Index

[Abstract 2](#_Toc98368576)

[CHAPTER 1: 5G communication scenario and Network slices 3](#_Toc98368577)

[1.1 VNF and Network slices 4](#_Toc98368578)

[1.2 MEC Architecture 7](#_Toc98368579)

[1.3 VNF Continuous Integration and Continuous development (CI/CD) 9](#_Toc98368580)

[CHAPTER 2: Federated Learning 12](#_Toc98368584)

[2.1 FL main characteristics and available architectures 14](#_Toc98368586)

[2.2: FL development cycle: MLops 18](#_Toc98368587)

[2.3: Federated Averaging algorithm (FedAVG) 19](#_Toc98368588)

[CHAPTER 3: Background on adopted Technologies 21](#_Toc98368589)

[3.1: Docker and Docker Compose 21](#_Toc98368590)

[3.1.1: Docker Hub 21](#_Toc98368591)

[3.2: Kubernetes 22](#_Toc98368592)

[3.2.1 KubeCtl, Kubeadm, kubelet 22](#_Toc98368593)

[3.2.2 Kustomize 22](#_Toc98368594)

[3.2.3: Master/worker architecture and components 22](#_Toc98368595)

[3.2.4: Kubernetes storage: Persistent Volume and Persistent Volume Claims 23](#_Toc98368596)

[3.2.5: Kubernetes components and deployment 23](#_Toc98368597)

[3.3: PyTorch 24](#_Toc98368598)

[CHAPTER 4: Kubeflow 25](#_Toc98368599)

[4.1: Istio Ingress Gateway 25](#_Toc98368600)

[4.2: Knative 26](#_Toc98368601)

[4.3: Katib 26](#_Toc98368602)

[4.4: Jupyter Notebooks 26](#_Toc98368603)

[4.5: Kubeflow storage: Minio and MySQL DB 27](#_Toc98368604)

[4.6 Dashboard v 1.4 27](#_Toc98368605)

[4.7: Kubeflow Pipelines (KFP) 27](#_Toc98368606)

[4.7.1: KFP Software Development Kit and python DSL 27](#_Toc98368607)

[4.8: Kubeflow Experiments and Runs 27](#_Toc98368608)

[CHAPTER 5: Experiment architecture and description 28](#_Toc98368609)

[5.1: Kubernetes experiment architecture: deployed pods and namespaces 28](#_Toc98368610)

[5.2: Notebook to Pipeline list of experiments: Basic, IID FedAVG, Non-IID FedAVG 28](#_Toc98368611)

[5.3: Results and comments 28](#_Toc98368612)

[CHAPTER 6: Conclusion 29](#_Toc98368613)

[REFERENCES 29](#_Toc98368614)

[Index of pictures 29](#_Toc98368615)

# Abstract

The main objective of my thesis work is to exploit the Google native and open-source platform Kubeflow, specifically using Kubeflow pipelines, to execute a Federated Learning scalable ML process in a 5G-like and simplified test architecture hosting a Kubernetes cluster and apply the largely adopted FedAVG algorithm empowered by the ML platform ‘s abilities to ease the development and production cycle of this specific FL process.

FL algorithms are more are and more promising and adopted both in Cloud application development and 5G communication enhancement through data coming from the monitoring of the underlying telco infrastructure and execution of training and data aggregation at edge nodes to optimize the global model of the algorithm ( that could be used for example for resource provisioning to reach an agreed QoS for the underlying network slice) and after a study and a research over the available papers and scientific articles related to FL with the help of the CTTC that suggests me to study and use Kubeflow to bear the algorithm we found out that this approach for the whole FL cycle deployment was not documented and may be interesting to investigate more in depth.

This study may lead to prove the efficiency of the Kubeflow platform itself for this need of development of new FL algorithms that will support new Applications and especially test the FedAVG algorithm performances in a simulated client to cloud communication using a MNIST dataset for FL as benchmark.

The document is articulated as follows:

* Introduction on 5G architecture and VNF development cycle
* Background information on Federated Learning
* Background knowledge on the technologies used: Docker, Kubernetes, PyTorch
* Kubeflow architecture and main components
* Experiment architecture, deployment, and tests taken
* Results and conclusions

# CHAPTER 1: 5G communication scenario and Network slices

As today in many research fields and daily used applications/platforms the help and power of technology is more and more important to grant a continuous evolution and achieve important results/offer good products to the customers the 5G communications technology introduced by the 3GPP WG in the 2016 is meant to support IT systems containing thousands or millions of end devices (i.e., in an IoT architecture) in a secure, fast (real-time requirements for specific projects) and reliable way.

Since the adoption of 4G communication technologies it was introduced the EPC (Evolved Packet Core) that with its architecture clearly separated the “data plane” and the “signaling” plane and the sessions of users were handled with IMS (IP over Multimedia Subsystem) the main evolutions from that 4G basis in 5G is that all the components of the Core Network (many of them evolved from 4G) are:

* Virtualized components:

In 5G architecture all the components of the Core Network are realized by Virtualized services and functions (NVF or Network Virtual Functions) over the real hardware (Antennas, Edge nodes) including components for radio access, signaling/control plane, data plane and other Telco specific added services.

* Computation at the edge:

In 5G networks (especially for IoT-related ones) an important role is played by the Edge nodes (intermediate nodes between the On-premises network and the Cloud one).

The paradigm of using Edge nodes to perform different computing Tasks is named “Edge Computing” and it introduces a lot of practical advantages such as low latency with respect to prem-cloud communication, data privacy (because if the aggregation/processing of sensible data is performed at the Edge it is possible to send to the Cloud only data obtained from this process that will not contain all the sensitive information)

* High Flexibility:

Especially for Industry 4.0 scenarios the 5G architecture provides high flexibility, composability, productivity, and it will in practice act as the enabler technology for emerging or developing applications such as Augmented Reality, Factory automation etc.

Another adopted approach as a possible replacement of Edge Computing especially in IoT scenarios (composed by resource-poor devices) is the Fog Computing paradigm that in the topology of the architecture represents an intermediate layer as well as Edges but it differs from the Edge approach that usually avails of a few powerful nodes because in the Fog Computing one there are many so-called Smart Gateways (SGs) with low resources that are unable to host heavy computations.

### VNF and Network slices

As introduced before one of the biggest challenges/innovations brought by 5G networks is of course the Network Function Virtualization (NFV) so the virtualization of every network component that in the previous generation were pure and specialized hardware components with the objective of run these evolved components in very powerful servers with enormous data storage capabilities, orchestrated automatically by software switches and remotely installed and configured.

This approach tries to overcome the limits of integration, automation, and scalability of the previous generations that were using specialized hardware technology.

The classical 5G use case scenario is the creation of different Network slices or (SDN software defined Network) that insists autonomously over the same underlying hardware resources that composes the network so for ex. one slice for an IoT cloud-based application, another one for automotive and eventually many others may run at the same time isolated the one from the others over the common hardware infrastructure as we may see in figure 1.

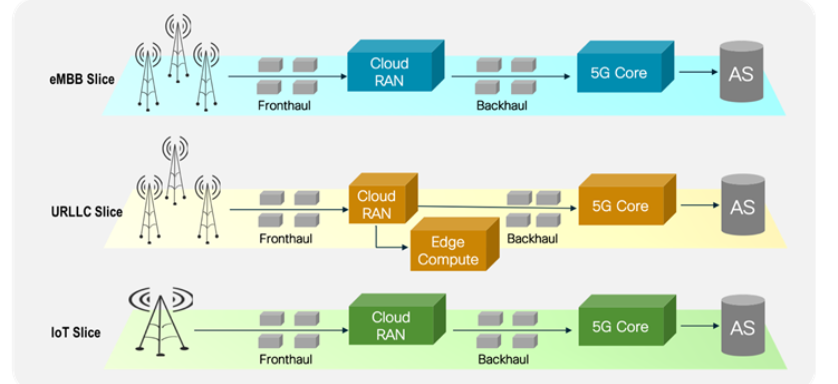


Figure 1: 5G networks slices example

This “slicing” is performed at the network level, and it provides many advantages for end-to-end management as:

* Composability:

All the entities and services related to a specific business domain/application are included in the same network slice resulting in a better management of them and ease and speeds up the creation/development of new versions

* Network flexibility:

It enables redesign of the network supporting the cloud native application from remote and flexible function allocation

* Orchestration:

It is necessary and provided for every software component of the slice and grants reliability and scalability of the same.

All this important features does not come out of the box with 5G but derives from good practices and previous knowledge related both to cloud computing and mobile cloud networking as Service oriented Architecture( SOA) so that every function or service ( in this case the VNF) should be decoupled and composable through a precise interface and provided and orchestrated by and agent and the microservices and Containerization approaches that are wide-adopted for different solutions and are at the base of the provisioning and the orchestration of the VNFs as well.

It is important to underline that all the VNF realizing Telco functions, the Cloud provider ones and the application-related ones coexist and collaborate to implement the service in case and a standardized and generic architecture it’s fundamental to grant an overall efficiency of the system and even more to reach a better and better integration between Telco providers and Cloud providers to ease the development of 5G cloud-based applications.

For that a common Management and Operation platform has emerged following the guidelines of the ETSI MANO standards that includes three main entities (extended in some open-source solution as Open Baton):

* Virtualized Infrastructure Manager (VIM):

Is the component responsible for the lifecycle management of compute, storage and network resources and it acts in practice as a cloud management system that exposes API for operation over those resources (usually CRUD one).

A very adopted implementation for this component is the open-source cloud platform Open Stack.

* Virtual Network Function Manager (VNFM):

Is the component responsible for the lifecycle of the VNF deployed in the system supporting the instantiation, creation, updating, scaling and termination of them?

Usually there is one VNFM per Network Function or one per group of VNFs.

* Network Function Virtual Orchestration (NFVO):

Is the component responsible for the lifecycle management of network services either in a single domain or over multiple datacenters, it may also apply policies for resource utilization a Requests the instantiation of VNFs via the VNF Managers

Another important aspect in dealing with VNFs is their location and placement in the overall architecture, in fact beyond deploy them over cloud providers or “central nodes” leveraging Edge computing we may deploy them over edge nodes as well.

This implies as well, since VNFs must be orchestrated adopting dynamic decisions based on the service state, network congestions and node mobility, that edges nodes for what regards networking play a big role in supporting this and if needed should be able to work autonomously in case of backhaul unavailability or disconnections from the central node.

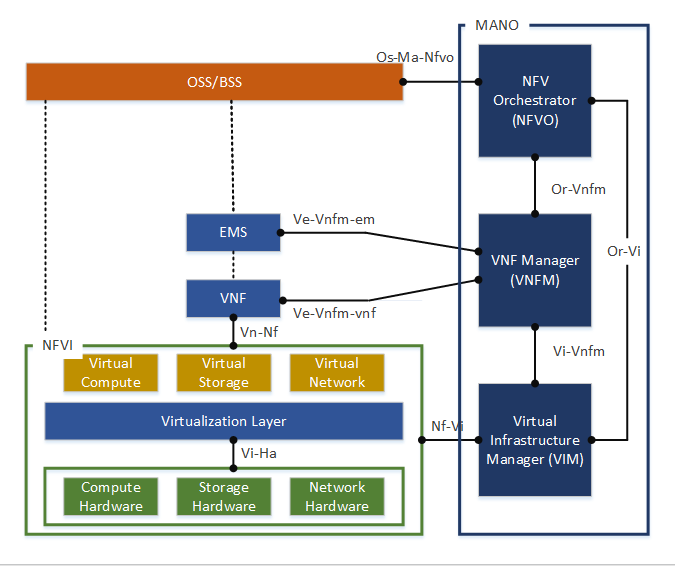


Figure 2: MANO basic architecture

### 

### 1.2 MEC Architecture

To organize and define an efficient and standardised way to adopt Edge computing in 5G architectures the MEC (Multi Access Edge computing) was introduced by the ETSI research group.

This standard is meant to overcome the limits of low-resources mobile devices by bringing computation closer to the devices, offering high bandwidth, low latency, and mobility of nodes

MEC enables the deployment of MEC applications as software only entities that runs on top of a virtualized infrastructure at the edge or close to it.

The MEC framework is composed by many entities that can be grouped in three main levels:

* Network level:

It is composed by entities providing the underlying network that will be exploit by the MEC application and contains elements for 3GPP, local and external networks

* MEC Host level:

It is the level that oversees managing the execution of MEC application.

The MEC Host itself is the element that contains the Virtualization Infrastructure (NVFI) and the host platform that provides compute network and storage for the MEC applications.

The Virtualization Infrastructure as showed below includes a data plane element to execute the traffic rules received by the MEC platform and route them to the underlying networks or some other application hosted at this level.

The Host Platform has many responsibilities such as furnish an environment to MEC application in which they can discover or offer MEC services possibly also services from other MEC systems, receives traffic rules from MEC platform manager, application or services and correctly instruct the data plane, handle DNS by receiving DNS records from the MEC platform manager and set properly a DNS proxy/server.

* MEC System level:

Its main component is the multi-access Edge orchestrator in charge of different tasks such as keep an overall state/view of the MEC system (deployed MEC host, available resources, and services); select a proper MEC host to fulfil the requests of instantiating an application bases on conditions to be met on the MEC host as resources availability, MEC services available; on boarding of application packages including authenticity checking and validation of application rules/requirement , keep a record of on boarded applications and prepare the Virtual Infrastructure manager for hosting them; it can also trigger applications instantiation , termination and migration.

The Operation Support System (OSS) in charge of receiving CSF portal or device application request for instantiation and termination of applications and after granting them are forwarded to the multi access edge orchestrator.

The user application Lifecycle Management Proxy (LCM proxy in figure 1) has the role of managing user applications that are MEC app (device app) instantiated in the MEC system after a request comes from a user who is running the application on his device, and it allow also to terminate or migrate them via requests from the user if needed. It must pass the requests coming from user’s applications to the OSS and the Edge orchestrator to fulfil them.

In the figure 1 it is described the general MEC reference architecture that complies to the ETSI MANO standards from [1]

The architecture in question is general and when it comes to use NFV as well as MEC given that they are complementary concept in the 5G context it needs some extensions and elements so that several different deployment options for MEC systems are available.

In such a contest the MEC platform is deployed as a VNF, the MEC applications appears as

VNFs for the NFV MANO elements; the Virtualization Infrastructure is deployed as a

NFVI and managed by a VIM; the MEC platform manager (MEPM) is replaced by a MEC

platform manager NFV(MEPM-V) that leaves the VNF lifecycle management to one or

more Virtual Network Function Manager (VNFM);the MEC orchestrator is replaced by a

MEC Application orchestrator that is supported by the NFV orchestrator for resource

orchestration and by one or more Network Services (NSs) for orchestration of the group of

MEC applications deployed.

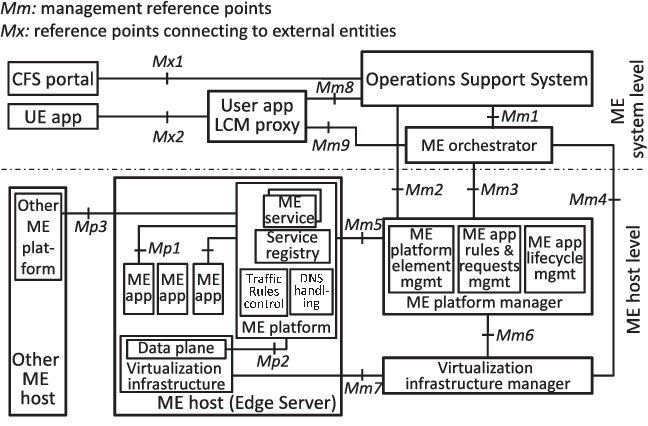


Figure 3: MEC Reference architecture

Since every component forming the architecture and used to deal with VNF in 5G in general is software it is important to reuse and adopt the best practices in software development and in particular DevOps rules to grant a smooth and endlessly evolving platform.

### 1.3 VNF Continuous Integration and Continuous development (CI/CD)

When it comes to deploy new components or upload new VNF’s versions the DevOps concepts and practices are mandatory and some of the most common ones are for sure the CI/CD principles and the derived pipeline and the so-called Agile delivery or in this case for usually the effort to produce a VNF and VNFC is the union of more parties “Joint Agile Delivery” (JAD).

With JAD more than that comes a series of “checkpoints” and tests corresponding to the key point across the whole CI/CD pipeline built among several carriers through which the code is tested and validated across all the different environments of the “delivery pipeline” from development and testing ones to the production one.

A more general view of the “delivery pipeline” in a multi-Operator delivery use case scenario can be found in figure 2 from the authors in [2].

# 

Figure 4: Joint Agile Delivery Multi-Operator Pipeline

# The delivery pipeline in figure is intended to represent and end-to-end process from the idea of the VNF in this case to the end product deriving the basic procedures from the Software Development Life Cycle (SDLC; classical SLDC procedures: design, coding, test, release) and applying them at a more fine-grained level following agile delivery methodologies too.

# CI is one of the most used DevOps practices and refers to automatically validate and uphold code artifacts along the delivery pipeline steps and when possible (for example a code artifact that correctly passes the tests of a specific pipeline step), autonomously trigger the next pipeline activity.

In VNF context this practice becomes fundamental since the artifact of code to deliver are usually smaller and more frequently updated with respect to the typical code artifact of SDLC development cycle and it usually involve software integration of heterogeneous components into a big entity on the fly.

This need of integration and the multi-operator nature of the VNF use cases highlights the need of a previous and clear agreement between operators on the requirements and the nature of the tests to be taken at the different checkpoints, usually it results that the nearer to the production environment the test is taken the more deepen and difficult to pass will be so that at the end of the development cycle the quality of the outcome will possibly satisfy the standards.

Another basic DevOps practice very important to reach a good ending product is the Continuous Delivery, in this context in fact it is quite normal that a consistent number of software components will need to frequently traverse all this development cycle.

The aid of CD in this flow of software artifacts is to automate as much as possible the passages between the steps by generating intermediate packages and package the released ones, to support that some complementary tool is required for environment management, resource management and test management.

More that that to support this process and make it effective the continuous monitoring is another important practice in general in DevOps and for dealing with VNF delivery and it aims at automating in a such scalable and complex context all the responses to alarms and different triggers on the pipeline through a deep and constant monitoring of the different components of the system.

To achieve that in this case a different design and implementation of eventing, monitoring and response to this type of events needs to be considered to make it successful, starting from the adoption in the early stages of the pipeline of a code repository handled through continuous integration to validate the new code uploaded to the repository with the less human intervention as possible and code checking to validate in the early steps automatically code based on its syntax or follows bests practices

Another important aspect of this VNF DevOps pipeline not yet considered is the definition of the interfaces between the organization who is producing the network function/application and the one who is serving and managing it so that the actors can smoothly follow the steps of the pipeline.

Regardless the complexity and the number of operators involved in the development process the actors usually involved can be summarized under these profiles:

* VNF provider: typically defined as the producer of the VNF
* VNF validator: entity that validates a VNF according to some KPIs
* VNF operator: entity that operates a certain VNF over a NFVI

The previous definitions and descriptions are meant to provide a general background on important concepts behind the ones that will be later explained and Kubeflow the selected platform to run the experiments might ease some of the underlined issues as scalability, container orchestration, multi-operator software development using CI/CD tools (in this case ML procedures) through Kubeflow Pipelines (KFP).

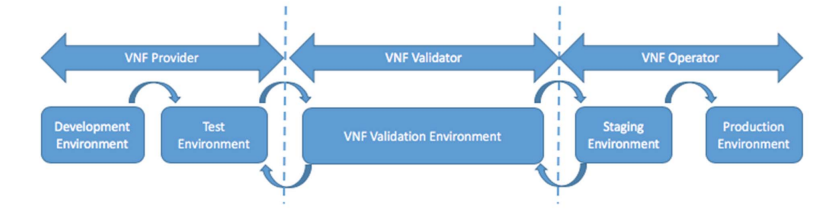


Figure 5: VNF DevOps cycle with all the actors

# CHAPTER 2: Federated Learning

# Parallelly to the development and affirmation of cloud computing, the evolution of communication technologies and protocols another important field in computing that was more and more studied and enriched is the Artificial Intelligence.

Between the disciplines and areas of interest of Artificial Intelligence in the recent years in particular Machine Learning (ML) has been widely adopted to cope with a very heterogeneous set of problems and research fields as Data analysis, Computer vision, model optimizations (i.e., Digital twins), Business Intelligence and many more.

The ML accepts under its umbrella a lot of different possible techniques to be used depending on the nature of the problem, the most general classification can be made based on the type of learning that needs to be performed:

* Supervised Learning:

A Supervised learning task is to be chosen in a scenario in which the data that is available it is already labelled or partially labelled, and the objective is to create a model that needs to be able to recognize a specific input through some learned pattern or some peculiarity of the train data in ingress and correctly classify it.

One of the possible limitations of this approach is that in the real-world data is usually unclassified and it is expensive to obtain labelled data, anyway once we have it can be very useful.

Some possible techniques to adopt in this scenario are Decision Trees, Bayesian Classifiers or Neural Networks (NN).

* Unsupervised Learning:

An unsupervised learning task is to be preferred when we have a lot of data available and no information at all (or little insight over the data) and the objective is to find common characteristics and patterns between the data to group the most similar examples and exclude from the same group inputs that have very few in common at the end of the learning process.

One fundamental consequence of identifying important feature of the input data is that when we have a huge dataset or a very sparse and incomplete one, we can keep in the later phases of the process only the most important features of the input data discarding the noisy one so that we can reduce the dimensionality of data.

Some of the possible algorithms to realize that are K-means, Hierarchical clustering, Principal Component Analysis (PCA) and NN.

* Reinforcement Leaning:

A reinforcement learning task is a particular ML procedure in which we have the known and effective concepts of “punishment” and “rewards” for it consists of a cyclic scheme in which an Agent that continuously take actions to evolve the state of the environment is judged by an entity called “teacher”; if the taken action has led the system towards the objective function the agent will be rewarded otherwise it will be punished in this way the agent should seeks for actions that minimizes or delay the punishments and maximizes the rewards.

Important applications of these approach are the Generative Adversarial Network (GAN), Game players (i.e., Computers playing Atari by Google DeepMind), controller optimizations, Self-driving cars and more…

The classification in question is not so strict and hybrid approaches are also available for example the so called “semi-supervised” learning in which only a very small portion of the data is available or sometimes more than one approach could be exploited to reach the objective (i.e., generate data similar to the real one with a GAN to be used as input of a Supervised network in order to improve its ability of generalization. )

Another possible classification of ML technologies can be taken, for what regards Neural Networks, based on the architecture of the network and its features as:

* Number of connections:

The architecture of a NN is strictly correlated to the task that drives its creation and the development of the same and it is usually very specific to the problem it is intended to solve.

As described below a NN can have a different number of layers from a minimum of 3 up to thousands, but a very important characteristic that can determine its behaviour is the number of connections between neurons.

As in our brain (from which all the NN theories are derived) the characteristic that make it so powerful is the high number of neurons and their connections, so is in neural network.

A connection here is defined as a directional link between two neurons i and j corresponding to different layers resulting in the weight Wij, that will be used to transfer data and activate the right portion of the network as a specific input will be presented.

The high-level classification of NN based on the number of connections is Fully connected networks and Layered networks.

In a Fully connected network as the name suggests each neuron is connected to all the others and the general behaviour of the net in case is resumed by an N x N connection matrix where N is the number of the neurons.

A lot of times instead it is preferred to select a layered architecture where not all the neurons of a layer are connected to all the others, but the connections are specific and designed following the nature of the problem.

* Number of layers:

A minimal network architecture it is composed by an Input Layer, a Hidden layer, and an Output layer, each one containing different weights that is going to be used for computations over the input data and will change their initial values while learning until the net can correctly generalize. There is a proportion between the number of suggested hidden layer, the number of inputs and the number of outputs so that for small datasets and very simple tasks this structure may be sufficient

Nevertheless, in many real problems where the dataset is huge (i.e., time-series, computer vision) the structure of the net is very complicated and the needed hidden layers from 1 can be up to hundreds or thousands. In this case we talk about Deep Learning.

The main conceptual distinction between ML in general and DL is that in the former the tasks of feature extraction and classification are well separated and pipelined (need of study the features of the data available to get the best out of the classification i.e. using decision trees), in the latter instead even if usually it is a very complicated and in a “black box” fashion task ( so that we only know that the net usually responds with a certain output when is stimulated with a specific input without knowing all the details of the processes that take place in the hidden layers)

Deep learning has been the pioneer technology for the disruptive evolution of AI, specifically over the last years, opening the possibility to exploit its potentiality in several different fields, but the common cloud-centric approach especially if considered in a 5G environment deployment scenario has many limits and potential issues.

That is why Federated Learning has become one of the more adopted approached to cope with the high number of entities and nodes in modern networks and moreover to get advantage of ubiquitous and powerful actors and their virtual counterpart disseminated all over the architecture from end devices passing through edge nodes and cloud/central nodes.

### 2.1 FL main characteristics and available architectures

In order to get advantage of valuable information provided everyday by the massive utilization of mobile devices and IoT devices the Federated Learning approach has proved to be one of the, if not the most promising, technologies to train models in a distributes system leveraging its characteristics to minimize the data transfer ( because all the local data used by devices to train their local models will remain there , contrary to the centralized model for global training ) and reach a good accuracy for the global trained model that will come out from the training process.

The downsides of this technology are for sure the possible security and privacy issues that may arise using user private or sensible data to train a model, and the communications costs related to the transfer of all the local models train outcomes onto the Aggregation Server or some similar entity at the Edge/Cloud premises.[3]

Many possible applications of this training scheme are still studied and in development and the reason is that is a very general scheme that, to be efficient as it should, has in a sense to be tailored to the problem that we are trying to solve using FL and all it’s possible peculiarities.

Starting from the data for example that FL will use for learning we may divide FL problems in IID (Independently and Identical Distributed) data and Non-IID(Non Independently and Identical Distributed one.

In the great majority of real problems and use-case scenarios data is coming from a multitude of different devices that might be different in terms of power/resources/connection availability/ Geo-location forming a global dataset implicitly Non-IID.

Anyway, studying and executing FL algorithms on IID datasets as well may be useful to understand the possible best case of execution and to adopt some optimization to its Non-IID counterpart such as sharing of a portion of the data between clients or similar techniques under experiment.

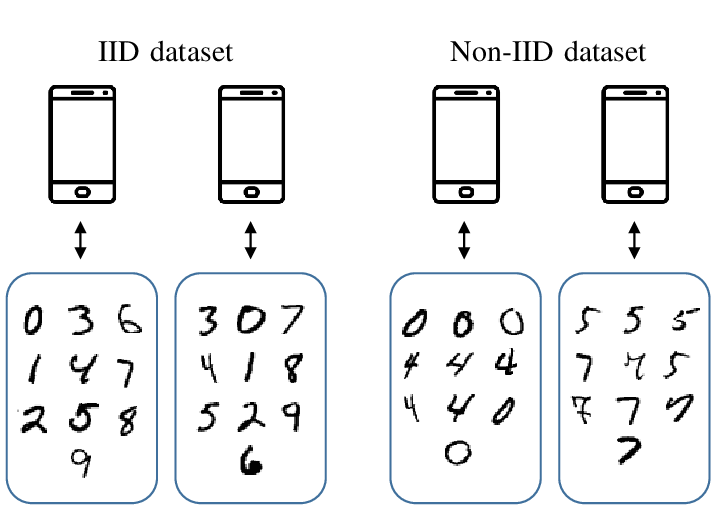


Figure 6: Differences between IID and Non-IID datasets (from Researchgate.com)

The figure 5 above gives a practical and intuitive view of the differences between an IID and a Non-IID datasets in the case in a digit recognition problem such as the one selected for the thesis work with the MNIST dataset for FL.

For the sake of generalization, it is evident that a heavy Non-IID dataset has intrinsic problems for probably every client/device will overfit or be specialized on his own portion of data and can inherit the ability of generalization from the global knowledge slower than its IID counterpart.

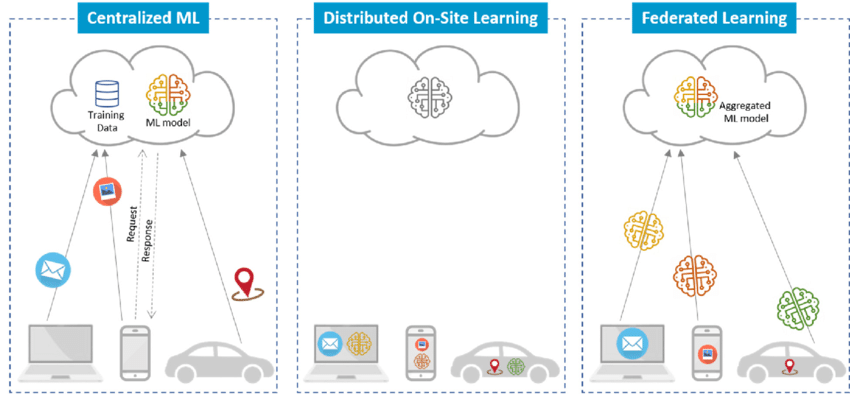


Figure 7: FL architectures: centralized, On-Site, distributed FL (from Researchgate.com)

The available architectures in the literature for FL depending on the fashion of global aggregation are:

* Centralized FL:

In this model the global aggregation takes place only at the centralized edge/cloud server and is the traditional approach for Deep learning tasks in cloud computing architectures to exploit the power of computation of the cloud centre

* Distributed FL:

This approach avoids the use of a centralized aggregation by performing aggregations at distributed servers (the aggregator servers).

* Collaborative FL:

In many scenarios amongst the end devices there are some of them that are very constrained and therefore cannot support Local Learning or for example are not able to send the local model update towards the server that will perform the aggregation.

Therefore Collaborative FL proposes a sort of “collaboration” between near nodes for example by sending model update to a more powerful and neat device towards which we have an already available connection and so on… all is for the sake of fairness ( in the sense that even the more constrained nodes can participate at the training round) and convergence of the algorithm , in fact if the number of needed devices is respected the algorithm probably will converge without additional latencies.

* Hierarchical FL:

This approach exploits a local area aggregation at the edge lever before a global one as the centralized. Is a mix between centralized and distributed FL.

* Dispersed FL:

This approach consists of exploiting subgroups of end devices for FL training.

Initially each subgroup of devices performs its local model update by iterating for some epoch over its local datasets.

After that before sending the updates to the global model a switch of local models is performed between subgroups and finally the models are sent to the Server for the global aggregation.

This approach can be very useful in domains such as self-driven vehicles, as each vehicle in that architecture have enough resource to act as a subgroup.

From a security point of view this can be good to because the global server aggregator cannot infer anymore the local modifications of end devices after they are shuffled between subgroup, by the way this problem it’s transferred at the subgroup server level that may infer sensitive information.

### 2.2: FL development cycle: MLops

As Data Science and ML are inherently very complex and pipelined processes given the number of steps usually involved the importance of handling this operation and fasten as a result the development cycle is critical and following the previously described best practices of DevOps its Machine learning counterpart MLOps is becoming fundamental when it comes to develop and deploy some ML task over any possible kind of hosting infrastructure.

We can sum up the basics and general steps needed (not everyone is mandatory, it will depend on the nature of the ML process in question) with the followings:

* Data Ingestion:

Given that data nowadays it is massively produced and available this first step is the one in charge of collecting huge amounts of data that may (and often is) be uncomplete, not necessary or with a lot of noise.

* Data Analysis:

In general sense the data analysis is the process of inspection and study of the data gathered in the previous step to get information on its nature and use it at its best along the data pipeline (achieved through Data Mining techniques).

* Data Transformation:

The data transformation is the process to obtain a dataset specific and optimized for the task where it is going to be used, some possible modification from the original dataset can be the discard of some features not relevant (it can be discovered through PCA) and basically to reduce the size of the data to process and use only important features to ease overall the computation over it.

* Data Validation:

To guarantee a final QoS of the model it is important that the data used to train it is valuable and to do so based on different policies and KPIs the dataset that has reached this step of the pipeline needs to be evaluated and validated by Data scientist or similar teams before the training.

* Data Splitting:

After ensuring the quality of data in the previous step the data splitting consists of a division of the whole dataset into two main sets one used for the training and the other to test and evaluate the performance of the ML model to augment its abilities of generalization.

* Building a model:

One important component of the whole ML process is the model, and its quality and accuracy are critical to get a good result; in addition to that the creation and definition of an ad hoc model is not an easy task.

In the case of NN for example it can be very difficult and tedious to find the best architecture of the network in terms of number of hidden layers and number of weights and in addition to many different tries and experience in creating ML model it can be very useful to exploit search algorithms to find the best architecture, such algorithms stand under the name of Neural Architecture Search (NAS) and it is implemented using Reinforcement Learning.

* Validate the model

After the previously described process of model creation, it has to undertake some evaluation to ensure that it meets the requirements in terms of model quality for the needed ML task

* Train the model at scale

In addition to the NN architecture of the model another difficult and usually long task in terms of time and resources is the search for the best “hyperparameters” of the networks for it is needed to take multiple runs of the algorithms (sometime a single run can take hours or days) to select the best value that let the algorithms converge faster ( or let it converge at all) and obtain good results accordingly to the predefined objective for the selected metrics of the model.

To achieve faster convergence/definition of hyperparameters value, especially in particular types of NN as CNN it is important to use ad hoc hardware for computing such ad GPUs or Tensor Processing Units (TPUs) so that the time of a single run is lower or when possible, parallelize the model and train it with different configurations.

* Serving:
* Monitoring and Logging of the processes

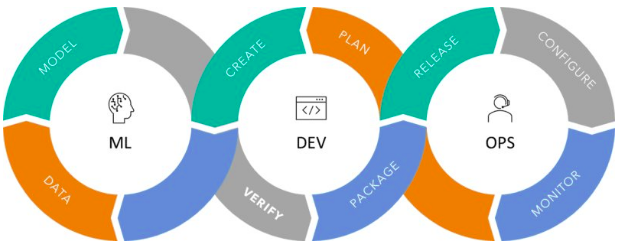


Figure 8: MLops cycle

### 2.3: Federated Averaging algorithm (FedAVG)

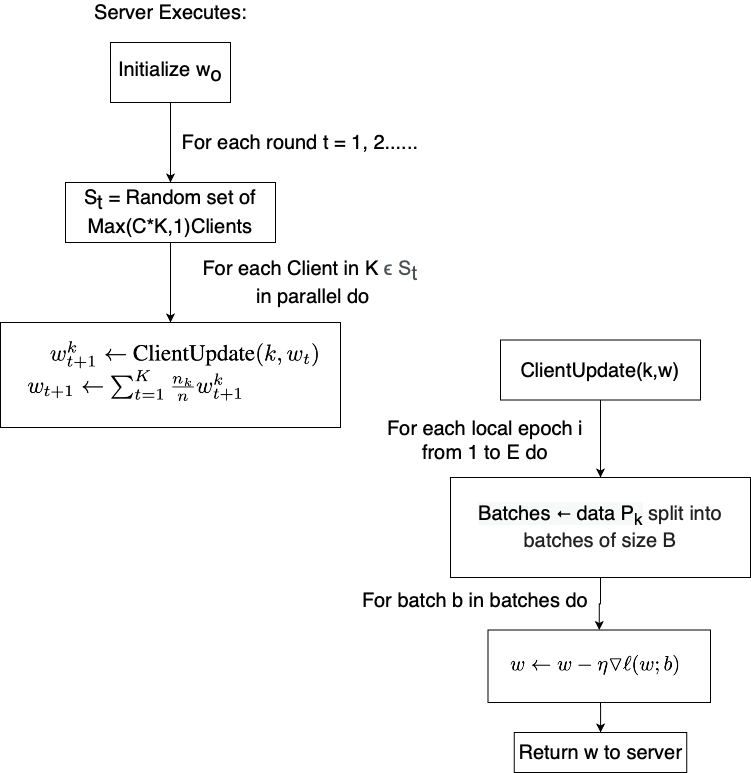


Figure 9: FedAVG algorithm scheme

# CHAPTER 3: Background on adopted Technologies

### 3.1: Docker and Docker Compose

### 3.1.1: Docker Hub

### 3.2: Kubernetes

### 3.2.1 KubeCtl, Kubeadm, kubelet

### 3.2.2 Kustomize

### 3.2.3: Master/worker architecture and components

A Kubernetes cluster is formed usually by a lot of nodes that in general can only be of two types:

- Master node

- Worker node

Each worker node must support a container runtime envinronment (e.g. Docker) and a kubelet process that it is used to interact both with the container and the node.

Kubelets starts the pods (Abstraction of a container) within a container inside the worker node and later it assigns resources from the node to the containers (CPUs, RAMs, Volumes).

More worker nodes can (and usually does) present replicas of the same pods that can btw them using the "Service " pods (Load Balancer that catches the application requests and forward them).

For implementing efficiently this kind of features on the master node 4 processes needs to run:

* Api Server:

entity used to interact with some client (UI, API or CLI as kubectl).

* Scheduler:

it is the second actor in the "request workflow" and it can schedule some process over a worker node.

It has the intelligent feature of deciding which one is the best pod to host the process so it

acts in a sense also as a Load Balancer (scheduling based on the resource occupation over

a node).

* Controller manager:

it detects state changes like crashing of pods or similar over the cluster and tries to resume the system's state as soon as possible (ex. by contacting the scheduler to restart the crashed pods).

As before once the scheduler has selected the best node to which it will assign the process it will contact its Kubelet entity that it oversees restarting the selected pod.

* ETCD:

It is a key-value storage of the cluster state. We can in a sense imagine it as the cluster brain.

All the changes that take place in the system will be stored here as a key-value pair (for ex when a new service is deployed, when a Pod crashes.).

The whole mechanism at the master node it's based upon the data stored here in the ETCD.

This key-value storage anyway does not store application-specific data too as databases or similar.

Given the important processes listed above that needs to run on the Master node to assure a good overall performance of the cluster in many K8s real distributions the cluster is composed not by a single master -multi worker, but both the master and the slave nodes are replicated for the sake of the availability and scalability of the cluster.

### 3.2.4: Kubernetes storage: Persistent Volume and Persistent Volume Claims

### 3.2.5: Kubernetes components and deployment

### 3.3: PyTorch

# CHAPTER 4: Kubeflow

### 4.1: Istio Ingress Gateway

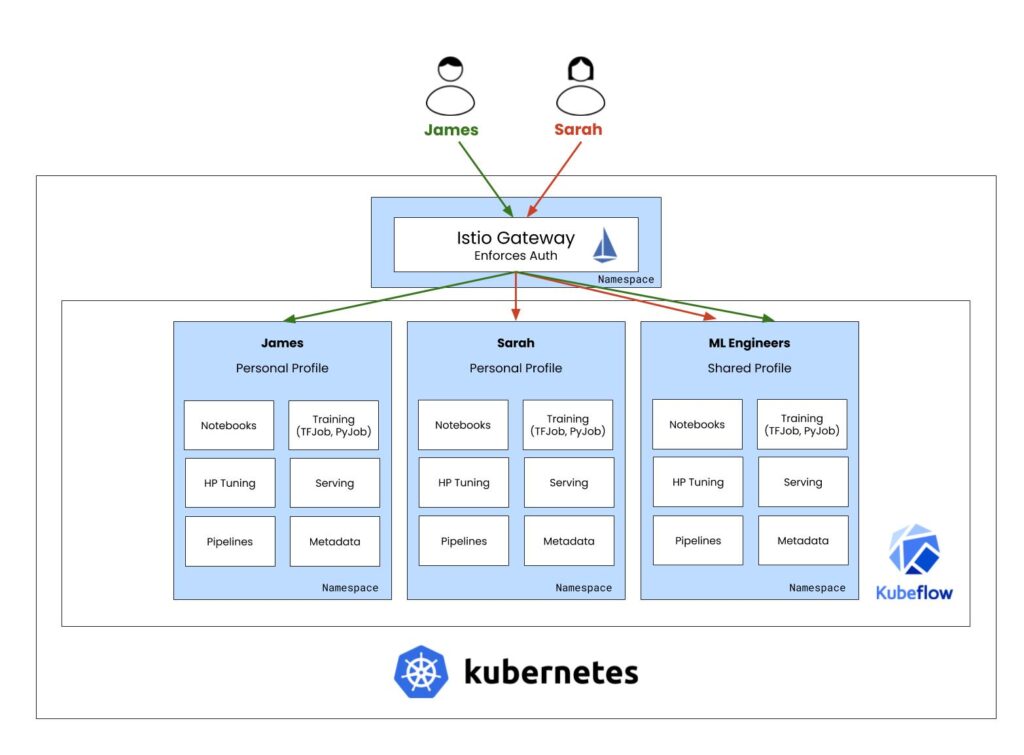


Figure 10: Multi-user Kubeflow separation through Istio

### 4.2: Knative

### 4.3: Katib

### 4.4: Jupyter Notebooks

### 4.5: Kubeflow storage: Minio and MySQL DB

### 4.6 Dashboard v 1.4

### 4.7: Kubeflow Pipelines (KFP)

### 4.7.1: KFP Software Development Kit and python DSL

### 4.8: Kubeflow Experiments and Runs

# 

# CHAPTER 5: Experiment architecture and description

### 5.1: Kubernetes experiment architecture: deployed pods and namespaces

### 5.2: Notebook to Pipeline list of experiments: Basic, IID FedAVG, Non-IID FedAVG

### 5.3: Results and comments

# CHAPTER 6: Conclusions

# REFERENCES

[1]: ETSI GS MEC 003: Multi-Access Edge Computing (MEC) Framework and reference architecture 2020-12

[2]: ETSI GR NFV-TST 006: Network Functions Virtualisation (NFV), Testing, Report on CICD and DevOps 2020-01

[3]: “A Survey on Federated Learning: The Journey from Centralized to Distributed On-Site Learning and Beyond”, IEEE INTERNET OF THINGS JOURNAL, VOL. 8, NO. 7, APRIL 1, 2021

# Index of pictures

[Figure 1: 5G networks slices example 4](#_Toc98369532)

[Figure 2: MANO basic architecture 6](#_Toc98369533)

[Figure 3: MEC Reference architecture 8](#_Toc98369534)

[Figure 4: Joint Agile Delivery Multi-Operator Pipeline 9](#_Toc98369535)

[Figure 5: VNF DevOps cycle with all the actors 10](#_Toc98369536)

[Figure 6: Differences between IID and Non-IID datasets (from Researchgate.com) 14](#_Toc98369537)

[Figure 7: FL architectures: centralized, On-Site, distributed FL (from Researchgate.com) 15](#_Toc98369538)

[Figure 8: MLops cycle 18](#_Toc98369539)

[Figure 9: FedAVG algorithm scheme 19](#_Toc98369540)

[Figure 10: Multi-user Kubeflow separation through Istio 25](#_Toc98369541)