



Data Science from Research to Production with Jupyter, Kubeflow & Nuclio

Or Zilberman, Data Scientist, Iguazio

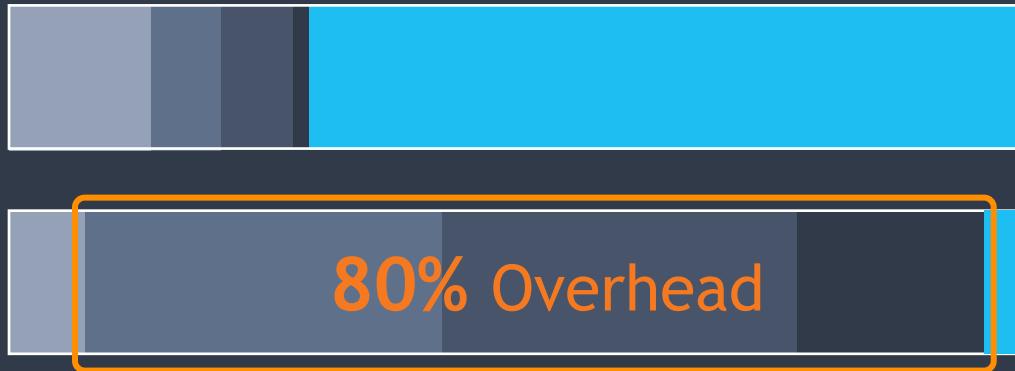
Nov 2019

Data-Scientists Don't Do Data Science !

Effort Allocation

Expectation

Reality



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

The need: Simpler Solutions, Better Data Integration

Goal

**Focus on the business goal
instead of boilerplate related tasks**

Goal

Focus on the **business goal**
instead of boilerplate related tasks

How?



Kubernetes
Cloud OS



Jupyter
DS IDE



Nuclio
Serverless Framework



Kubeflow
Orchestration

Or Zilberman

- Chief Data Scientist @ Iguazio
- Data Scientist @ Glispa Global Group:
 - Managed Glispa's Data Management Platform (User Personalization)
 - Owned Glispa's SSP Platform's Data Science efforts
 - RTB Process & Revenue Optimization
 - Ad serving network optimizations
- Data Scientist @ Teddy Sagi's Group:
 - Supplied solutions to multiple companies in the group as part of a specialized AI team
 - E-Commerce: Recommendation engines based on Image, Text and user actions
 - Forex: Risk management,
 - Customer Life Time Value, Customer Churn
 - Automated lead generation
- Patents: 3
- Latest publication: An FPGA Scalable Parallel Viterbi Decoder @ IEEE MSOC 2018 Vietnam

Additional Experience:

- Head of SEO & CRO @ Dataworks
- VP Product @ NegevJobs
- Director @ Streamono/Gameono

Certifications:

- Google Analytics Certified Individual Qualification (01684861)
- iCDE (18012817THDUUZHU)

Education:

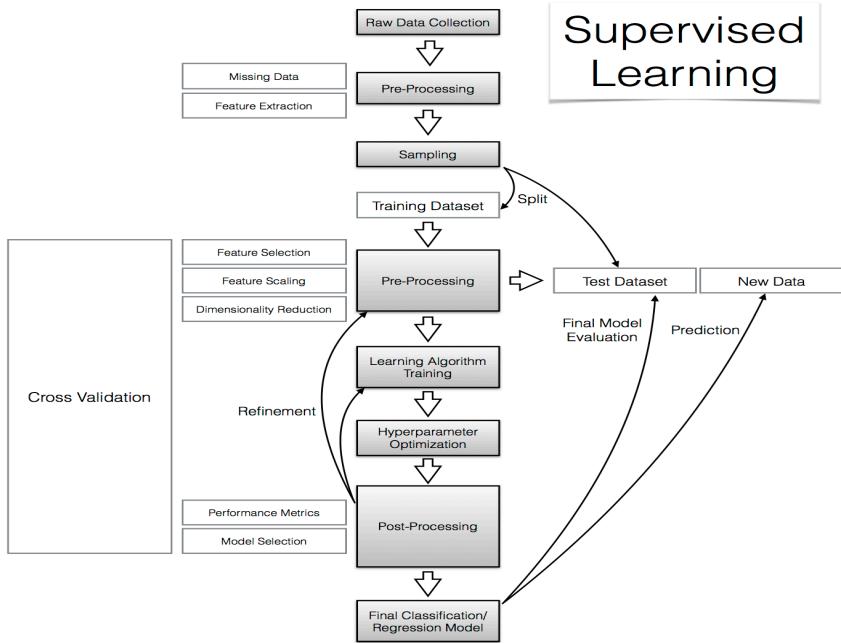
- University of Haifa

What Will We Do Today?

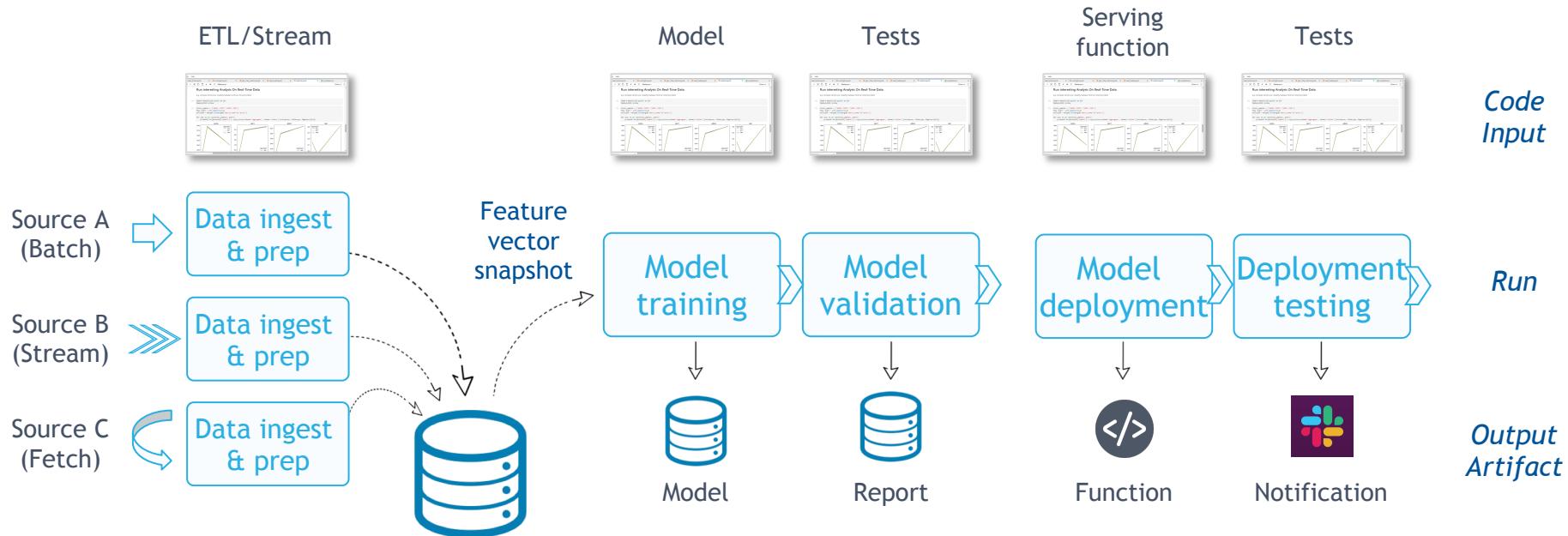
- Review the data science process from a production deployment point of view
- Technological review - base & architecture
 - Docker
 - Kubernetes
 - Serverless
- Nuclio
 - What is Nuclio
 - How to deploy serverless functions from our Jupyter notebook
 - Training: Deploy a function
- Kubeflow & MLRun
 - What is Kubeflow
 - What is MLRun
 - Training: Deploy a full pipeline

The Data Science Process

- Get the data
- Pre-Processing
 - From raw data to a Feature Vector
- Training
 - Hyperparameter optimization
 - Cross validation
- Post processing
- Final model



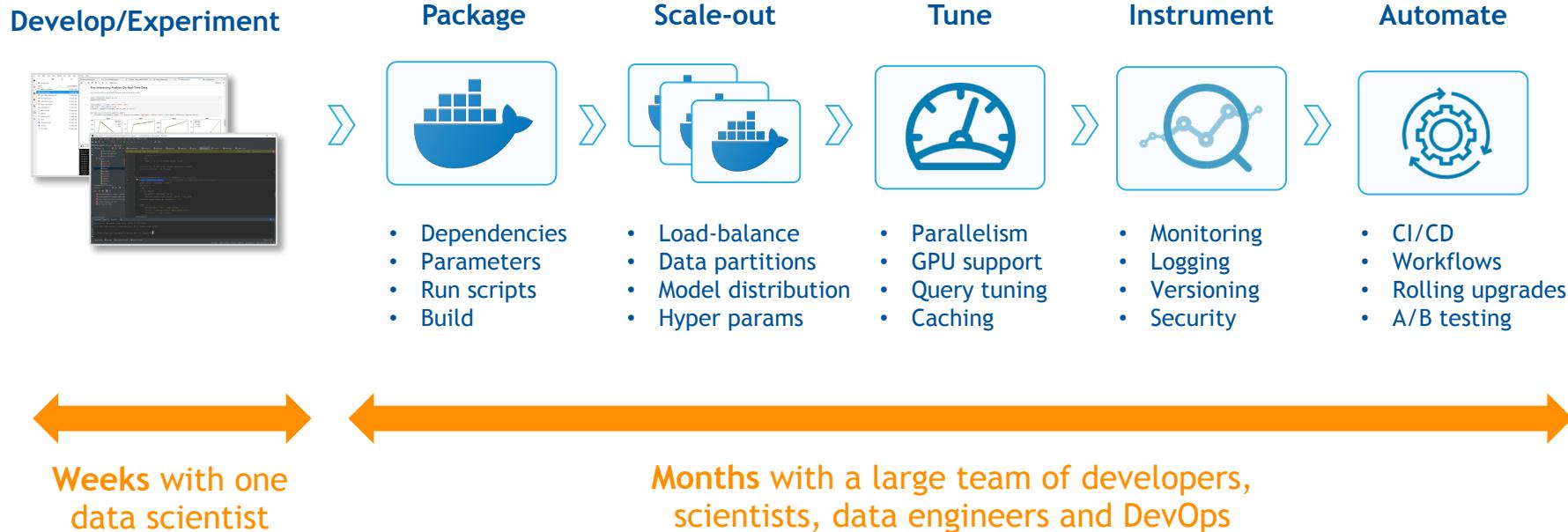
Typical Data Science Pipeline



Pipelines must be automated!

Code/Model Development Is Just The FIRST Step

Every piece of code, data science algorithm, or data processing task must be built for production



Docker



Docker is a platform for developers and sysadmins to **build, share, and run** applications with containers. The use of containers to deploy applications is called *containerization*. Containers are not new, but their use for easily deploying applications is.

Containerization is increasingly popular because containers are:

Flexible: Even the most complex applications can be containerized.

Lightweight: Containers leverage and share the host kernel, making them much more efficient in terms of system resources than virtual machines.

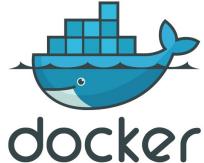
Portable: You can build locally, deploy to the cloud, and run anywhere.

Loosely coupled: Containers are highly self sufficient and encapsulated, allowing you to replace or upgrade one without disrupting others.

Scalable: You can increase and automatically distribute container replicas across a datacenter.

Secure: Containers apply aggressive constraints and isolations to processes without any configuration required on the part of the user.

Docker



Docker Basics



Image

The basis of a Docker container. The content at rest.



Container

The image when it is 'running.' The standard unit for app service



Engine

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.



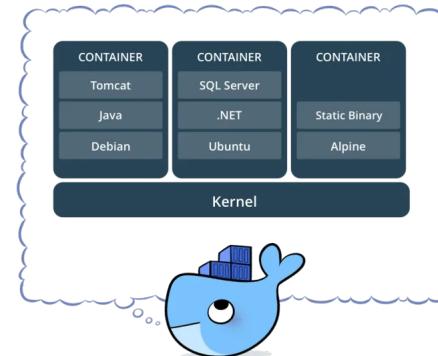
Registry

Stores, distributes and manages Docker images



Control Plane

Management plane for container and cluster orchestration



What is Kubernetes?

“Kubernetes” = Greek for governor, helmsman, captain

- Open source container orchestration system
- Originally designed by Google, maintained by CNCF
- Aims to provide “a platform for automating deployment, scaling and operations of application containers across clusters of hosts”

Kubernetes - Pod

The basic, atomically deployable unit in Kubernetes.
A **Pod** consists of one or many *co-located containers*.
A Pod represents *a single instance of an application*.

The containers in a Pod share the loopback interface (localhost) and can share mounted directories.

Each Pod has it's own, uniquely assigned and internal IP.
Pods are **mortal**, which means that if the node the Pod runs on becomes unavailable, the workload also goes unavailable.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  containers:
  - image: nginx:1.13.9
    name: nginx
    ports:
    - name: http
      containerPort: 80
```

Kubernetes - Deployment

With a [Deployment](#), you can manage Pods in a **declarative and upgradable** manner.

Note the *replicas* field. Kubernetes will make sure that amount of Pods created from the template always are available.

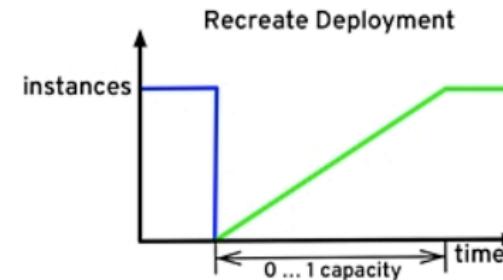
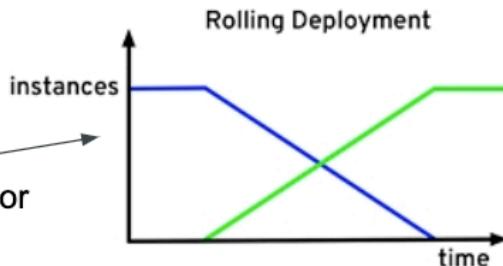
When the Deployment is updated, Kubernetes will perform a **rolling update** of the Pods running in the cluster. Kubernetes will create one new Pod and remove an old until all Pods are new.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
    name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - image: nginx:1.13.9-alpine
        name: nginx
        ports:
          - name: http
            containerPort: 80
```

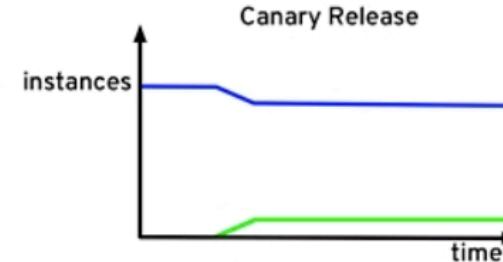
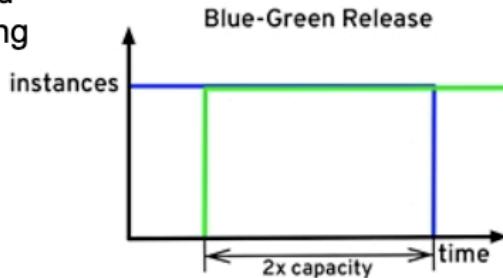
The Pod Template

Kubernetes - Deployment strategies

The built-in
Deployment behavior



The other strategies
can be implemented
fairly easily by talking
to the API.



Kubernetes - Service

A Service exposes one or many Pods via a *stable, immortal, internal IP address*.

It's also accessible via cluster-internal DNS:
{service}.{namespace}.svc.cluster.local, e.g.
nginx.default.svc.cluster.local

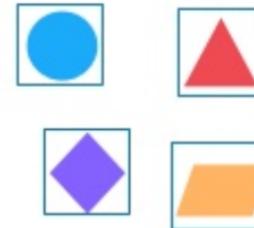
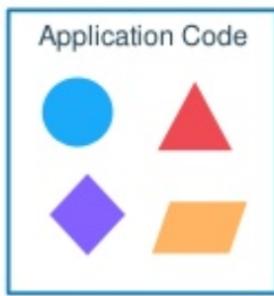
The Service selects Pods based on the **label key-value selectors** (here app=nginx)

A Service may expose multiple ports. This ClusterIP can be declaratively specified, or dynamically allocated.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  type: ClusterIP
  ports:
  - name: http
    port: 80
    targetPort: 80
  selector:
    app: nginx
```

Serverless - Moving to Micro-Services

Application Modernization



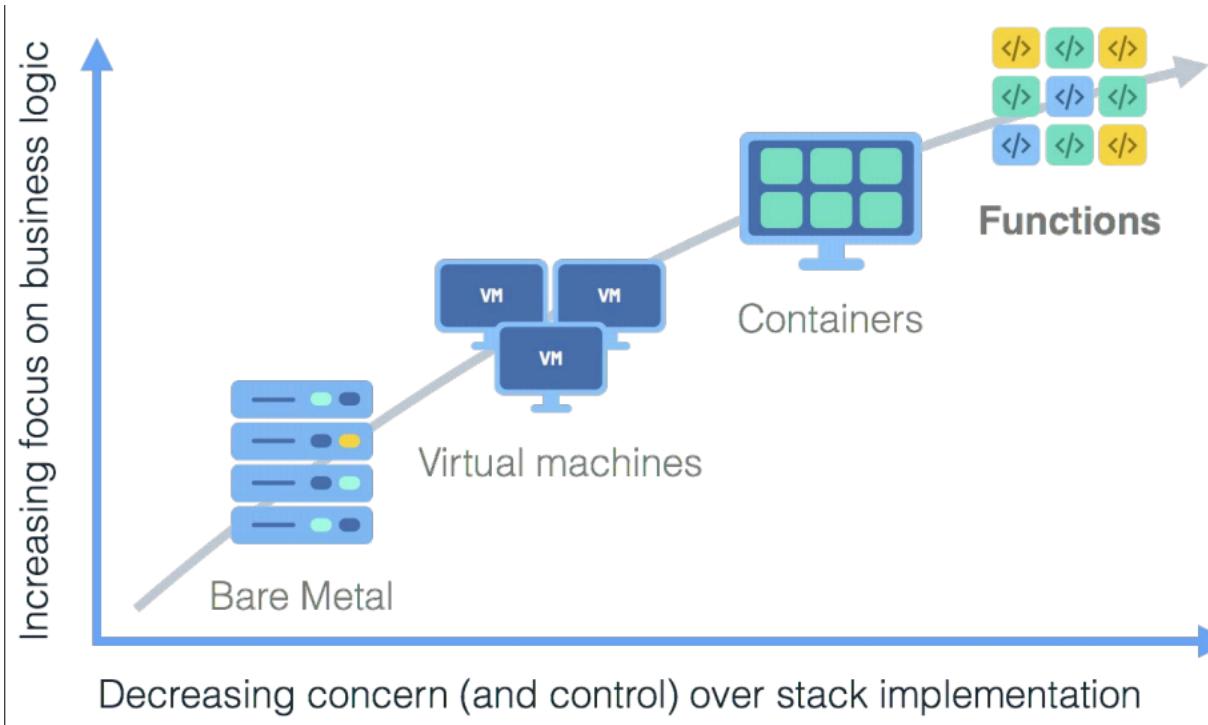
Developer Issues:

- Minor code changes require full re-compile and re-test
- Application becomes single point of failure
- Application is difficult to scale

Microservices: Break application into separate operations

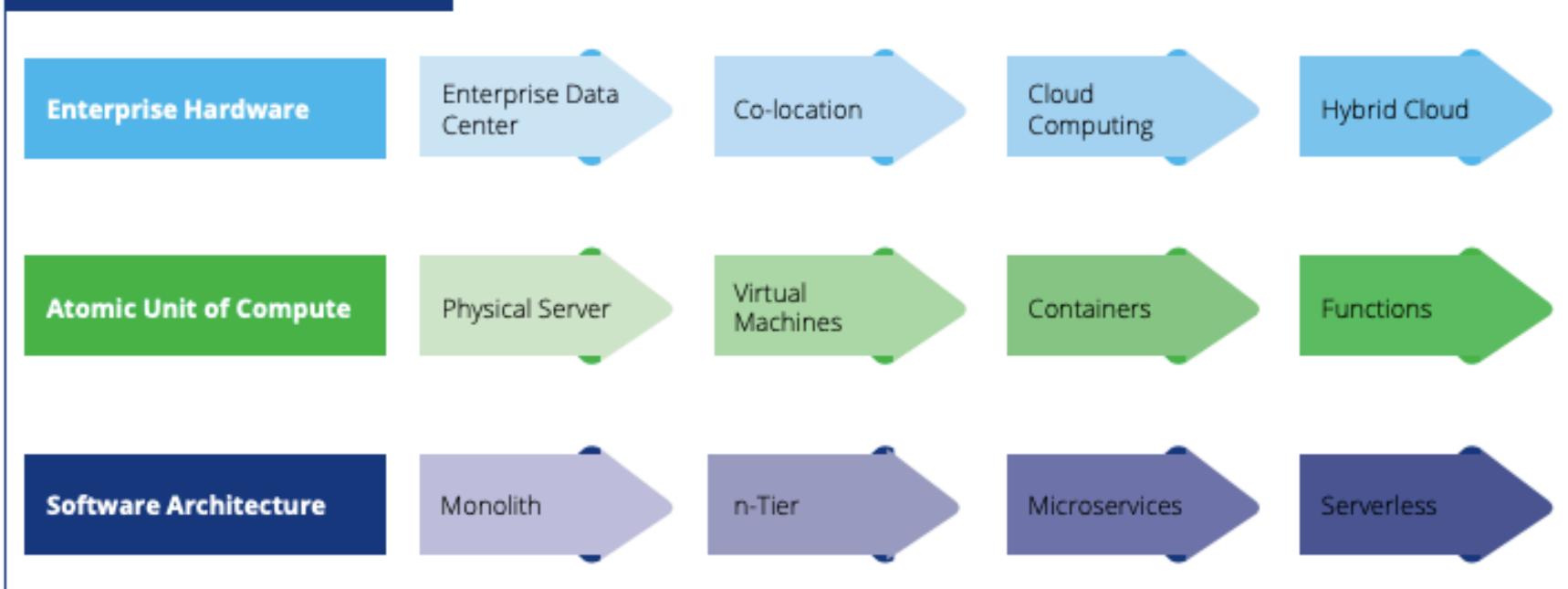
12-Factor Apps: Make the app independently scalable, stateless, highly available by design

Serverless

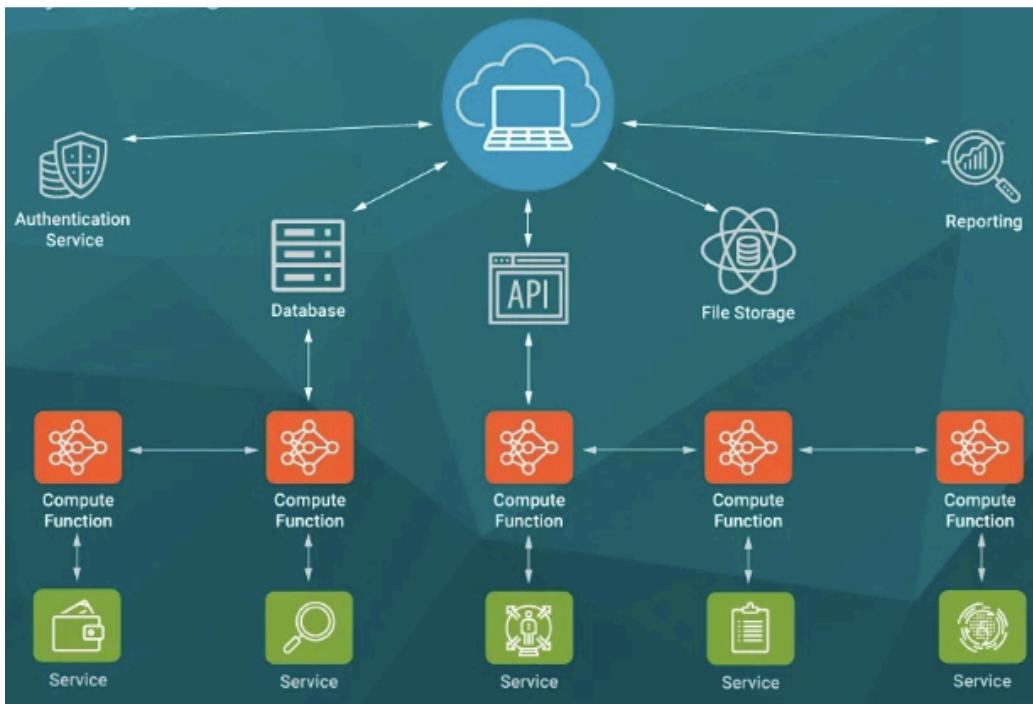


Serverless

Figure: Evolution of technology



Serverless Architecture



The Agility Challenge:

How to Innovate and Iterate Faster, Deploy Everywhere

Modern Apps Today:

Complex, long development

- Many moving parts, lack of skilled developers, scientists, and DevOps

Production readiness

- Error-handling, security, logging, auto-scaling, live updates and more

Slow to respond or act

- High-latency, or fast but inaccurate
- Cloud latency/bandwidth barriers



The Need:



Serverless

Automate dev and ops



High performance

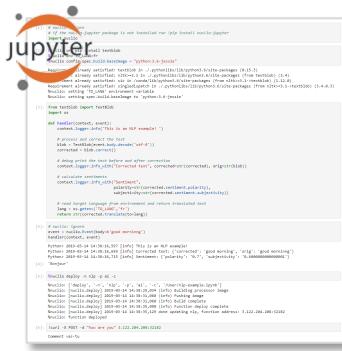
Data, AI and app services



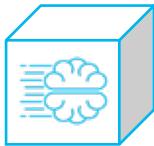
Federated

In the cloud, on-prem or edge

Nuclio: Fast Serverless for Data Science & RT Analytics

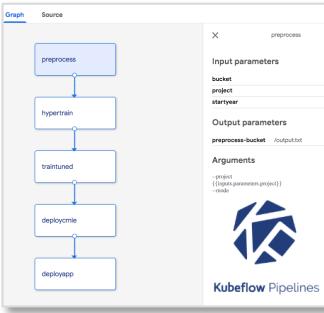
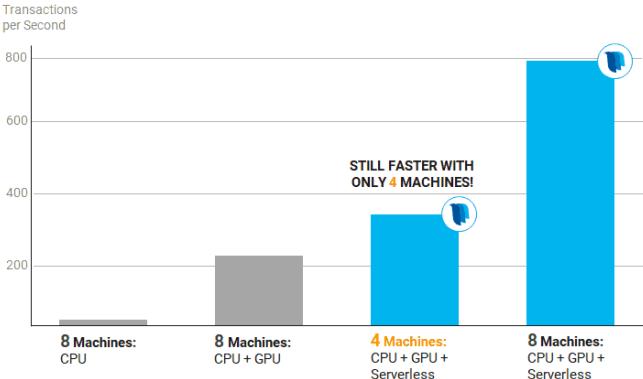


```
# For the first time, chapter package is not installed nor `jupyter` exists.
# Python 3.x+Jupyter
# Run this command to install required dependencies:
# pip install jupyter[all]
# Run this command to install required dependencies:
# pip install nuclio[notebook]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml,aws]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis,streaming]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis,streaming,rt]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis,streaming,rt,ml]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis,streaming,rt,ml,aws]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis]
# Run this command to install required dependencies:
# pip install nuclio[notebook,serverless,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis,streaming,rt,ml,aws,redis,streaming]
```



Magic commands from notebook to function

High performance IO and computation
+ GPU optimization



Extending ML pipelines from batch:

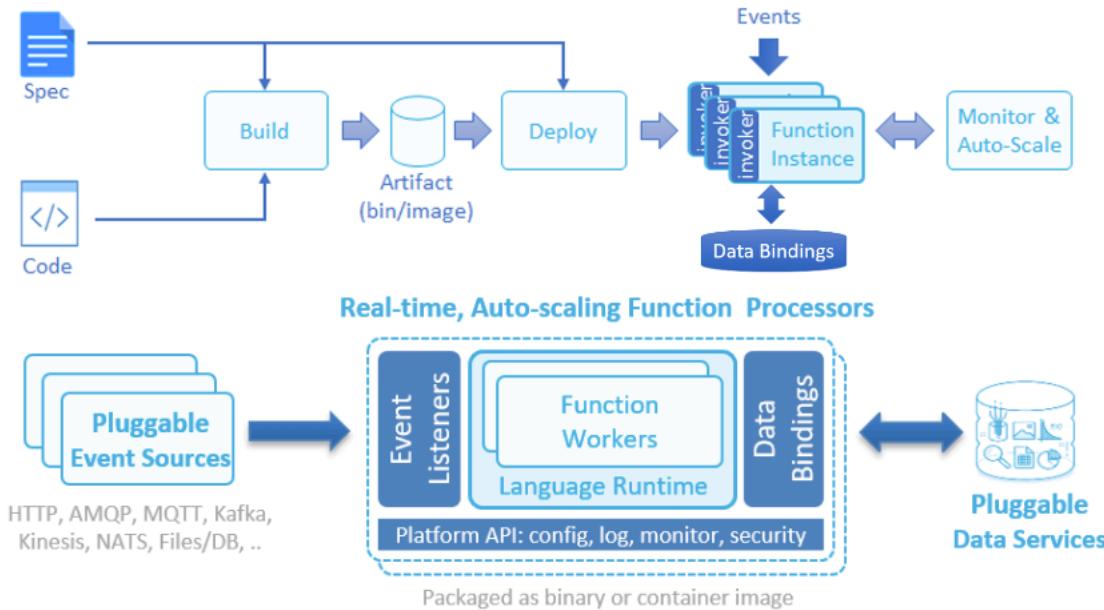
1. Parallel processing
2. Code build/deployment
3. Stream processing
4. API/Model Serving



Code + DevOps automation:

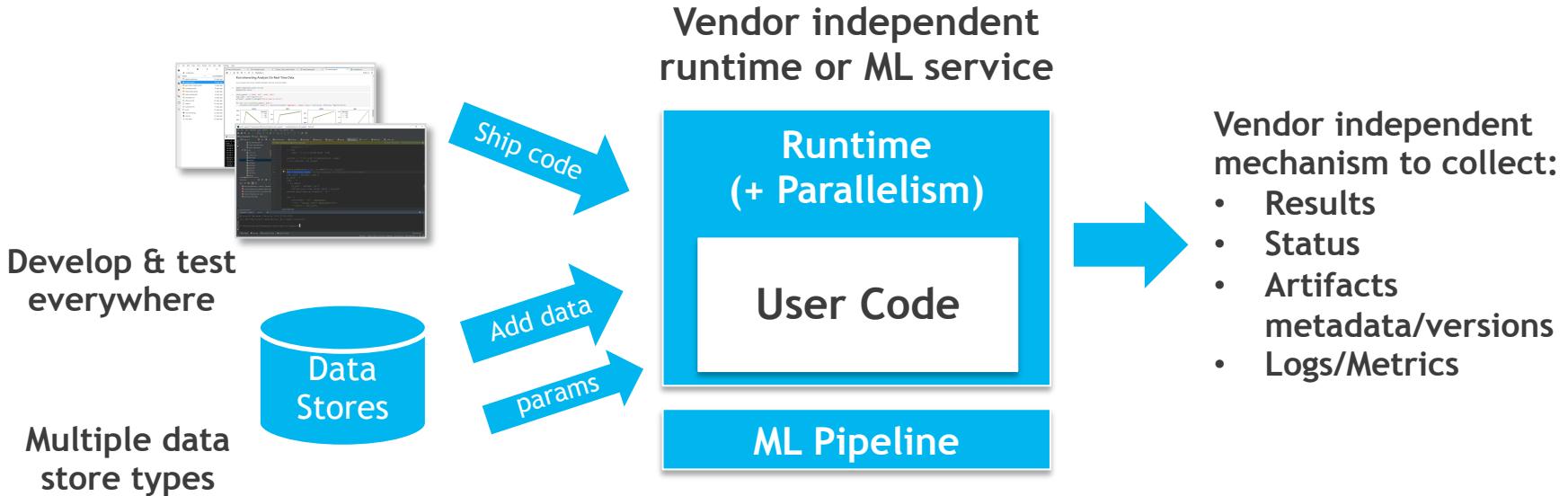
1. Auto-scaling (to zero)
2. Automated logging & monitoring
3. Security hardening
4. Auto-build and CI/CD
5. Workload mobility (cloud/edge/..)

Nuclio - Under the Hood



MLRun

Vendor and Runtime Independent ML Deployment and Tracking Automation



Check it out on github : [mlrun](#)

Demos !
