



# The Complete Guide to Using the Iguazio Feature Store with Azure ML

In this e-book, we will showcase an end-to-end hybrid cloud ML workflow using the Iguazio MLOps Platform & Feature Store combined with Azure ML. The first part gives an overview of the solution and the types of problems it solves, and the following parts are a technical deep dive into each step of the process.

## Part 1: Feature Store Motivation

## Part 2: Data Ingestion + Transformation via Iguazio's Feature Store

- Detailed overview of Iguazio feature store functionality
- Ingest and transform dataset into feature store
- Retrieve features in batch and real-time

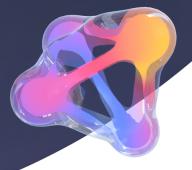
## Part 3: Model Training via Azure ML

- Upload/register features from the Iguazio feature store into Azure ML
- Orchestrate Azure ML training job from the Iguazio platform
- Download trained model(s) + metadata from Azure back into the Iguazio platform

## Part 4: Hybrid Cloud + On-Premise Model Serving + Model Monitoring

- Deploy models to a real-time HTTP endpoint
- Combine multiple models into a voting ensemble
- Integrate model serving with real-time feature retrieval

# **Part 1: Feature Store Motivation**

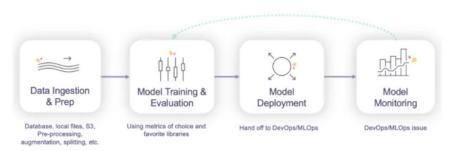


# The Gaps: Challenges When Operationalizing Data Science

Regardless of the environment, one of the main challenges when operationalizing data science is fostering collaboration between teams, and eliminating tech silos. This is both a technological and organizational challenge, requiring the right processes in place and the right tools to support these processes.

In a typical data science project, there are usually three different teams involved at different points in the pipeline:

- Data Engineer: Ingest and transform raw data from various sources
- Data Scientist: Utilize transformed data to train model
- MLOps Engineer: Containerize and deploy model at scale with monitoring, drift detection, and re-training capabilities



While the pipeline itself looks straightforward, there are more than a few places where things can go wrong mostly at the handoff points between teams. What happens when a data scientist needs additional or different features? What happens when the MLOps Engineer cannot anticipate a model issue the data scientist knew about? Or a data issue the Data Engineer knew about?

These organizational challenges can be solved, in part, with central artifact/job management and a feature store.

# Filling the Gaps with Iguazio's Feature Store

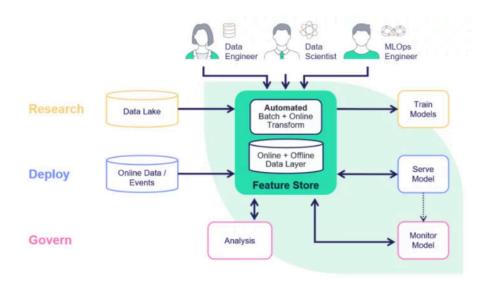
The feature store is a relatively recent concept in the world of MLOps. The purpose and function of a feature store is slightly different depending on who is asked. However, most can agree on the main definition: a central place where teams can store and access features, and share them across projects. You can read more about the necessity of feature stores for scaling data science here.



For Iguazio's feature store, storing and retrieving features is the bare minimum. Also available out of the box are custom batch/real-time pipelines to transform data upon ingestion, dual storage formats to facilitate batch and real-time workloads, and integration with model monitoring and serving.

This allows the feature store to be used as a central communication plane across projects, jobs, models, artifacts, transformation pipelines, etc. The Iguazio Feature Store also functions as a data transformation service, enabling complex feature calculations such as the customers' mean purchases in the last 24 hours, financial transactions in the last 12 hours, or other sliding window aggregations.

Not only does this standardize the overall ML workflow, it uniquely benefits each team in the pipeline:



- Data Engineer: Allows for batch/real-time data ingestion and transformation pipelines
- Data Scientist: Allows for easily accessing features and reducing duplicated work, and the harnessing of complex real-time features for their predictions
- MLOps Engineer: Allows for access of features in real-time for model serving and monitoring

## Next Steps: Iguazio + Azure

Despite the number of fantastic services available in the Azure ecosystem, a feature store as described above is not one of them. Additionally, due to the inherent nature of cloud services, they typically cannot be used for on-premise workloads or serving. This is where the Iguazio MLOps Platform can fill in some gaps.

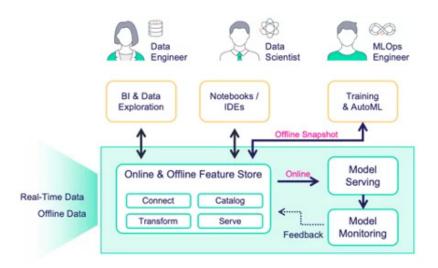
In the next three sections, we will build out a full end-to-end hybrid cloud ML workflow using features from Azure and Iguazio.



# Part 2: Data Ingestion + Transformation into the Iguazio Feature Store



While some feature stores are more focused in their functionality, Iguazio's feature store is designed to facilitate the entire end-to-end workflow:



In part one, we discussed the Iguazio feature store at a high level. The functionalities include (but are not limited to) the following:

- Ingest and transform data sources in batch or real-time
- Easily retrieve features in batch or real-time
- Dual storage formats to facilitate batch and real-time workloads
- Complex real-time feature engineering (eg. Sliding window aggregations)
- Integration with model monitoring
- Integration with model serving

We will be exploring all of this functionality in this e-book.



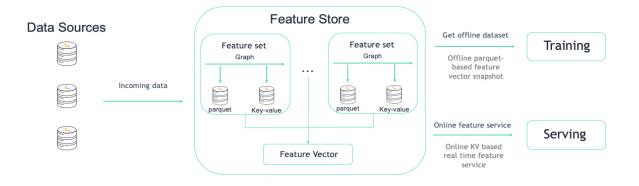
## Iguazio Terminology: FeatureSet and FeatureVector

The Iguazio feature store introduces two new terms:

- FeatureSet: group of features from one data source (file, data frame, table, etc.)
- FeatureVector: group of features from one or more FeatureSets (i.e. a few columns from here, a few columns from there)

These act as the building blocks of the Iguazio feature store and will be used heavily throughout this blog series.

They fit into the overall picture as described in this diagram:



Each data source will be ingested as a `FeatureSet` with an optional transformation graph. Once ingested, the data will be stored as parquet for offline usage and KV for online usage.

From there, a `FeatureVector` is created as a combination of multiple `FeatureSet`s. This allows for simple joining of multiple `FeatureSet`'s during retrieval using either the offline or online file format.

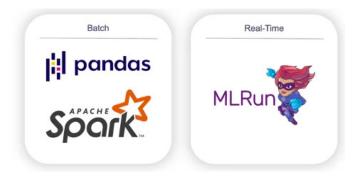
You can read more about the Feature Store, **FeatureSet** and **FeatureVector** in the <u>MLRun</u> <u>documentation</u>.

## Feature Store Transformation Engines

Not only is the Iguazio feature store useful for ingesting and cataloging features, it can also be used to perform data transformations in batch and real time. These transformations will be added to the `FeatureSet` in the form of a transformation graph, where each step is a Python class. The Python class itself can be a pre-built class or a custom implementation.



What exactly the Python class does depends on the desired transformation engine. Currently, the feature store supports the following:



#### • Pandas

- Batch transformations that fit into memory
- Input Pandas data frame → transform → output pandas dataframe
- Spark
  - Batch transformations that to not fit into memory
  - Input Spark data frame → transform → output Spark data frame
- MLRun
  - Real-time transformations (one record at a time)
  - Input OrderedDict → transform → output OrderedDict

Implementing each of the different engines is quite simple and explained fully in the Feature Set transformation documentation.

## Dataset Overview

For this demo, we will use the Heart Disease Dataset from the UCI Machine Learning Repository to train a model that detects whether a given patient has heart disease.

For the purposes of this demo, we have generated a unique ID per record called `patient\_id`, separated the features into a file titled `features.csv`, and separated the target into a file titled `target.csv`. We will be using these CSV files as the inputs for our feature store ingestion.

## Feature Store Resources to Create

For this project, we will be creating the following resources in the feature store:

- FeatureSet #1-`heart disease features`
  - Ingest data from local `features.csv` file
  - Perform transformations in real-time to incoming data
- FeatureSet #2 `heart\_disease\_target`
- Ingest data from local `target.csv` file
- No transformations
- `FeatureVector` `heart\_disease\_vec`
- Combine all features from `heart\_disease\_features` and target variable from `heart\_ disease\_target` into singular dataset for training and serving



## Ingest and Transform Feature Sets

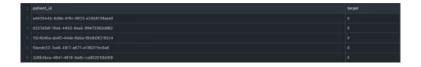
Let's get into the actual code, starting with imports and creating the project:

```
# MLRun imports
import mlrun.feature_store as fstore
from mlrun.feature_store.steps import MapValues, OneHotEncoder
from mlrun.datastore.sources import CSVSource
from mlrun import get_or_create_project
# Create MLRun project
project = get_or_create_project(name="azure-fs-demo", context="./")
```

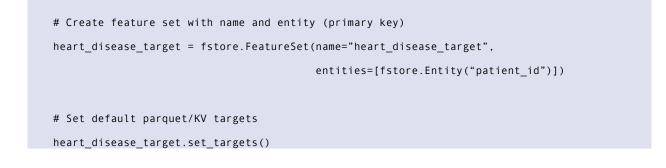
#### FeatureSet #2 Ingestion - `heart\_disease\_target`

The `FeatureSet` ingestion itself is fairly straightforward. We will start with `heart\_disease\_target` as there is no transformation component.

The raw `target.csv` dataset looks like this:



First, create desired `FeatureSet`:





To ingest, we simply pass in our newly created `FeatureSet` and desired source:



## FeatureSet #1 Ingestion/Transformation - `heart\_disease\_ features`

Next, we will ingest `heart\_disease\_features` and specify a transformation graph using the real-time engine. This will allow us to transform records as we ingest (in addition to re-using this graph later in the project during model serving).

The raw `features.csv` dataset looks like this:

1. patient_id			treathpa	chel		restoog	thelach	exang	aldpeak	viope	thei
2 #4435448-829#-d%c-9523-e345613%axe0											
3 82274341-1640-4452-8+46-004723824962	mate	typical_angina								upstoping	
1 10:4543a-a540-45de-8ata-6500060100:4		typical_angina									
5 Kacdu22-7ee8-4817-a671-e1363116xGa8										downstoping	
1 245535cs-4541-5018-9a8c-ca9020109000	female.	typical_angina			Trae					Fat.	reversable_date

First, create desired `FeatureSet` with transformation graph:



```
# Transformation mapping - specific to MapValues component
    age_mapping = { 'age': { 'ranges': [{ 'range': [0, 3], "value": "toddler"},
                                       {'range': [3, 18], "value": "child"},
                                       {'range': [18, 65], "value": "adult"},
                                       {'range': [65, 120], "value": "elder"}]}}
    # Transformation mapping - specific to OneHotEncoder component
    one_hot_encoder_mapping = {'age_mapped': ['toddler', 'child', 'adult', 'elder'],
                                'sex': ['male', 'female'],
                                'cp': ['typical_angina', 'atypical_angina', 'non_anginal_pain',
'asymtomatic'],
                                'exang': ['no', 'yes'],
                                'fbs': [False, True],
                                'slope': ['downsloping', 'upsloping', 'flat'],
                                'thal': ['normal', 'reversable_defect', 'fixed_defect']}
    # Add Python classes to transformation graph (with configuration)
    heart_disease_features.graph.to(MapValues(mapping=age_mapping, with_original_features=True))\
                                 .to(OneHotEncoder(mapping=one_hot_encoder_mapping))
    # Print out transformation graph
    heart_disease_features.plot(rankdir='LR', with_targets=True)
                                             parquet
    start
              MapValues
                             OneHotEncoder
                                              nosql
Like before, to ingest we simply pass in our newly created FeatureSet and desired source:
    # CSV source - customizable
```

```
source = CSVSource("mycsv", path="features.csv")
# Ingest into feature store - returns transformed dataframe
resp = fstore.ingest(featureset=heart_disease_features, source=source)
```

🕽 iguazio

FIND OUT MORE ON www.iguazio.com

# Print top 5 rows of newly ingested data

# resp.head() age.mapped\_tx88 age.mapped

5 rows × 27 columns

## Create FeatureVector

Now that we have two `FeatureSet`s, we will combine them into a singular dataset called a `FeatureVector`. The backend API will handle joining multiple data sources for us.

Creating the `FeatureVector` is quite simple as well:

```
# Create feature vector with list of features + desired label
# Specify in format of FEATURE_SET.FEATURE (or * for all)
vector = fstore.FeatureVector(
    name="heart_disease_vec",
    features=["heart_disease_features.*"],
    label_feature="heart_disease_target.target",
    with_indexes=True
)
```

That's it.

## **Retrieve Offline Features**

To retrieve the `FeatureVector` using the offline (parquet) format, use the following:

# Retrieve feature vector as pandas dataframe

# Specify in format of PROJECT/FEATURE\_VECTOR

fstore.get\_offline\_features("azure-fs-demo/heart\_disease\_vec").to\_dataframe().head()



	age_mapped_toddler	age_mapped_child	age_mapped_adult	age_mapped_elder	836	sex_male	sex_female	cp_typical_angina	cp_atypical_s	p_non_anginal_pain	03	ang_yes	oldpeak	slope_downsloping	slope_upsloping	slope_flat	ca	thal_normal	thal_reversable_detect	thal_fixed_defect	target
patient_id																					
e443544b- 8d9e-4%c- 9623- e24b6139aae0	0	0	1	0	52	1	0	1	0		0		1.0	1	0	0	2.0	1	0	0	٥
8227d3df- 16ab-4452- 8ea5- 99472362d982	0	0	1.	o	53	1	0	1	0		1		3.1	•	1	o	0.0	1	0	a	0
10c4b4ba- ab40-44de- Baba- Godb062192o4	0	0	0	1	70	1	0	1	0		1		2.6	0	1	0	0.0	1	0	0	0
f0acdc22- 7ee6-4017- a671- e136211bc0a6	0	0	1	0	61	1	0	1	0		0		0.0	1	0	0	1.0	1	0	0	0
2d6b3bca- 4341-4618- 9a8c- cx602010b009	0	o	1	o	62	0	1	1	0		0		1.9	•	0	,	3.0	0	,	0	•

5 rows × 28 columns

## **Retrieve Online Features**

In addition to offline features, we can utilize the online feature service that uses the online (KV) format:

- # Retrieve online feature service
- # Specify in format of PROJECT/FEATURE\_VECTOR

```
feature_service = fstore.get_online_feature_service("azure-fs-demo/heart_disease_vec")
```

This allows us to do things like retrieve statistical information:

```
feature_service.vector.get_stats_table().head()
```

	count	mean	std	min	max	hist
age_mapped_toddler	968.0	0.000000	0.000000	0.0	0.0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 968, 0, 0, 0,
age_mapped_child	968.0	0.000000	0.000000	0.0	0.0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 968, 0, 0, 0,
age_mapped_adult	968.0	0.875000	0.330890	0.0	1.0	[[121, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
age_mapped_elder	968.0	0.125000	0.330890	0.0	1.0	[[847, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
age	968.0	54.073347	9.127036	29.0	77.0	[[4, 0, 21, 18, 25, 84, 61, 64, 38, 79, 98, 89

Or retrieve individual records with real-time speed:

```
# Retrieve record via entity (primary key)
feature_service.get([{"patient_id" : "e443544b-8d9e-4f6c-9623-e24b6139aae0"}])

[{'patient_id': 'e443544b-8d9e-4f6c-9623-e24b6139aae0',
    'age_mapped_child': 0,
    'age_mapped_adult': 1,
    'age_mapped_adult': 1,
    'age_mapped_elder': 0,
    'age': 52,
    'sex_male': 1,
    'sex_female': 0,
    'cp_typical_angina': 1,
    'cp_atypical_angina': 0,
    'cp_asymtomatic': 0,
    'trestbps': 125,
    'chol': 212,
    'fbs_Frue': 0,
    'restecg': 1,
    'fbs_frue': 0,
    'cexang_no': 1,
    'sang_wes': 0,
    'oldpeak': 1.0,
    'slope_downsloping': 1,
    'slope_upsloping': 0,
    'slope_flat': 0,
    'thal_normal': 1,
    'thal_reversable_defect': 0,
    'thal_fixed_defect': 0]
```



## Feature Store UI

Finally, we are able to view everything we've done in the Iguazio project UI:

Projects > azure-fs-den	mo > Feature Store (Beta)	
Feeture Sets - Feetures	Peelure Vectors Defavets	
Name:	Labet key1.Jeg2/vesker	c
Name	heart_disease_features	Cancel Apply Changes B
~ heart,disease,features	25 Oct, 15-46:44 .	
+ latest	Dverview Features Transformations Preview Statistics Analysis	
<ul> <li>heart_disease_target</li> </ul>		Configuration
+ latest	Source	
		> Function template
		> Source
	· · · · · · · · · · · · · · · · · · ·	v Steps
	MapMalues	Selected Shep
		O Tank O Revolt O Q
	OneHotEncoder	After + On Error (Optional)
Projects		8 0
Projects > azure-fs-den	no > Feature Store (Beta)	
Feature Sets Features	Feature Vectors Datasets	
Name	Label keyl Jeg2-value,	o :
Name		
	heart_disease_target	Centrol Apply Changes
<ul> <li>heart_disease_features</li> </ul>	25 Oct, 15-47-03 •	
+ latest	Overview Peatures Transformations Preview Statistics Analysis	
v heart_disease_target	matters id	
	patient_id target	
+ latest	patient;,13 carpet = e443544b.849e-4fbc:9623.e34bb139ase0 0	
+ latest		
+ latest	o443544b.849e-4fbc-9623-e24bb139aae0 0	
e latest	#443544b.847be48b;9822.942464139aae0         0           82227830716ab;4452,8468,998723628982         0           100458b;e346-4436-48ab;40cb0239204         0           100458b;e346-4436-48ab;40cb0239204         0           10ac6227;ee6-4817;ec71 e1302116cbab         0	
e brest	#443544b.849e.48b.9822.942463139aae0         0           R2222530714a8-4852.8ea8-998723823982         0           10x0630a-944-6456-8aa-60x500231204         0           10x06222.7ea5-4817.ed571.e135211bc3a6         0           206680aca.4847.4639.9a8r.ce0220730509         0	
• latest	#443544b.849e.4159.9829.#24691398ae0         0           R2229530F14ab-4452.8ea6.49847238329882         0           10x1040x-0446.464e.484x-60x500279204         0           10xa0x222.7ea6-4817.4659.9a8r-ce9220730009         0           26660bce.4447.4659.9a8r-ce9220730009         0           386612x2.257bc4ab.60x0.00x264515         5	
• latest	#443544b.849e.419.09229.e24641378aae0         0           822225301114ab-4452.84e8.4984723832882         0           10x1040xe.9446-4484e.4884.948723832882         0           10x1040xe.9446-4484e.4884.9502712024         0           208600xe.4847.4659.9484.e0020732009         0           38461228.2926.4884.960.902302032009         0           38461228.2926.4884.960.9024288457570264         8           08050x116777b-4204xe864647570264         0	
• latest	#443544b.84bc.44bc.9422.824843138aan3         0           82229364714abc.4452.8428.9947233282882         0           50x08bba-8446.448bc.482bb02379204         0           846522.7ee64317.82771e134211bc8a6         0           2866502.82845.4459.984720730009         0           2866502.82845.4459.984720730009         0           2866502.82845.4459.984720730009         0           2866502.92874.4459.9848c.e09220730009         0           2866502.92874.4459.9848c.e0920730009         0           2866502.92874.4459.98475.9515         8           0065ce143775-4206-8364.646ea54757bc4         0           3945bef1.056e-4399.9431.23857bs156         0	
• locat	#443544b.849e.419.09229.e24641378aae0         0           822225301114ab-4452.84e8.4984723832882         0           10x1040xe.9446-4484e.4884.948723832882         0           10x1040xe.9446-4484e.4884.9502712024         0           208600xe.4847.4659.9484.e0020732009         0           38461228.2926.4884.960.902302032009         0           38461228.2926.4884.960.9024288457570264         8           08050x116777b-4204xe864647570264         0	
	#443544b.84bc.44bc.9422.824843138aan3         0           82229364714abc.4452.8428.9947233282882         0           50x08bba-8446.448bc.482bb02379204         0           846522.7ee64317.82771e134211bc8a6         0           2866502.82845.4459.984720730009         0           2866502.82845.4459.984720730009         0           2866502.82845.4459.984720730009         0           2866502.92874.4459.9848c.e09220730009         0           2866502.92874.4459.9848c.e0920730009         0           2866502.92874.4459.98475.9515         8           0065ce143775-4206-8364.646ea54757bc4         0           3945bef1.056e-4399.9431.23857bs156         0	
Projects	#44554bi #3bi #45:9423 #2454139aan3         0           #2275bit714abi #452 #485 #947233028982         0           \$50:685ba #846 #45b #63b0279204         0           \$50:685ba #846 #45b #63b0279204         0           \$84:0522 7ee6 #437 #4751 #135211bc546         0           \$28:6520 #245 #446 #45b #64b02792040         0           \$28:6520 #245 #456 #64b #65b #64b02792040         0           \$28:6520 #245 #456 #64b02795040         0           \$28:6520 #245 #456 #64b02795040         0           \$29:65bc#10075b-204+0456 464b02795040         0           \$29:65bc#10075b-204+0245 464b02795040         0           \$29:65bc#10056+0215 #5051 hotho00191         0           \$29:65bc#10056+0215 64615 hotho001912e         0	å ¢
Projects	#443544b.84bc.44bc.9422.824843138aan3         0           82229364714abc.4452.8428.9947233282882         0           50x08bba-8446.444bc.482bc.95073004         0           8460522.7665431782714130211bc.846         0           286603bac.84541.44599.842502730009         0           2866120.237bc.8466.4569575010         0           2866120.237bc.8466.4569575010         0           2866120.237bc.8466.456857557bc4         0           2965bc11056c4339.9431.2359.15c58039         0	å ¢
Projects	#44554bi #3bi #45: 9422 #2454139aan3         0           #2275bit714abi #452 #485 #94723502882         0           \$50:68bb #346 #45b #45b 593723501         0           \$50:68bb #346 #45b #45b 593723501         0           \$50:68bb #346 #45b #45b 593723501         0           \$84:522 meth-4317 #4359 148b exe0520735050         0           \$28:6120_237b #438 #600 502736509         0           \$36:6120_237b #439 #431 *2356 h505         5           \$36:6120_237b #439 #431 *2356 h505         0           \$36:6120_237b #439 #431 *2356 h505         0           \$36:6120_4305 #431 *5 4568 h505 h506         0           \$36:6120_4305 #431 *5 4568 h506 h505 h506         0           \$36:6120_4305 #431 *5 4561 h506 h507 h506         0	å ¢
Projects Projects → azure-fs-derr Petern Sets – Peternes	e4455445-845e-485e-9837293039882     0     8259439714ab-4852-8468-9937293039882     0     90x66222-7466-4817-48279-47393039882     0     90x66222-7466-4817-48279-473930318e-046     0     36665bae-4841-4858-8488-047302783609     0     38665328-2485-4816-48250783609     0     38665328-2485-4816-48250783609     0     396654748-008-3339-043-24856488     0     0     9     Peduture Store (Beta)     Datasets     Pedure Viscler     Datasets	Å ¢
Projects Projects > azure-fs-derr Pesture Sets Pestares Top Intest +	e4455445-845e-485e-9837293039842     0     8259439714a8-4852-8468-9937293039842     0     90x662227496-4817-6879-47293039842     0     90x662227496-4817-6879-47293039842     0     90x662227496-4817-6879-487020739009     0     38665208-2884-4836-680207384     0	
Projects Projects > azure-fs-derr Perture Bets Pertures Sig. Intext * Name Deer dease ac	#443544b-84bc-9422-82454139aac0         0           82593687164b-9452-824549347234028982         0           100654bc-9546-454b-9347234028982         0           100654bc-9546-454b-9347234028982         0           100654bc-9546-454b-9347234028982         0           100654bc-9546-454b-9347234028982         0           100654bc-9546-454b-9347234028982         0           100654bc-9546-454b-9347234028982         0           100654bc-9546-454b-934726050         0           100654bc-9547157b-10         0           100654bc-9547157b-10         0           100654bc-954715-8517b-951250         0           100654bc-954715-8517b-951250         0           100654bc-954715-8517b-951250         0           10055         Feeture Store (Beta)           Norm:         Labritiky1/kyg2-sabac           heart_disease_vec         Labritiky1/kyg2-sabac	
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#445544b-847be-448b-948723828982         0           822293697164b-6452-4684b-948723828982         0           1500854b-9484-644b-642b-920738209         0           1500854b-9484-644b-642b-920738209         0           1500854b-9484-644b-642b-920738209         0           1500854b-9484-644b-642b-920738209         0           1500854b-9484-644b-642b-920738209         0           1500854b-9484-644b-642b-920738209         0           256452b-9347-942b-042b-920738209         0           256452b-9347-942b-042b-648b-6468257b-7b-04         0           2765berth C356e-43399-9437-9235013b-056059         0           26452bell-6495-4315-64817-0b-100019915b         0           1500 > Feature Store (Beta)         0	a
Projects Projects > azure-fs-derr Perture Bets Pertures Sig. Intext * Name Deer dease ac	#443544b.4912-9421-943139aac0         0           #252954b7.194b-4452.4824.9947/33028482         0           \$5005656.9946-4946.9947/33028482         0           \$5005656.9946-4946.9947/33028482         0           \$5005656.9946-49520735000         0           \$3045120.2937.4130321100486         0           \$3045120.2937.44349.600.091200.87501         0           \$3045120.2937.44349.600.091200.87501         0           \$3045120.2937.44349.600.091200.87501         0           \$3045120.2937.44349.600.091200.87501         0           \$3045120.2937.44349.600.091200.87501         0           \$3045120.2937.44349.600.091200.87501         0           \$3045120.2937.44349.600.091200.87501         0           \$3045120.2937.44349.600.091200.87501         0           \$3045120.2937.4435.6437.56451.2951.2951.295         0           \$3045120.916.6435.4931.56451.56450.5760         0           \$3045120.916.6435.4931.56451.56450.5761         0           \$3045120.916.6435.4931.56451.56451.56451.56450.5981.56         0           \$3045120.916.6435.4931.56451.56451.56451.56450.5981.56         0           \$3045120.917.6431.591.64513.56451.56451.56451.56451.56451.56451.56451.56451.56451.56451.56451.56451.56451.56451.56451.56451.56551.56451.56551.56451.565551.56551.565551.565551.565551.565551.5655555555	a
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#443544b.4912-842b.49139aac0         0           #201954714ab-4852.482b.991733028482         0           \$201954714ab-4852.482b.991733028482         0           \$201954714ab-4852.482b.991733028482         0           \$201954714ab-4852.482b.991733028482         0           \$20195472ab-4845.482b.882b.9917330291         0           \$2019582ab-9847.482b.882b.991738029         0           \$2019582ab-9847.482b.882b.991738029         0           \$2019582ab-9847.482b.882b.991738059         0           \$2019584714072b-420b-482b.50757264         0           \$2019584716056e-43930.9433.235619266939         0           \$20195847164845-8817584573584         0           \$20195847164865-8317584573584         0           \$20195847164865-8317584573584         0           \$20195847164865-831758457584         0           \$201958471564837584         0           \$201958471564837584         0           \$2019584715648375844         0           \$201958471564837584         0           \$2019585         \$201947156485           \$2019585         \$201947156485           \$2019585         \$201947156485           \$2019585         \$201947156485           \$2019585         \$201947156485           \$201	Cause Apply Okeyam I
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#445546-845e46669-94723828982         0           #20295871646-64524e364994723828982         0           \$5008568-9546-458-645060299204         0           \$5008568-9546-458-645060299204         0           \$5008568-9546-458-645060299204         0           \$5008568-9546-458-645060299204         0           \$5008568-9546-458-645060299204         0           \$5008568-9546-458-645060299204         0           \$5008568-9546-458-645060299204         0           \$5008568-9547-94579-42050-85515         5           \$500856417575-4205-43505566         0           \$2955647-15664-32059-9437-255073606059         0           \$6055411755-4205-94575564         0           \$2955647-15664-3205         0           \$6055411755-4205         0           \$6055411756-42059         0           \$60554115755-4205         0           \$6055411566457         0           \$6055411564637         0           \$6055411564637         0           \$6055411564637         0           \$605541156481-0         0	Cator Aphi Olargan E sabala Tap Preg Histogram as
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#445546b #49b 4(4b):94229 ±2454139aac0         0           #2529367 164b 4452 #64b 99817234028982         0           150:0545b 9546 4456 443b 463b 99817234028982         0           150:0545b 9546 4456 443b 463b 95817234028982         0           150:0545b 9546 4456 443b 463b 95817234028982         0           150:0545b 9546 4456 443b 463b 95817334028982         0           150:0545b 9546 4456 443b 4630 952018         0           150:0545b 9546 4456 4431 45035 1506 0         0           150:05547 15056 4300 9430 9431 235615b 1506 0         0           150:05471 15056 4300 9431 235615b 1506 0         0           150:05487 15066 4300 9431 235615b 1506 0035         0           150:05120 1375 44035 4431 56451 150 8455         0           150:05 Feature Store (Beta)         0           150:05 150 150 8637         0           150:05 150 150 8637         0           150:05 150 8637         0           150:05 150 8637         0           150:05 150 8637         0           150:05 150 8637         0           150:05 150 8637         0           150:05 150 8637         0           150:05 150 8637         0           150:05 150 8637         0           150:05 150 160         150 8638 837 84018	Constr. Apply Chargen E whysie Tap Preq Histogram 6 100
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#445546-845e-845e-845e-845135aae3         0           #25295871646-8452-8464598723629882         0           \$52095871646-8452-8464598723629882         0           \$52095871646-8452-846459987236298         0           \$52095871646-8452-846450379201         0           \$52095871647546798201         0           \$5209587164646-8458-845050203200500         0           \$58055220-2354-8456-84520738000         0           \$58055220-2354-8456-84520738000         0           \$58055220-2354-8456-84575640         0           \$295584716266-8330-9437-235673520         0           \$6857284-8456-84312-8457564         0           \$295584716266e-8330-9437-235673520         0           \$6857284-8456-83715-84573-84700278824         0           \$6857284-8456-83715-84573-84700278824         0           \$7045987         \$28645837564           \$80451-81175-84573-84700278824         0           \$80451-81175-84573         0           \$80451-81175-84573         0           \$80451-8115-84573         0           \$80451-8115-84573         0           \$80451-8115-84573         0           \$80451-8115-84573         8           \$80451-8115-84574         8           \$80451-81178         8	Concer Apply Charges E solyddi Top Preq Hotogram 6 L 200 8 J 200 8 J 200
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#445546-849e4/8496429-82153628882         0           #252958671646-8452-8645-981723628882         0           15205858-984723628882         0           15205858-984723628882         0           15205858-984723628882         0           15205858-984723628882         0           15205858-98472362882         0           15205858-9847236289         0           15205958-948174679-913231100586         0           15205958-948174679-9132301100586         0           15205958-9481746797493029120039         0           1520596471520-2315-0630732009         0           1520596471520-2315-0630732009         0           1520596471520-2315-0637372604         0           1520596471520-2315-06373572604         0           15205971520509         0           15205971520509         0           1520597152050971520509         0           152059715205097152050971520509         0           152059715205         1           152059715205         1           152059715205         1	Concer Apply Changes E adjulis Top Final Hologram Main E No. 1 0 0 0
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#445546-845e-4816-981738xee3         0           #2529587164-6452-824545138xee3         0           #2529587164-6452-8245-981723829882         0           #2629587164-6452-8245-981723829882         0           #2629587164-6452-8245-991723829882         0           #26295871642-8454-6458-9817238209882         0           #26295871642-8454-6458-9917238209         0           #26295871642-8454-6458-9917238209         0           #262958716420-237126003         0           #26295871777-4209-8454-6458-0737260         0           #2629587284-6495-6312-84512-845073826         0           #2629587284-6495-6312-84512-845073826         0           #26295887284-6495-6312-84512-845029         0           #26295887284-6495-6312-84512-845029         0           #26295887284-6495-6312-84512-845029         0           #26295887284-6495-6312-84512-84512-845029         2           #26295887284-6495-6312-84512-84512-845029         4           Marm:         Laber:kry1,3xy2-sabac         2           Marm:         Laber:kry1,3xy2-sabac         4           Marm:         Count         Marm:         Marm:         Marm:           Versiver         Regarmanged_scille         0         8         4           Marm:	Concer Apply Changen E subjets Top Prog Histogram So 1 40 0 0 0 1 0
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#4455465-8452-42451338aad3         0           #22750071640-6452-424541338aad3         0           #22750071640-6452-424549847233028847         0           No.0635a-9447-2457914322116c8a6         0           2945040-6457-457914322116c8a6         0           2945040-6457-457914322116c8a6         0           2945040-645714532116c8a6         0           2945040-645714532116c8a6         0           2945040-645714532116c8a6         0           2945040-64571453203732050         0           2945040-645714559         0           2945040-645714504         0           2945040-64571656079         0           29450471636e-432959437 23359150-66079         0           29450471636e-432959437 23359150-66079         0           Americ         Laber:kyr1/kyr2+eable           Norrer:         Laber:kyr1/kyr2+eable           Norr	Canor Apply Chargen E adyols Top Preq Honogram 6 1 80 80 1 80 1 80 1 80 1 80 1 80 1 80 1
Projects Projects > azure-fs-derr Perture Sets Freshres Say, Intent • Name > * bated	#445546-845e-4816-981738xee3         0           #2529587164-6452-824545138xee3         0           #2529587164-6452-8245-981723829882         0           #2629587164-6452-8245-981723829882         0           #2629587164-6452-8245-991723829882         0           #26295871642-8454-6458-9817238209882         0           #26295871642-8454-6458-9917238209         0           #26295871642-8454-6458-9917238209         0           #262958716420-237126003         0           #26295871777-4209-8454-6458-0737260         0           #2629587284-6495-6312-84512-845073826         0           #2629587284-6495-6312-84512-845073826         0           #26295887284-6495-6312-84512-845029         0           #26295887284-6495-6312-84512-845029         0           #26295887284-6495-6312-84512-845029         0           #26295887284-6495-6312-84512-84512-845029         2           #26295887284-6495-6312-84512-84512-845029         4           Marm:         Laber:kry1,3xy2-sabac         2           Marm:         Laber:kry1,3xy2-sabac         4           Marm:         Count         Marm:         Marm:         Marm:           Versiver         Regarmanged_scille         0         8         4           Marm:	Conterne Apply Changen E andydda Tep Freq Horlegwen 40 40 40 1 40 1 1 1 1 1 1 1 1 1 1 1 1 1

# Next Steps

In the next two blogs, we will build upon the features we just ingested to build out a full end-to-end hybrid cloud ML workflow using features from Azure and from Iguazio.



# Part 3: Model Training with Azure ML

In this part, we will:

- Upload data from Iguazio into Azure and register dataset in Azure ML
- Train several models in Azure using Auto ML
- Retrieve trained models from Azure back to Iguazio
- Log trained models with experiment tracking and metrics

## MLRun Functions: Overview

Before running any code, we need to take a moment to discuss one of the core tenants of the Iguazio platform - the MLRun function.

This abstraction allows for simple containerization and deployment of code. Users are able to execute workloads on Kubernetes by specifying code and configuration options via high level Python syntax. An MLRun function will look something like this:



The function will have its own name/tag, code, docker image, resources, and runtime. The syntax for creating an MLRun function looks like the following:



From there, the code can be executed locally in Jupyter or on the cluster using several runtime engines including Job, Spark, Dask, Horovod, and Nuclio real-time functions.



## Create Azure MLRun Function from Python File

Now that we have the background on what an MLRun function is, we are going to create one with our Azure code. We have written a Python file called `azure\_automl.py` that performs several tasks such as:

- Upload the dataset from the feature store
- Register the dataset in AzureML
- Execute a training job using Azure AutoML
- Download the trained models back to Iguazio
- Log models with experiment tracking metadata such as labels and metrics

First we will configure our code to use the same project that our features reside in:

```
import mlrun
from mlrun import get_or_create_project, code_to_function, build_function, run_function
project = get_or_create_project(name="azure-fs-demo", context="./")
function
```

Next, we will use `code\_to\_function` from the MLRun library to convert our Python file into an MLRun function:

This will create our MLRun function and store it in the project. Now, we can execute our code on top of the Kubernetes cluster with ease.

## Build Docker Image

However, the code requires some Python packages for Azure services that are not included in the default `mlrun/mlrun` Docker image. We can easily build our docker image and update our function with the following:

build_function(	
function="azure",	# Name of MLRun function we created
<pre>skip_deployed=False,</pre>	# Force rebuild
with_mlrun=False,	<pre># MLRun already installed - no need to reinstall</pre>
<pre>base_image="mlrun/mlrun:0.8.0",</pre>	# Base Docker image
<pre>requirements="requirements.txt"</pre>	# Required packages



Because of how we configured our MLRun function, this one line of code will do the following:

- Fetch the function titled `azure` from our project
- Start building an image with the base image `mlrun/mlrun:0.8.0`
- Install packages specified in `requirements.txt`
- Update the `azure` function in the project with the newly built image

## View MLRun Function in Project

Now that we have created our MLRun function and built the required Docker image, we can see what we built in the MLRun UI:

Projects	
Projects > azure-fs-	demo > ML functions
Name:	Show untapped
Name	
22.50	azure
> latest Jan 25, 12:56:20 PM	Jan 25, 12:56:20 PM
	Overview Code Build Log
	Name 22370
	Kind job
	Hash 2xdb657bb107cd0e25ee2a66a990536559fe0e2
	Code origin https://github.com//gz-us-sales/igz-blogs.gitHSDe2729e8d3937ecf735fcd3aab3cb3dfbf5790.azurent_stills.gy
	Updated Jan 25, 12, 58, 20 PM
	Command
	Image minur/fano azure/is-demo-azure/aitest
	Description

Here we can see some of the high level information including the name, code origin, runtime, and image. Additionally, we can see the code itself:

1	Projects > azure-fs-	demo > ML functions
	Name:	Show untagged
	Name	azure
	> latest Jan 25, 12,58,20 PM	Jan 25, 12:58:20 PM .
		Overview Code Build Log
		import on
		import json import logging
		from typing import Tuple, List
		from mlrun import MLClientCtx, DataItem, get dataitem
		import mlrun.feature_store as f_store
		from mlrun.api.schemas import ObjectKind
		from azureml.core.authentication import ServicePrincipalAuthentication
		from azurenl.core.workspace import Workspace
		from azureml.core.experiment import Experiment
		<pre>from azurenl.core.dataset import Dataset from azurenl.core.model import Nodel</pre>
		from azurenl.core.compute import ComputeTarget, AmlCompute
		from azureml.core.compute target import ComputeTargetException
		from azureml.core.script_run import ScriptRun
		from azureml.train.automl import AutoMLConfig
		from azureml.train.automl.run import AutoMLRun
		<pre>def _env_or_secret(context, key):</pre>
		if key in os.environ: return os.environ[key]



Note that these MLRun functions can be versioned. This is not a replacement for version control software like Git, but is a useful tool to track which models are tied to which version and how your code has changed over time.

## Add Kubernetes Secrets

An additional step before executing our code is to create some project secrets. This will allow us to securely retrieve the Azure credentials without storing sensitive information in the code itself. We can easily do this like so:

This only ever needs to be run once to store the credentials in a Kubernetes secret. **Do not commit these** secrets to version control as they will allow anyone to spin up Azure resources.

To insert our secrets into the MLRun function, we will create a MLRun task like so:

We will use this when executing the code itself. Notice that we are only specifying the names of the secrets we want to retrieve - the values are stored securely as a Kubernetes secret.



## Configure AutoML Job

Before executing our job, we need to specify what we want to run. Because we have wrapped all the functionality in our MLRun function, we only need to pass in some configuration parameters to our function.

There are many configuration parameters in this MLRun function including:

- Which `FeatureVector` or dataset to use
- Registered dataset name and description in Azure
- Azure experiment and compute settings
- Registered model name and AutoML settings in Azure
- How many models to save to Iguazio

The most important settings for configuring the job itself are specified here:

```
# Azure ML settings for model training
automl_settings = {
    "task": 'classification',
    "enable_early_stopping" : False,
    "allowed_models": ['LogisticRegression', 'SGD', 'SVM'],
    "iterations" : 5,
    "n_cross_validations": 5,
    "primary_metric": 'accuracy',
    "featurization": 'off',
    "model_explainability": False,
    "enable_voting_ensemble": False,
    "enable_stack_ensemble": False
}
# MLRun input for dataset - FeatureVector that we previously created
inputs = \{
    "dataset" : "store://feature-vectors/azure-fs-demo/heart_disease_vec:latest"
}
# MLRun parameters for job
params = \{
    "experiment_name" : "azure-iguazio-blog",
    "cpu_cluster_name" : "azureml-cpu",
    "dataset_name" : "iris",
    "dataset_description" : "iris training data",
```



```
"register_model_name": "iris-model",
"label_column_name" : "target",
"save_n_models" : 3,
"automl_settings" : automl_settings
}
```

From this configuration, we can see:

- We are doing a classification task
- We are using the `FeatureVector` from the previous blog
- The label for the training set is titled `target`
- We will train 5 models total without early stopping
- The models will be of type `LogisticRegression`, `SGD`, or `SVM`
- We will save the top 3 models based on accuracy back to Iguazio

There are many options that we can specify in `automl\_settings` - an exhaustive list can be found in the Azure documentation.

## Run Azure Auto ML

Last but certainly not least, we can run our AutoML job on Azure. The culmination of all our work so far can be expressed in a single line of code:



As explained above, this will:

- Upload the dataset from the feature store
- Register the dataset in AzureML
- Execute a training job using Azure AutoML
- Download the trained models back to Iguazio
- Log models with experiment tracking metadata such as labels and metrics



## View Azure ML Training Job

Once our Azure job kicks off, we can use the `Microsoft Azure Machine Learning Studio UI` to keep track of everything.

This includes the registered dataset:

	eritine Learning Datalos									4.0.7	C Transfer Morris	
Figure	1 Delante > Search	diana.										
heart	disease Verset	1 datests 1										
- Derive	toround the											
13.54	manine v C has	an 🕨 taranta poli	St thread line									
14.14												
	ne inche											
-		Number of save: 52 (cf										
		···· spr. mapped, child				(Francische	Wass, Sanake			Fig.m.asph.		
	a.		1	0	0	1	5		4	8		83
2					18							14
č .		C 1			-				÷		0	1.0
	Č.	÷		Č.	47			1		3 I I I I I I I I I I I I I I I I I I I	÷	1.0
č - 1	Č.	5 I I I			14		2		Č.		÷	
÷ .	č. –			Č			2	1		2	÷	1
				S			2		2	1	0	-
		0		Č		- 2		- C				
÷.					14				0		÷	12
÷.,	Q	Q	2	Q	11				÷	21	2	10
÷.	÷		2		**			1.	2	2		100
			1	<u>.</u>	14	2					Q	10
	*				11							140
16	0			0	11		0	1				128
	*		1		14		¥ .					
												140
			2	2	14			1.2				104
18												

Also the completed job output:

-	Home > Experiments > asse-guasie blog > g	reen, red, 34053495	
+ 0 8	green, rod_sl05snf9 🍠		
5	Properties.		Best model summary
	Data Completed Completed Constend Jan 25, 2022 12:41 PM Duration Jan 55:505 Compare duration Jan 65:505 Compare duration Jan 65:505 Compare stript compare stript compare stript compare stript	Sort name 	Appreten name Sandard Societ (Wappen; SMM Hypequanettes BL Van hyperpravates) Accessor Serging 1000 % © Regressed acadits In-model 135 Disploy state To disployment yet Ren sammary
	AutoML_065a2547-77aa-4494-x93e-1f7c308b	8a1a	Task type
	Tags		Classification III New configuration set     Featurization     Auto
	O No tega		Primary metric Accuracy
	Description		Deperiment name     azure-iguazio-blog
			ecore directo.ox08

This page displays a lot of useful information including the time taken, the best model, evaluation metrics, and more. We can also view the individual child runs for the model training:

	Home > Experiments > azura-iguado	blog > green,r	od_sl05sn19					
	green_rod_sl05snf9 🖉	ancel 💼 Delete						
1	Details Data guardicails Models	Outputs + logs	Child runs Scapsh	ot				
8	Display name	Status	Submitted time	Duration	Submitted by	Compute target	Run type	Taga
9	poine, potato, Silfjeds	Completed	Jan 25, 2022 12:42 PM	2m 21s	Service Principal	азыний сро	Sorigit	
Δ	heroic_chayste_conf96yd	O Completed	Jan 25, 2022 12:42 PM	95	Service Principal	aturent cpu		
9 9	sharp_sprout_k300d4vt	O Completed	Jan 25, 2022 12:42 PM	121	Service Principal	aturerti-cpu		
0	gende_spring_bk?wwfp5	Completed	Jan 25, 2022 12:42 PM	25	Service Principal	azurenti ope		
2	boring_spinach_0jytydmh	O Completed	Jan 25, 2022 12:42 PM	94	Service Principal	anoperi cpu		
2	yellow_cat_s426w78v	Completed	Jan 25, 2022 12:42 PM	01	Service Principal	azuremi-cpu		
	boring,hand,1w77webj	Completed	Jan 25, 2022 12:41 PM	431	Service Principal	atureril cpu	Script	
đ p.								



We can see there are 6 jobs total - 1 setup job and the 5 training jobs we specified. Each of these child runs has additional information on time taken, evaluation metrics, and more.

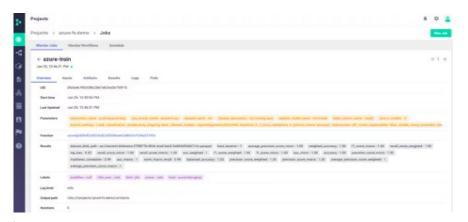
## View MLRun Job

Although Azure ML is incredibly powerful and robust, the goal of this blog series is to integrate with Iguazio and bring the models back into the platform. Within the Iguazio UI, we can also view information about the job that ran.

This includes an overview of the job with parameters, results, and even a link to the function code itself:

Paojecto	4 0
rojects > az	urorito demo 🔸 Joba
Marchar Jobs	Mandar Wardson Barnadain
← azure-tr Jan 26, 12-46-21	
Overview 1	quite Arthuris Beaulte Gage Pada
ve	dbs4k795048c38kVebuds/10915
Disting	APRICE TRACE
Last lipited	34-31(13-631 PM
Parameters	representations interpretations (socializations and interpretation) discontances in the data description in tensing data representation in tensing data rep
Punchan	energistikettestendinduschettestendintete
femile	lease (id. ph) as Associations (1967). Not and an Unit Onli Onli Onli Onli Onli Onli Onli Onli
Labels	webber of stinger oil. Bed ph. prove oils. Bed anythingpy
Legimet	aŭ
Output path	sites of property sources for demonstrations
Autofante	

Additional tabs include the model artifacts:



#### And Function logs:

Projects		٠	0
Projects > acur	-to-demo > Jaba		-
Maniter Julia	Annhar Horiffonn Setuduk		
+ azure-trail Jac25-12-621760			æ 1
<ul> <li>2012-01-02 (2012)</li> <li>2012-01-02 (2014)</li> <li>2014-01-02 (2014)&lt;</li></ul>	ME, MARTER FRANKER FROMMEN MENNENNENNEN FRANKER FROMMEN MENNENNENNENNENNENNENNENNENNENNENNENNENN	will 7-41	164-1
	substantia and substant		
TYPE: Classification of the second se	s balancing association De post annu anny part, and a'i) classes are balanced in your training date. In are alanci Malancie data Hojai (Vasa.actantaminado Sebulancedo)		



All of this information is stored per run and is available to retrieve via the UI as seen here or programmatically through a Python SDK.

## View Logged Models

Finally, we can view the models that were logged as part of this training run. We are storing the model itself plus metadata on the algorithm, evaluation metrics, and custom labels:

Inter Stategy         *         Marce         Uddes/Set/Met/velow         Interest Methods         Management Advance         Manadva									11.0	Real Time Pauline		Nodel Englished	March 1
r molel,manager (r) scores na articlette angegenericative (th installation of the scores of the scores)							ations	int		and the reason			-
				Processorie & Adjustition			Interior	factored	Delar	Protost	Labelto		Inter
) moli (upstregenin it) souters est and/(0.4019) angegening.commercell induced (1.5) April 4	. 10	*	•	Accentili, SHOWINGOUT	1.16	wepter(anney 1.00	amp.perset, sort, new 100	Jan 25, 12 46,00 PM		active tests.	165	C, normager	-
	8	*		Assentia, Lagarteritegenerice	100	seighted, annuary 3.34	amprovement and more that	Japa 25, 12-46-18 Per	nick	-	18	Unperimpretion	t mobil, i
) modi_Lingetregenese (i) powner un Jar76/04/31 Per senge_presecutor_httl wegter_ensure 146 in19 Appendix Coperchapterios		*	•	Asserted, LogisticRegression	+10	segret.envior.1.H	away person and more that	Jan 25, 12, 46, 21 PM	NOR	-	41	Lippingesse	1 mobile

We can also view more detailed information per model:

noperates		
icjects > azune-fs-dem	c > Models	Pages
Martin Made Delperints	na kad Tina Pipalina	
Ting latest +	State Libbits leg Used States	
Marter	Overview Provine Function Statistics	
> mobil.C.setwinppet	agettes Silvingon	
model, 1, hypotic agreeses	Labels Furnicol: Acceld.	
> model 2 highlinepresent	Manu mengaperan aten init menjada anany 100 Fasika nane 100 selatana nepele 130 jegana 200 metalam nene 138 selata anangketa Fasika nangketa 100 Fasika nepeleka 100 kong 100 kong 100 pengangketana nangketa 100 jengangketa manangketa 100 jengapangka nangketa 100 penganangketa 100 jengapangketana nangketa 100 jengapangketana nangketa	manu: 1
	v Roses Unes	
	Auros Publ	
	Excern discriptional failure failure failure failure des exceptions the constraints des exceptions of the exception	
	Producer	
	Note Appriati	
	End to	
	In several several database (Vince Andreas Vince Andre	

This allows us to relate metrics, models, runs, code, and custom metadata all within the experiment tracking UI.

Overall, this combination allows for powerful integration between Iguazio's feature store and Azure's highly customizable AutoML training jobs - something that no single tool can do on its own.



# Part 4: Hybrid Cloud + On-Premises Model Serving + Model Monitoring



In this final part, we will:

- Discuss the benefits of a hybrid cloud architecture
- Define model load and predict behavior
- Create a model ensemble using our top 3 trained models
- Enable real-time enrichment via the feature store during inferencing
- Deploy our model ensemble in a Jupyter notebook and on a Kubernetes cluster
- Enable model monitoring and drift detection
- View model/feature drift in specialized dashboards

## Hybrid Cloud Benefits and Motivation

Hybrid clouds are all the rage. While cloud computing has given many organizations access to near infinite compute power, the reality is that on-premise infrastructure is not going away. From data privacy concerns, to latency requirements, to simply owning hardware, there are many legitimate reasons for having an on-premise footprint.

However, with this increased flexibility comes increased complexity – the right tools are needed for the job. With the multitude of end-to-end platforms, SaaS services, and cloud offerings, it is increasingly difficult to find those correct tools—especially those that work across cloud and on-premise environments.

In the last blog of this series, we will combine the feature store and on-premise model serving of the Iguazio MLOps Platform with the models trained via Azure's AutoML to create a full end-to-end hybrid cloud ML workflow.

# Define Model Behavior

Now that we understand the motivation behind the on-premise deployment, the first step to deploying our models is to define the inferencing behavior. Essentially, how do we load our model and use it for predictions? MLRun Serving Graphs allow users to define a high level Python class for these behaviors, much like KFServing and Seldon Core.

Since all of the models we downloaded from AzureML are `pickle` files and have the same prediction syntax, this is a very straightforward class. The inferencing class resides in the `azure\_serving.py` file and includes the following:

```
import numpy as np
from cloudpickle import load
from mlrun.serving.v2_serving import V2ModelServer
```

class ClassifierModel(V2ModelServer):



```
def load(self):
    """load and initialize the model and/or other elements"""
    model_file, extra_data = self.get_model('.pkl')
    self.model = load(open(model_file, 'rb'))

def predict(self, body: dict) -> list:
    """Generate model predictions from sample"""
    print(f"Input -> {body['inputs']}")
    feats = np.asarray(body['inputs'])
    result: np.ndarray = self.model.predict(feats)
    return result.tolist()
```

Note that your class can be as complex as you need it to be with built-in hooks for `preprocessing`, `postprocessing`, `validation`, and more. For more information on MLRun Serving Graphs, check out the <u>documentation</u>.

## **Define Serving Function**

After writing our simple Python class, the MLRun framework will take care of the building, provisioning, routing, enriching, aggregating, etc.

First we will configure our code to use the same project that our features and models reside in:

```
import os
from mlrun import get_or_create_project, code_to_function
project = get_or_create_project(name="azure-fs-demo", context="./")
```

Like in the previous blog, we will be using MLRun's `code\_to\_function` capabilities to containerize and deploy our function. However, unlike last time, we will be using a different runtime engine. Instead of a batch `job`, we will be deploying a model `serving` function:

```
Configure Model Ensemble with Feature Store Enrichment
```



In addition to the initial `code\_to\_function` configuration, we will be adding additional options to the serving function. Behind the scenes, we are really constructing a graph with different topologies, preprocessing steps, models, etc.

In our case, we are looking for a model router that can:

- 1. Enrich incoming data using the feature store
- 2. Impute any missing values using statistics from the feature store
- 3. Utilize multiple models in a voting ensemble
- 4. Return a single result from the ensemble

MLRun has such a router that comes out of the box called the `EnrichmentVotingEnsemble`. The configuration looks like this:

```
serving_fn.set_topology(
    topology='router',
    class_name='mlrun.serving.routers.EnrichmentVotingEnsemble',
    name='VotingEnsemble',
    feature_vector_uri="heart_disease_vec",
    impute_policy={"*": "$mean"}
```

Notice that we are configuring the desired `FeatureVector` to enrich with as well as the imputation method for missing values.

Now that the model router is set up like we want, we can add our models. We will be referencing the models using their URI's available via the MLRun UI. The "copy URI" icon is circled in red below:

	Restaria Model Endjanist	8.5m	Real-Time Pipelines									
	Trois Series 1	mene		Labels Rer Unit-value	. Shoe bear	tiona						0
	None	Labelto	Producer	Deter	Reduced	Metros			imperature k Algorithm			
ŕ	model, 6, personages	300	accertain.	and a	Jan 28, 12-66 (8 PM	awaya person, sore, ress. 1.08	seighted, annexes 7.00	118	Aranditi, SACHINGPORT	•	*	0
	mathin, 1, Juga Perspection	1880	anata base.	-	Jan 25, 12-46 18 PM	awaya purma, sona, men. 1910	might(arrest) 1.0	1.18	Aramidi, Lagarochegenation		*	0
5	model_1.teprtrupterent	12	-	nick	Jan 25, 12 49 21 PM	aways present some non- 041	segled arrange 3.86	1.16	Asserbit. Logisticitygheason	•	-	0

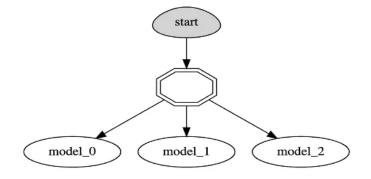
We can add the models using very simple syntax. This sets the name of the model in the graph, the desired Python class to use for load/predict behavior, as well as the path to the model itself (using the URI we copied). This looks like the following:

```
serving_fn.add_model(
    key="model_0",
    class_name="ClassifierModel",
```



```
model_path="store://models/azure-fs-demo/model_0_svcwrapper#0:latest"
)
serving_fn.add_model(
    key="model_1",
    class_name="ClassifierModel",
    model_path="store://models/azure-fs-demo/model_1_logisticregression#0:latest"
)
serving_fn.add_model(
    key="model_2",
    class_name="ClassifierModel",
    model_path="store://models/azure-fs-demo/model_2_logisticregression#0:latest"
)
```

Finally, we can save our serving function in our project and visualize the computation graph:



#### serving\_fn.save()

serving\_fn.spec.graph.plot()

Before continuing, consider the complexity of what we just set up using a few lines of code:

- Define model load and predict behavior
- Create a model ensemble using our top 3 trained models
- Enable real-time enrichment via the feature store during inferencing
- Enable real-time imputation of missing values using feature store statistics

## Test Locally in Jupyter Notebook

Now that our serving function and model ensemble are setup, we should test it out in our local Jupyter environment. MLRun makes this quite simple to do:

from azure_serving import ClassifierModel	<pre># Need our serving class available for local</pre>
server	
local_server = serving_fn.to_mock_server()	# Create local server for testing



> 2021-12-15 23:08:35,858 [info] model model\_0 was loaded > 2021-12-15 23:08:35,931 [info] model model\_1 was loaded > 2021-12-15 23:08:36,000 [info] model model\_2 was loaded

From here, we can invoke the model using one of the `patient\_ids` from our dataset. This will:

- Retrieve the corresponding record in the feature store
- Impute any missing values
- Send the fully enriched record to our model ensemble
- Aggregate and return the results of the model ensemble

This looks like the following:

```
local_server.test(
        path='/v2/models/infer',
        body={
            'inputs': [
                ["d107db82-fe26-4c02-b264-a3749510ed9b"]
            1
        }
   )
   Input -> [[0, 0, 1, 0, 62, 0, 1, 1, 0, 0, 0, 138, 294, 0, 1, 1, 106, 1, 0, 1.9, 0, 0, 1, 3.0,
0, 1, 0]]
   Input -> [[0, 0, 1, 0, 62, 0, 1, 1, 0, 0, 0, 138, 294, 0, 1, 1, 106, 1, 0, 1.9, 0, 0, 1, 3.0,
0, 1, 0]]
   Input -> [[0, 0, 1, 0, 62, 0, 1, 1, 0, 0, 0, 138, 294, 0, 1, 1, 106, 1, 0, 1.9, 0, 0, 1, 3.0,
0, 1, 0]]
    {'id': 'e36dedd8c72c48c48ff01594064c84fa',
     'model_name': 'VotingEnsemble',
     'outputs': [0],
     'model_version': 'v1'}
```

🚺 iguazio

FIND OUT MORE ON www.iguazio.com

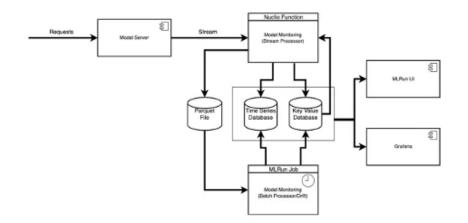
Not only do we receive the desired prediction and model information, we also see that the fully enriched record is printed out three times. This is due to the `print` statement in our `predict` function in `azure\_serving.py` being called for each of our three models.

Now that we have verified that our model ensemble is working as expected in our Jupyter environment, we need to deploy it on our Kubernetes cluster.

## Enable Model Monitoring

In addition to the deployment itself, we want to enable model monitoring and drift detection on our serving function. There are many excellent resources available on why model monitoring is important as well as how to build a drift-aware ML system. This is not a trivial task, especially for online drift calculations using real-time production data.

Luckily for us, this is available out of the box using the Iguazio platform. Behind the scenes, Iguazio will be deploying a real-time stream processor as well as an hourly scheduled batch job to calculate drift. The architecture looks something like this:



However, for our purposes, we don't need to worry about any of that. We can simply enable model monitoring for our serving function with the following snippet:

#### serving\_fn.set\_tracking()

project.set\_model\_monitoring\_credentials(os.getenv('V3I0\_ACCESS\_KEY'))

## Deploy on Kubernetes Cluster

Finally comes the moment of truth - production deployment on Kubernetes. In many cases, this is where projects go to die. In fact, some estimate that 85% of AI projects fail moving between development and production.

Not for us. Because we are using MLRun for containerization/configuration/deployment, we are almost done with our journey. To do the final deployment, we run this single line:

serving\_fn.deploy()



This will build a new Docker image using the base image/requirements we specified, followed by deploying our serving graph to a real-time Nuclio function. Additionally, it will automatically deploy the stream/batch processor for model monitoring purposes.

We can see these changes displayed in the MLRun project UI:

Projects > apure-fa-demo										
azure-fs-demo										() Dragte bea
Pastame	Models				Feat.			Files		
Paget pain		3				2	2			0
Balver Delme Dresteriat 73m, 1624-60	Jobs and workflow						Real-time functions (	Nuclic)		
Dane: Ink Dange Menters 2 Manage	0	0	i	0		1	2		0	0
Labels	Running jobs	Raming an	stitland	Failed		Schulder	Reverg		uler -	it/highneseyr
Dark Lines	Name	1994	Dates	Darhel et		Danite	Barro			lister
Durde Lineau Madata	modernanturepta.	•		11 Dec, 14 D	ier.	001				Record
Faulure store (Seta)	andreases par	de l		8 Dec. 11(23)		01557	made we stop			Record
File										
John and applificant										
Schedule alter M. Receber										
Real Lines Scientistes, Pilosites)						644-48				(execution)

We can test our newly deployed function using the default HTTP trigger like so:

```
serving_fn.invoke(
    path='/v2/models/infer',
    body={
        'inputs': [
           ["d107db82-fe26-4c02-b264-a3749510ed9b"],
           ["4d05f307-b699-4dbe-b51d-f14627233e5a"],
           ["43f23da3-99d0-4630-9831-91d7b54e757e"],
```



```
["e031ed66-52f8-4f49-9881-aeff00be2be1"],
["31ff724d-b29b-4edb-9f70-f4da66902fe2"]
]
}
> 
> 2021-12-15 20:33:56,189 [inf0] invoking function: {'method': 'POST', 'path': '<http://
nuclio-azure-fs-demo-model-serving.default-tenant.svc.cluster.local:8080/v2/models/infer>'}
{'id': 'f178c7da-0dd7-4d1b-8049-5328231229d7',
'model_name': 'VotingEnsemble',
'outputs': [0, 1, 1, 0, 1],
```

```
'model_version': 'v1'}
```

The default invocation trigger is via HTTP, but we can also add additional triggers like Cron, Kafka, Kinesis, Azure Event Hub, and more.

## Simulate Production Traffic

Before viewing our model monitoring dashboards, we need something to monitor. We are going to simulate production traffic by continuously invoking our function using the `patient\_ids` from our `FeatureVector`.

We can retrieve a list of the `patient\_ids` like so:

```
import mlrun.feature_store as fstore
records = fstore.get_offline_features(
    feature_vector="azure-fs-demo/heart_disease_vec",
    with_indexes=True
).to_dataframe().index.to_list()
```

Then, we can invoke our serving function with random records and delays like so:

```
from random import choice, uniform
from time import sleep
for _ in range(4000):
    data_point = choice(records)
    try:
        resp = serving_fn.invoke(path='/v2/models/predict', body={'inputs': [[data_point]]})
        print(resp)
        sleep(uniform(0.2, 1.7))
```



```
pass
> 2021-12-15 21:37:40,469 [info] invoking function: {'method': 'POST', 'path': '<http://
nuclio-azure-fs-demo-model-serving.default-tenant.svc.cluster.local:8080/v2/models/predict>'}
{'id': '65e2c781-31e5-46de-8e1d-2254d7c1b8af', 'model_name': 'VotingEnsemble', 'outputs':
[1], 'model_version': 'v1'}
> 2021-12-15 21:37:41,707 [info] invoking function: {'method': 'POST', 'path': '<http://
nuclio-azure-fs-demo-model-serving.default-tenant.svc.cluster.local:8080/v2/models/predict>'}
{'id': 'da267484-3b00-4752-87a9-157e2e3ca31c', 'model_name': 'VotingEnsemble', 'outputs':
[1], 'model_version': 'v1'}
> 2021-12-15 21:37:43,411 [info] invoking function: {'method': 'POST', 'path': '<http://
nuclio-azure-fs-demo-model-serving.default-tenant.svc.cluster.local:8080/v2/models/predict>'}
{'id': '4502196f-b9bb-45ea-a909-22670aa4cc58', 'model_name': 'VotingEnsemble', 'outputs':
[0], 'model_version': 'v1'}
```

## View Monitoring Dashboards

except OSError:

Once our simulated traffic has run for a bit, we can check our model monitoring dashboards.

The first dashboard is high level drift information in the MLRun Project UI.

Here we can see statistics, histogram, and drift metrics per feature.:

Projects > acure-fe-der	to > Models														
Madel Endpoint	-														
Labora Rev Ukey Ervalue	Sort By: Function +														9
Name	model_0													1	
model(2	hear														
model, 1	themine	-	uturas kontpa	*											
model_3	Feature	inerest N	and and	Sec. 1	N	Terret	and a	Supervel Ma		nya	Helinger	6.0	Histor	-	
VelogEnservice	age, rapped, tables								ę	φ.			1	1	
	epe,mapped,child		2	٠				a	0	0	8		1	1	
	ept.meportl.adult	1.05	9.90	9.30	0.25	۰.	*	,		0.05	8.07	0.04	· 1		
	app,mapped,miter	0.18	9.57	8.33	0.25				1	0.08	8.07	0.54	1 .	1	
	-10	95.07	52.15	8.12	1.15	28	.15	17	10	0.39	0.30	1.18			
	uss, make	672	0.74	1.45	0.44		. 4		1	9.82	8.08	9.20	1 L		
	sex, lense	0.38	9.29	3.41	0.44	٩	4	1	1	9.12	8.04	9.26	1 .	I	
										0.05	0.09	0.30			



We can also view overall model drift analysis:

Labers RepflaceD-volue.	Sortilly Punction +				-0
Marrie	model_0				1
model.8	latent				
model.1	Georgian Drift Analysis Freek	ures Analysis			
sinded.2	Marie	119	Inallinger	45,0	
wenglisseste	Sam.	2.81	3.13	21.06	
	Mage	8.01	1.12	0.44	

The second dashboard is more in depth drift and inferencing information in a Grafana dashboard. Here we can see drift per model, average number of predictions, and average latency per project:

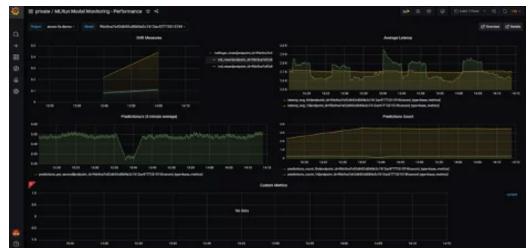
-	anne forderro 1							diversion () was	
	Side-1	-	Prodution	into (1 Minute Annuape)		Annupristance (Sant Hour)	Drove.		
	4			0.50		4.612 ms	0		
					Webb				
Renards	CORPORATION	store for democrately serving	rouble, Thereat	Classifie Model	2 hours ago	a few ancontrage			
elehtitz	TOLINOWINDELTS.	azure fo denormadei eening	VotingEnsembles/	<b>Gridmenootogtneersle</b>	1 Nouro ago	a film anomidi aga		904	
In case of	1002341-01264	ances for democranship serving	readed, Distant	CasellarModel	1 fearmage	a line ascends age			
MICOL	deligibilities.	same to democroated earing	madel_Distant	CassileMotel	2 hours age	a fere ancoroto ago			
		mikrisna'ı (Salinaia Anmaşti			umage Labora; () Final)		tun (		
			•امريبيون				Bros.		
			- 		*******				

We can also view a graph of incoming features over time. This is one of my personal favorite dashboards as it lets you visualize how your incoming features change over time:

Result Statistics Statis Statis Statis Statistics Statistics Statistics Statistics Stat										
Yet/perg/         Yet/perg/         Mathyan (perg/ stationer/ stationer/ stat	forward internet	erete, metel, then	en exerted	and the second second	Cashariaded	048	£108-0		Chours repr	a few seconds age
Yet(yord)         Yet (yord)         Ket(yord)         <										
3.1         5.1         5.2         12         12         14           Indext Stripts           Number Stripts         Numer Stripts         Number Stripts										
Falanta Sanghan         Falanta Sanghan </td <td></td>										
France:*         Accurations         Accurations         Expension										5.00
Name         XB         XL         XL         XL         XL         XL           AppL:MODELLAN         0         8.00         5         1         888         6         66.09         6.609										
No.110041.001         0         8.00         1         6.00         6         6.00         1.60           No.110041.001         0         6         0										
ада-падаяк, блай 0 6 0 6 0 6 3 0 0 ада-падаяк, блай 0 8.866 7 1 5.55 6 5.699 6.657 8.568 ада-падаяк, блайи 6 6 8 0 0 0 0 Боловор Наблаяк										
Appl. Analos. 2 100 7 5 123 6 209 207 200 Appl. Analos 2 4 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8										
repunged (Miller 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6										
Roomy Heaven			1.866							
	age, mapped, tottler									
					TOPY	ng Peerlunes				
• A state of the state of th			ALMAN ASS							an reserve Allowand
add arthau filliad Mitar Jibleth Katt - Tarki Mitark Techa an Mitalaka Josh Lutholitis. Maladilase -	PERSONAL PROPERTY.	Mall rolling the	MARY MARKEN	- Addad	n. hinas	Margaret	Concertion 1	Million Street		-
- Marchine Miller Mallar - Malay Mark Lange (1994)										ter mener methoderet



Finally, you can see model performance, latency, drift metrics over time, and any custom metrics you are logging:



All of these dashboards and visualizations come out of the box with little to no additional configuration required.

If and when model drift is detected, an event with the drift information will be posted to a stream. From there, we can add consumers to the stream to perform tasks like sending notifications or kicking off a re-training pipeline.

# Closing Thoughts

In this e-book, we have accomplished quite a lot. Using a combination of the Iguazio MLOps platform and Azure's AutoML capabilities, we have created a full end-to-end training flow that does the following:

- Ingest and transform dataset into feature store
- Upload/register features from Iguazio feature store into Azure ML
- Orchestrate Azure AutoML training job from Iguazio platform
- Download trained model(s) + metadata from Azure back into Iguazio platform
- Deploy top 3 models in model ensemble to real-time HTTP endpoint
- Integrate model serving with real-time enrichment and imputation via feature store
- Integrate model serving with model-monitoring and drift-detection

This whole flow is reproducible and modular in nature - meaning it would be simple to add additional steps, datasets, models, etc.

Want to play around with the code examples in this document? Check out the repo here

Want to try out the Iguazio MLOps Platform? Start Your 14-day Free Trial

