

Hydra Architecture:

Orchestrating ML across clusters,
regions, and clouds

`donny@run.house`



Runhouse



Who am I?

Previously **Product Lead for PyTorch at Meta** - worked with hundreds of ML teams and led multiple large ML platform rearchitectures

Now **Co-founder of Runhouse** - a **distributed computing platform** for **enterprise ML** (NY-based, venture-backed, founded 2022)

Snowflake Won by Abstracting Complexity

Before Snowflake:

- SQL worked fine for normal sized queries
- For large scale queries, managing Hadoop was an infrastructural disaster

After Snowflake (and OLAP databases generally):

- Write regular SQL queries, completely non-optimized for scale
- Magic
- Executes over ephemeral allocated cloud compute that is entirely opaque to you

This is better

ML Has Not Gone Through the Same Transformation

- ML engineers and data scientist spend **50%+ of their time** wrestling with infrastructure.
- Teams hit multiple walls scaling up:
 - Going from a few models getting manually unblocked by ML platforms team to many teams who need enablement
 - Moving from CPU training to GPUs, and single GPU to multi-node

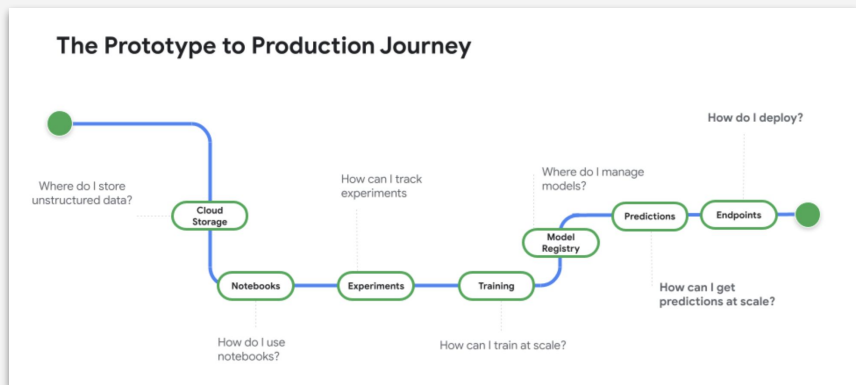
>>> History and Evolution of AI/ML Platforms

"Platform in a box": Opinionated Point-Solutions

c. 2017-2020 - AWS SageMaker, GCP Vertex AI, Azure ML, Dataiku

Clunky - Enterprise ML teams outgrew their rigidity

Poor fault-tolerance and debuggability - built to sell



Benefits of SageMaker

Choice of ML tools

Enable more people to innovate with ML through a choice of tools—IDEs for data scientists and no-code interface for business analysts.

Poor incentives - compute inefficiency and gatekeeping

Studio Classic	JupyterLab	Code Editor	RStudio	Notebook Instances	Processing	TensorBoard
Data Wrangler	Feature Store	Training	MLflow	Real-Time Inference	Asynchronous Inference	
Batch Transform	Serverless Inference	JumpStart	Profiler	HyperPod	Inference optimization	



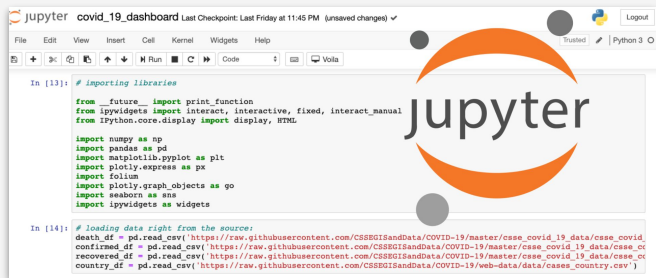
"MLOps": Fragmented Research+Production Stacks

c. 2020-2022 - Kubeflow, MLFlow, Airflow, Flyte, etc.

Slow down every
project by 6-9 months

Research
High velocity
Toy compute and data

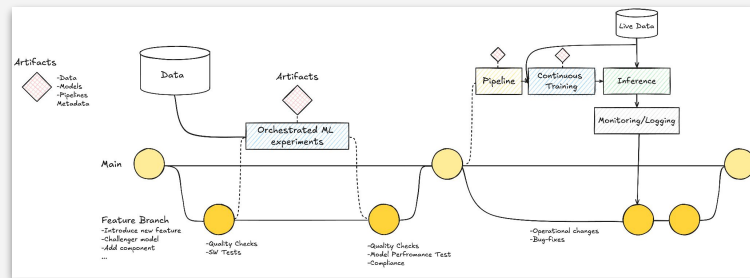
Production
Stable and reproducible
Push-and-pray development



```
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
from IPython.core.display import display, HTML

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import folium
import plotly.graph_objects as go
import seaborn as sns
import ipywidgets as widgets

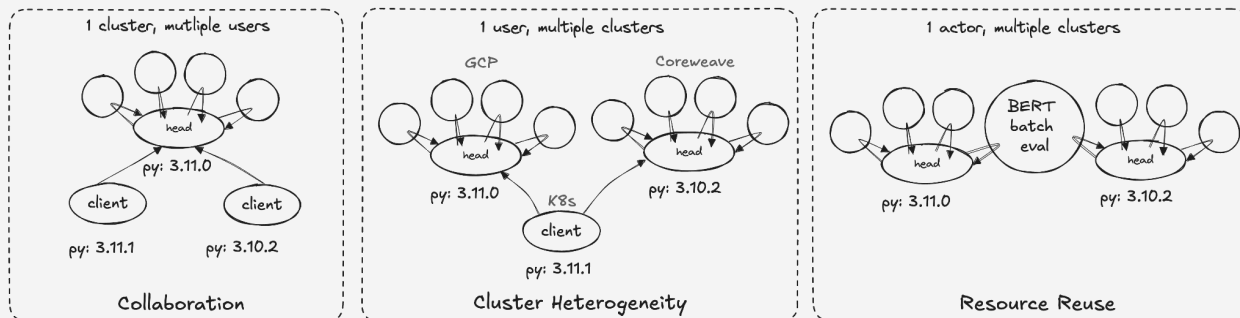
In [141]: # Loading data right from the source:
death_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_data/death_df.csv')
recovered_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_data/recovered_df.csv')
country_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/web-data/data/cases_country.csv')
```



"Unified Runtime": A Walled Garden

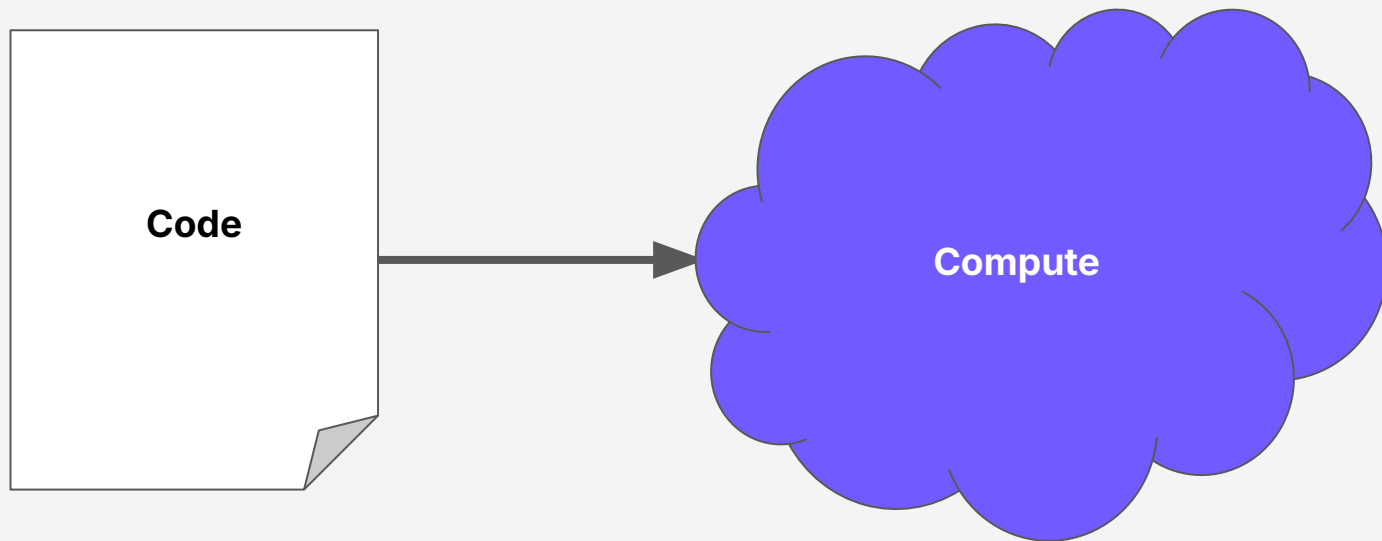
c. 2022-Present - Anyscale/Ray, Snowflake ML, Databricks ML

- Execution must be in-platform
 - Highly disruptive - Imagine no remote SQL submission to Snowflake, only UIs
 - Breaks in-platform flow 🧑 - Snowflake and Databricks only support distributed training in notebooks
- Platform opinionation / rigidity
 - e.g. Ray as "universal runtime" vs. "handy distributed DSL" - framework layering
 - Cascading failures, DSL / Code migration, heterogeneous jobs, reuse



>>> A Generic Runtime for AI/ML

What Should an ML Platform Look Like?



Hydra Architecture == Generic Runtime for ML

- We're not proposing a fancy, complicated system for enabling execution in this "hydra" approach
- It's the **opposite**: ML platforms team should provide an abstraction over compute that is robust enough for any task

Benefits of this Approach

Developer Experience:

- Just write code
- Fast research to production - identical execution

Cost:

- Picking the right-sized instances for each task in a pipeline
- Ability to access discounts or cloud credits without moving your entire stack

Availability / quota:

- Cloud provider sales team are opaque gatekeepers to quota
- Platforms team manually load-balance work

Portability:

- No migration costs when adopting new frameworks, tools, or infra

Runhouse: "Snowflake for ML"

Dispatch **arbitrary** ML code to be distributed and executed

Future-proof at every level:

- Any hardware
- Any Python code, any dependencies, any distribution (PyTorch, TF, Ray, etc.)
- Any CI/CD, orchestration, or deployment toolchain
- Compute and data remain entirely within your own cloud(s)

Sub-second iteration for research, fault-tolerance for production - identical and reproducible everywhere

Case study:

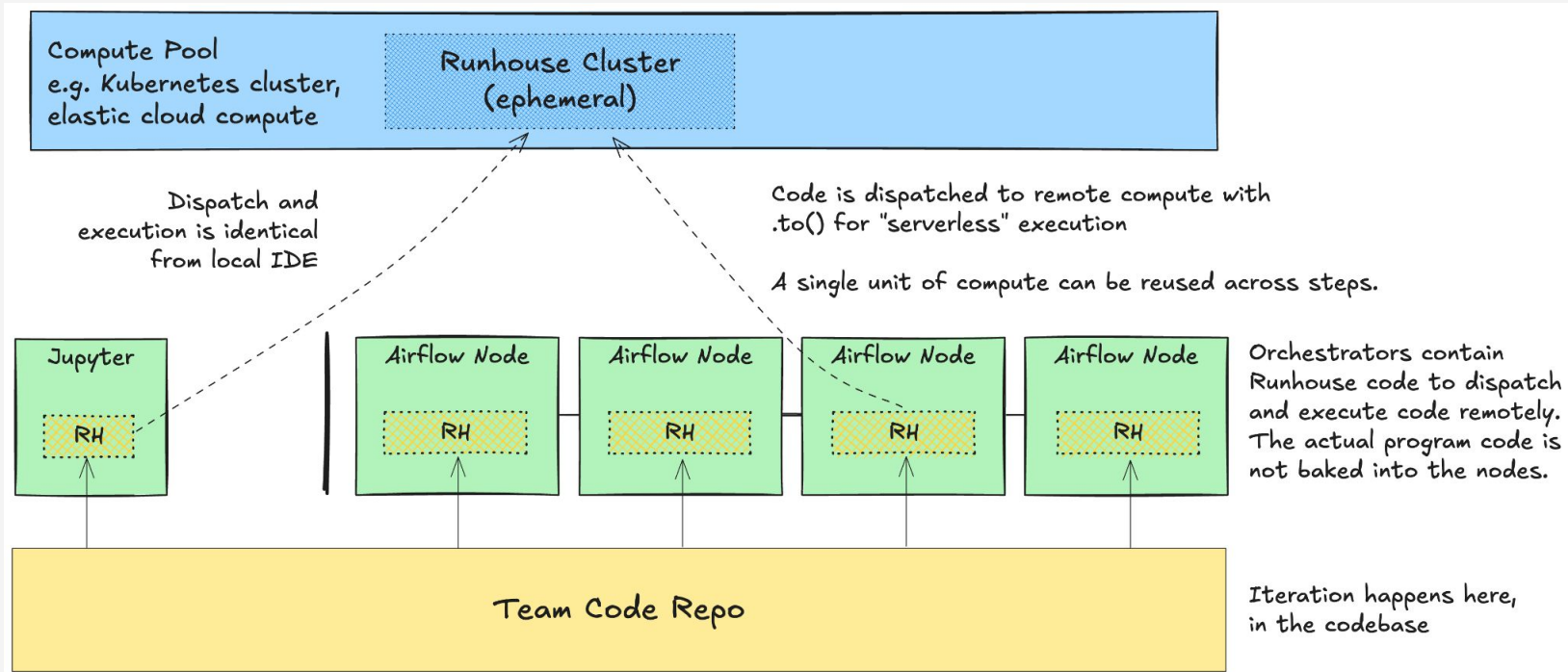
- Ranking team training
100s of models per day
- SageMaker→Runhouse
- 40% failure rate→<.5%
- 50% compute cost reduction
- 2 hour debugging loop→seconds

True Platform-as-a-Runtime - Execute ML from anywhere, like Spark or SQL

No runtime:

- No disruptive DevX
- No DSL / code migration - Removes framework layers instead of adding
- Orchestrate across regions/clouds
- No cascading failures

How Does It Work?



Any ML engineer can do distributed training on day 0

```
def train_gpu():
    gpu = rh.cluster(name="rh-a10g",
                    instance_type="A10G:1",
                    provider="aws")
    train_pytorch_gpu = rh.function(train_pytorch).to(gpu)
    train_pytorch_gpu(
        model="resnet18",
        dataset_path="s3://bucket/data",
        epochs=10,
        batch_size=32,
        learning_rate=0.001,
        momentum=0.9,
        weight_decay=0.0001,
        output_path="s3://bucket/output",
    )

if __name__ == "__main__":
    train_gpu()
```

```
import runhouse as rh
from ray import tune

from hpo_train_fn import train_fn

def find_minimum(num_concurrent_trials=2, num_samples=4, metric_name="score"):
    search_space = {
        "width": tune.uniform(lower=0, upper=20),
        "height": tune.uniform(lower=-100, upper=100),
    }

    tuner = tune.Tuner(
        train_fn,
        tune_config=tune.TuneConfig(
            metric=metric_name,
            mode="max",
            max_concurrent_trials=num_concurrent_trials,
            num_samples=num_samples,
            reuse_actors=True,
        ),
        param_space=search_space,
    )
    tuner.fit()
    return tuner.get_results().get_best_result()

if __name__ == "__main__":
    cluster = rh.cluster(
        name="rh-cpu",
        default_env=rh.env(reqs=["ray[tune]"]),
        instance_type="CPU:4+",
        provider="aws",
    ).up_if_not()
    remote_find_minimum = rh.function(find_minimum).to(cluster).distribute("ray")
    best_result = remote_find_minimum()
```

Unified "Compute Estate" - Collaboration, Auth, Observability, Admin

The image displays four overlapping screenshots of the Runhouse Den web interface, illustrating its unified management capabilities:

- Cluster Logs:** Shows a sidebar with navigation options (Dashboard, Resources, Account & Token, Secrets, Documentation, Examples, Github, Runhouse) and a main area with "Cluster Logs" and "Logs Tail" sections. The logs tail shows server logs for a cluster.
- Resource Monitoring:** Displays a "Runhouse Den" dashboard with navigation options and monitoring graphs for "CPU Memory Usage" and "CPU Utilization" over time.
- User Access:** Shows a "User Access" management page with options for "Overview", "Activity", and "User Access". It includes a form to "Invite new members" with fields for "Email or Username" (house.guest@example.com) and an "Add more" button.
- Resource Details:** Shows a detailed view of a resource named "sashab/rh-basic-gpu". It includes a sidebar with navigation options (Dashboard, Resources, Account & Token, Secrets, Documentation, Examples, Github) and a main area with "Resource Metadata" (Instance Type: g4dn.xlarge, Region: us-east-1) and "Resource Data" (resource_subtype: "OnDemandCluster", ips: ["54.165.144.56"], server_port: 32300, etc.).