DESIGN OF A SERVERLESS OGC WPS BASED GEOPROCESSING SERVICE SOLUTION

M.E. Pakdil[1], R.N. Çelik[1]

[1]Istanbul Technical University, Istanbul, Turkey – (pakdilme, celikn)@itu.edu.tr

KEY WORDS: Cloud Computing, Geographic Information Systems, Serverless Architecture, OGC WPS, Geoprocessing.

ABSTRACT:

Geospatial data and related technologies have increasingly become a crucial part of big data analysis processes and even a prominent player in most of them. Serverless architectures have become today's trending and widely used technology within the cloud computing paradigm. In this paper, we review the serverless paradigm advantages over traditional cloud architecture models and infrastructures. Moreover, we examined the deployment of Open Geospatial Consortium (OGC) Web Processing Service (WPS) specification based geoprocessing Application Programming Interface (API) with serverless architecture. In this context, we propose a system design and review it in detail together with the results discussed along with use cases.

1. INTRODUCTION

The recent developments in cloud computing have led to the achievements of essential milestones in science and technology. This was primarily because highly scalable processing infrastructures are offered at low cost, allowing researchers to access high computing power readily and quickly. Today, it is almost impossible to think of big data scenarios without cloud computing. Researchers now have the opportunity of testing their algorithms on cloud computing platforms by using their ready and tailorable computing infrastructures so that they can efficiently complete their works without considerable investments in infrastructure. They have contributed to the development of science and technology and played a significant role in the emergence and evolution of geospatial informatics.

Geospatial data and related technologies have increasingly become a crucial part of big data analysis processes and even a prominent player in most of them. The most challenging requirement in these analysis processes is the setup and management of fulfilling computing infrastructure that can handle complex and resource-consuming algorithms. Thanks to cloud computing, infrastructure deployment and management have become much easy, even out of requirement. Therefore, in today's world, geospatial informatics is more frequently mentioned together with cloud technologies. Although a geospatial project is attempted to be designed and implemented on cloud computing infrastructure, lack of skilled human resources and cost issues could be blocker and deal-breaker items in project resource planning. The employment of skilled human resources for cloud computing technologies is considered an obstacle by companies eager to use this technology due to the increasing demand for these skills in the market. Besides, hiring a person with skills in cloud computing and geospatial technologies poses extra difficulty.

Technical challenges must be carefully evaluated as a cloud-based geographical information system is designed and planned. These challenges are directly proportional to the volume of data and the number of users. Although cloud infrastructures can promise limitless physical resources and scalable platforms, project budgets could limit their usage in practice. Moreover, even though many cloud providers offer their services based on a pay-as-you-go pricing model, project owners may face unexpected bills unless resources are correctly configured. It can cause colossal resource consumption mainly because of misconfigured scaling parameters. (Berbotta et al., 2020).

Serverless architectures have become today's trending and widely used technology within the cloud computing paradigm. The major advantage of the serverless approach over others is its high scalability and higher-level abstraction in infrastructure management. Thus, it provides a computing platform with fewer technical challenges and maintenance requirements. Serverless computing services can be examined in two categories as following;

- Function as a Service (FaaS)
- Container as a Service (CaaS)

(Schachar 2019, Chowhan 2018).

These service types are specifically designed to run deployed applications with abstracted infrastructure management delegated to cloud providers. Therefore, skilled human resources needed for both solving technical challenges in complex deployment environments and infrastructure management are reduced a lot. With the use of serverless technology in geographical information system applications, geospatial professionals with a lack of cloud computing competency may have the opportunity to get acquainted with cloud technologies.

FaaS applications are event-based. Events can be fired from different sources and devices, usually under the internet of things (IoT) umbrella. Nowadays, various smart sensors are widely used as a vital player in many smart city ecosystems. It is essential for decision-makers to process the data produced by these sensors based on environmental events and activities in near real-time. For this reason, the infrastructure of the geographical information system to process the data from these sensors must be very responsive and easily scalable to handle massive data flow. In this way, the data can be stored and analysed in the fastest way (Evans et al. 2018). To give a comparative example, consider a smart city scenario based on an IaaS or PaaS cloud architecture; the system's success will depend on the technical competence of human resources responsible for the management of infrastructure. On the other hand, in the same scenario based on serverless architecture, the success will depend only on the software architecture design (Van et al. 2020). Thus, system

sustainability can be prolonged with proper optimisation and maintenance of the application.

Even if, at first glance, smart city systems based on serverless architectures cannot be run without the infrastructure provided by public cloud providers, they can also run on on-premise infrastructures in the same way. On-premise serverless infrastructures make it possible to deploy serverless GIS architectures for projects where cloud computing is not an option because of limitations by local data regulations. Thus, resource optimisation, one of the primary virtues of serverless architectures, has also become applicable to on-premise systems. Nevertheless, skilled human resources requirements could remain a trade-off in on-premise deployments that need the management of on-premise infrastructures.

This study proposes a system design to run Open Geospatial Consortium (OGC) Web Processing Service (WPS) based web service to execute complex and long-running geoprocessing tasks in a serverless architecture. The proposed system design is evaluated on a selected cloud provider, and the results are reviewed. This system design could prove that geoprocessing workflows can deploy to serverless platforms using a well-known industry standard to eliminate interoperability issues.

2. SYSTEM DESIGN

The proposed system is designed based on the serverless components so that the system's computing components are run when called and terminated when the execution is done (Figure 1). The WPS API follows the OGC WPS standard's definitions and runs asynchronously. The API executes the requested operation on serverless containers. Task statuses are stored in a serverless database and updated based on container and execution states. The design is implemented with AWS serverless services, and each service name is depicted together with the component name.

In this design, we aim to accomplish the following goals;

1. The design should not consume any computational resources while the system is idle.
2. The API should work with an industry standard to provide interoperability and avoid a steep learning curve for those who want to consume the API.
3. The API must be asynchronously run to be deployed to stateless service models such as FaaS.
4. The design should be able to deploy in both on-premises and public cloud infrastructures.
5. The system must be extensible with different technologies and new task definitions without changing core components.
6. The design should be testable and runnable in the developer's machine so that it can be developed.
7. The system should be fault-tolerant. Application or tasks errors should be logged and monitored.

Two different roles are defined in the design; these are `user` and `task developer`.

- The user can be a person who knows how to use desktop or web GIS applications. These end-user applications must support the OGC WPS standard.
- The task developer adds new Docker container images that contain code to run a task. The task developer should understand the basics of Docker image development and deployment. There is no limitation to using a programming language so that the developer has the freedom to use any programming language for the task development.

In the diagram, we divide the system design into three layers called "OGC WPS Service", "Task Execution Service", "Task Repositories". In the following three subsections, we will explain these layers in detail.

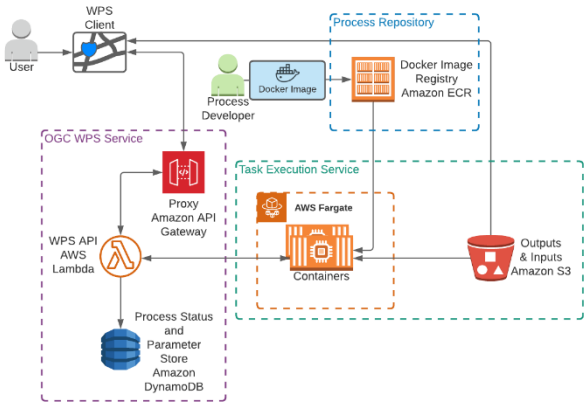

Figure 1. Serverless OGC WPS based geoprocessing system architecture design

2.1 OGC WPS Service

The WPS API is the core component of the whole system. A full-fledged API handles HTTPS (Hypertext Transfer Protocol Secure) requests from an API client and a running container. It runs asynchronously and stateless. Thus, it does not support session-based authentications or data, and each request must be authenticated in every request.

The API endpoints are categorised as public and private endpoints. Private endpoints are only available to containers, and containers may use these internal endpoints to update progress, status, and estimated completion time of the running task. In addition, public endpoints are permission trimmed by role. User and developer roles are defined in hardcoded ACL (access-control list) that grants permissions. Process management endpoints are restricted to developers. The API obeys REST (Representational state transfer) design constraints except for "/wps" endpoint because it must be OGC WPS 2.0 compliant (Mueller and Pross, 2015).

Endpoint	Request Method	Description	Role
/wps	POST and GET	WPS operations	User
/outputs/{task-id}/{filename}	GET	Download produced output file	User
/process/	POST	Add new process definition	Developer
/process/{process-name}	PUT	Update process definition	Developer
/process/{process-name}	DELETE	Delete process definition	Developer

Table 1. WPS API Public Endpoints

"/wps" endpoint follows OGC WPS 2.0 specification for both request and response models. On the other hand, process management endpoints, which start with "/process/" path,

consume and produce JSON (JavaScript Object Notation) payloads.

Endpoint	Request Method	Description
/status/{task-id}	PUT	Update status of running task
/inputs/{task-id}	GET	Download given inputs
/outputs/{task-id}/{filename}	POST	Upload produced output file

Table 2. WPS API Private Endpoints

All input and output endpoints consume and produce a binary payload for file management. These endpoints play a crucial role in making WPS API asynchronous. A running task can use these endpoints to store its outputs and read large input objects. Once the task execution is complete, temporary download links can be generated and passed as a reference value in the WPS execution response.

The API can be deployed to a FaaS service as a serverless function. Serverless functions, one of the serverless computing services, are utilised to run a deployed piece of code to execute a task for a purpose such as processing data from a sensor. This piece of code may be written in any programming language supported by the FaaS service. Functions can be triggered easily by any event that can be sourced from either another platform service or an external source (Malawski et al., 2020). Fundamentally, FaaS services are a kind of container orchestrator that runs deployed a piece of code within a short time on special containers that are mainly designed for the FaaS system (Figure 2). A typical FaaS platform loads the function code on demand. The first load is called a "cold start", and it usually takes milliseconds (Katzer, 2020). Then the platform immediately gives the function a request payload from the event. When processing completes, it terminates the running container. Subsequently, the platform may reuse the created container with a "warm start" to save time, but the function cannot rely on this case, and it must be designed stateless.

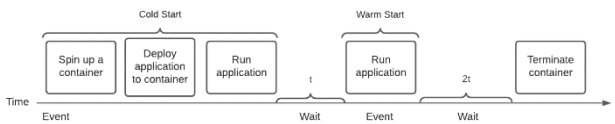

Figure 2. A life cycle of a FaaS execution

AWS offers a FaaS service called AWS Lambda. It supports major programming languages and various event sources, including HTTP events. Supporting HTTP traffic sourced events makes it possible to deploy a Web API application. AWS Lambda keeps regularly used functions warm for a limited time (40 - 60 minutes) (Cui, 2020). AWS provides a logging platform called CloudWatch that can integrate with AWS Lambda. It is helpful to monitor the system for possible errors. Alerts can also be created in CloudWatch to notify system administrators in case of any failure in near real-time.

The proxy service is to route HTTPS traffic to WPS API in a secure way. Most importantly, the proxy service cloud also provides to work with multiple functions so that we can split the API codebase into different functions based on optimisation or deployment needs.

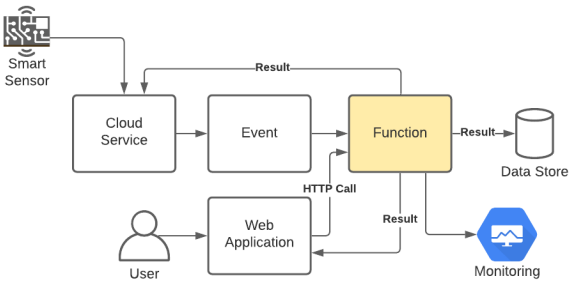

Figure 3. A function and its integrations with other resources

We leverage AWS API Gateway as a proxy service to transform HTTP requests into AWS Lamba events in our proposed design. The AWS API Gateway provides authentication and request throttling, but we do not add these extra protection layers to keep our design simple. Since the design is decoupled from these protection features, they can be added later in available services and libraries based on needs.

The WPS API functions store process statuses and parameters' data in a serverless NoSQL database. NoSQL database technology is selected as data storage because it is easier to find an available serverless service in public cloud providers over serverless relational SQL databases. Each asynchronous WPS request is stored in the data store to monitor the progress of the executed process. Thus, a WPS client can poll WPS API frequently to track progress.

The WPS specification describes three different data types for input and output parameters (Table 3).

Data Type	Description
Complex	It can be an extended ASCII or Binary (Base-64) data such as GML vector or Base-64 encoded raster data.
Literal	Simple inputs like buffer size or distance value
Bounding Box	Bounding Box definition in geographic coordinates

Table 3. WPS Data Types

The WPS API stores complex input and outputs to blob storage to reduce the load on the data store in our design. Literal and bounding box parameter values are stored in the data store. We will explain how processes utilise these values in the next section.

The data store consists of two tables that define two entity models; process and task (Figure 4). The process entity model is designed to handle process information comply with the WPS specification. Each process and task have a unique identifier. System components identify and distinguish all task-related objects based on these unique identifiers. When a developer adds a new process definition to the system, the process model will be populated with version, identifier, title, inputs, outputs, container image name and container image tag as mandatory fields. On a new task submission or a task event, the WPS API adds a new record to the task table or sets additional attributes denoting status information such as percentage, timing, and logged message.

Process	
Id	String
Version	String
Identifier	String
Title	String
Abstract	String
Keywords	String
Metadata	String (JSON)
Inputs	String (JSON)
Outputs	String (JSON)
ContainerImageName	String
ContainerImageTag	String

Task	
Id	String
ProcessId	String
Version	String
TimeStart	Number
TimeEnd	Number
ProcessIdentifier	String
Message	String
PercentDone	Number
Status	Number

Figure 4. Data models for task and process entities

Amazon DynamoDB is a NoSQL database provided by AWS as a key-value store. In the implementation, Amazon DynamoDB is chosen as a serverless data storage solution. Unlike relational databases, Amazon DynamoDB provides HTTP API endpoints to perform data operations on tables instead of SQL. For this reason, the WPS API communicates Amazon DynamoDB throughout the HTTP API wrapped by the DynamoDB library. Since it is a key-value store, we choose "Id" fields as keys in our table configurations.

2.2 Task Execution Service

The task execution service is the layer where all processes are executed in containers based on demand. The WPS API calls the container orchestration platform to create a new container to execute the process with given inputs. The container orchestration platform pulls the published image from the image registry.

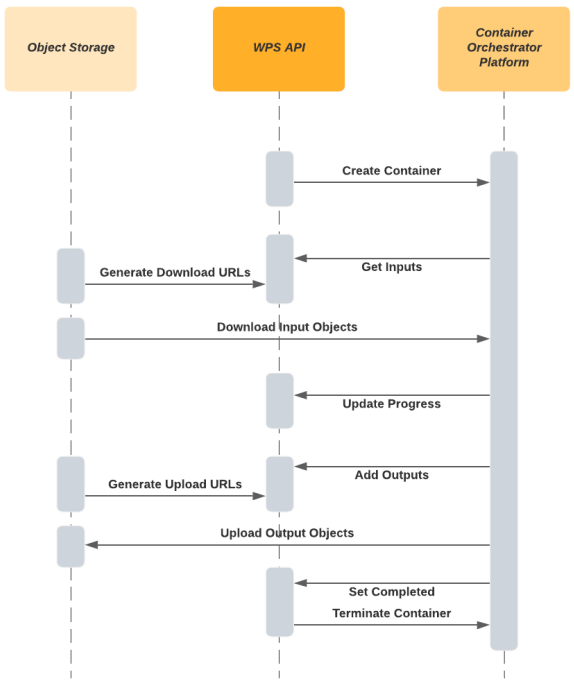

Figure 5. Sequence diagram for task execution service

The container and WPS API must communicate via private endpoints; thus, the container's network configuration should allow traffic between the API and the container without any restriction. In some cases, the process may need to access public geospatial APIs or external resources via an internet gateway. Therefore, internet access may be needed to allow for containers.

During the process execution, the container process updates the progress and forward outputs to the WPS API, so the user is acknowledged the status of the running or completed execution.

Amazon Fargate is an AWS service for the serverless container orchestration platform that uses the Docker framework. Docker is the de facto containerisation framework and has revolutionised the packaging and deployment of software. In our design, we use Amazon Fargate to run deployed container images. We choose Amazon Fargate because it allows running containers without managing an infrastructure to handle a high workload and long-running processes (Culkin et al., 2021). In addition, it consumes infrastructure resources only while containers are running.

The processes store output data to object storage as objects. Object storages manage binaries as objects instead of other storage architectures like file systems that manage data as a file hierarchy. Objects are partitioned by process and task id as follows; *"{process-id}/{task-id}/output-file-name"*. The WPS API interacts with the object storage to generate signed URLs to provide secure addresses for file uploads and downloads inside the container.

Amazon S3 is the serverless object storage service provided by AWS. The implantation of the proposed design uses Amazon S3 as object storage to store a WPS process' binary inputs and outputs. Moreover, it allows generating unique temporal URLs to access files from a container or a WPS client.

2.3 Process Repository

In this layer, the process developer builds and publishes custom container images to the container image registry and register the new process to the WPS API. The custom container image is inherited from a previously published base image. This base image contains necessary initialisation binaries to execute deployed code pieces as a forked process and forwards all standard inputs and outputs to the WPS API. This embedded binary also calls the WPS API to update progress and get the given inputs to download.

```
FROM [BASE IMAGE FOR PROCESS BINARY] as
function

COPY --from=function
/function runner/function runner
/usr/bin/function_runner
RUN chmod +x /usr/bin/function_runner

...REST OF THE IMPLEMENTATION...

ENV FUNCTION_COMMAND="[COMMAND TO BE FORKED]"

CMD ["function_runner"]
```

Figure 6. An example Dockerfile structure

The process developer builds an image based on the structure we propose (Figure 6). This custom structure declares a custom base docker image that injects the binary to fork deployed app as a child process.

3. USE CASES

In geospatial data processing applications as well as in smart city applications, there are various situations where serverless WPS implementation can be useful. Geospatial workflows can leverage WPS processes where intensive calculations are needed.

Geospatial big data applications can utilise cloud infrastructure to scale out the process by splitting it into small batches for parallel processing. Concurrent analyses are the most appropriate use case for a serverless method (Roberts and Chapin, 2017). For example, generation of contours from DEM (Digital Elevation Model) raster files are supplied from a public data server that resides in the cloud (Figure 7).

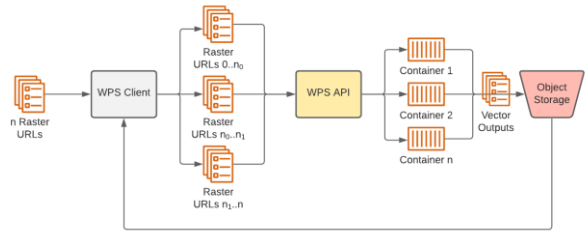

Figure 7. An example concurrent batch raster to vector conversation

Smart sensors or Internet of Things (IoT) devices are another use case for serverless architecture. IoT devices usually have little processing capacity and rely on external resources by delivering event signals, which are well-suited to cloud computing (Shekhar et al., 2012). They can deliver messages to WPS API and trigger processing execution. In high message traffic flow cases, the system can allocate more containers to handle requests. The stored process outputs can be directly used by another cloud serverless analytic service to generate valuable outcomes for decision-makers. Therefore, the data is processed and analysed in the cloud without transferring to another platform (Figure 8).

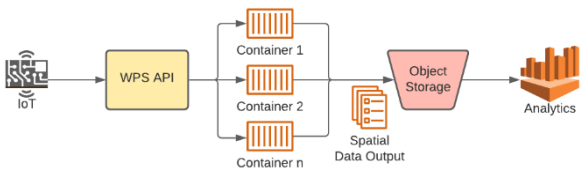

Figure 8. Data processing flow triggered by smart sensors

It is possible to add more use cases for scenarios requiring data processing in a performant and cost-effective way for geospatial data throughout an open industry standard; in our case, it is OGC WPS. We hope to see numerous use cases of serverless geospatial data processing in the smart city architectures near future.

4. RELATED WORKS

PyWPS is an open-source project that implements WPS specifications in Python (URL 1). PyWPS allows developers to serve Python processes using the WPS standard. Not to mention the fact that our API's WPS endpoints are implemented inspired by the PyWPS library.

De Sousa et al. introduce new features in PyWPS version 4, and they give some use cases that can be readily tested on our proposed design.

Baldini et al. review the advantages of serverless architectures and list limitations of the serverless platform. They emphasise that there has not been a commensurate level of interest among researchers for serverless technologies.

OGC released a new specification for geoprocessing service as a part of the OGC API standard. This new specification also supports JSON format.

Zhang et al. introduce a geoprocessing workflow that couples W3C PROV and OGC WPS. This new model allows users to define a workflow execution plan to run a set of geoprocessing tasks in order. Their implementation can be applied to our proposed architecture design and make it deployable on serverless infrastructures.

Read et al. develop a new library called "geoknife". The idea behind the library overlaps with our purpose. Briefly, the idea is to delegate geoprocessing operations to managed web servers and returned manageable datasets for subsequent local investigations. This approach can be very well achieved with our proposed design. The data can be processed in a cloud platform with public datasets also served in the same cloud platform. Low latency network calls between the geoprocessing and dataset locations can reduce execution durations.

5. CONCLUSIONS

In this paper, we reviewed our proposed design in detail. Our design is applied on OGC WPS specification; however, OGC has been working on a new geoprocessing specification called OGC API – Processes (URL 2). We choose the existing WPS specification over OGC API – Processes because the new standard is still in draft, and major GIS applications do not widely support it. We expect the new geoprocessing standard to be more suitable for sensors directly communicating with an OGC API service that allows JSON requests and responses. JSON is considerably smaller than XML, resulting in quicker transmission and processing.

We prefer CaaS over FaaS because CaaS provides fewer restrictions on configurations to deal with complex workloads. When simple calculations are needed to be executed, the FaaS model can also be preferred instead of the CaaS model in the design. Thus, we do not add anything into the design to couple tightly with CaaS infrastructure. When the FaaS is used, the developer role needs to be slightly changed, and it will be responsible for deploying code pieces that can be run on the FaaS platform instead of publishing container images.

Our use cases are designed as generic as possible to give a fundamental idea for readers to design specific systems based on the requirements.

We prefer to implement our design with Amazon Web Services as a public cloud provider because it is one of the most widely used cloud providers and provides serverless services that we need to realise our design. The design can also be deployed on Microsoft Azure or Google Cloud, and both provide similar serverless services for storage, data store, and container orchestration.

In conclusion, we discuss the serverless paradigm together with geospatial processing as locate it as a core component in systems that produce meaningful information from raw data that may be sourced from the environment and people. We believe that smart city systems should leverage serverless architectures to build highly optimised, real-time, highly scalable, fault-tolerant systems requiring minimum infrastructure management and advanced skills for geospatial professions.

REFERENCES

Baldini, I., Castro, P.C., Chang, K., Cheng, P., Fink, S.J., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017). Serverless Computing: Current Trends and Open Problems. Research Advances in Cloud Computing.

Bebortta, S., Das, S., Kandpal, M., Barik, R. K., Dubey H. (2020). Geospatial Serverless Computing: Architectures, Tools and Future Directions. ISPRS International Journal of Geo-Information. 9. 311. doi.org/10.3390/ijgi9050311.

Chowhan, K. (2018). Hands-On Serverless Computing. Pact Publishing

Culkin, J., Zazon, M., Ferguson, J. (2021). AWS Cookbook. O'Reilly Media, Inc.

Cui, Y. (2017). How long does AWS Lambda keep your idle functions around before a cold start? https://acloudguru.com/blog/engineering/how-long-does-aws-lambda-keep-your-idle-functions-around-before-a-cold-start (Accessed on: 25 July 2021)

de Sousa, L. M., de Jesus, J. M., Čepicky, J., Kralidis, A. T., Huard, D., Ehbrecht, C., Barreto, S., & Eberle, J. (2019). PyWPS: overview, new features in version 4 and existing implementations. Open Geospatial Data, Software and Standards, 4(1). doi.org/10.1186/s40965-019-0072-0

Evans, M. R., Oliver, D., Yang, K., Zhou, X., Ali, R. Y., & Shekhar, S. (2018). Enabling spatial big data via CyberGIS: challenges and opportunities. GeoJournal Library, 143–170. doi.org/10.1007/978-94-024-1531-5_8

Katzer, J. (2020). Learning Serverless. O'Reilly Media, Inc.

Malawski, M., Gajek, A., Zima, A., Balis, B., & Figiela, K. (2020). Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. Future Generation Computer Systems, 110, 502–514. doi.org/10.1016/j.future.2017.10.029

Mueller, M., Pross, B. (2015). OGC WPS 2.0.2 Interface Standard Corrigendum 2. Open Geospatial Consortium, http://docs.opengeospatial.org/is/14-065/14-065.html (Accessed on: 25 July 2021).

Read, J. S., Walker, J. I., Appling, A. P., Blodgett, D. L., Read, E. K., & Winslow, L. A. (2015). geoknife: reproducible web-processing of large gridded datasets. Ecography, 39(4), 354–360. doi.org/10.1111/ecog.01880

Roberts, M.; Chapin, J. (2017). What is Serverless? O'Reilly Media, Incorporated: Sebastopol, CA, USA.

Schachar, A. (2019). Serverless CaaS: Rethinking app infrastructure. https://spot.io/blog/serverless-caas-rethinking-app-infrastructure/ (Accessed on: 27 May 2021)

Shekhar, S., Gunturi, V., Evans, M. R., & Yang, K. (2012). Spatial big-data challenges intersecting mobility and cloud computing. Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access - MobiDE '12. the Eleventh ACM International Workshop. doi.org/10.1145/2258056.2258058

URL 1. https://pywps.org (Accessed on: 25 July 2021)

URL 2. https://ogcapi.ogc.org/processes/ (Accessed on: 25 July 2021)

Van Eyk, E., Toader, L., Talluri, S., Versluis, L., Uta, A., Iosup, A. (2018). Serverless is more: from PaaS to present cloud computing. IEEE Internet Computing, 22(5), 8–17. doi.org/10.1109/mic.2018.053681358

Zhang, M., Jiang, L., Zhao, J., Yue, P., & Zhang, X. (2020). Coupling OGC WPS and W3C PROV for provenance-aware geoprocessing workflows. Computers & Geosciences, 138, 104419. doi.org/10.1016/j.cageo.2020.104419