithms, and cyberinfrastructure (CI). On each time step, near-term iterative forecast models are updated with observations using data assimilation methods and fed back into the workflow to generate future forecasts [27]. Forecasting workflows can build upon a combination of empirical and process models and observed data from various types of sensors [11]. To support seamless and uninterrupted data assimilation of environmental observations (e.g., at the scale of minutes) with model forecast

runs occurring regularly (e.g., daily), it is necessary to deploy a robust yet flexible end-to-end CI encompassing sensors at a lake/reservoir site, networking, data transfer storage, and model execution [27]. Water quality forecasters need such a CI to compose and orchestrate integrated model-data systems that create forecasts [27].

While the infrastructure for national weather forecasting systems is well-established [1], forecasting water quality in lakes and reservoirs poses a problem with unique complexity due to the heterogeneous characteristics of the millions of lakes and reservoirs across the world [31]. Accurate forecasts for these water bodies require observations from sensors locally deployed at the site, in addition to widely-available national-scale or global-scale remote sensing satellite data. These sensors may be attached to buoys and/or platforms on or near the water and may need to be deployed in remote locations with poor network connectivity. Furthermore, lakes and reservoirs have different characteristics that require different types of sensors and models to properly capture temporal and spatial variability, such as the number and size of inflows and outflows, bathymetry, winter ice conditions, trophic state, and mixing regime [34]. In essence, there is no single solution in terms of sensors and models that can be applied to all lakes. Therefore, water quality forecasts need an underlying CI that can deliver high degrees of flexibility and customization.

This customizability is very challenging for the CI since it needs to encompass the entire end-to-end workflow, from sensors (data capture) to the edge (data staging, processing, and transfer), to the cloud (long-term storage, model execution, and data assimilation). This entails dealing with the variety of sensors in the field, constrained network connectivity at the edge (possibly performing computations at the edge to reduce data transfers), securely transferring the data to the cloud, and finally deploying a customized forecasting workflow for every lake or reservoir. In addition to being highly customizable and robust, the CI also needs to be effectively usable by domain experts: e.g., drinking water treatment plant operators, freshwater ecologists, and water utility managers. Furthermore, the cloud computing modules of the CI should be deployable on both public and private clouds and, in the former case, support a Function-as-a-Service (FaaS) model that minimizes deployment costs. This falls into what is often referred to as the "long tail of science" and calls for an interdisciplinary approach where the CI is co-designed with input from the various stakeholders [3].

To address these demands, we have developed FLARE [27] (Forecasting Lake And Reservoir Ecosystems), a novel end-to-end platform for a flexible, customizable, end-to-end near real-time iterative water temperature forecasting system [27]. It has been successfully deployed and operational for over two years in a drinking water reservoir (Falling Creek Reservoir in Vinton, Virginia, USA). It effectively predicted the beginning of fall turnover (i.e., the process of a lake's water turning over from epilimnion to hypolimnion) 4–14 days in advance in three consecutive autumns [27].

FLARE open-source platform for lake and reservoir water quality forecasting [27] builds upon and integrates modern open-source software frameworks to implement its desired functionality. At its core in the cloud, it uses Docker containers for microservice deployment and Apache OpenWhisk [42] for orchestration of event-driven FaaS actions. At the edge, FLARE leverages the EdgeVPN.io [25], [43] (Evio) software-defined overlay virtual private network for edge-to-cloud transfers and remote management. For storage and transfer, FLARE uses well-known data services: Git is used for staging data at the edge, versioning, and reliable transfer of time-series differences from edge to cloud, and S3 is used for data staging for FaaS services. The numerical forecasting core uses the General Lake Model (GLM [16]) to model lake processes (physics, chemistry, and biology) and data assimilation based on the ensemble Kalman filter (EnKF) [13].

The modular design of FLARE (Figure 1) enables it to be deployed for various use-case scenarios. These scenarios range from an individual researcher beginning to develop a workflow with a single computer, to operational forecasts overseeing sensors/edge nodes deployed across multiple lakes and sharing a cloud platform. Using a containerized structure as the backbone of FLARE facilitates complex workflows while enabling customization and expanding the use-cases. The event-driven architecture enabled by OpenWhisk allows increased modularity, in which complex workflows are triggered based on data availability, and allows cost-efficient deployment of FLARE in commercial FaaS clouds, such as IBM Cloud Functions [37].

To enable seamless network connectivity among nodes from edge to cloud, FLARE integrates EdgeVPN.io [25], [43] (Evio), an open-source software for deploying scalable virtual private networks across distributed edge resources. Using Evio, a private network is established between edge and cloud devices, providing authentication and privacy at the network layer for both data transfers and remote management of edge nodes. This is an essential layer of the broader security framework needed for a trustworthy CI for water supply management. A recent cyber-attack on Oldsmar's water system [48] highlights the crucial importance of security in water quality management systems. Besides leveraging state-of-the-art technologies, our interdisciplinary co-designed approach ensures that FLARE integrates with frameworks and environments well-known by ecologists, such as the R programming language and the Git distributed version control system supported by cloud platforms, including GitHub.

This paper makes the following contributions:

- We describe the architecture of the CI underlying FLARE, which is novel in 1) how it creates a distributed virtual cluster with containers on edge and cloud resources supporting end-to-end water quality forecasting workflows, 2) how it uses event-triggered functions to deploy workflows at low cost, and 3) how it integrates Git for distributed storage and reliable transfers of file deltas from edge to cloud while preserving a file-based abstraction that is convenient to FLARE's target user base.

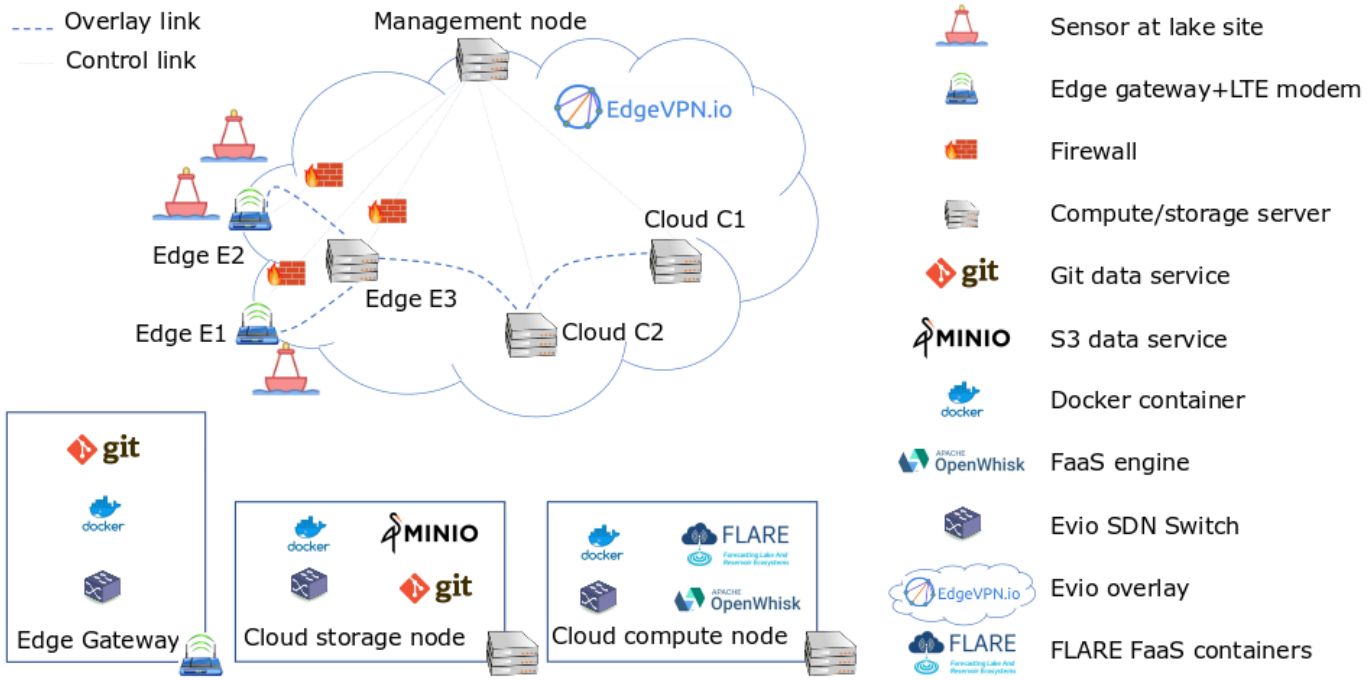

Fig. 1. Overview of CI and an example end-to-end workflow in FLARE. In edge site E1, sensor data are captured and staged in a local data Git repository within a Docker container running in E1's sensor gateway, where local edge processing may be applied. The sensor gateways are connected to the Internet via wireless cellular 4G LTE modems and may be firewalled and have private Internet IP addresses. The EdgeVPN.io (Evio) overlay provides connectivity across all gateway/edge/cloud resources through Software-Defined Network (SDN) switches and overlay links. Event-driven actions trigger on-demand execution of FLARE FaaS containers at cloud resources, orchestrated by OpenWhisk; FaaS containers access cloud storage data via Git (for time-series observation data) and S3 (holding state in S3 buckets/objects for FaaS functions). Deployment, monitoring, and management are performed across the Evio network.

- We report on key aspects of the performance of FLARE based on a realistic end-to-end deployment with sensors and edge computing gateways deployed on-site at Falling Creek Reservoir, virtualized resources deployed on the XSEDE Jetstream [24], [29] academic cloud, as well as FaaS execution on the commercial IBM Cloud Functions [37].

II. Related Work

FLARE is related to systems including DataTurbine [28], an open-source streaming data middleware for real-time data stream processing and visualization for environmental observing systems. It is also related to the work by Watson et al. [32], which presents an Internet of Things (IoT) CI for environmental sensing as a part of the Jefferson Project at Lake George. Simmhan et al. [23] design and deploy an IoT software platform and calibration models that provide a precise prediction of the scientific variables from the low-accuracy raw sensor signals for air quality monitoring. Korambath et al. [19] introduce an edge-cloud architecture called Streaming Workflows to collect data from sensors connected to edge devices and pass on computation to popular on-demand cloud services. Ramamurthy [22] introduces the Unidata CI facility, which uses cloud computing for accessing, analyzing, and visualizing geosciences data to develop data-driven scientific workflows. Gutierrez-Polo et al. [14] describe the Great Lakes Monitoring CI as an integrated resource for hydrologic scientists. This system cleans and organizes the data collected from different sources into a semi-standardized schema, stores them in a database, and provides a data visualization user interface. Huang et al. [18] introduce EcoPAD, a platform for automatic data transfer and processing from sensors to ecological forecasting in a web-based workflow. Compared to FLARE, EcoPAD focuses on the back-end cloud execution of forecasting models and does not extend end-to-end to deploy and manage containers on sensor gateways and edge devices.

Key differences in FLARE with respect to these systems include the event-driven model for end-to-end forecasting workflows, handling of sensor data from the edge to the cloud using versioning and file deltas for time-series data files, and the integration of stateless microservices and virtualization technologies in the form of containers and virtual networks.

With respect to its CI middleware, FLARE is related to workflow systems such as Pegasus [8], which allows users to define computational workflows for cloud execution. It is also related to efforts in applying serverless technologies in scientific computing [6] using a CI that allows researchers to concentrate on novel ideas rather than systems management. FuncX [5] is a federated FaaS framework that can be used in the development of serverless applications.

A key difference of the serverless computing approach taken in FLARE compared to the related work above is the ability to deploy containers on-demand in response to events, integrated with network virtualization that spans from edge to cloud.

While conceivably FLARE could use such platforms, we have opted for the use of OpenWhisk [42] due to its ability to support unmodified applications in containers and its availability in a commercial cloud [37]. Serverless computing significantly reduces the cost of provisioning ephemeral microservices in FaaS clouds, as opposed to long-running virtual machines in Infrastructure-as-a-Service (IaaS) clouds. The edge-to-cloud self-organizing VPN simplifies software/container deployment and provides network-layer authentication and privacy for data transfers and remote management.

III. Cyberinfrastructure

Despite the critical need for ecological forecasts, developing a CI that seamlessly integrates sensors, software, data repositories, and models to create daily forecasts remains a substantial challenge [10]. The individual pieces needed for ecological forecasting to be widely implemented for all lakes on the globe are already in place: Millions of sensor observations are being collected every minute and ecological models have successfully been developed and calibrated [15], [33]. Yet, there remains the substantial challenge of linking wireless sensor data to models for assimilating data and forecasting in near real-time [15], [17].

To address this challenge, we designed and developed a distributed CI to support FLARE. The CI leverages and integrates various open-source frameworks and builds on standards. The workflow starts with collecting sensor data with data loggers, retrieving and staging sensor data using Git on edge gateways, transferring the data to cloud storage, and the orchestrated execution of containers that implement data pre-processing, model execution, data assimilation, and visualization.

A. CI Resources

The major resources that make up the FLARE end-to-end CI are as follows:

- *Sensors:* Collect water quality data. Sensors may be deployed in the water body or watershed around the lake. Environmental sensors gather data from the environment and transfer them to the data loggers. The connection between the sensors and the data loggers can be either wired (both analog and digital) or wireless.
- *Data loggers:* Data loggers are responsible for reading and buffering the raw sensor data and transfer them to the edge gateways. The connection between the data logger and the edge gateway can be either wired or wireless. Data loggers popular in this industry are capable of sending the data to a locally-connected edge gateway via Wi-Fi or wired Ethernet [39].
- *Edge gateways:* Read and stage sensor data from the data logger and transfer them from the edge to the cloud. Edge gateways are field-hardened commodity computers capable of running Linux and middleware and connected to the Internet via a cellular modem.
- *Cloud storage nodes:* Store the data in cloud resources to drive forecasting, including observation data and state across FaaS invocations. In FLARE, the state consists of a storage object encapsulated in files produced by a function and consumed downstream. These nodes also store raw and/or processed sensor data pushed by the edge gateways.
- *Cloud Compute nodes:* Execute the serverless functions that implement data pre-processing, assimilation, forecasting model, and visualization. Cloud nodes are responsible for executing the forecasting engine for FLARE.

B. Containers and Microservices

Distributed computing is continuously expanding, spanning multiple clouds and IoT devices at the network's edge. Lightweight virtualization solutions, particularly containers, are increasingly being used as a foundation for deployment. They are faster to instantiate and have lower run-time overhead compared to virtual machines. This makes containers suitable for application packaging and orchestration spanning edge and cloud resources, end-to-end. In FLARE, Docker containers form the foundation for deploying tasks across the end-to-end CI.

The FLARE architecture is based on microservices, which can be defined as cohesive, independent processes interacting via messages [12]. In essence, a microservice encapsulates a function with well-defined inputs and outputs that can be retrieved from and stored back into a distributed data storage system. Microservices can thus be made "stateless": The microservice itself does not hold a state but rather a function that, when executed, transforms a state. They can be invoked on-demand and only operate as needed to perform a function, thereby reducing costs.

Docker containers have emerged as a widely-adopted open-source technology for microservice deployment. Containers host microservices that run in a logically-isolated environment while sharing a common physical server. Container-hosted microservices enhance the modularity and manageability of a distributed system. It happens through decomposing complex workflows into small, independent, stateless modules using well-defined protocols for communication and data storage and retrieval.

C. Versioned Storage

FLARE leverages a versioned storage system (Git) to handle reliable, efficient transfers of time-series data from the edge to the cloud.

Git is a distributed revision control system, widely and freely available. An important characteristic of Git is that it promotes file revisions to first-class citizens. While Git is typically used for collaborative software development, its applicability is not limited to source code development. It can be used in other environments where it is essential to track changes, manage provenance, and provide a robust mechanism to verify and commit updates as differences over the network. In FLARE, we use Git as a platform to manage the transfer, updates, and storage of time-series observation data.

One of the key motivations is that Git is an open, mature, widely-used, and well-understood protocol, offering both reliable data transfers and storage. Git has several open-source

implementations (e.g., GitLab) for private deployment, as well as free hosted services (e.g., GitHub) with which users in the ecology community are already familiar.

In FLARE, Git is particularly useful in managing the movement of data from the edge to the cloud. It provides a mechanism that allows a local repository to be maintained at the edge despite intermittent power and network connectivity, and also provides cryptography techniques for data assurance. Proper use of Git can ensure that no matter how many times an edge gateway gets restarted or a cellular link gets disconnected, when a transfer is eventually accomplished, it is reliably committed to storage.

In FLARE, time-series data retrieved by sensor gateways are text-based, such as a CSV (Comma-Separated Values) file where each row has a timestamp and columns with the various fields/observations read from sensors (a common practice in the ecology field). With Git, we can be assured that, although a file grows in size with appends, only the differences are transferred. This reduces network usage, which is essential when using cellular data plans, while keeping the process of buffering and sending data simple. By appending measurements to a file and using Git to transfer differences reliably, the gateway software can be kept simple and does not need to worry about corner cases such as failures and restarts in the middle of multi-file transfers.

D. Serverless Execution

The microservices used by FLARE are amenable to serverless execution in FaaS platforms. FLARE leverages Apache OpenWhisk as a platform for serverless computing.

Apache OpenWhisk is an open-source, distributed serverless platform. In OpenWhisk, the functional logic called Actions can be encapsulated in containers and dynamically run in return for related events via Triggers. That enables the user to run applications as event-driven workflows via a series of containers.

In FLARE, OpenWhisk is used as the cloud platform that drives the forecasting workflow. Actions encapsulate the various FLARE microservices for pre-processing, forecasting, and post-processing. Events provide a basis for invocation of the various functions to realize the end-to-end workflow. Triggers in FLARE are fired when data becomes available from external sources (e.g., observational data committed to a Git repository triggers a webhook), as well as events generated upon completion of a microservice leading to triggers downstream in the workflow.

A FLARE deployment may use a private OpenWhisk cluster deployed on a cloud. For instance, our cluster runs on VMs in the Jetstream academic cloud. In addition, it is possible to deploy in a hosted cloud service, such as IBM Cloud Functions [37], and pay only per service invocation, leading to a cost-effective FaaS model.

E. Object Storage

FLARE uses S3 object storage to hold state across OpenWhisk serverless invocations. The main functions that make the FLARE workflow operate on files as inputs and outputs, allowing FLARE to seamlessly support different modalities of operation, as described in Section IV. In serverless operation (Workflow Mode), FLARE aggregates inputs and outputs as file archives that are retrieved from/committed to S3 storage at the beginning/end of an action invocation. The files are named according to a convention, where each different lake has an S3 bucket, each container produces data inside a directory in the bucket, and files are tagged with a date identifier. The use of files allows FLARE to reuse S3-compatible implementations (e.g., the MinIO [36] server or Amazon's S3 service [45]). The S3 endpoint and access credentials are passed to the action via configuration.

F. Overlay Virtual Private Network

In order for FLARE to work end-to-end, all services must be able to communicate, regardless of their geographical location. This presents a major challenge, as end-to-end workflows may span resources distributed across multiple cellular, Internet, edge, and cloud computing providers. Furthermore, as data traverse the public Internet, it is crucial to protect privacy and integrity in communication to avoid data leakage and corruption by unauthorized users and provide strong authentication such that only authorized resources (e.g., drinking water managers at a water utility) may join the CI. FLARE nodes, such as edge gateways, often have private IP addresses and are behind firewalls and Network Address Translators (NATs), which add significant complexity to establish bi-directional communication.

To address connectivity and security requirements, FLARE uses Evio [25], [43] to connect all edge and cloud resources. The key benefit of the VPN abstraction is that it supports existing, unmodified software, allowing reuse of the key CI modules that run across the virtual cluster, including Docker, Git, OpenWhisk, and S3. It also enables remote management of the system. Evio enables the reuse of these frameworks without any modifications and uses standard public key cryptography techniques to encrypt data for privacy and integrity, as well as for authentication of nodes to join the end-to-end workflow.

G. Collaborative Development

The choices made during the development of the various CI modules in FLARE have been guided by close interdisciplinary collaboration (following [3]) and reflect a balance of capabilities and usability. One of the key outcomes of this approach is a modular design that can be deployed in different modalities (with different levels of complexity), depending on the use case.

The desire to support different use cases from the ecological forecasting community and achieve a system that is accessible to individual researchers and can scale to an automated operational development led to several design choices reflected in the system:

- Containers: The use of containers was perceived to be not only beneficial from the standpoint of cloud deployment,

but also as a mechanism to deliver complex pre-packaged software to facilitate deployment on end-users' personal computers.
- File abstraction: The use of a file abstraction to store and transfer time-series data allowed seamless integration with existing data loggers widely used in the community. It was deemed easy to understand and use by ecologists and allowed the CI to reuse widely-used storage services (Git, S3) without modifications.
- Git: The use of Git as a foundation for data transfers and storage (in combination with using a file abstraction) was also deemed easy to understand by ecologists who are familiar with platforms including GitHub. The ability to track versions and commit updates while keeping history is also proved to be valuable during manual data cleaning efforts.

IV. FLARE Application

This section elaborates on FLARE as an application that builds upon the CI modules described in the previous section.

A. FLARE Deployment Modalities

The use cases and resources available to a variety of potential FLARE users may vary considerably. In one extreme, FLARE can be used by an individual researcher developing and calibrating a model for a new lake using a local desktop. In another extreme, FLARE can be automated in operational forecast mode. Our design accounts for these different modalities and builds up progressive complexity: from core forecasting numerical packages, to containers encapsulating packages and associated scripts, to a workflow orchestrating container invocations and data movement.

The FLARE workflow (Figure 2) starts with the downloading of 16-day ensemble weather forecasts (NOAA Global Ensemble Forecasting System) for the specific lake/reservoir site. This numerical weather forecasting model output is retrieved from public NOAA servers. FLARE also downloads the observational sensor data for the lake/reservoir site, a process divided into two independent steps. First, the edge gateway commits sensor data to a Git repository, which generates a webhook event. The event triggers an action (container) that pulls data from the repository. Both datasets are processed to verify data completeness and prepare inputs for forecasting. FLARE then runs the General Lake Model from the previous date to the current day using the EnKF to assimilate new sensor observations and set initial conditions for a 16-day forecast, and sets a checkpoint for the next day's FLARE run. The forecast generation process needs to be executed once a day to generate a forecast for the next 16 days for any specific site. In the last step, the FLARE visualization container prepares a graphical output of the forecasts for visualization.

Currently, FLARE is implemented in the following modes:

- *Interactive Mode:* FLARE runs as an R package on a personal computer, and the steps above are initiated interactively by the end-user via R or RStudio.
- *Manual Mode:* FLARE runs as a set of containers invoked manually by the user via Docker.
- *Workflow Mode:* FLARE runs as a set of containers invoked automatically by Apache OpenWhisk when the data from the previous step is ready and verified.

1) Interactive Mode: The simplest deployment of FLARE uses functions in an R package that can be combined into a single R script and deployed in an interactive environment, such as RStudio. This environment is familiar to many ecologists, thus providing an on-ramp path to adoption with minimal barriers to entry. The package is structured such that modules, naming, and data conventions are predefined, and lake example templates are provided. End-users with a personal computer capable of running R or RStudio can use FLARE in interactive mode to customize their own lake forecast.

2) Manual Mode: While the interactive mode is well-suited for development and testing, the Manual Mode is intended to run a forecast manually (or triggered by a timer, e.g., a cron job) by the end-users. The FLARE package, all the dependencies, and helper scripts are encapsulated in containers. Users can expect to obtain exactly the same result on any platform or operating system capable of running Docker containers. In the Manual Mode, FLARE containers use a shared volume mounted from the host machine as the common space to share files and directories with other containers. This also gives the end-users easy access to modify configuration files as needed. In this way, data, output, and configurations from one container can be used as an output for another container through a local host directory. The FLARE modules are encapsulated into the following containers:

- *flare-download-noaa:* This container is responsible for downloading forecasts of meteorological data from NOAA, on a 6-hourly basis, for a specific geo-coordinate. The data is downloaded from NOMADS (NOAA Operational Model Archive and Distribution System) [40] using a custom downloader module with fault-tolerant capabilities to deal with data unavailability and timeouts (which, in our experience, are very common).
- *flare-process-noaa:* This container is responsible for processing meteorological forecast data from NOAA already downloaded by flare-download-noaa (i.e., spatial and temporal downscaling), and for preparing data for consumption by flare-generate-forecast.
- *flare-download-observations:* This container is responsible for downloading sensor data for a specific location required for running the forecast. The data is pulled from a Git repository that stores the time-series data pushed from edge gateways.
- *flare-process-observations:* This container is responsible for processing sensor data for a specific location required for running the forecast, and for preparing data for consumption by flare-generate-forecast.
- *flare-generate-forecast:* This container is responsible for running the forecast, including the lake model (e.g. GLM) and EnKF data assimilation.

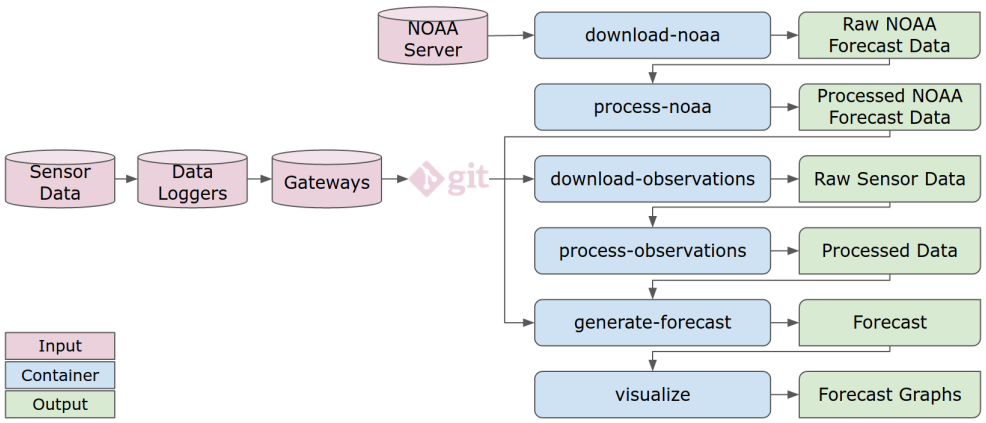

Fig. 2. Overview of FLARE Workflow. Each container reads from a YAML configuration file (not depicted) to configure lake-specific parameters.

- *flare-visualize:* This container is responsible for visualizing forecast output using figures generated by R. Figure 3 shows a snapshot of visualization of turnover and water temperature forecasts.

3) Workflow Mode: In this mode, instead of running modules manually or on a timer, event-driven execution is triggered by Apache OpenWhisk to run them with predefined rules. When a rule or a set of rules is satisfied (e.g., data is verified and available to a downstream container), it triggers the action to execute the corresponding container. Modules communicate through S3 cloud storage endpoints in the Workflow Mode rather than a shared volume on the host in the Manual Mode. The data and configuration needed for each module can be pulled from the cloud storage before running the module, and the output and configuration pushed to the cloud storage after the run is finished.

The Workflow Mode is configured as follows:

- The latest stable version of FLARE containers is uploaded to DockerHub. These containers are associated with actions in OpenWhisk.
- An S3 service is configured for storage. Each lake/reservoir is assigned a separate bucket, and each container is assigned a bucket within it. The buckets store YAML configuration files for each container and store files generated by the execution of that container.
- Actions are invoked with a JSON payload that contains the S3 endpoint and key, as well as OpenWhisk endpoint and key.
- Containers that produce data for downstream consumption push outputs to S3 storage and generate triggers for the next action after a step that verifies data completeness. A custom action is created to fire a trigger for flare-generate-forecast only after two data sources (NOAA forecasts and sensor observations) are complete.

V. Evaluations

A. Deployment

Meteorological data for Falling Creek Reservoir have been collected starting in 2015 using a research-grade meteorological station from Campbell Scientific at 1-minute intervals, with data stored on a CR3000 Micrologger from Campbell Scientific. In-water sensors have collected measurements starting from July 2018 at 10-minute intervals and the data are stored on a CR6 Campbell Scientific data logger. We also measure pressure and stream temperature for an inflow stream every 15 minutes and store them on a CR310 Campbell Scientific data logger. All observations are recorded in UTC-05:00 (Eastern Standard Time) without Daylight Saving Time changes in the data record. The types of sensors being used at Falling Creek Reservoir are: 1) meteorological sensors: wind monitor, rain gauge, temperature and relative humidity probe, barometer, quantum sensor, radiometer; 2) in-water sensors: thermistor, dissolved oxygen sensor, EXO2 multiparameter sonde with sensors for temperature, conductivity, dissolved oxygen, fluorescent dissolved organic matter, and total algae, and also pressure transducer. The gathered data for each data logger are accumulated in a single CSV file and pushed to their respective Git repository and branch a few times per day. AC-powered sensor gateways are up 24/7 and push their data every 6 hours. On the other hand, battery-powered sensor gateways are up during four 15-minute time windows each day for pushing the data via Git and remote maintenance, if required. The overall size of the data for the whole site is about 1 MB per day. The deployment currently uses GitHub as the storage service for sensor data.

B. Experiments

We designed and performed a set of experiments to evaluate the performance of different aspects of FLARE's CI, including overheads associated with the overlay virtual network, performance of data transfers, and container execution performance in both Manual Mode and Workflow Mode. The experiment environment includes FLARE v21.01.3 [7], Evio overlay v20.12.2 [26], and Apache OpenWhisk release 1.0.0 on Verizon Wireless 4G LTE network and Gigabit Ethernet.

C. Network Characteristics Test

We conducted a set of experiments to measure the overhead of running Evio with respect to round-trip latencies and

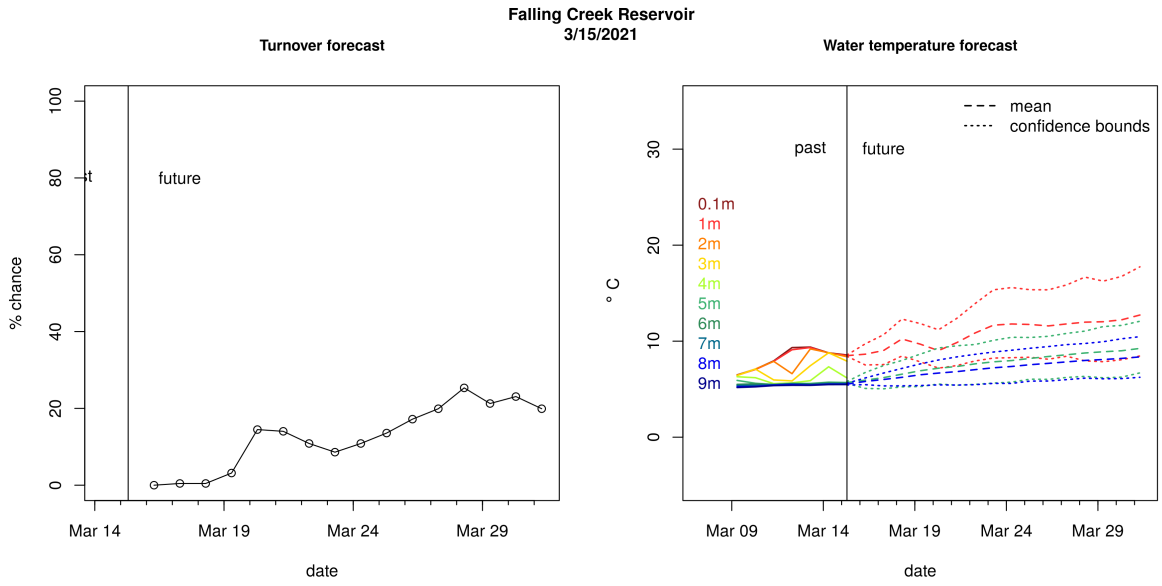

Fig. 3. Output of visualization container: turnover probability (left) and water temperature forecast at different depths for the next 16 days (right)

throughput, considering both LAN and cellular 4G LTE connections. We used 'ping' and 'iperf' tools between two nodes in 5-minute periods to compare four different scenarios when the nodes are connected by Ethernet or 4G LTE connection, with and without Evio overlay. The results (Table I) show that the overhead of using Evio is negligible on a 4G LTE connection; the virtual network adds a few milliseconds of RTT and can deliver up to 183Mbps. The standard deviation for all the experiments is less than 15%.

TABLE I
LATENCY AND THROUGHPUT BETWEEN NODES

	Eth w/o Evio	Eth w/ Evio	Wi-Fi w/o Evio	Wi-Fi w/ Evio	Cell w/o Evio	Cell w/ Evio
RTT (ms)	0.5	1.9	33.1	33.5	79.6	88.0
BW (Mb/s)	941	183	83.8	51.8	0.6	0.5

[a]Average round-trip time (RTT) and maximum bandwidth (BW) measured using ping and iperf

D. System Resource Usage Test

We designed a set of experiments to evaluate how much Evio utilizes system resources in edge devices. We used 'vnstat' and 'pidstat' tools to measure Evio system resource usage in 5 minutes on a CompuLab fitlet2 edge gateway device [38] connected with Ethernet or cellular network when the system is idle. The results shown in Table II indicate that the cost of running Evio overlay on system resources is slim. The standard deviation for all the experiments is less than 5%.

E. Git Transfer Test

To assess the effectiveness of using Git for edge-cloud data transfer size, we employed the 'nethogs' tool. We first transferred a sample 63 MB CSV data file storing cumulative time-series data up to day 0. Then, we appended the new observations for day 1 (0.2 MB), pushed the file to Git again, and compared it with the case that we push the whole day 1 data file (63.2 MB) from scratch to Git. The experiments are done in four different scenarios with LAN and cellular connection, with and without Evio overlay. The results (Table III) show the transfer size is drastically reduced when Git has to transfer just the differences instead of the whole new file. The standard deviation for all the experiments is less than 2%.

TABLE II
NETWORK, CPU, AND MEMORY UTILIZATION

	LAN w/ Evio Idle	Cell w/ Evio Idle
Bandwidth (Kb/s)	0.08	0.02
CPU (%)	0.12	0.20
Memory (MB)	83.6	84.2

[a]Average Evio bandwidth, CPU, and memory utilization measured using vnstat and pidstat

TABLE III
GIT TRANSFER SIZE

	LAN w/o Evio	LAN w/ Evio	Cell w/o Evio	Cell w/ Evio
Day0 All (MB)	17.9	18.5	20.1	20.6
Day1 Diffs (MB)	0.06	0.07	0.07	0.08
Day1 All (MB)	18.0	18.7	20.3	20.8

[a]Average Git transfer size measured using nethogs

F. Execution Time

In order to estimate how long it takes for each FLARE container to finish execution, in this set of experiments, we measured execution time for every FLARE container in both

Manual Mode and Workflow Mode. The 'generate-forecast' container included the generation of a 1-day simulation using observed meteorology collected by the sensor network, an invocation of the EnKF using water temperature observations from the sensor network, and a 16-day water temperature forecast using 221 ensemble members. For the Workflow Mode, we used an M1.medium virtual machine with 16 GB of memory and six cores of Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz on the XSEDE-Jetstream academic cloud. The results (Table IV) indicate that a whole daily run of ecological forecasts can be done in less than half an hour. The standard deviation for all the experiments is less than 10%.

The main difference in performance between the Manual Mode and Workflow Mode execution is due to differences in how data movement is handled. In the Manual Mode, the data is available locally and mounted as a volume, whereas in the Workflow Mode, data is retrieved and stored from the S3 object storage service. The latter introduces overheads in data copies. The execution time difference between the running modes is mainly due to the difference in these I/O methods.

In addition to experiments in an IaaS cloud resource, we have been able to successfully complete a run of the same FLARE containers from DockerHub on the IBM Cloud Functions free tier. With a 16-day temperature forecast using the GLM model and 221 ensemble members, the resource allocation for each FLARE container fits within the IBM Cloud Functions limits (10 minutes per invocation, 2 GB maximum memory). Based on IBM's pricing model as of this paper's writing ($0.000017 per second of execution, per GB of memory allocated), the estimated cost of running 16-day forecasts daily, for a month, would be less than $1.

TABLE IV
CONTAINERS EXECUTION TIME IN MANUAL AND WORKFLOW MODE

	process-noaa	**download-observations**	**process-observations**	**generate-forecast**	**visualize**
MN (s)	15.8	47.7	62.0	321.0	12.5
WF (s)	17.9	153.3	129.4	470.9	22.5

[a]Average execution time in Manual Mode (MN) and Workflow Mode (WF) on XSEDE-Jetstream

G. Running on Solar Panel Powered Batteries

There are many scenarios where data loggers and edge gateways in the field do not have access to the AC power supply. It is vital to run them on long-lasting batteries or ideally using a sustainable setup employing renewable energy such as a solar panel. We have been able to successfully deploy and operate FLARE edge nodes with such a setup, where sensors, data loggers, and edge gateway are powered by an 84 Ah 12 V rechargeable battery [35] powered by a 50 W solar panel [46].

In this setup, a timer switch is added to turn on the edge gateways in predefined 20-minute time windows during the day. The edge gateway, upon startup, automatically receives the new observations from the data logger, connects to the virtual network, and attempts to commit the differences appended to the observations file to the Git repository. If power fails or the network disconnects during a time window, the process is retried in the next available window.

Figure 4 shows the battery level during one week. The system works from battery power and is properly recharged on a daily basis. The voltage level does not drop below a safe operating range.

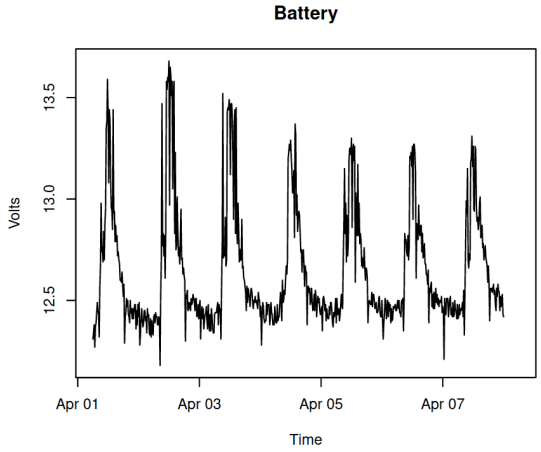

Fig. 4. Solar panel powered battery level over time

VI. DISCUSSION AND FUTURE WORK

A. Data Storage Services

FLARE uses two different approaches to networked data storage services, both with well-defined standard APIs. The motivation is to exploit the advantages of different storage approaches to suit different data needs.

Files that are retrieved from data loggers by sensor gateways tend to be text-based, such as CSV files, where each row is an observation with a timestamp and columns storing the measurements read from the sensors. The new observations are appended to the end of the sensor data file. Although the Git protocol does not have a file size limit, there is a 100 MB file size limit imposed by the GitHub provider. Our approach allows the choice of using either a private Git server or a public Git service such as GitHub for storing the sensor data. In the latter case, file rotation is needed to keep the size of the files within the size limit. In our Falling Creek Reservoir deployment, this rotation has been performed once a year.

Aside from the sensor data, the intermediate data and output generated by forecasting processes are generally neither incremental/append-only nor text-based. Hence, there are no clear benefits of using the Git protocol for storing and transferring them. For these files, the Simple Storage Services (S3) protocol is used. We have tested the system to utilize both private S3 servers (using MinIO containers) and hosted services (including OpenStack Swift [41] on Jetstream).

B. Generalization and Scaling

FLARE has been successfully implemented for ecological forecasting at Falling Creek Reservoir. The system has been

designed with modularity and extensibility in mind, and to scale to incorporate additional sensor types and multiple lakes. Our group is deploying FLARE across several new sites, including two additional reservoirs in Virginia (Carvins Cove and Beaverdam Reservoir), Lake Sunapee in New Hampshire, Lough Feeagh in the Republic of Ireland, as well as several lakes that are part of the U.S. National Science Foundation's National Ecological Observatory Network (NEON) [30]. FLARE supports running forecasts for all these sites by following a generalizable workflow and reusing the Docker containers, OpenWhisk triggers, and OpenWhisk actions that implement the workflow. Per-lake customization support is being added by providing the ability to pull site-specific scripts and configurations at run-time.

Another effort in generalization is supporting other lake models besides the General Lake Model. Ongoing work is considering the integration of ensembles consisting of multiple lake models by reading the model name from configuration and pulling and running the proper files and executables at run-time.

FLARE uses open-source software stacks and services that allow it to be deployed in a way that can be customized by a particular user. For instance, OpenWhisk, S3, and Git services may be deployed on public cloud provider resources (e.g. IBM Cloud, Amazon Web Services, or GitHub), as well as deployed in private infrastructure (e.g. a custom OpenWhisk deployment, MinIO S3 services, and private Git servers). The end-user has the flexibility to choose all-private, all-public, or mixed deployments depending on the requirements of privacy and cost, among other factors.

We plan to investigate the use of Kubernetes [44] to orchestrate the deployment of OpenWhisk, storage services (S3, Git), and edge containers across the end-to-end CI. Currently, containers are deployed via SSH and Ansible through a management node. We also plan to investigate the integration of sensors and data loggers connected point-to-point to edge gateways via LoRa [47] radios for devices in remote locations without cell connectivity and expanding the Evio network to reach LoRa endpoints.

VII. Conclusions

In this paper, we propose a novel end-to-end architecture that builds on serverless computing and virtualization for customizable, flexible ecological forecasting workflows. The resulting system has been co-designed and has evolved iteratively through a close interdisciplinary collaboration and lessons learned from a deployment spanning over more than two years in a drinking water reservoir (Falling Creek Reservoir). A set of experiments conducted on local computers, academic cloud, and commercial cloud quantitatively shows that the overhead of using an edge-to-cloud virtual network (Evio) to facilitate and secure deployment, management, and data transfer is acceptable. It also shows that using Git structure to transfer Git differences instead of whole files substantially reduces the data transfer size, in addition to providing a platform that is well-understood by ecologists, supports versioning, and reliable transfers in the event of failures. The experiments also show that the approach supports deployment not only in a private/academic cloud but also in a public cloud, with significant cost savings due to the serverless execution.

Acknowledgment

We would like to express our gratitude to the Falling Creek Reservoir Group field crew for keeping sensors and gateways working in the field, especially Bethany Bookout, Nick Hammond, Alexandria Hounshell, Dexter Howard, Abigail Lewis, Mary Lofton, Ryan McClure, Heather Wander, and Whitney Woelmer. CIBR team members (Kathleen Weathers, Bethel Steele, Tadhg Moore) provided useful feedback. We are grateful to the Western Virginia Water Authority for their support and access to field sites.

References

[1] P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction," Nature, vol. 525, no. 7567, pp. 47–55, 2015, doi: https://doi.org/10.1038/nature14956.

[2] J. D. Brookes, C. C. Carey, D. P. Hamilton, L. Ho, L. van der Linden, R. Renner, and A. Rigosi, "Emerging Challenges for the Drinking Water Industry," Environmental Science & Technology, vol. 48, no. 4, pp. 2099–2101, 2014, doi: https://doi.org/10.1021/es405606t.

[3] C. C. Carey, N. K. Ward, K. J. Farrell, M. E. Lofton, A. I. Krinos, R. P. McClure, K. C. Subratie, R. J. Figueiredo, J. P. Doubek, P. C. Hanson, P. Papadopoulos, and P. Arzberger, "Enhancing collaboration between ecologists and computer scientists: lessons learned and recommendations forward," Ecosphere, vol. 10, no. 5, 2019, doi: https://doi.org/10.1002/ecs2.2753.

[4] C. C. Carey, W. M. Woelmer, M. E. Lofton, R. J. Figueiredo, B. J. Bookout, R. S. Corrigan, V. Daneshmand, A. G. Hounshell, D. W. Howard, A. S. Lewis, R. P. McClure, H. L. Wander, N. K. Ward, and R. Q. Thomas, "Advancing lake and reservoir water quality management with near-term, iterative ecological forecasting," Inland Waters, pp. 1–14, 2021, doi: https://doi.org/10.1080/20442041.2020.1816421.

[5] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "funcX: A Federated Function Serving Fabric for Science," Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 2020, doi: https://doi.org/10.1145/3369583.3392683.

[6] K. Chard and I. Foster, "Serverless Science for Simple, Scalable, and Shareable Scholarship," 2019 15th International Conference on eScience (eScience), 2019, doi: https://doi.org/10.1109/escience.2019.00056.

[7] V. Daneshmand, Y. Jin, Y. Ku, and R. Figueiredo, "FLARE-forecast/FLARE-containers: 21.01.3," doi: https://doi.org/10.5281/zenodo.4695036.

[8] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," Future Generation Computer Systems, vol. 46, pp. 17–35, 2015, doi: https://doi.org/10.1016/j.future.2014.10.008.

[9] C. A. Dieter, Estimated use of water in the United States in 2015. Reston, VA, USA: U.S. Department of the Interior, U.S. Geological Survey, 2018.

[10] M. Dietze, "Ecological Forecasting". Princeton, NJ: Princeton University Press, 2017.

[11] M. C. Dietze, A. Fox, L. M. Beck-Johnson, J. L. Betancourt, M. B. Hooten, C. S. Jarnevich, T. H. Keitt, M. A. Kenney, C. M. Laney, L. G. Larsen, H. W. Loescher, C. K. Lunch, B. C. Pijanowski, J. T. Randerson, E. K. Read, A. T. Tredennick, R. Vargas, K. C. Weathers, and E. P. White, "Iterative near-term ecological forecasting: Needs, opportunities, and challenges," Proceedings of the National Academy of Sciences, vol. 115, no. 7, pp. 1424–1432, 2018, doi: https://doi.org/10.1073/pnas.1710231115.

[12] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, Today, and Tomorrow," Present and Ulterior Software Engineering, pp. 195–216, 2017, doi: https://doi.org/10.1007/978-3-319-67425-4_12.

[13] G. Evensen, "The Ensemble Kalman Filter: theoretical formulation and practical implementation," Ocean Dynamics, vol. 53, no. 4, pp. 343–367, 2003, doi: https://doi.org/10.1007/s10236-003-0036-9.

[14] I. Gutierrez-Polo, Y. Zhao, S. Bradley, E. Roeder, M. Pitcel, K. TePas, P. Collingsworth, and L. Marini, "Monitoring Water Quality in the Great Lakes Leveraging Geo-Temporal Cyberinfrastructure," 2017 IEEE 13th International Conference on e-Science (e-Science), 2017, doi: https://doi.org/10.1109/escience.2017.50.

[15] D. Hamilton, C. Carey, L. Arvola, P. Arzberger, C. Brewer, J. Cole, E. Gaiser, P. Hanson, B. Ibelings, E. Jennings, T. Kratz, F.-P. Lin, C. McBride, D. de Motta Marques, K. Muraoka, A. Nishri, B. Qin, J. Read, K. Rose, E. Ryder, K. Weathers, G. Zhu, D. Trolle, and J. Brookes, "A Global Lake Ecological Observatory Network (GLEON) for synthesising high–frequency sensor data for validation of deterministic ecological models," Inland Waters, vol. 5, no. 1, pp. 49–56, 2015, doi: https://doi.org/10.5268/iw-5.1.566.

[16] M. R. Hipsey, L. C. Bruce, C. Boon, B. Busch, C. C. Carey, D. P. Hamilton, P. C. Hanson, J. S. Read, E. de Sousa, M. Weber, and L. A. Winslow, "A General Lake Model (GLM 3.0) for linking with high-frequency sensor data from the Global Lake Ecological Observatory Network (GLEON)," Geoscientific Model Development, vol. 12, no. 1, pp. 473–523, 2019, doi: https://doi.org/10.5194/gmd-12-473-2019.

[17] M. R. Hipsey, D. P. Hamilton, P. C. Hanson, C. C. Carey, J. Z. Coletti, J. S. Read, B. W. Ibelings, F. J. Valesini, and J. D. Brookes, "Predicting the resilience and recovery of aquatic systems: A framework for model evolution within environmental observatories," Water Resources Research, vol. 51, no. 9, pp. 7023–7043, 2015, doi: https://doi.org/10.1002/2015wr017175.

[18] Y. Huang, M. Stacy, J. Jiang, N. Sundi, S. Ma, V. Saruta, C. G. Jung, Z. Shi, J. Xia, P. J. Hanson, D. Ricciuto, and Y. Luo, "Realized ecological forecast through an interactive Ecological Platform for Assimilating Data (EcoPAD, v1.0) into models," Geoscientific Model Development, vol. 12, no. 3, pp. 1119–1137, Mar. 2019, doi: https://doi.org/10.5194/gmd-12-1119-2019.

[19] P. Korambath, H. Malkani, and J. Davis, "Streaming Workflows on Edge Devices to Process Sensor Data on a Smart Manufacturing Platform," 2019 15th International Conference on eScience (eScience), 2019, doi: https://doi.org/10.1109/escience.2019.00088.

[20] J. Loperfido, "Surface water quality in streams and rivers: introduction, scaling, and climate change: Chapter 5," in Comprehensive Water Quality and Purification, 1st ed. Calabash, NC, USA: Elsevier, 2013, ch. 5, pp. 87-105.

[21] C. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures – A Technology Review," 2015 3rd International Conference on Future Internet of Things and Cloud, 2015, doi: https://doi.org/10.1109/ficloud.2015.35.

[22] M. Ramamurthy, "Toward a Cloud Ecosystem for Modeling as a Service," 2018 IEEE 14th International Conference on e-Science (e-Science), 2018, doi: https://doi.org/10.1109/escience.2018.00046.

[23] Y. Simmhan, S. Nair, S. Monga, R. Sahu, K. Dixit, R. Sutaria, B. Mishra, A. Sharma, A. SVR, M. Hegde, R. Zele, and S. N. Tripathi, "SATVAM: Toward an IoT Cyber-Infrastructure for Low-Cost Urban Air Quality Monitoring," 2019 15th International Conference on eScience (eScience), 2019, doi: https://doi.org/10.1109/escience.2019.00014.

[24] C. A. Stewart, G. Turner, M. Vaughn, N. I. Gaffney, T. M. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, and S. Tuecke, "Jetstream: a self-provisioned, scalable science and engineering cloud environment," Proceedings of the 2015 XSEDE Conference on Scientific Advancements Enabled by Enhanced Cyberinfrastructure - XSEDE '15, 2015, doi: https://doi.org/10.1145/2792745.2792774.

[25] K. Subratie, S. Aditya, V. Daneshmand, K. Ichikawa, and R. Figueiredo, "On the Design and Implementation of IP-over-P2P Overlay Virtual Private Networks," IEICE Transactions on Communications, vol. E103.B, no. 1, pp. 2–10, 2020, doi: https://doi.org/10.1587/transcom.2019cpi0001.

[26] K. Subratie, R. Ganesh, P. Nagaraj, and S. Aditya, "EdgeVPN.io Maintenance Release version 20.12.2," doi: https://doi.org/10.5281/zenodo.4579871.

[27] R. Q. Thomas, R. J. Figueiredo, V. Daneshmand, B. J. Bookout, L. K. Puckett, and C. C. Carey, "A Near-Term Iterative Forecasting System Successfully Predicts Reservoir Hydrodynamics and Partitions Uncertainty in Real Time," Water Resources Research, vol. 56, no. 11, 2020, doi: https://doi.org/10.1029/2019wr026138.

[28] S. Tilak, P. Hubbard, M. Miller, and T. Fountain, "The Ring Buffer Network Bus (RBNB) DataTurbine Streaming Data Middleware for Environmental Observing Systems," Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007), 2007, doi: https://doi.org/10.1109/e-science.2007.73.

[29] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr, "XSEDE: Accelerating Scientific Discovery," Computing in Science & Engineering, vol. 16, no. 5, pp. 62–74, 2014, doi: https://doi.org/10.1109/mcse.2014.80.

[30] R. M. Utz, M. R. Fitzgerald, K. J. Goodman, S. M. Parker, H. Powell, and C. L. Roehm, "The National Ecological Observatory Network: An Observatory Poised to Expand Spatiotemporal Scales of Inquiry in Aquatic and Fisheries Science," Fisheries, vol. 38, no. 1, pp. 26–35, 2013, doi: https://doi.org/10.1080/03632415.2013.748551.

[31] C. Verpoorter, T. Kutser, D. A. Seekell, and L. J. Tranvik, "A global inventory of lakes based on high-resolution satellite imagery," Geophysical Research Letters, vol. 41, no. 18, pp. 6396–6402, 2014, doi: https://doi.org/10.1002/2014gl060641.

[32] C. Watson, E. M. Dow, A. P. Praino, M. Kelly, V. W. Moriarty, G. Auger, and L. A. Treinish, "The Application of an Internet of Things Cyber-Infrastructure for the Study of Ecology of Lake George in the Jefferson Project," AGU Fall Meeting Abstracts, vol. 2018, pp. H53P-1814, 2018.

[33] K. C. Weathers, P. C. Hanson, P. Arzberger, J. Brentrup, J. Brookes, C. C. Carey, E. Gaiser, E. Gaiser, D. P. Hamilton, G. S. Hong, B. Ibelings, V. Istvánovics, E. Jennings, B. Kim, T. Kratz, F.-P. Lin, K. Muraoka, C. O'Reilly, K. C. Rose, E. Ryder, and G. Zhu, "The Global Lake Ecological Observatory Network (GLEON): The Evolution OF Grassroots Network Science," Limnology and Oceanography Bulletin, vol. 22, no. 3, pp. 71–73, 2013, doi: https://doi.org/10.1002/lob.201322371.

[34] R. G. Wetzel, Limnology: lake and river ecosystems. San Diego etc.: Academic Press, an imprint of Elsevier, 2015.

[35] "BP84 - 84 Ah 12 V Sealed Rechargeable Battery," BP84: 84 Ah 12 V Sealed Rechargeable Battery. [Online]. Available: https://www.campbellsci.com/bp84. [Accessed: 15-Apr-2021].

[36] I. MinIO, "High Performance, Kubernetes Native Object Storage," MinIO. [Online]. Available: https://min.io/. [Accessed: 12-Jul-2021].

[37] "IBM Cloud Functions - Overview," IBM. [Online]. Available: https://www.ibm.com/cloud/functions. [Accessed: 15-Apr-2021].

[38] "fitlet2," fit IoT. [Online]. Available: https://fit-iot.com/web/products/fitlet2/. [Accessed: 15-Apr-2021].

[39] "FTP Streaming," Campbell Scientific. [Online]. Available: https://s.campbellsci.com/documents/us/technical-papers/ftp-streaming.pdf. [Accessed: 15-Apr-2021].

[40] N. O. A. A. N. W. S. N. C. E. P. N. C. E. P. C. Operations, "NOAA Operational Model Archive and Distribution System," NOMADS, 01-Jan-2009. [Online]. Available: https://nomads.ncep.noaa.gov/. [Accessed: 15-Apr-2021].

[41] "Open Source Cloud Computing Platform Software," OpenStack. [Online]. Available: https://www.openstack.org/software/releases/wallaby/components/swift. [Accessed: 07-Jul-2021].

[42] "Open Source Serverless Cloud Platform," Apache OpenWhisk is a serverless, open source cloud platform. [Online]. Available: http://openwhisk.apache.org/. [Accessed: 15-Apr-2021].

[43] "Open-source VPN for Edge Computing," EdgeVPN.io. [Online]. Available: https://edgevpn.io/. [Accessed: 15-Apr-2021].

[44] "Production-Grade Container Orchestration," Kubernetes. [Online]. Available: https://kubernetes.io/. [Accessed: 25-Jul-2021].

[45] "S3," Amazon, 2002. [Online]. Available: https://aws.amazon.com/s3/. [Accessed: 12-Jul-2021].

[46] "SP20 - 20 W Solar Panel," SP20: 20 W Solar Panel. [Online]. Available: https://www.campbellsci.com/sp20. [Accessed: 15-Apr-2021].

[47] "What Is LoRa®?," Semtech. [Online]. Available: https://www.semtech.com/lora. [Accessed: 25-Jul-2021].

[48] WFLA 8 On Your Side Staff, "Hacker altered chemicals in Oldsmar water supply to 'damaging' levels, sheriff says," WFLA, 09-Feb-2021. [Online]. Available: https://www.wfla.com/news/local-news/hacker-caught-altering-chemicals-in-oldsmar-water-supply-to-damaging-levels. [Accessed: 15-Apr-2021].