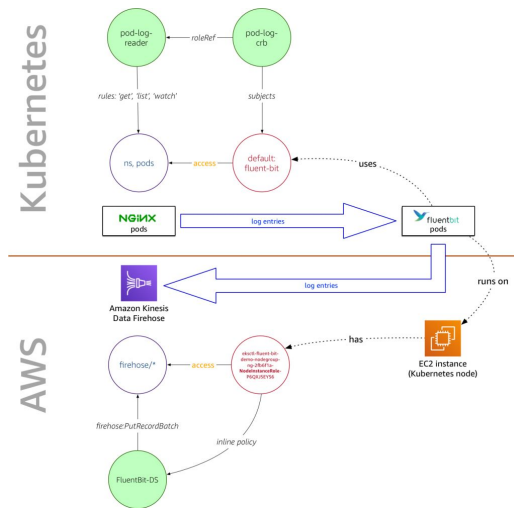


Taking the Reigns of the Cloud: A Human-Centric Approach to ML Experiments

Artem Trofimov, JetBrains

MLOps: It's Not Rocket Science...

- Experiments may take days/weeks
- GPUs are expensive and it is hard to allocate them due to shortage
- Dev!=Prod (many computers)
- Complex unmanaged migration (data & env)
- A lot of data that is hard to transfer
- Reproducibility issues
- Inference is expensive or inefficient



Problem setup

- ML engineer often writes code using local laptop
- Transition from local experiment to remote runtime is non-trivial
 - A lot of data
 - GPU shortage
 - Huge variety of tools
 - ...
 - VERY COMPLEX INTERFACES

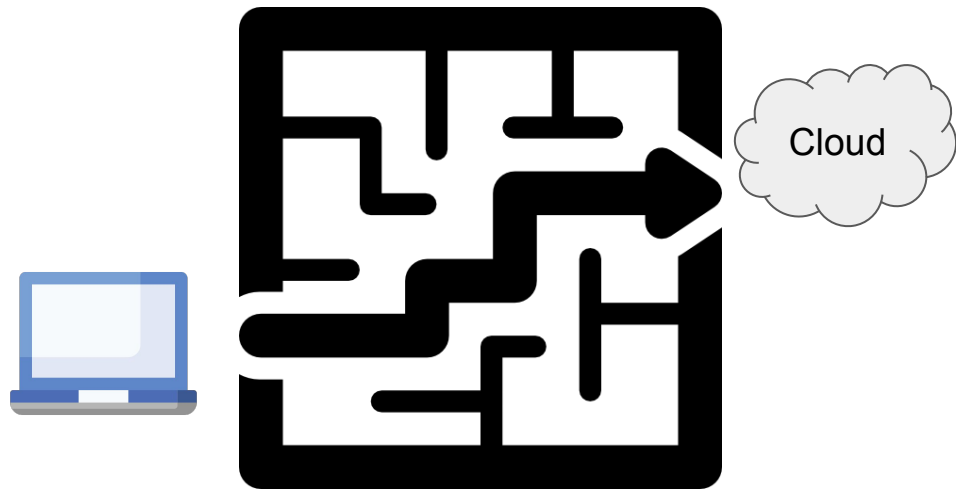


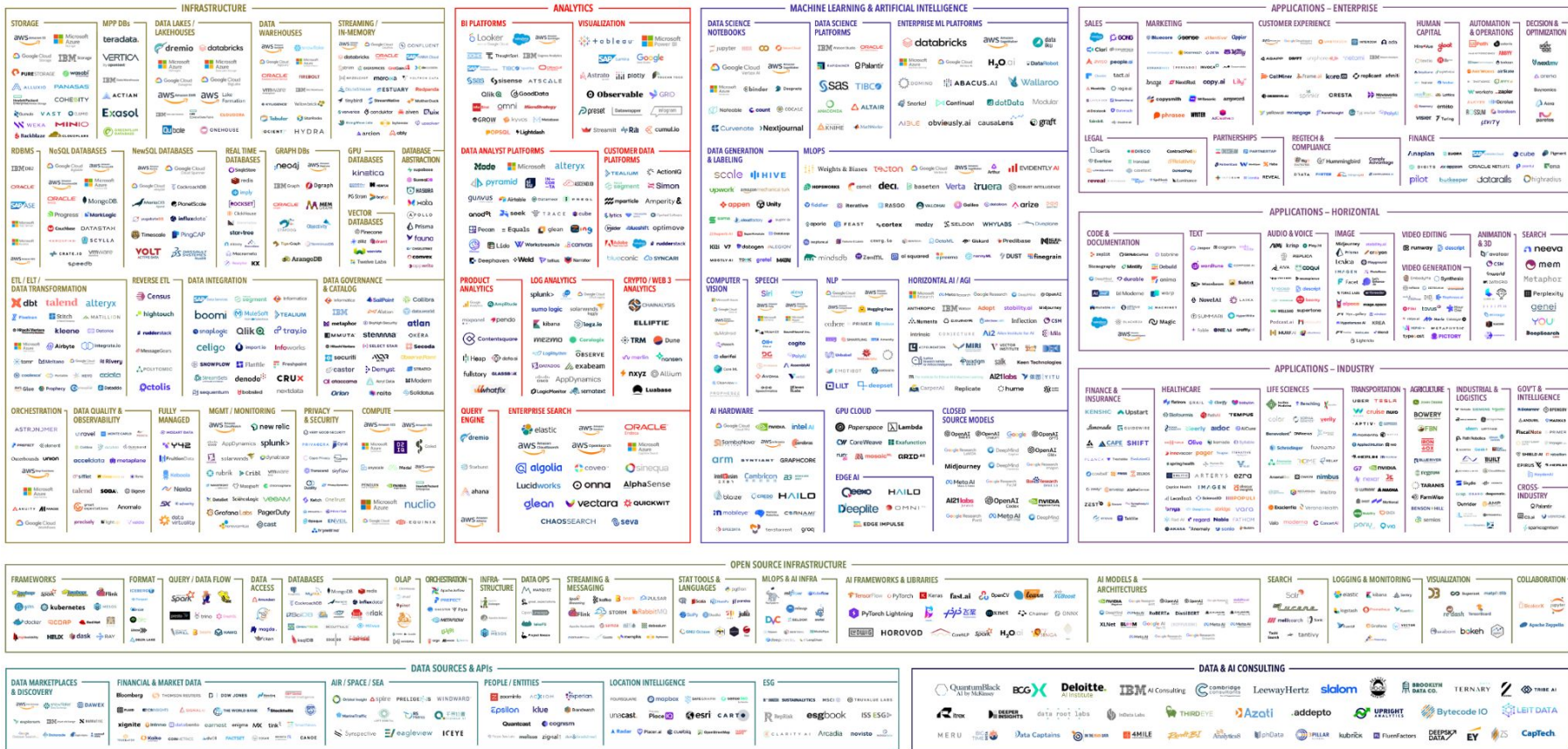
Image: Flaticon.com

Part 1

MLOps Toolkit: the Good the Bad, and the Complicated



THE 2023 MAD (MACHINE LEARNING, ARTIFICIAL INTELLIGENCE & DATA) LANDSCAPE



Version 1.0 - Feb 2023 © Matt Turck (@mattturck), Kevin Zhang (@ykevinzhang) & FirstMark (@firstmarkcap)

Blog post: mattturck.com/MAD2023

Interactive version: MAD.firstmarkcap.com

Comments? Email MAD2023@firstmarkcap.com

FIRSTMARK
EARLY STAGE VENTURE CAPITAL

<https://mattturck.com/landscape/mad2023.pdf>

VMs + SSH

- ✓ Full control
- ✓ Highly customizable



- Boilerplate for env/data synchronization
- Poor utilization due to manual lifecycle
- Full control -> reproducibility issues
- A lot of BASH

Jupyter Notebooks

- ✓ Full control*
- ✓ Highly customizable*
- ✓ Integrated with IDE

*depends on a specific managed solution



- Poor utilization (remote dev on a GPU machine)
- Reproducibility issues due to black-box state

Pipeline tools (Kubeflow, Ray, Flyte, Airflow, Prefect, etc.)

- ✓ Resources allocation on demand
- ✓ Structured stages
- ✓ Various resources for various stages*



- Pretty high entry level
- Vendor lock on the code level
- No remote debug

*supported by some engines only

Task scheduling tools (MosaicML, SkyPilot, DStack, etc.)

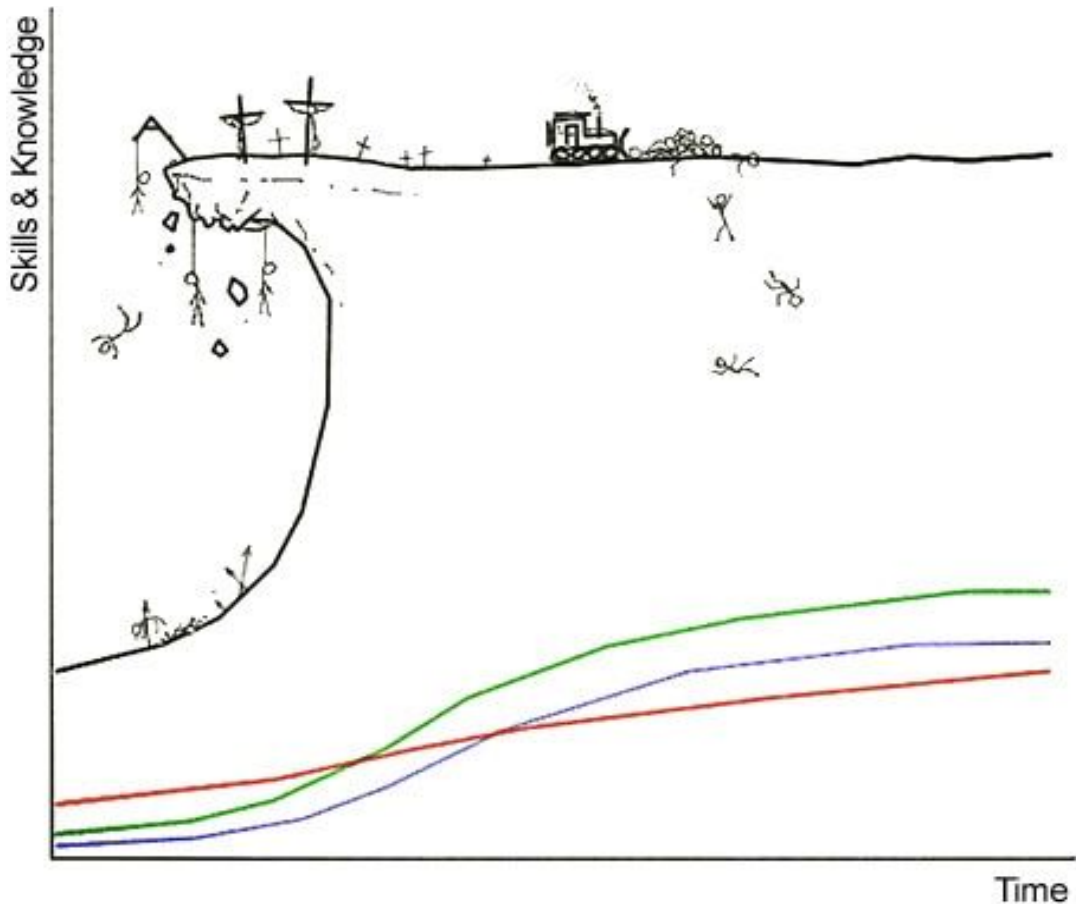
- ✓ Resources allocation on demand
- ✓ Simpler than pipeline tools



- YAML configs
- Cloud-specific settings
- No remote debug

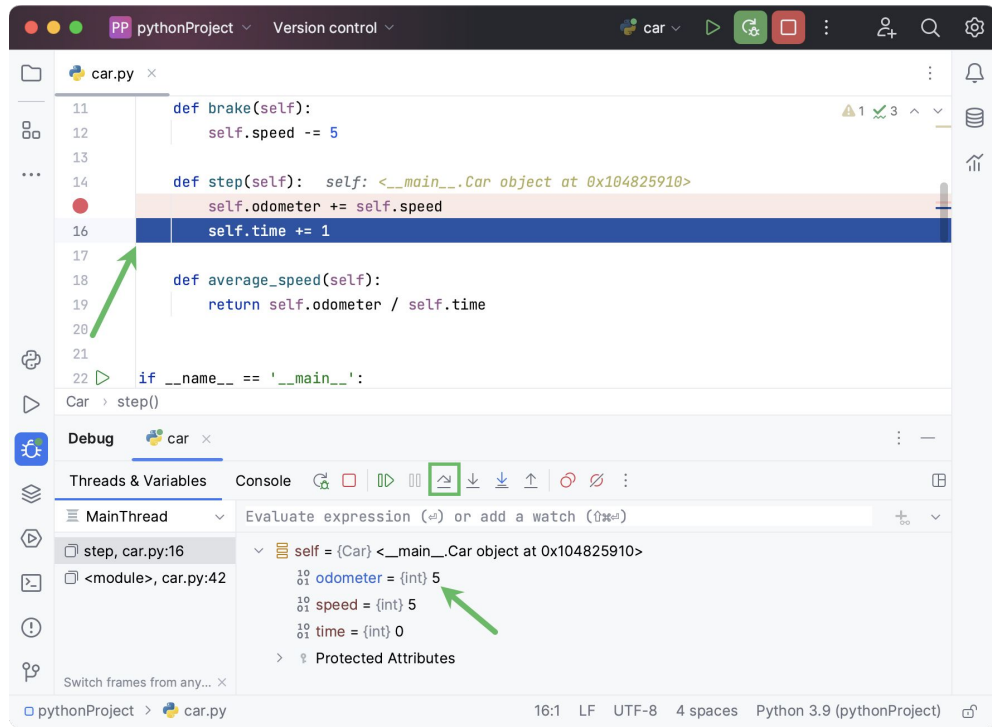
Overview

- Dev tools are simple and powerful, great for start but are not suitable for complex workflows
- Ops tools are ensuring reproducibility and scalability but have too complicated UX



Part 2

Can ML folks
stay within
IDE? Or we
have to use
other tools?



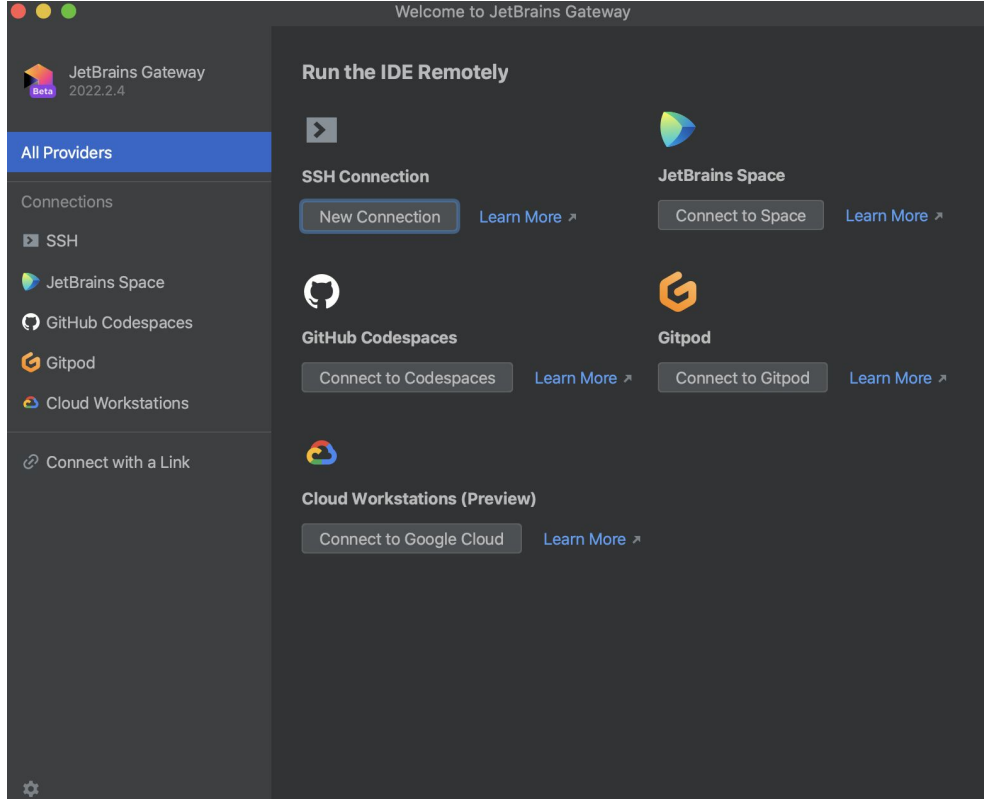
The screenshot shows a Python IDE window titled 'pythonProject' with a 'Version control' dropdown. The main editor displays the code for a `Car` class in `car.py`. The code includes methods `brake`, `step`, and `average_speed`. The `step` method is currently selected, and its execution is being debugged. A green arrow points to the line `self.time += 1` in the `step` method. Below the editor, the 'Debug' panel is open, showing the 'Threads & Variables' tab. The 'MainThread' is selected, and the 'Evaluate expression' field shows the state of the `self` object: `self = {Car} <__main__.Car object at 0x104825910>`. The object's attributes are listed: `odometer = {int} 5`, `speed = {int} 5`, and `time = {int} 0`. A green arrow points to the `odometer` attribute. The status bar at the bottom indicates the file is `pythonProject > car.py`, the cursor is at line 16, column 16, and the Python version is 3.9.

```
def brake(self):  
    self.speed -= 5  
  
def step(self):  
    self.odometer += self.speed  
    self.time += 1  
  
def average_speed(self):  
    return self.odometer / self.time  
  
if __name__ == '__main__':  
    Car().step()
```

Debug Console:

```
self = {Car} <__main__.Car object at 0x104825910>  
odometer = {int} 5  
speed = {int} 5  
time = {int} 0
```

IDE natively supports remote development!



Does remote development solve the problem?

- Remote development is great for *development*
- It is very inefficient to write code using VM with 8xA100 GPUs
- We need a tool at the intersection of development and MLOps

Can we delegate *some* runs to a remote VM?

- User should indicate:
 - Hardware requirements
 - Overridden parameters
 - Data to mount
- System should start VM, transfer env/data to it and run the task



IDE has enough meta-information for remote run!

CMD to run

Env to
setup/install

Data/code to
sync

The screenshot shows a configuration window for a remote run. The 'Name' field is 'fl-training'. The 'Module name' is 'train'. The 'Parameters' field contains a command: `-S hardware.per_device_batch_size=50 -S cloud_logger_project_name=python-training -S model=gpt2-12L-1024H -S hardware.precision=bf16`. The 'Environment' section is expanded, showing 'Environment variables' with `PYTHONUNBUFFERED=1;MLFLOW_TRACKING_USERNAME=mlflow`. The 'Python interpreter' section shows 'Use specified interpreter' selected, with 'Python 3.9 (py39)' chosen. The 'Working directory' is `/Users/Artem.Trofimov/Work/FL`. The 'Add content roots to PYTHONPATH' and 'Add source roots to PYTHONPATH' checkboxes are checked. The 'Execution' section is collapsed.

Name: ☒ Allow multiple instances ☐ Store as project file

Configuration Logs

Module name:

Parameters:

Environment

Environment variables:

Python interpreter: ☐ Use SDK of module: ☒ Use specified interpreter:

Interpreter options:

Working directory:

☒ Add content roots to PYTHONPATH
☒ Add source roots to PYTHONPATH

> Execution

Running ML experiments from IDE: UX

The screenshot displays the JetTrain IDE interface. At the top, a 'New JetTrain execution' dialog box is open, allowing users to configure a new experiment. Below this, a table lists existing experiments. A context menu is open over the first row of the table, showing options like 'Cancel', 'Rerun...', 'Terminal', 'Remote debug', 'Logs', and 'Results'. A curved arrow highlights the 'Run' button in the dialog box and the 'Run' button in the context menu.

New JetTrain execution

Alias:

Run configuration:

VM configuration:

Parameters:

JetTrain

#3	exp v2	train.py	m7gd.xlarge	2023-11-09 13:36:48	Running	36m 24s	⋮
#2	experiment	train.py	m7gd.xlarge	2023-11-08 21:44:13	Error	2m 01s	⋮
#1	first try	train.py	m7gd.xlarge	2023-11-08 10:14:02	Success	46m 10s	⋮

- Cancel
- Rerun...
- Terminal
- Remote debug
- Logs
- Results

IDE-native ML experiments

- ✓ UX similar to local runs
- ✓ No context switching
- ✓ On-demand resources



- No pipelines
- User should be familiar with IDE

Open questions

- Tasks scheduling (under GPU shortage)
- Env synchronization
- Data synchronization

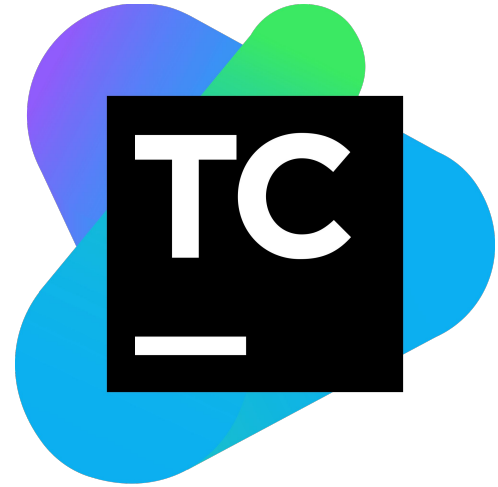
Part 3

Confronting The Obstacles: Which Technical Challenges We Have?



Tasks scheduling, user management, billing, etc.

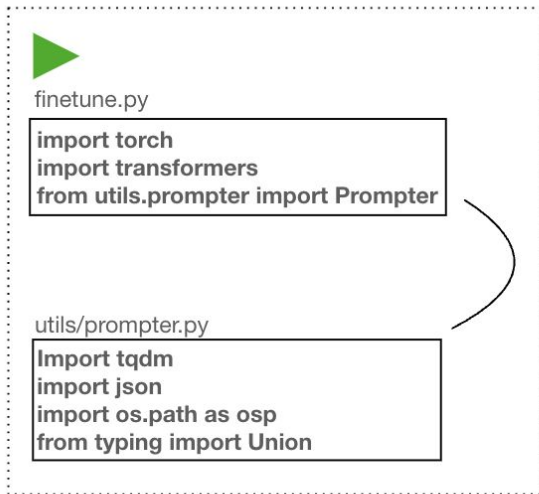
- These problems are really hard but fortunately we have TeamCity!
- We can delegate cloud integration, scheduling, logs delivery, etc., to TeamCity
- TeamCity brute forces available cloud providers and their zones to find VMs quickly!



Mirror, mirror on the wall, sync the best environment of all

- It is easy if user has requirements.txt/conda.yaml/etc.
- To do it automatically we can recursively analyze modules of global variables starting from main module
- This algorithm is precise but does not detect imports inside functions

Automatic env detection algorithm



Result

conda.yaml:	local modules:
python==3.9.7	/alpaca-lora/
torch==2.0.1	/alpaca-lora/utils/
transformers==4.30.0	
tqdm==4.65.0	

Data synchronization: a naive way

- aws s3 sync/cp
- If data and VM are in different data centers, data transfer can be slow and expensive

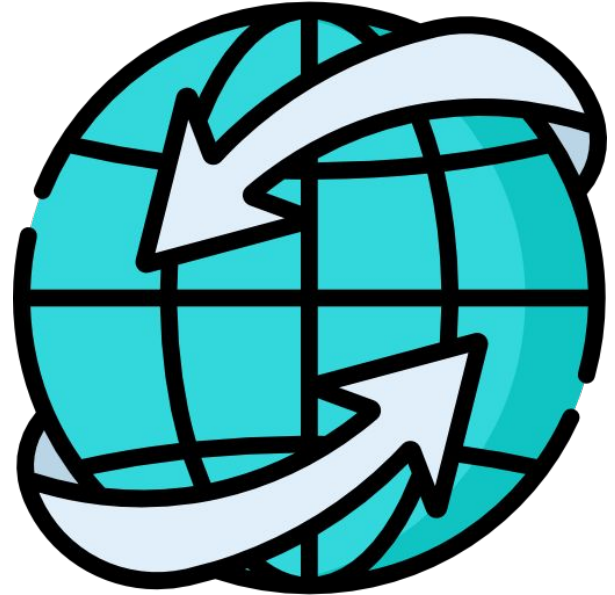
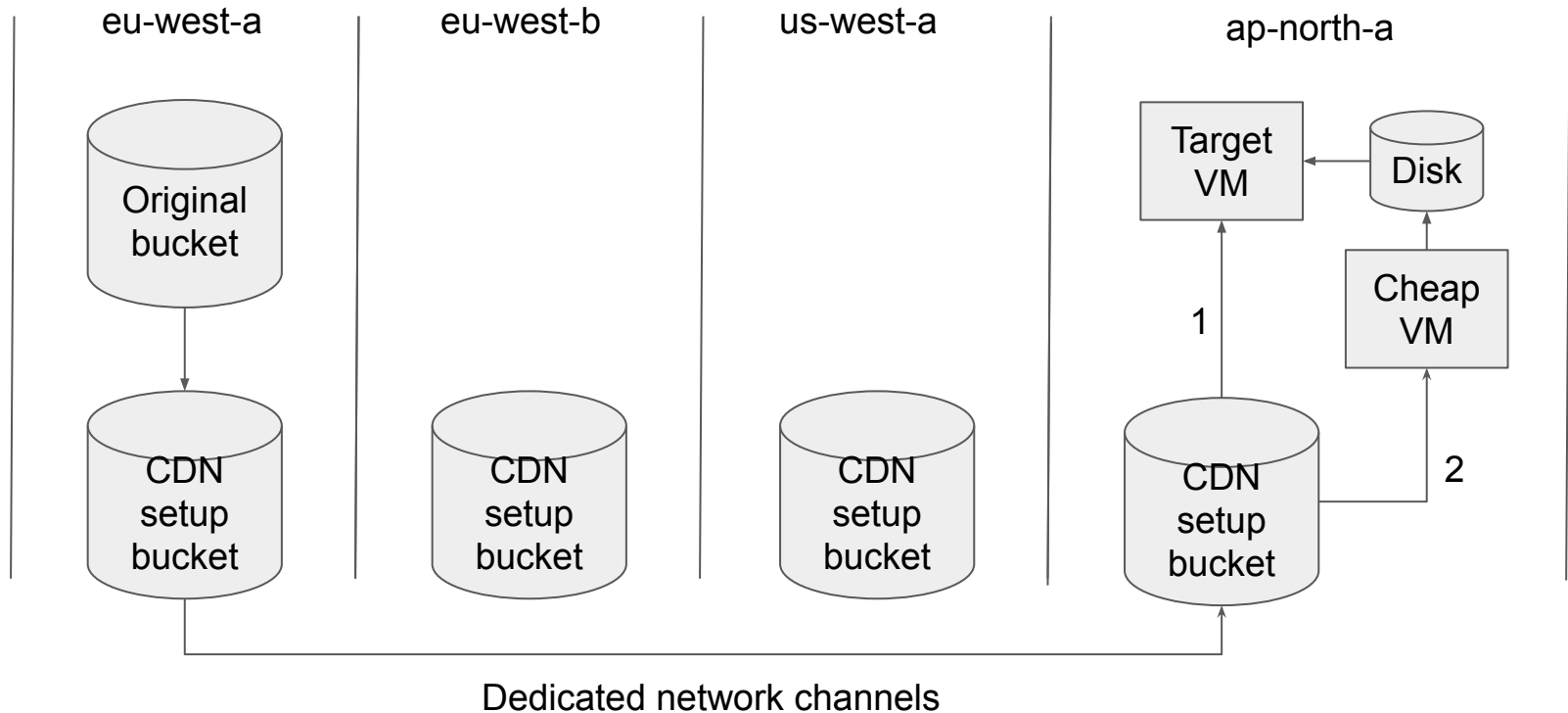


Image: Flaticon.com

Optimized way: data dances amid the great GPU drought



A Peek at Our Current MLOps Expedition



Generated by DALL-E

- First adopters: 5 internal ML teams
- 500+ hours/week VMs consumption
- Stable for long trainings (tested on 8+ days trainings)
- Private preview for external clients in Q1 2024

Our current status



- First adopters: 5 internal ML teams
- 500+ hours/week VMs consumption
- Stable for long trainings (tested on 8+ days trainings)
- Private preview for external clients in Q1 2024

IDE-native ML experiments?

Let's discuss!

feel free to contact me regarding any questions/ideas:

artem.trofimov@jetbrains.com

telegram: @tyooma