



Nuclio : KubeFlow Serverless's component

Orit Nissan-Messing, VP R&D, Iguazio

Data Science Teams Don't Do Data Science

Effort Allocation

Expectation



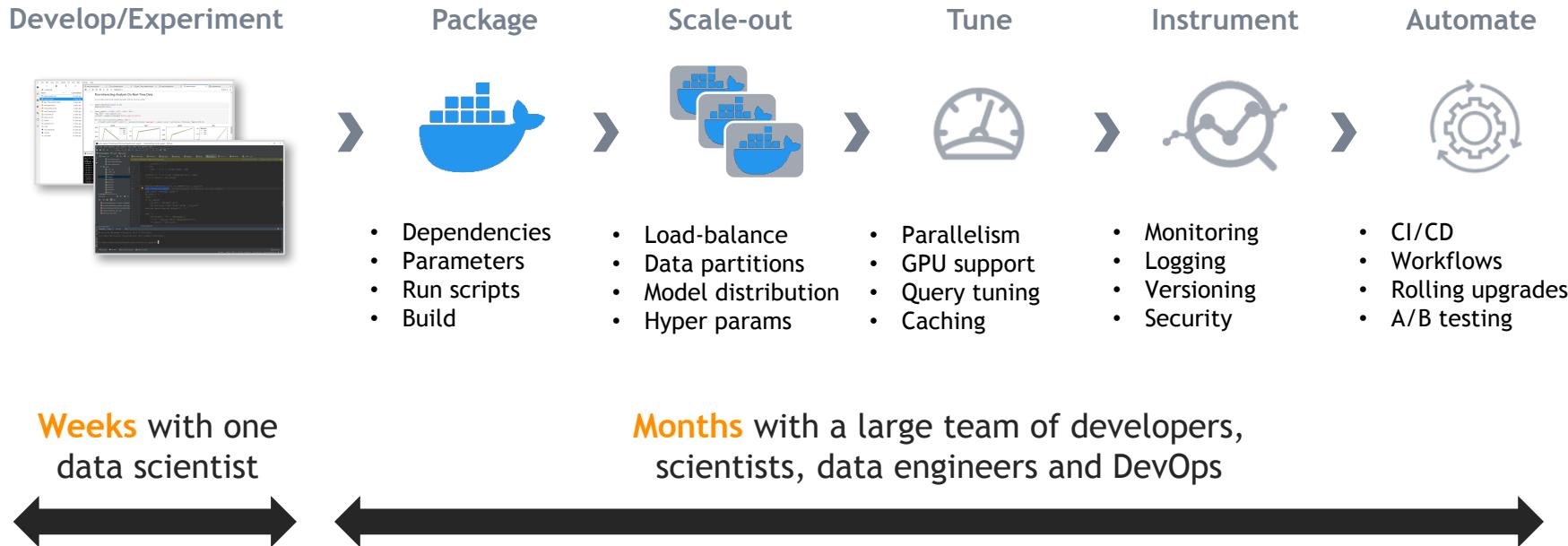
- Defining KPIs
- Collecting Data
- Infrastructure & DevOps
- Integration
- Optimizing ML Algorithm

Reality



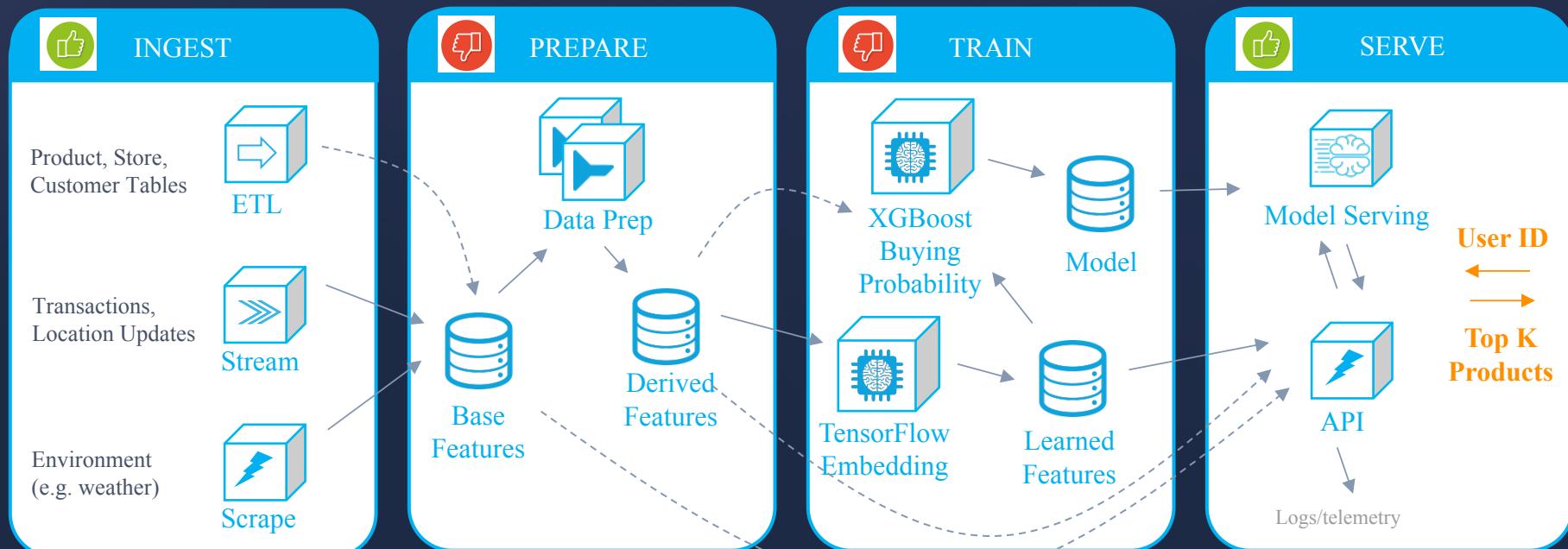
Source: Google Developers Launchpad

The need: Simpler Solutions, Better Data Integration



Automate DevOps to Deploy Projects in One Week as Opposed to Months!

Example: Real-time Product Recommendations



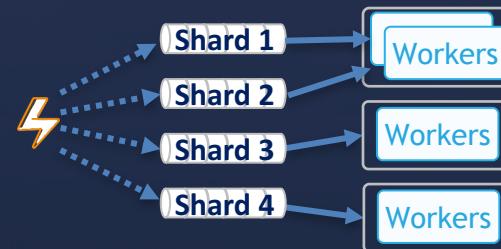
Nuclio: Taking Serverless to Data Intensive Apps

Extreme Performance



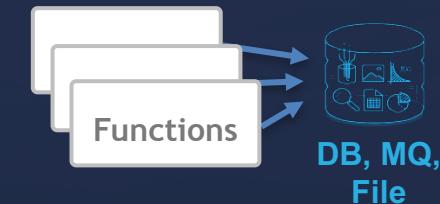
- Non-blocking, parallel
- Zero copy, buffer reuse
- Up to 400K events/sec/proc
- **GPU** optimizations

Advanced Data & AI Features



- Auto-rebalance, checkpoints
- Any source: Kafka, NATS, Kinesis, event-hub, iguazio, pub/sub, RabbitMQ, Cron, ..
- NVIDIA Rapids integration

Statefulness



- Data bindings
- Shared volumes
- Context cache

Natively integrated with Kubeflow and Jupyter Notebooks

Ingest: Using Nuclio to Accelerate ETL and Streaming

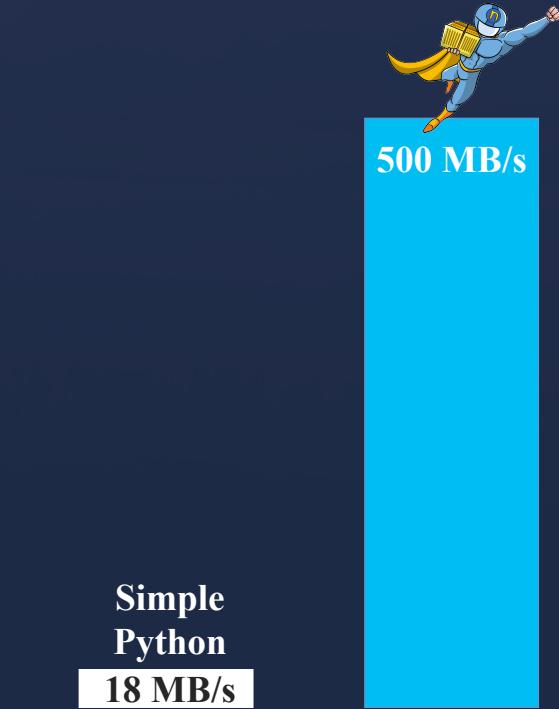
Simple code! Automated DevOps ! Any Source!
(e.g. read JSON Stream + aggregate + dump to Parquet)

```
def init_context(context):
    os.makedirs(sink, exist_ok=True)

def handler(context, event):
    add_log_to_batch(context, event.body)

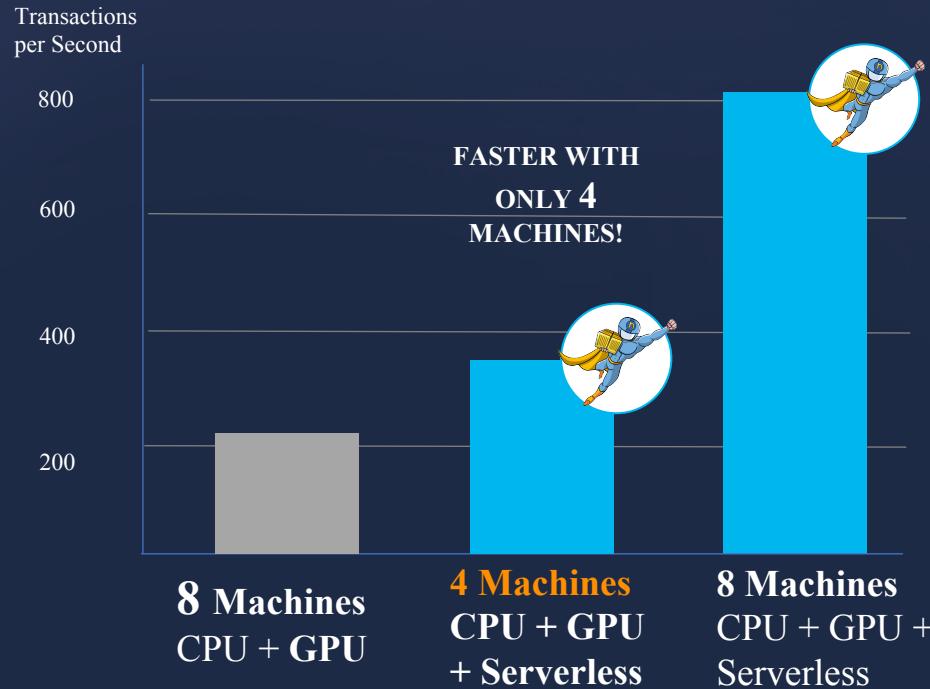
    if len(batch) > batch_len:
        df = _batch_to_df(context)
        if not df.empty:
            df = df.groupby(['log_ip']).agg({'feconn': 'mean',
                                              'beconn': 'mean',
                                              'time_backend_response': 'max',
                                              'time_backend_response': 'mean',
                                              'time_queue': 'mean',
                                              'time_duration': 'mean',
                                              'time_request': 'mean',
                                              'time_backend_connect': 'mean'
                                              })
            df_to_parquet(df)
            reset_batch()

    df_to_parquet(df)
    reset_batch()
```



Serving: Using Nuclio for Real-time Model Serving

4X Faster model serving on GPU system



Single command from notebook to function

```
# Create a Nuclio function
@nuclio.function(name="myfunc", kind="serverless")
def myfunc(context, event):
    context.logger.info("This is an ML model!")
    context.logger.info("The input is: %s", event)

# Print the message and other information
context.logger.info("Connected to Nuclio function %s", context.function_name)
context.logger.info("Nuclio version: %s", context.function_version)
context.logger.info("Request ID: %s", context.request_id)
context.logger.info("Request timestamp: %s", context.timestamp)

# Read request parameters and return function result
long = event.get("long", 100)
short = event.get("short", 100)

return long + short, "Good morning!"

# Create a Nuclio function
@nuclio.function(name="myfunc", kind="serverless")
def myfunc(context, event):
    context.logger.info("This is an ML model!")
    context.logger.info("The input is: %s", event)

# Print the message and other information
context.logger.info("Connected to Nuclio function %s", context.function_name)
context.logger.info("Nuclio version: %s", context.function_version)
context.logger.info("Request ID: %s", context.request_id)
context.logger.info("Request timestamp: %s", context.timestamp)

# Read request parameters and return function result
long = event.get("long", 100)
short = event.get("short", 100)

return long + short, "Good morning!"

# Create a Nuclio function
@nuclio.function(name="myfunc", kind="serverless")
def myfunc(context, event):
    context.logger.info("This is an ML model!")
    context.logger.info("The input is: %s", event)

# Print the message and other information
context.logger.info("Connected to Nuclio function %s", context.function_name)
context.logger.info("Nuclio version: %s", context.function_version)
context.logger.info("Request ID: %s", context.request_id)
context.logger.info("Request timestamp: %s", context.timestamp)

# Read request parameters and return function result
long = event.get("long", 100)
short = event.get("short", 100)

return long + short, "Good morning!"
```



Why Not Use Serverless for Training and Data Prep?

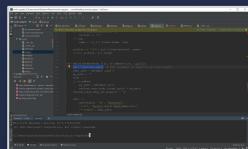
What about Training and data prep ?

	Serverless Today	Data Prep and Training
Task lifespan	Millisecs to mins	Secs to hours
Scaling	Load-balancer	Partition, shuffle, reduce, Hyper-params, ring allreduce
State	Stateless	Stateful
Input	Event	Params, Datasets

Serverless: resource elasticity and automated deployment and operations

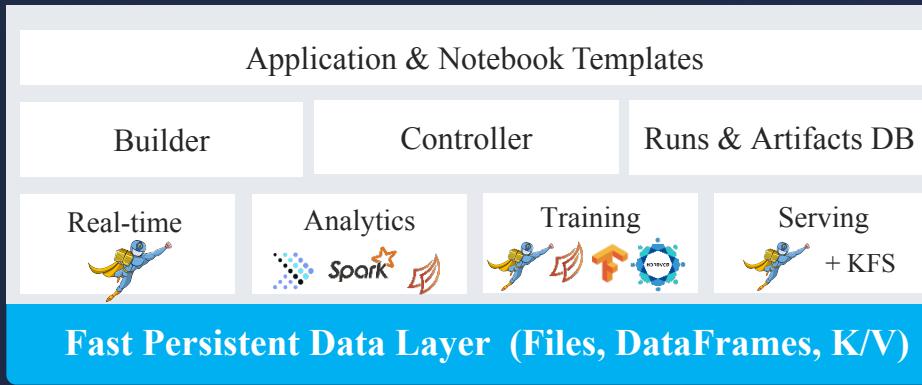
Introducing Nuclio ML Functions

Access from your notebook, IDE, or KubeFlow



Common
APIs &
Automation

Multiple
Engines



Built-in Artifacts &
Runs Tracking

Elastic Scaling

Demo: Fast and Serverless KubeFlow Pipeline



All demos can be found in github: <https://github.com/mlrun/demos>





Thank You

oritn@iguazio.com, www.iguazio.com

Iris Model Nuclio Function

The screenshot shows the Iguazio UI interface for managing machine learning models. The left sidebar contains icons for Projects, Pipelines, Functions (highlighted in blue), Services, Data, Clusters, Storage, Networks, and Identity. The main area displays the configuration for a specific function.

Project Path: Projects > iris > xgb-train > \$LATEST

Configuration Tab: The Configuration tab is selected, showing the following details:

- Volumes:** A table listing a single volume named "fs" of type V3IO, with a mount path of "User; Access Key: abb3f38a-e1bd-462a-b9ac-7b...". A button to "Create a new volume" is also present.
- Build:** Configuration for building the function image.
 - Image name:** Enter image name...
 - Base image:** python:3.6-jessie
 - Onbuild image:** Enter onbuild image...
 - Build commands:** pip install sklearn
pip install xgboost
pip install matplotlib
pip install mlrun
- Readiness timeout (seconds):** A dropdown menu with options 1, 2, 5, 10, and 30.
- Disable cache:** A checkbox that is unchecked.

Actions: Buttons for Actions (dropdown), Deploy, and other management functions.

Iris Model Pipeline

Pipelines

Experiments > xgb

← **xgb 1**

Retry **Clone run** Terminate Archive

Graph Run output Config

```
graph TD; ingest[ingest-iris] --> train[xgb-train]; train --> serving[deploy-iris-serving]; train --> plot[plot]
```

ingest-iris

xgb-train

deploy-iris-serving

plot

Artifacts

Input/Output

Volumes

Manifest

Logs

my-xgboost-training-pipeline-9h6v4-4169956791

0 10 20

0.90 0.92 0.94 0.96

① Runtime execution graph. Only steps that are currently running or have already completed are shown.

Iris Model Serving

The screenshot shows the Iguazio UI interface for managing machine learning models. The left sidebar contains icons for Projects, Pipelines, Functions, Services, Data, Clusters, Storage, Networks, and Identity. The main area displays the 'iris-srv' project under the 'iris' organization, specifically the '\$LATEST' version. The top navigation bar includes tabs for CODE, CONFIGURATION, TRIGGERS, and STATUS. The CODE tab is selected, showing the source code for the model serving function:

```
9
10 BOOSTER_FILE = "model.bst"
11
12 class XGBoostModel(kfserving.KFModel):
13     def __init__(self, name: str, model_dir: str, booster: xgb.XGBModel = None):
14         super().__init__(name)
15         self.name = name
16         self.model_dir = model_dir
17         if not booster is None:
18             self._booster = booster
19             self.ready = True
20
21     def load(self):
22         model_file = os.path.join(
23             kfserving.Storage.download(self.model_dir), BOOSTER_FILE)
24         self._booster = xgb.Booster(model_file=model_file)
25         self.ready = True
26
27     def predict(self, body: List) -> List:
28         try:
29             dmatrix = xgb.DMatrix(body)
30             result: xgb.DMatrix = self._booster.predict(dmatrix)
31             return result.tolist()
32         except Exception as e:
33             raise Exception("Failed to predict %s" % e)
34
```

Distributed TensorFlow Pipeline

Pipelines

Experiments > horovod1

← ✓ hvd_pipeline 2019-11-18 17-34-06

Retry Clone run Terminate Archive

Graph Run output Config

```
graph TD; download[download] --> label[label]; label --> train[train]; train --> deploy[deploy-tf-image-...]
```

Input parameters

images-path	/User/mlrun/examples/images
label-categories-map	/User/mlrun/examples/images/categories_map.json
label-file-categories	/User/mlrun/examples/images/file_categories_df.csv
source-dir	/User/mlrun/examples/images/cats_n_dogs

Output parameters

train-model	None
-------------	------

ⓘ Runtime execution graph. Only steps that are currently running or have completed successfully are shown.

MLRun UI - Distributed TensorFlow Train Job

 **MLRunUI**

[SEE ON GITHUB](#) 

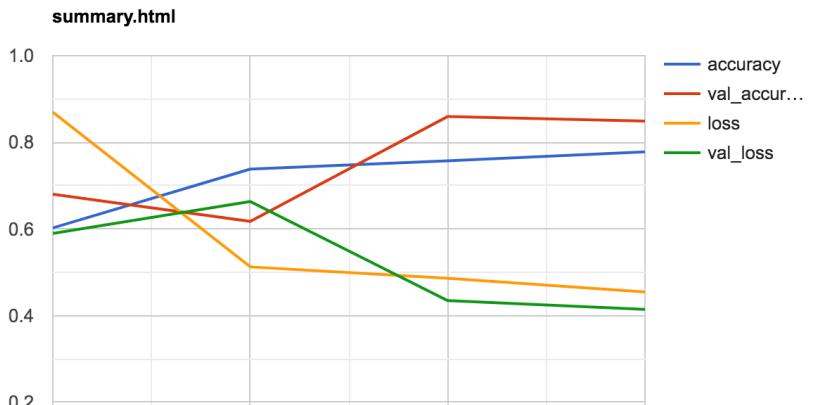
Nov 19 18:30:51

train ✓ 4e0d2d4ff4934a7eb76ef511a5929398

INFO	INPUTS	ARTIFACTS	RESULTS	LOGS
model /User/mlrun/e...			Size: 1 KiB Created: Nov 19 21:00:06	

summary.html /User/mlrun/e...

summary.html



Iteration	accuracy	val_accuracy	loss	val_loss
0	0.60	0.65	0.85	0.60
6	0.70	0.60	0.55	0.65
12	0.75	0.65	0.50	0.60
18	0.78	0.85	0.45	0.55
24	0.78	0.85	0.40	0.40

v1.0