

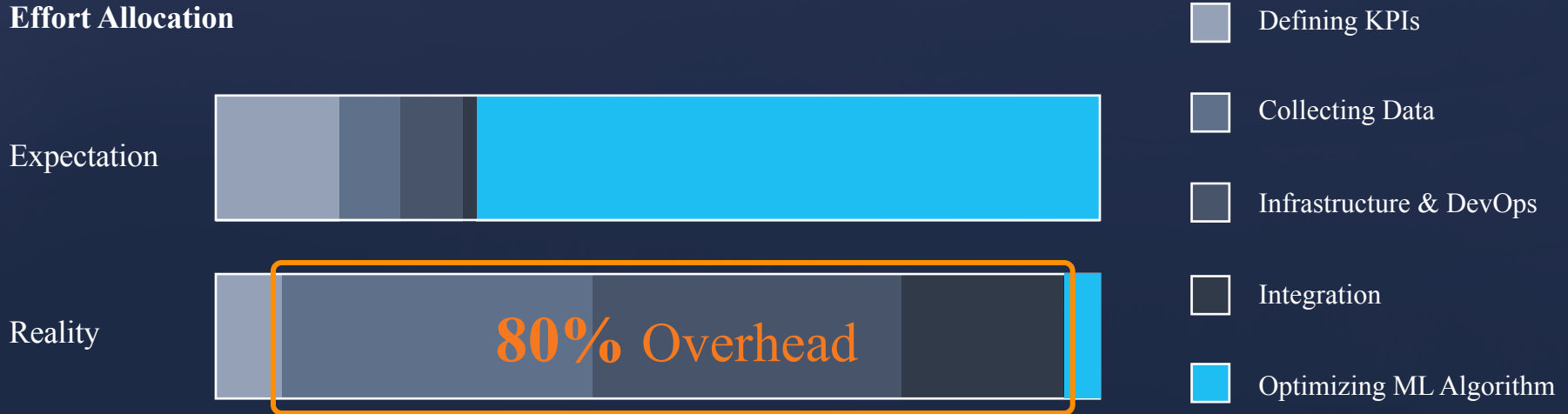


Nuclio : KubeFlow Serverless's component

Orit Nissan-Messing, VP R&D, Iguazio

Data Science Teams Don't Do Data Science

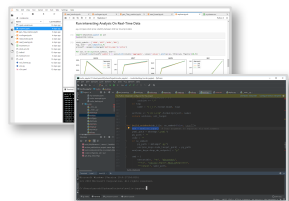
Effort Allocation



Source: Google Developers Launchpad

The need: Simpler Solutions, Better Data Integration

Develop/Experiment



Package



- Dependencies
- Parameters
- Run scripts
- Build



Scale-out



- Load-balance
- Data partitions
- Model distribution
- Hyper params



Tune



- Parallelism
- GPU support
- Query tuning
- Caching



Instrument



- Monitoring
- Logging
- Versioning
- Security



Automate



- CI/CD
- Workflows
- Rolling upgrades
- A/B testing

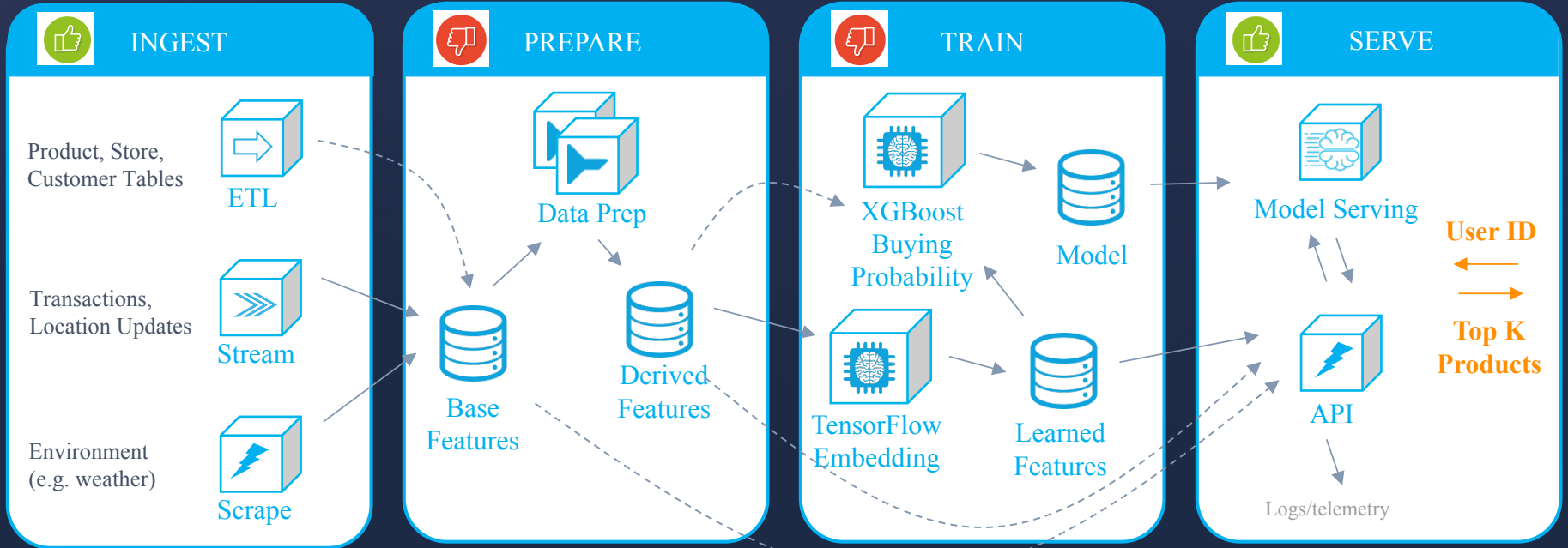
Weeks with one data scientist

Months with a large team of developers, scientists, data engineers and DevOps



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

Example: Real-time Product Recommendations



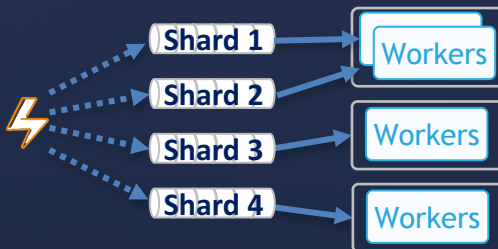
Nuclio: Taking Serverless to Data Intensive Apps

Extreme Performance



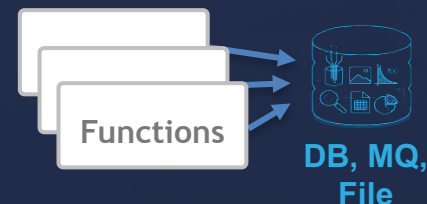
- Non-blocking, parallel
- Zero copy, buffer reuse
- Up to 400K events/sec/proc
- **GPU** optimizations

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()
```



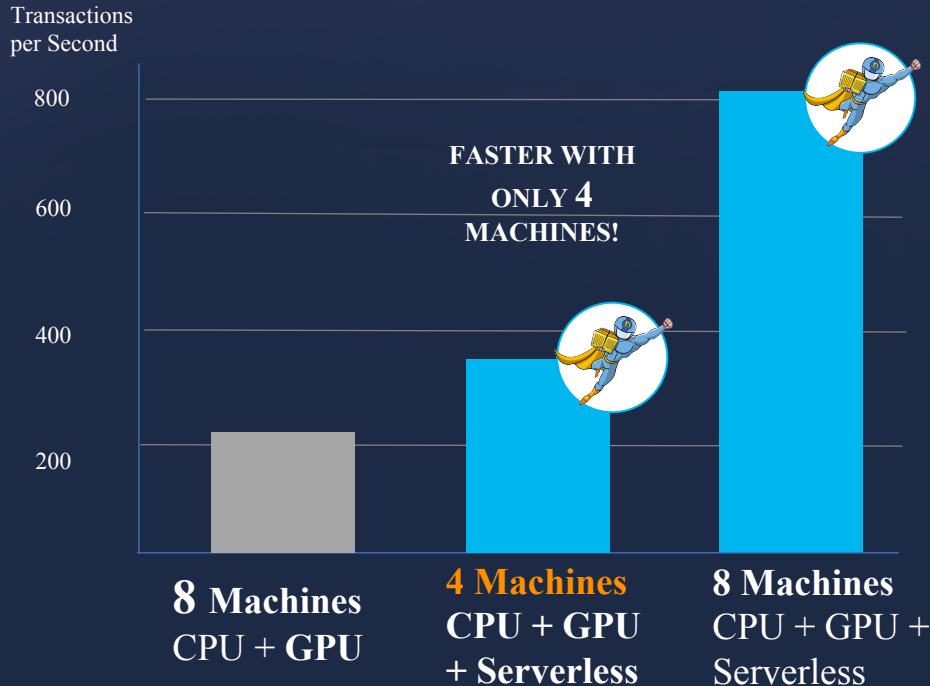
500 MB/s

Simple
Python

18 MB/s

Serving: Using Nuclio for Real-time Model Serving

4X Faster model serving on GPU system



Single command from notebook to function

```
%%bash
# Create a new Nuclio function package to use installed on this kernel with Jupyter
nuclio new

# Create and set function metadata
nuclio set --name my-function --type python --runtime python --memory 128MB --timeout 30s

# Create a simple function that prints "Hello World"
cat > my-function.py
def handler(context, event):
    print("Hello World")
    return {}

# Deploy the function
nuclio deploy --name my-function --runtime python --memory 128MB --timeout 30s

# Test the function
nuclio invoke --name my-function --runtime python --memory 128MB --timeout 30s
```



Why Not Use Serverless for Training and Data Prep?

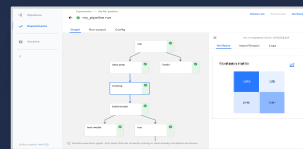
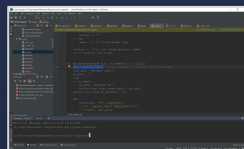
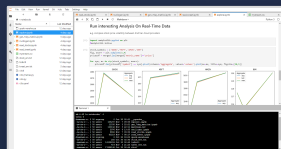
What about Training and data prep ?

	Serverless Today	Data Prep and Training
Task lifespan	Milliseconds 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

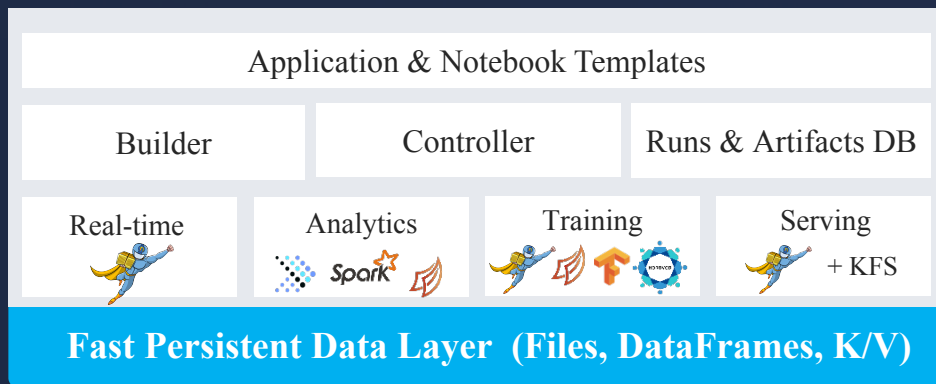
Introducing Nuclio ML Functions

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

Projects > iris > xgb-train > \$LATEST

PIPELINES

FUNCTIONS

SERVICES

DATA

CLUSTERS

STORAGE

NETWORKS

IDENTITY

ACTIONS

DEPLOY

CODE CONFIGURATION TRIGGERS STATUS

Volumes

Name	Type	Mount Path & Params
> fs	V3IO	Mount Path: User, Access Key: abb3f38a-e1bd-462a-b9ac-7b...

+ Create a new volume

Build

Image name

Base image

Onbuild image

Build commands

```
pip install sklearn
pip install xgboost
pip install matplotlib
pip install mirun
```

Readiness timeout (seconds)

Disable cache

Iris Model Pipeline

The screenshot displays the Databricks Pipelines interface. On the left is a navigation sidebar with icons for Pipelines, Functions, Services, Data, and Clusters. The main area shows the 'Pipelines' section for an experiment named 'xgb'. The pipeline 'xgb 1' is shown in a 'Graph' view, consisting of four steps: 'ingest-iris', 'xgb-train', 'deploy-iris-serving', and 'plot'. All steps are marked with green checkmarks, indicating successful completion. Below the graph, a note states: 'Runtime execution graph. Only steps that are currently running or have already completed are shown.'

On the right, a modal window titled 'my-xgboost-training-pipeline-9h6v4-4169956791' is open, showing an 'Artifacts' bar chart. The chart displays the volume of artifacts for different accuracy levels. The x-axis represents accuracy (0.90 to 0.96) and the y-axis represents volume (0 to 20+).

Accuracy	Volume
0.90	8
0.92	0
0.94	25
0.96	8

Iris Model Serving

Projects > iris > iris-srv > \$LATEST

CODE CONFIGURATION TRIGGERS STATUS ●

Code entry type: Source code (edit online) Runtime: Python 3.6 Handler: nuclio_serving.handler

Source code *

```
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

The screenshot displays the Iguazio Pipelines interface. On the left is a navigation sidebar with icons for Pipelines, Functions, Services, Data, Clusters, and Storage. The main area is titled "Pipelines" and shows the execution details for an experiment named "horovod1". The specific pipeline is "hvd_pipeline 2019-11-18 17-34-06", which is in a successful state (indicated by a green checkmark). The pipeline is visualized as a vertical flow graph with four steps: "download", "label", "train", and "deploy-tf-image-...". The "train" step is highlighted with a blue border, indicating it is the current focus. To the right of the graph, a panel titled "image-classification-training-pipeline-2qj9q-1020041149" shows the "Input/Output" parameters for the selected step. The input parameters include "images-path", "label-categories-map", "label-file-categories", and "source-dir". The output parameter is "train-model", which is set to "None".

Pipelines

Experiments > horovod1

hvd_pipeline 2019-11-18 17-34-06

Graph Run output Config

download label train deploy-tf-image-...

image-classification-training-pipeline-2qj9q-1020041149

Artifacts Input/Output Volumes Manifest Logs

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

MLRun UI - Distributed TensorFlow Train Job



MLRunUI

SEE ON GITHUB

demo

Nov 19 18:35:56

ffeaf5b410014799aa85f...

Iterations: 0



train

Nov 19 18:30:51

4e0d2d4ff4934a7eb76e...

Iterations: 0



label

Nov 19 18:30:30

a93ed4cbbad6424ca89...

Iterations: 0



download

Nov 19 18:30:05

f4fe28b8cacc4e79a7de...

Iterations: 0



xgb_train

Nov 19 18:28:40

c864c0d07b084ed0b9...

Iterations: 24



iris_gen

Nov 19 18:28:30

8aa415c1ac9d400baf0...

Iterations: 0



train

4e0d2d4ff4934a7eb76ef511a5929398

Nov 19 18:30:51

INFO

INPUTS

ARTIFACTS

RESULTS

LOGS

model

/User/mlrun/e...

summary.html

/User/mlrun/e...

/User/mlrun/examples/images/summary.html

Size: 1 KiB

Created: Nov 19 21:00:06

