Subramanya, Rakshith; Räisänen, Paula; Sierla, Seppo; Vyatkin, Valeriy

Cloud Computing Design Patterns for MLOps

Published in: IECON 2023 - 49th Annual Conference of the IEEE Industrial Electronics Society

DOI: 10.1109/IECON51785.2023.10312212

Published: 01/01/2023

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.
Cloud computing design patterns for MLOps: applications to virtual power plants

Rakshith Subramanya
Department of Electrical Engineering and Automation
Aalto University
Espoo, Finland
rakshith.subramanya@aalto.fi

Paula Räisänen
Department of Electrical Engineering and Automation
Aalto University
Espoo, Finland
paula.c.raisanen@aalto.fi

Valeriy Vyatkin
Department of Electrical Engineering and Automation, Department of Computer Science, Electrical and Space Engineering
Aalto University, Luleå University of Technology
Espoo, Finland. Luleå, Sweden
valeriy.vyatkin@aalto.fi

Seppo Sierla
Department of Electrical Engineering and Automation
Aalto University
Espoo, Finland
seppo.sierla@aalto.fi

Abstract—Virtual Power Plants (VPPs) are a key factor in smart grids, and they use cloud computing to integrate and manage Distributed Energy Resources (DERs). VPPs use Machine Learning (ML) methods to optimize various tasks. Machine Learning Operations (MLOps) methodology is a set of techniques that targets to develop, deploy and maintain ML applications smoothly on production. Cloud design patterns (CDPs) are general reusable solutions for common cloud problems that can improve the reliability, scalability, and quality of cloud applications. This paper discusses how CDPs can help in building complex ML applications on cloud with MLOps practices which can help VPPs to optimize their workloads. The paper also provides an example implementation on a public cloud provider.

Keywords—cloud computing, design patterns, MLOps, machine learning, smart grid, virtual power plant

I. INTRODUCTION

Various Distributed Energy Resources (DERs) are used on different energy markets like Primary Frequency Reserves (PFRs) [1], [2] thanks to the development of smart grids and Virtual Power Plants (VPPs). Some of the DERs include, batteries, wind turbines, combined heat and power generators, photovoltaics, and smart loads. PFRs are a type of frequency reserve that typically consists of spinning reserves that consume fossil fuels and have a quick response time to frequency changes [3], [4]. In an effort to minimize carbon emissions, these reserves are currently being replaced by DERs. The primary task of a VPP is to manage DER. Since DERs capacity are small, they are unable to supply the minimal amount of controllable power needed to participate in frequency reserves [5]. In order to trade the resources on the frequency reserves markets, VPPs have been employed to aggregate them [6].

VPPs are a key factor in smart grids [7], and smart grid applications use cloud computing. Like smart grid applications, cloudification is essential in VPPs. Application scalability, flexible processing and storage capabilities, remote controlling and monitoring of smart grid applications are some of the reasons smart grid uses cloud computing [8]. Additionally, for a VPP, one of the main advantages of using cloud computing is the integration of DERs that are not co-located. With different cloud computing services, connecting and managing such DERs would be easy. Providing ancillary services to the grid is the primary task of a VPP, but VPPs are also used for other applications like, energy market participation with optimal bidding, energy sale at optimal price, and load shifting or shaving. To improve the reliability and efficiency of such VPP applications, different Machine Learning (ML) methods are used [9]. Forecasting such as power generation of DERs and market prices along with optimization of DER operation for maximizing the profits are some of the typical ML applications [10].

As VPPs and ML applications are expected to work within the cloud computing framework, hosting VPPs and ML applications on the existing Cloud services is a major milestone. VPP cloudification is difficult because power systems and distributed energy resources frequently employ interfaces and protocols from a prior era. Our previous research targets these issues and provides solutions for interfacing VPP with DERs on cloud [11]. Considering such VPP cloudification solutions, in this paper we target hosting a ML application on cloud for VPPs need.

DevOps emerged as a result of the challenges that Development and Operations teams had in achieving seamless software product rollouts. Stakeholders can now resolve challenging and complex business use cases, thanks to the recent investments in machine learning (ML) technologies. Various ML applications are now required to work in conjunction with DevOps enabled components like web applications [12]. MLOps techniques can be aimed at maintaining and deploying such ML code with high reliability and efficiency. Due to the advantages in terms of security, flexibility, reliability, and maintainability, most of the modern application hosting happens on cloud [13]. Complexity of the cloud deployment varies on the application parameters and its sub-systems. For building such complex applications on cloud with the said advantages, cloud design patterns (CDPs) are defined by different cloud vendors. CDPs are general reusable solution to commonly recurring problems in cloud eco systems. These design patterns can boost cloud architecture in terms of rapid development and support highly scalable applications. Also, CDPs are field-tested solutions which are easily repeatable, and hence can improve the reliability and quality of the system.

There is no generic, vendor neutral CDPs, which can be implemented out of the box. Microsoft Azure (MS Azure) is a
major cloud service provider, which defines 42 different CDPs across different categories. Some of these CDPs provide ready solutions for a VPP use case. For instance, by using MS Azure ‘Edge Workload Configuration pattern’ an edge component can be placed within each DERs, which can be used for configuring different workloads from VPP. Software implementation within the DevOps pipeline can be easily implemented using these CDPs and production rollout can be done. The same CDP principles could be used for MLOps pipelines also, ensuring the reliability and efficient working of VPP on cloud.

This paper makes three contributions.
1. Related work on the VPP & ML cloudification, along with different CDPs.
2. An overview of the CDPs relevant to the implementation of MLOps.
3. National electricity consumption forecasting application using MLOps framework, hosted on MS Azure by using different CDPs.

II. RELATED WORK

Various authors researched on the VPP cloudification. Rouzbahani et.al [14] defines VPP as a cloud-based platform that aggregate DERs. Zhenan [10] et.al use a cloud platform and ML to collect and process data from various energy sources and optimize the operation of the VPP. For a German power market, Candra et.al [15] configure VPP on a cloud computing platform for optimal implementation. When the VPPs are deployed on a cloud, they should support cloud-native principles. Cloud-native applications are designed for cloud that support scalability, flexibility and resilience [16], [17]. Not all the applications are cloud ready and to make them as cloud-native applications several authors propose different strategies. For instance, Cai et.al [18] use a reusable pattern-based transformation approach for migrating software applications on the cloud. Markoska et.al [19] develop an interoperable cloud environment with various classic software design patterns for connecting two or more cloud APIs. In similar terms, for data portability between two cloud databases, Shirazi et.al [20] propose design patterns. Di Martino et al. [21] map classic design patterns to cloud patterns for porting an application to cloud. Cloudification of ML applications along with VPP can be done via MLOps. Several authors have investigated on MLOps. John et.al [22] derive an MLOps framework that defines the ML model activities also outline different stages in MLOPs. Banerjee et al. [23] propose MLOps for hybrid clouds. Karamitos et al. [24] apply DevOps principles to ML applications.

Several authors have investigated CDPs. Malcher [25] explores the Circuit breaker pattern for a web application and use circuit breaker pattern to detect database connection. For designing, building and managing cloud applications, Fehling et.al [26] define and implement various cloud computing patterns. Cope et.al [27] provide a detailed set of cloud computing design patterns based on industry tools, technologies, products, and platforms. Considering the current increase in the cyber-attacks, cloud security is one of the most pressing issues for any cloud hosted application. Torkura et.al [28] use CDPs in the context of Security as a Service (SecaaS) to achieve things such as scalability. Some of the cloud vendors provide an alternative terminology to CDPs, for example, IBM cloud provides Code Patterns similar to CDPs [29]. Amazon Web Services (AWS) has nine set of CDPs defined by the community [30] whereas MS Azure has defined 42 CDPs [31].

III. METHODOLOGY

Reliable operation of a VPP with ML system is complex. For ML application, this complexity is not only in terms of ML algorithm implementation, but also with communication and collaboration between data scientists, conventional software developers and operational people. They all have to work together to increase the software quality. To simplify the overall development to deployment process and automate the ML application in large-scale production environments MLOps is needed. MLOps workflow is also beneficial as majority of the ML projects are experimental in nature and have more modular components than conventional software systems. ML systems are significantly more complex to build and most importantly to maintain them. MLOps helps organizations to overcome issues like system downtime and generate more commercial value. Because of the MLOps process, models can be aligned with business needs and regulatory requirements.

Thanks to the cost savings, security, and performance most of the web and mobile application infrastructure runs on cloud. ML applications usually part of such applications where DevOps is already implemented, and hence it makes sense to run ML applications on cloud infrastructure, with MLOps pipeline. A generic MLOps components include everything from developing a ML model till Deploying and maintaining it as discussed in our previous research [12]. The workflow components in MLOps mainly depends on the type of ML application like image classification or time series forecasting. To further explore and implement MLOps pipeline, this paper considers Finnish National Electricity Consumption Forecast as the ML application. Finnish Transmission System Operator (TSO) defines the electricity consumption, and it is formulated as follows:

\[ \text{Consumption} = \text{Production} + \text{Import} - \text{Export} \]  

The prime target in the Electricity Consumption forecast is predicting an hourly day-ahead electricity consumption. For getting the prediction, data collection, data processing, model training and prediction operations should be performed. The first step in the implementation of this ML application is to create the MLOPs pipelines. A typical MLOPs workflow components include the operations related to ML application in terms of Planning, Data operations, Model building, Testing, Software release, ML deployment, Running the ML application and Monitoring [12], [24]. Not all these operations are mandatory, but depending on the use case the MLOPs pipeline components can be selected.

Data collection was performed from the Finnish TSO and weather data was collected from the Finnish Meteorological Institute (FMI) websites. These data were collected via online REST (Representational State Transfer) API (Application Programming Interface). Once we collect the data from the two different sources, the next step would be to store the data on a database. The stored data is subjected to Exploratory Data Analysis (EDA). In the current context, performing EDA every time is important as it finds any trends and patterns in the data. Also, EDA can reveal any data issues like missing values, data drift, outliers, and presence of NULL values in the series. Once the data is analyzed and stored in the database,
the next step is to train the model. The structure of the ML model is same as discussed in [12].

ML model is trained every time with the new data and evaluated against evaluation data to assess the performance. The model performance is tracked continuously using Performance monitoring. The resulting trained model is then saved on a model registry. The resulting MLOps pipeline architecture with all the components are as shown in Figure 1.

Each of the pipeline components are implemented on a local computing resource. Once they run on an on-premise machine, then they are converted into modular components for the cloud deployment. Each of these modular components are created with Python language, and the scripts run sequentially as per the set schedule. The MLOps pipeline execution starts in the following sequence:

- Weather data and the data from Finnish TSO is collected via REST API.
  - After the data is collected, it is stored to a database instance.
- EDA is performed on this stored data.
- After the EDA, a Deep Learning Model is trained and evaluated.
- After the model is evaluated, the performance is tracked, and the trained model is saved in the model registry. The model then performs a day-ahead prediction and stores the predictions to the database along with CSV (Comma Separated Values) files to a file storage.
- Finally, a web application with REST framework exposes the forecasted values to the outside world. The REST API are usually interfaced to a cloud hosted VPP for market participation planning.

Python based libraries are used for all these operations and GPU based Docker is used for the code execution. For storing the trained model to Model registry and Performance monitoring, MLflow is used. In the on-premise setup, this pipeline seems to work fine, but this is not an ideal setup in terms of accessibility and scalability. The same application is now migrated to one of the public clouds, Microsoft Azure.

Many factors affect the cloud deployment process starting from the service provider itself. Different cloud services provided by the cloud vendor should be compatible and interoperable with the existing systems and applications. For instance, in the current context - Docker, Database and web application hosting. Apart from this, cloud reliability, features, SLAs, pricing, and regional availability also matters. Once these factors are met and a cloud is selected like in the current context MS Azure, the software components can be migrated to the cloud services. Following are some of the major advantages and issues of hosting a software application on cloud. As explained earlier, not all the software components in ML are cloud native and following issues should be kept in mind while migrating ML applications to the cloud.

ML application cloudification benefits are as follows.

- **Accessibility:**
  - Because of the geographic replication feature ML applications remain accessible with the same performance from anywhere around the world.

- **Disaster Recovery:**
  - When an on-premises ML application crashes due to issues like hardware failure, disaster recovery would be difficult and time consuming. From Figure 1, MLOps introduces numerous modular components. On the unlikely events of disasters, managing all the components manually and on a local setup would be difficult. However, on a cloud this can be easy to handle.

- **Scalability:**
  - To handle peak load conditions, the running instances have to be replicated or hardware like Memory and CPU/GPU needs to be added. A cloud hosting platform can automatically do these via horizontal and vertical scaling, at all the geo replicated areas at once. On a local setup, this is time consuming and might be costly. Large and complex ML workloads can be easily handled with cloud scalability.

- **Security:**
  - A cloud hosted ML application might be a lot secure due to various measures like encryption, firewalls, request authentications and network isolation via Virtual Private Networks. ML data and model security can be assured with the security measures available on cloud.

However, a ML application cloud hosting can be an issue as well. The issues could be caused by making improper deployments, misconfiguring the cloud services or configuration issues with the ML application on cloud. Here are such common issues.

- **Cost:**
  - ML application needs GPU for the ML training. GPU on a cloud infrastructure might lead to increased cost.
• **Portability:**

Not all the tools and technologies are supported on a cloud, and this leads to installing the needed software via Dockers or Virtual machines, which should be maintained regularly.

CDPs reduce such issues with the application hosting and makes sure that the cloud computing environment always works with the best possible performance. The relevant CDPs corresponding to the current MLOps pipeline implementation shown in Figure 1 are as follows.

- Ambassador pattern
- Circuit Breaker pattern
- Sidecar Pattern
- Static content hosting pattern

The first step of MLOps pipeline is the data collection module. This is implemented via Ambassador pattern.

A. **Ambassador pattern**

By using this pattern, helper services are developed that issue network requests on a main application's behalf. The common client connectivity tasks like data collecting from a remote service can be offloaded with the help of this pattern. The requests are sent to Weather and TSO REST APIs for collecting the data. This pattern creates an Ambassador service that is co-located with the main application code. The main application code offloads two different data collection jobs to the Ambassador proxy. When the job is done, each process will separately send the data back to the main application. In this way, different authentication methods needed for each REST API can be maintained easily. Figure 2 depicts a data ingestion pipeline implementation accessing remote services through an Ambassador proxy. The Ambassador pattern uses Circuit breaker pattern to check the state which is explained in the next sub-section B. **Circuit Breaker pattern**. Along with Circuit Breaker pattern Retry pattern can also be used.

B. **Circuit Breaker pattern**

When connecting to several remote services for the data acquisition, this pattern can stop an application from submitting repeated requests in the event of a failure. In a distributed network architecture, service calls can fail due to multiple issues like timeouts, slow networks, or target resource temporarily unavailable. After some time, a few of these problems might be resolved, and the application should be prepared to handle some of these failures. An open, closed, and half-open state machine can be used to implement the proxy.

The Half-Open state keeps track of how many times the operation has been successfully invoked. If a remote call is unsuccessful, the circuit breaker enters the Open state right away, and the success number is reset to 0 the next time it reaches the Half-Open state. Periodically, a time based failure counter automatically resets. The Timeout count in Figure 3 is kept high for the weather API as the expected response time is high due to the large payload.

C. **Sidecar Pattern**

After the data ingestion pipeline, the data is now in the database. The next component in the MLOps pipeline is to perform EDA and primarily train & evaluate the model. For training the model there is no dependency on the EDA module. But the only constraint is, EDA should run on the same dataset which model uses to train, hence the trigger should be on the same data lifecycle. The libraries and frameworks used for EDA and Model training are different. In such scenarios, we can consider Model training as the Primary application and EDA can be a peripheral task. The Sidecar pattern exactly supports this scenario as shown in Figure 4.

The sidecar service – EDA, is not part of the Primary Application – Model training, but it is connected to it.
D. Static Content Hosting pattern

Once the Prediction Service MLOps pipeline component runs, it saves the data to the database and CSV. These CSV files are delivered to client directly from the cloud-storage. Web servers are tuned for dynamic content rendering and caching those contents. But, if these CSV files are served through the web applications, the processing in the web servers will consume some CPU cycles, which is a resource waste. The same architecture is used for serving some static contents which are required by the HTML pages such as images and client-side JavaScript files.

Fig. 5. Static Content Hosting Pattern for static file serving

When the model is trained, the model registry stores the artifacts of every run. The model registry is implemented via Azure Storage container. In this implementation, MLflow is used as a Performance monitoring tool. For the MLflow analysis, all the required artifacts are served via Storage container. Figure 5 shows the static content hosting pattern.

IV. IMPLEMENTATION

The MLOps pipeline is implemented using various MS Azure cloud services. To implement the pipelines via CDPs Figure 6 shows different services used and how they are interlinked.

Table 1 lists different service types created and linked on Azure as shown in the previous figure.

### Table I. SERVICES CREATED ON AZURE

<table>
<thead>
<tr>
<th>TYPE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action group</td>
<td>Global</td>
</tr>
<tr>
<td>API Connection</td>
<td>West Europe</td>
</tr>
<tr>
<td>App Service plan</td>
<td>West Europe</td>
</tr>
<tr>
<td>Azure Cosmos DB for MongoDB account (RU)</td>
<td>West Europe</td>
</tr>
<tr>
<td>Container registry</td>
<td>West Europe</td>
</tr>
<tr>
<td>Smart detector alert rule</td>
<td>Global</td>
</tr>
<tr>
<td>Container instances</td>
<td>West Europe</td>
</tr>
<tr>
<td>App Service</td>
<td>West Europe</td>
</tr>
<tr>
<td>Application Insights</td>
<td>West Europe</td>
</tr>
<tr>
<td>Container instances</td>
<td>West Europe</td>
</tr>
<tr>
<td>Virtual network (classic)</td>
<td>West Europe</td>
</tr>
<tr>
<td>Log Analytics workspace</td>
<td>West Europe</td>
</tr>
</tbody>
</table>

Fig. 6. Cloud services for implementing MLOps using CDPs

For building the MLOps pipeline components using the above mentioned CDPs within Azure, microservice architecture is used. Microservices are a well-liked architectural design pattern for creating highly scalable, and quick-evolving applications. Small, independent services are arranged in a collection known as a microservices architecture. Each service should implement a single business capability inside a defined environment.

V. RESULTS

One full cycle execution of MLOps pipeline on Azure generates multiple types of outputs. The Finnish National consumption day-ahead forecast values are stored in the database and as a CSV file. The stored values are shown in a dashboard and exposed to VPPs via REST API. Figure 7 shows the web interface for visualizing the results for electricity consumption forecast, where the light blue line and dark blue lines represent the actual and forecasted values respectively. Figure 8 shows the REST API server web interface.

Fig. 7. Web application interface for the electricity consumption forecast.
The services mentioned in Table 1 are created in a shared tier plan on Azure, and this can cause issues like resource unavailability. The cloud design patterns act in these scenarios. For instance, while running the ML model training, validation & EDA via sidecar pattern as shown in Figure 4, the ML training Docker has failed to start due to GPU unavailability. However, the sidecar task for EDA has already started. Due to the usage of Sidecar pattern, if the Main application has fails, then the Sidecar task must stop, and the result is as expected. Figure 9 shows the main application failure message log and Figure 10 shows the Sidecar termination log.

Fig. 8. REST API interface for the electricity consumption forecast

The services mentioned in Table 1 are created in a shared tier plan on Azure, and this can cause issues like resource unavailability. The cloud design patterns act in these scenarios. For instance, while running the ML model training, validation & EDA via sidecar pattern as shown in Figure 4, the ML training Docker has failed to start due to GPU unavailability. However, the sidecar task for EDA has already started. Due to the usage of Sidecar pattern, if the Main application has fails, then the Sidecar task must stop, and the result is as expected. Figure 9 shows the main application failure message log and Figure 10 shows the Sidecar termination log.

Fig. 9. Main application failure log from Azure

Fig. 10. Sidecar termination log from Azure

VI. CONCLUSION

The use of cloud computing to implement VPP has been presented in this article. VPPs utilize different services including ML applications for the optimal operation. A successful ML application lifecycle needs MLOps process and CDPs would make sure the implementation and execution of MLOps pipeline components on cloud. Using Azure, these concepts were demonstrated, and this work can be further extended to other cloud providers also.

This work could be further improved in terms of security by implementing a more standard authentication and authorization scheme for the exposed public API like OAuth 2.0. OAuth 2.0 could replace the current token-based authentication scheme. The current implementation relies on Azure's auto-scaling capabilities to handle concurrent loads, and if necessary, additional scaling mechanisms could be implemented.

ACKNOWLEDGMENT

This research was funded by Business Finland grant number 1516/31/2022.

REFERENCES


