KubeCon | CloudNativeCon
North America 2023

Supercharge your AI Platform with KubeRay: Ray + Kubernetes

Archit Kulkarni, Software Engineer, Anyscale
Winston Chiang, Product Manager, Google

Presenter Information

Archit Kulkarni
Software Engineer, Anyscale

Winston Chiang
Product Manager, Google

Agenda

- Introduction
- What is Ray
- Why Kubernetes
- What is KubeRay
- Demo: LLM lifecycle with KubeRay
- Conclusion and Q&A

AI is all around you

NIANTIC

cohere

Spotify

shopify

samsara

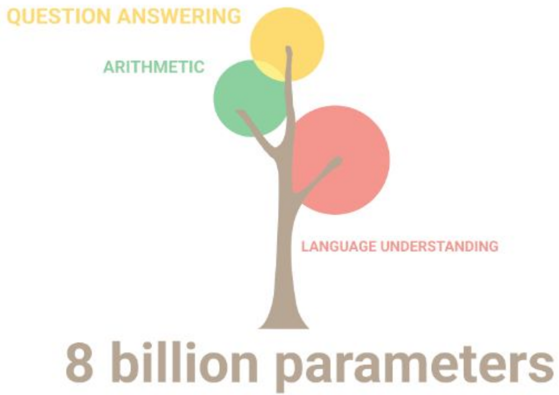
Uber

Adobe

OpenAI

ByteDance

Large models gain new abilities



Source: "Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance" by Sharan Narang and Aakanksha Chowdhery, Google Research Blog

AI Platform Trends

As AI capabilities have grown, so have the challenges

Scale

Cost

Future Proof

Unified AI Platform

Team 1 | Team 2 | Team 3 | Team 4 | Team 5

Workspaces | Jobs | Services

Ray unified framework for Scalable Compute

Ray Data
Ray Train
Ray Tune
RLLib
Ray Serve

Ray AI Libraries

Kubernetes unified compute

Autoscaling | Placement | Provisioning

Multi-Instance
TimeSharing
Local SSD
GCS Fuse
Fast Socket
gVNIC

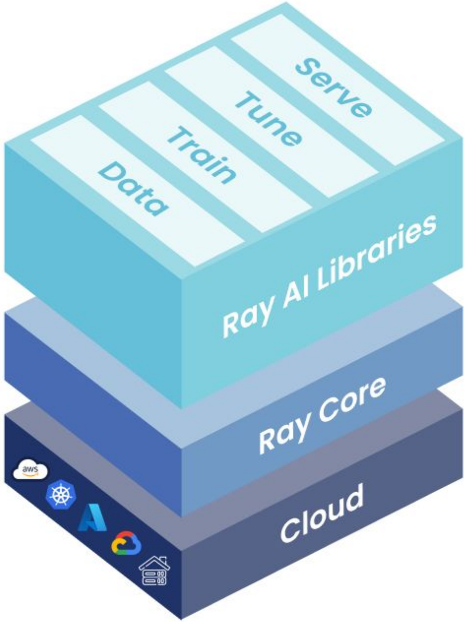
Compute
GPU
TPU
Storage
Network

Why Ray

→ Open source unified framework for scaling AI and Python

→ User doesn't need to be a distributed systems expert!

→ Ray automatically handles orchestration, scheduling, fault tolerance, autoscaling



high-level libraries that enable simple scaling of AI workloads

a low-level distributed computing framework with a concise core and Python-first API

Developing AI Applications with Ray

→ Ray AI Libraries:

+ Data: Scalable, framework-agnostic **data loading and transformation** across training, tuning, and prediction.
+ Train: Distributed multi-node and multi-core **model training** with fault tolerance that integrates with popular training libraries.
+ Tune: Scalable **hyperparameter tuning** to optimize model performance.
+ Serve: Scalable and programmable **serving** to deploy models for online inference, with optional micro-batching to improve performance.
+ RLlib: Scalable distributed **reinforcement learning** workloads.

When to use Ray & Ray AI Libraries?

Scale a single type of workload

- Data ingestion for ML
- Batch Inference at scale
- Distributed Training
- Only serving or online inference

Scale end-to-end ML applications

Data → Train → Tune → Serve

Run ecosystem libraries using a unified API

Ray Train

Build a custom ML platform

- Spotify, Instacart
- Pinterest & DoorDash
- Samsara & Niantic
- Uber Eats & LinkedIn

Ray Core API

Functions -> Tasks

```
def read_array(file):
    # read array “a” from “file”
    return a

def add(a, b):
    return np.add(a, b)
```

Ray Core API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)
```

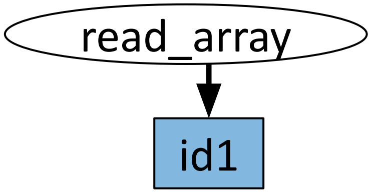
Ray Core API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(“/input1”)
```



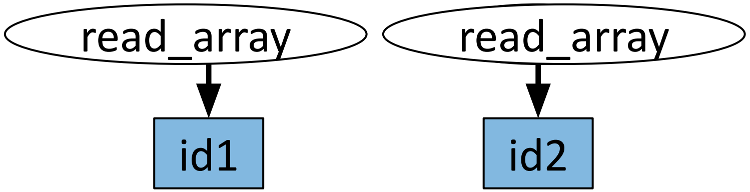
Ray Core API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array "a" from "file"
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote("/input1")
id2 = read_array.remote("/input2")
```



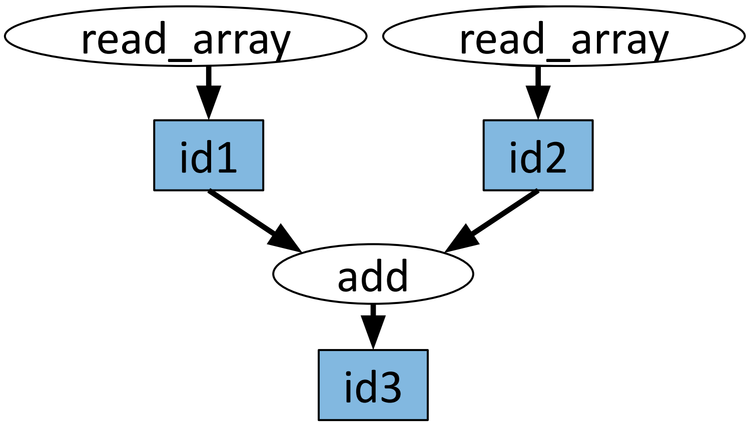
Ray Core API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array "a" from "file"
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote("/input1")
id2 = read_array.remote("/input2")
id3 = add.remote(id1, id2)
```



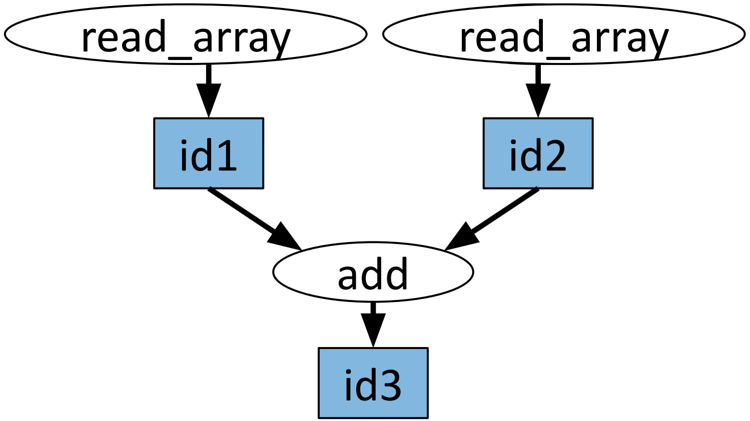
Ray Core API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(“/input1”)
id2 = read_array.remote(“/input2”)
id3 = add.remote(id1, id2); ray.get(id3)
```



Ray Core API

Functions -> Tasks

```python
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(“/input1”)
id2 = read_array.remote(“/input2”)
id3 = add.remote(id1, id2)
```

Classes -> Actors

Ray Core API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(“/input1”)
id2 = read_array.remote(“/input2”)
id3 = add.remote(id1, id2)
```

Classes -> Actors

```
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value
```

Ray Core API

Functions -> Tasks

```python
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(“/input1”)
id2 = read_array.remote(“/input2”)
id3 = add.remote(id1, id2)
```

Classes -> Actors

```python
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
ray.get([id4, id5])
```

Ray Core API

Functions -> Tasks

```python
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote(num_gpus=1)
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(“/input1”)
id2 = read_array.remote(“/input2”)
id3 = add.remote(id1, id2)
```

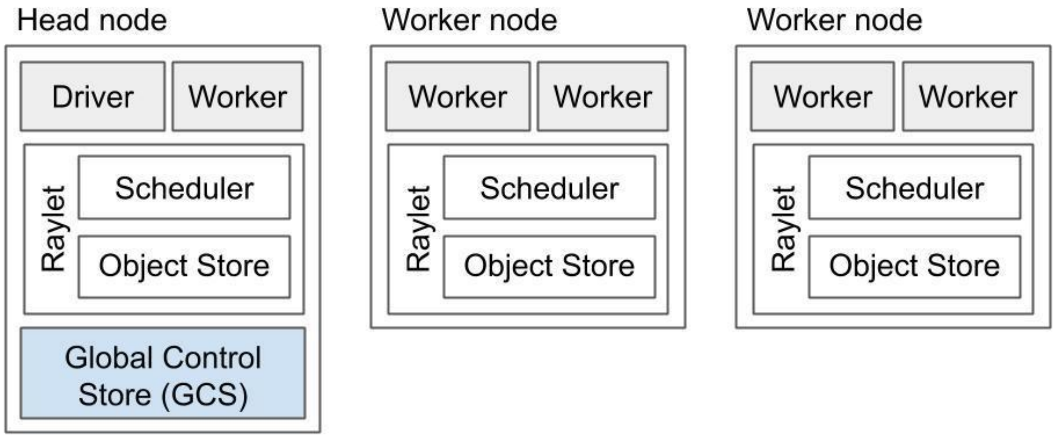
Classes -> Actors

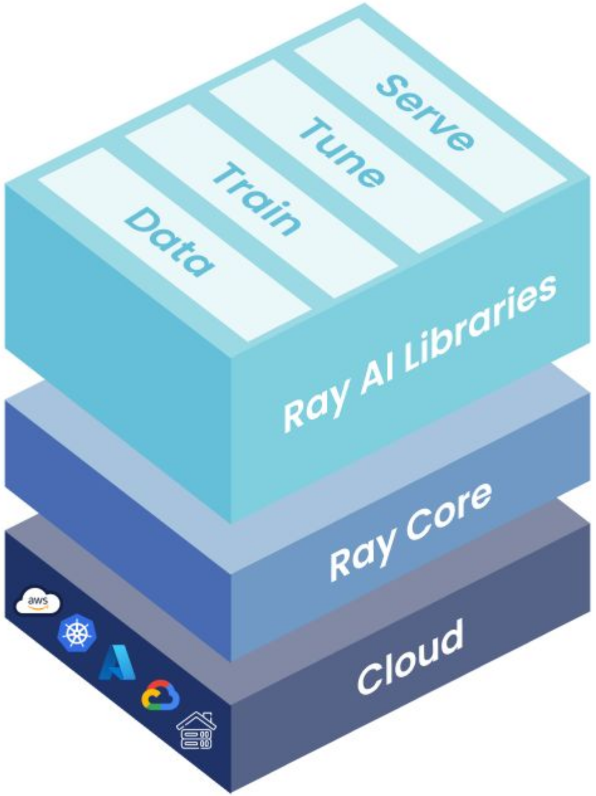
```python
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
ray.get([id4, id5])
```

Ray Architecture

- **Raylet**: Manages shared resources on each node
- **GCS**: Manages cluster-level metadata
- **Worker process**: Task/Actor execution
- **Driver process**: Special worker process executing the top-level application (e.g. __main__)





high-level libraries that enable simple scaling of AI workloads

a low-level distributed computing framework with a concise core and Python-first API

Why Kubernetes with Ray?

→ Get the unified Python experience delivered by Ray

→ Together with the operational benefits of Kubernetes.

- Infrastructure Automation
- Scalability
- High Availability and Reliability
- Resource Efficiency
- Complex, multi-cloud deployments

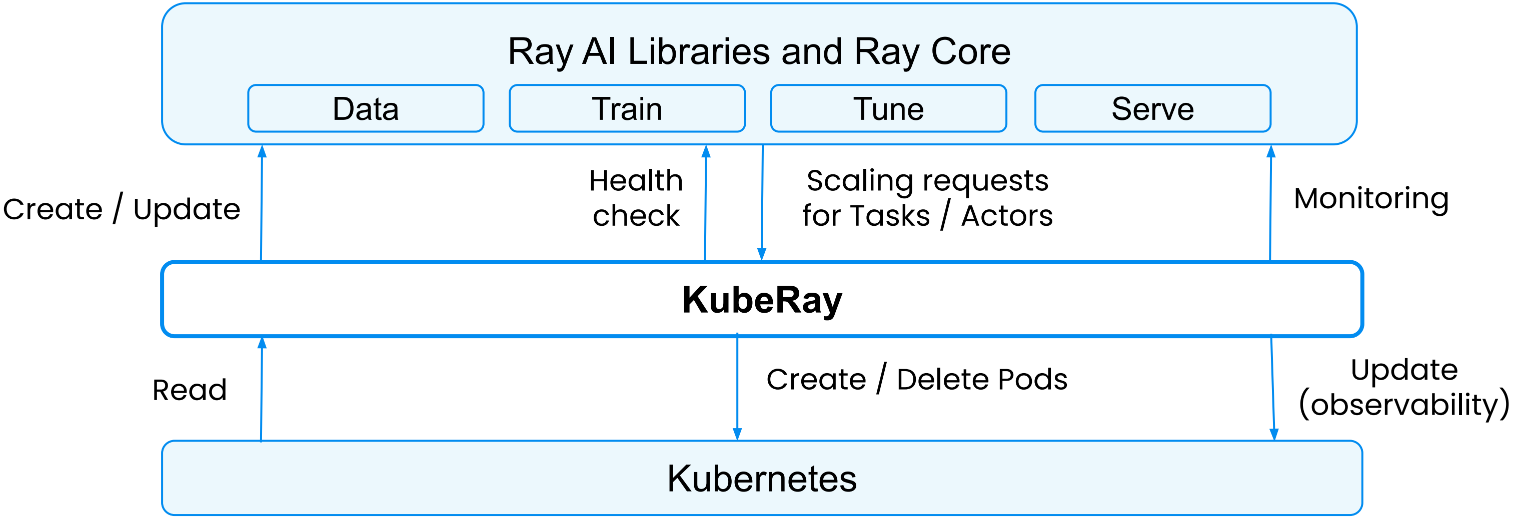
Ray on Kubernetes

- Both **Ray** and **Kubernetes** serve as resource orchestrators.
- Ray focuses on **computation** with **Tasks and Actors** as its scheduling unit.
- Kubernetes focuses on **deployment** with **Pod** as its scheduling unit.

KubeRay: The best solution for Ray on Kubernetes

- KubeRay enables **data/ML scientists** to focus on computation while **infra engineers** concentrate on Kubernetes.

data/ML scientists: Develop Python scripts.



infra engineers: Integrate KubeRay with Kubernetes ecosystem tools, e.g. Prometheus, Grafana, and Nginx.

KubeRay: 3 core components for different workloads

RayCluster

- Manage lifecycle of Ray cluster
- **Autoscaling**
- **GCS fault tolerance**

RayJob = RayCluster + Job

- Creates a RayCluster
- Submits job when ready
- RayCluster can be **recycled automatically**

RayService = RayCluster + Ray Serve

- Creates a RayCluster and deploys Ray Serve applications on it
- **In-place update**
- **Zero-downtime upgrade**
- **High-availability**

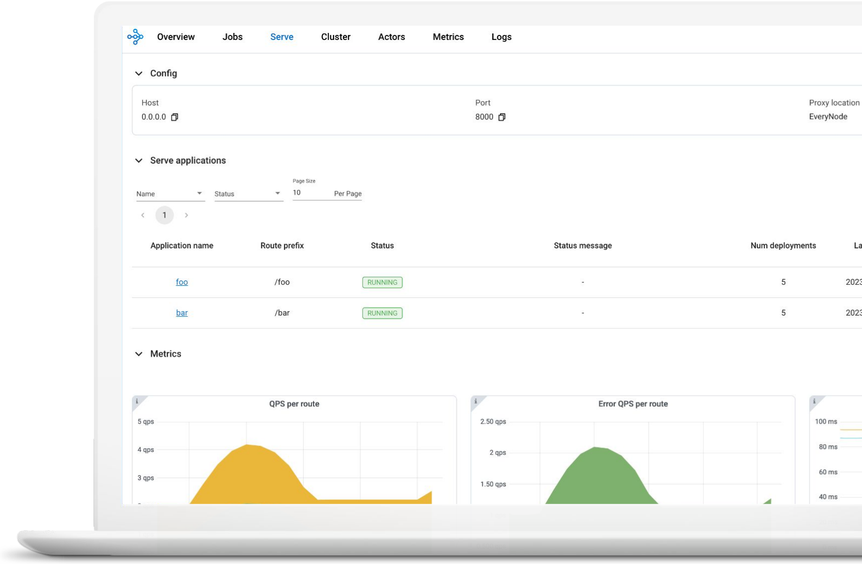
Benefits of KubeRay

Scalability and Performance

Cost and Efficiency

Customizability

Portability

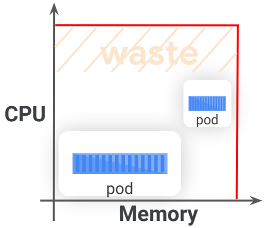


Scalability and Performance

Fine tune performance and scale the platform.

- Distributed, high performance accelerators
- Scale the platform from 1 to 1000's to meet the needs of all of your AI workloads
- Ray achieves zero-downtime by monitoring unhealthy states

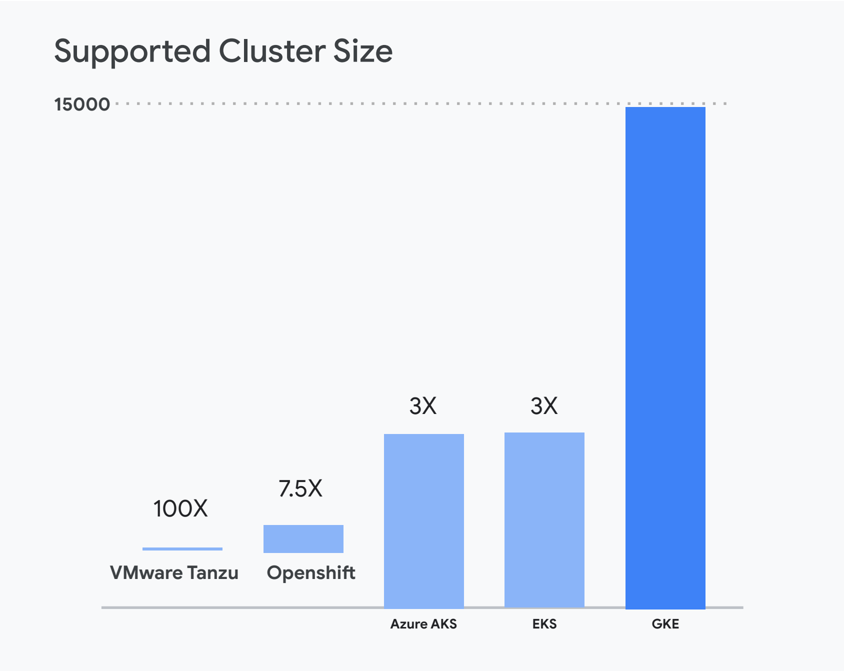
Utilization



Bin Packing

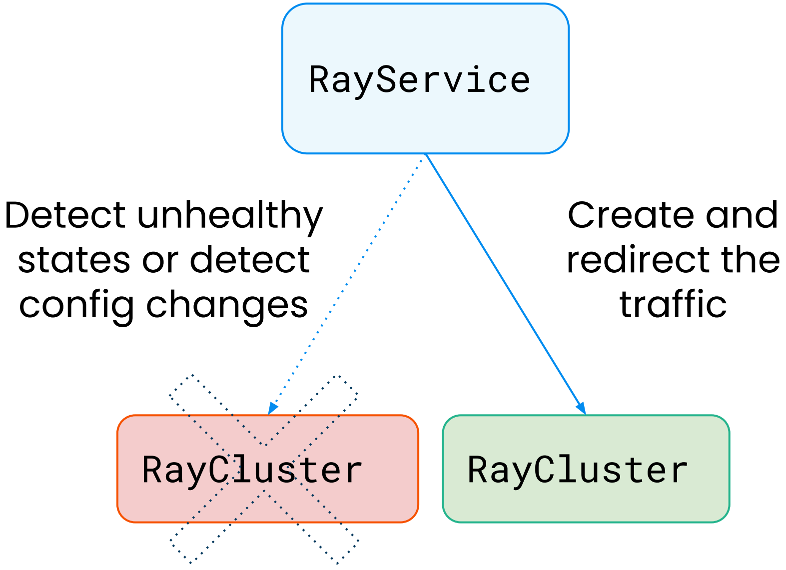
Kubernetes helps you achieve and manage scale

- AI and GenAI models demand massively distributed compute
- Kubernetes is the proven solution
- The largest models were trained on Kubernetes

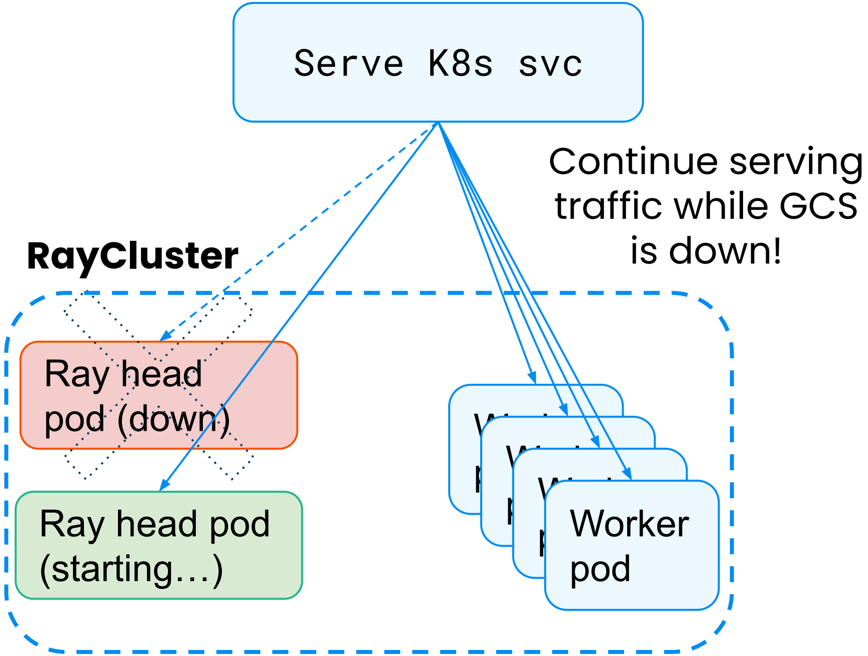


RayService stability features

Zero-downtime

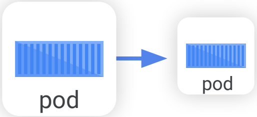


High-availability + GCS fault tolerance



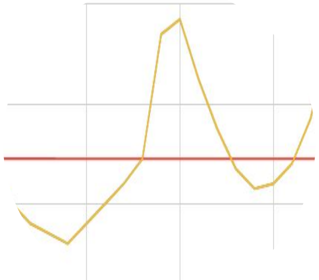
Pay for what you need when you need it

- Higher utilization of compute resources (CPUs, GPUs, TPUs) and cost savings with Spot
- Reduced operational costs for unified platform



Workload Rightsizing

Actual utilization
vs
Requested resources

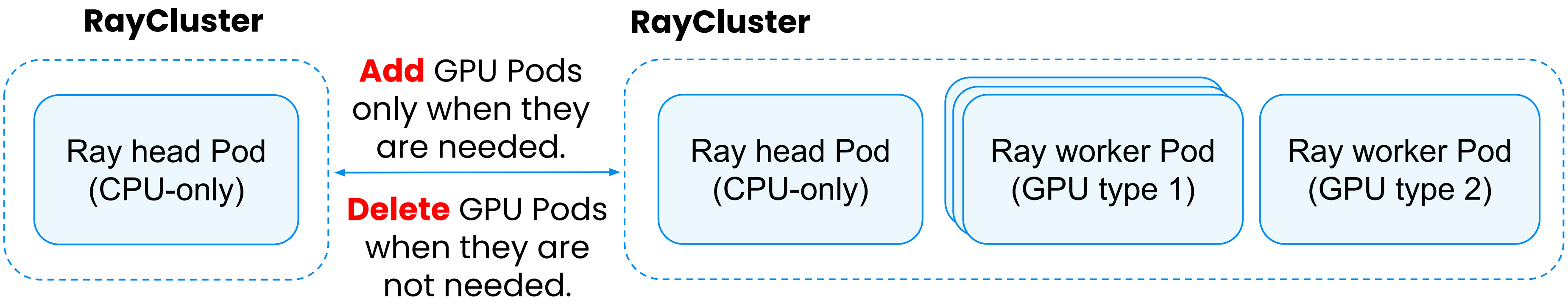


Off-peak hours

Low demand
should drive
cluster scale down

Ray Autoscaler on KubeRay

- KubeRay seamlessly integrates with Ray Autoscaler.
- This is fine-grained **application-level** autoscaling (smarter than default K8s autoscaler, which can't see into the application!)



Supports heterogeneous resources (e.g. high-end GPUs for LLM serving)!

Customizability

Choose the best framework(s) for the job

- Meet the needs of multiple teams with their framework of choice
- Customize the platform to meet your structure and requirements
- Ray is platform-agnostic.

Spark RAY W&B RAPIDS LangChain Hugging Face

Vibrant ecosystem of frameworks from which to choose!
www.ray.io/integrations

Portability

Write Once.
Run Everywhere.

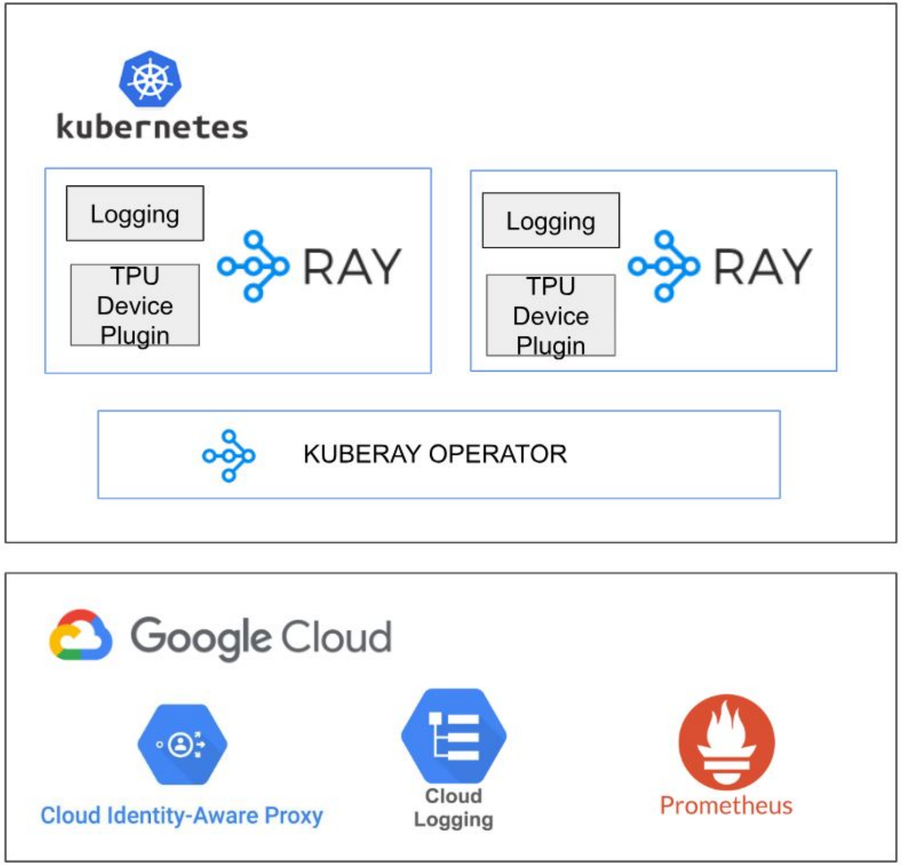
- Train and serve with the same ML program across clouds and on-premises
- Cloud native, Open standards

K8s is the industry standard compute orchestration platform available anywhere you need it
Ray is simply Python

Getting Started: Ray Solution Templates on GKE

- Out-of-box Terraform integrations:
  - GPU/TPU-enabled clusters
  - Kuberay operator
  - IAP-enabled endpoint
  - Cloud Logging
  - Prometheus monitoring
  - Jupyter notebook server
- User guide is available at link
- Kubecon Session by Richard Liu – Accelerate your GenAI Model Inference with Ray and Kubernetes

  Thursday, November 9 • 2:55pm - 3:30pm



Kuberay Benefits

Faster

Cheaper

Kuberay Customers achieving greatly improved efficiencies

Customer	Result
instacart	12x faster
samsara	50% cheaper
Pinterest	40% cheaper
amazon	10x cheaper
Clari	5x faster
DOORDASH	30% cheaper

NIANTIC

https://nianticlabs.com/news/ray

Thanks to Ray … we are able to reduce scan processing time by 75%, cost per scan by more than 60%, and number of lines of code by more than 85%….
We heavily rely on KubeRay to reliably create, autoscale, and shutdown our production Ray clusters.

Massive growth in the number of KubeRay clusters

+37% per month avg growth over the last 6 months

4 major releases

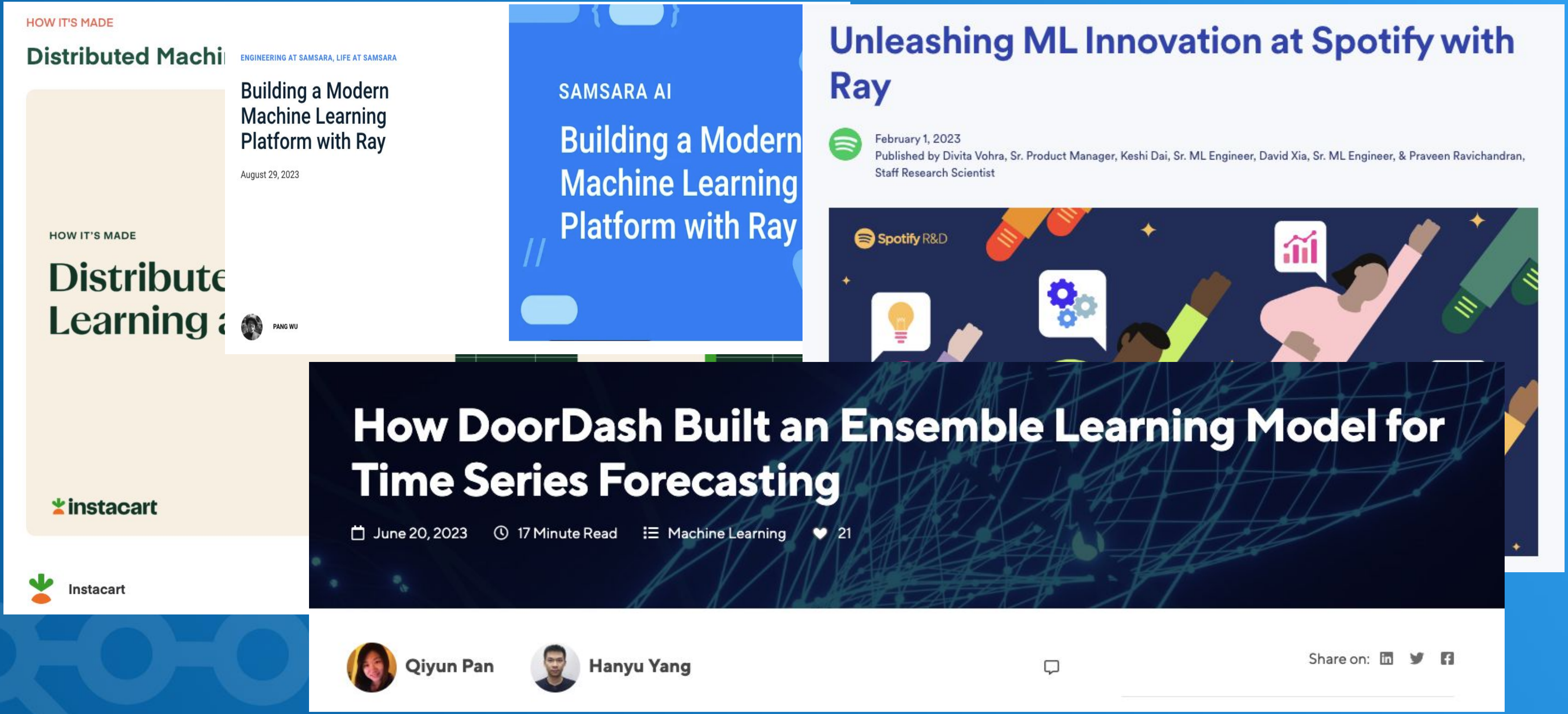
230 commits → 600+ commits

100+ contributors

100+

contributors

10+ External Blogs

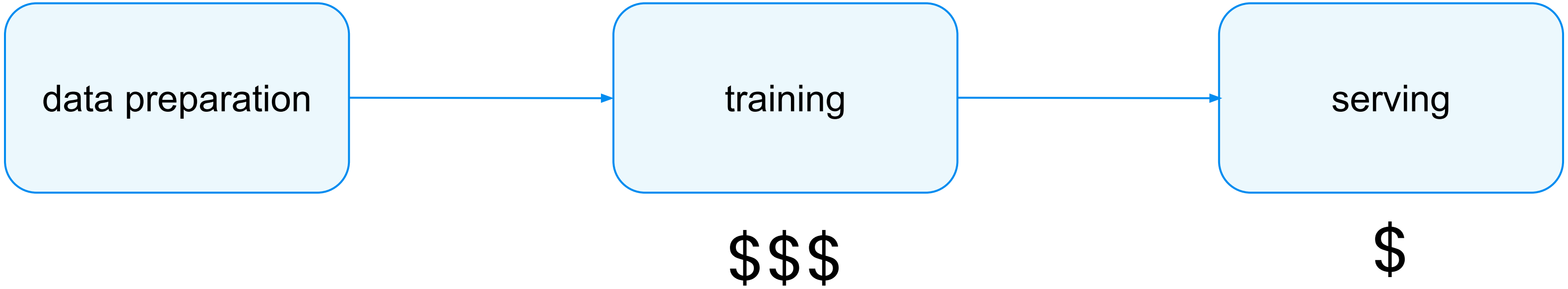


KubeRay is GA!

We're happy to announce that KubeRay is generally available, packed with enhanced features, robust stability, and community-driven improvements. KubeRay v1.0.0 is now available.

LLMs with KubeRay

Traditional ML model lifecycle



- CPU-inference
- Single/Small GPU inference
- Compute-intensive

LLM model lifecycle

data preparation → fine-tuning → serving → LLM applications

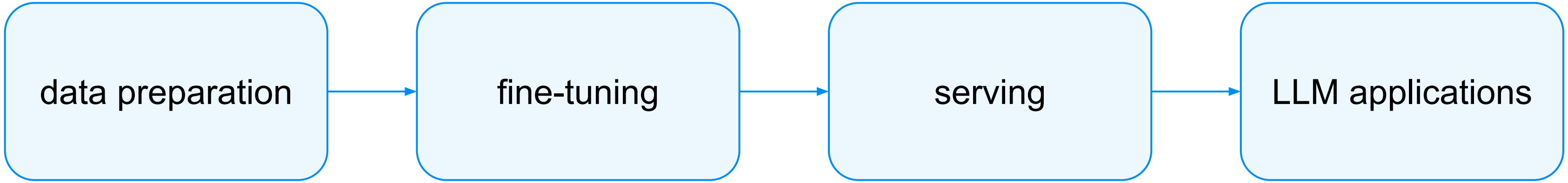
pre-training → fine-tuning

Only a few big techs can afford it!

$

Most users use open-source pretrained LLMs

LLM model lifecycle



$ $$$

Serving becomes much more expensive and important!

- Multiple / Large GPU(s) inference
- Memory-intensive

LLMs with KubeRay

- KubeRay can manage the end-to-end LLM lifecycle on Kubernetes.
- KubeRay/Ray are the best solution for LLM on Kubernetes, especially for **LLM serving** cost.
    - **Autoscaling:** It is hard to predict the traffic for online serving. KubeRay supports autoscaling, which adjusts based on dynamic load, to save costs.
    - **Heterogeneous:** High-end GPUs are in high demand. Supporting heterogeneous computing resources, such as different types of GPUs, TPUs, and CPUs, is important.

Demo: End-to-end LLM lifecycle with KubeRay

Demo: End-to-end LLM lifecycle with KubeRay

- LLM lifecycle

RayCluster + Ray Core

RayService + RayLLM

data preparation → fine-tuning → serving

RayJob + Ray Train

serving → LLM applications

prototyping

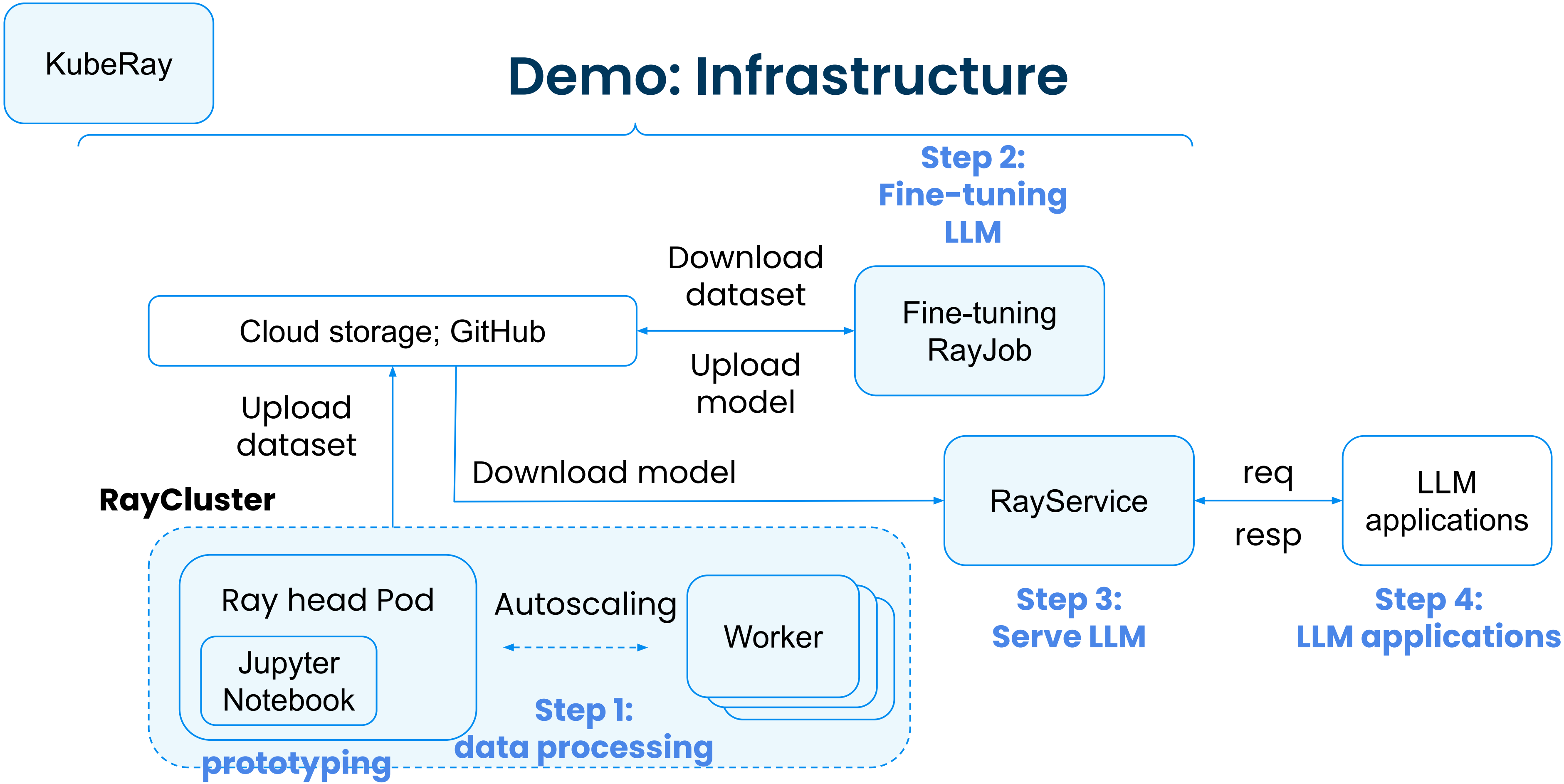
RayCluster + Jupyter Notebook

open-source pretrained LLMs

LLM applications

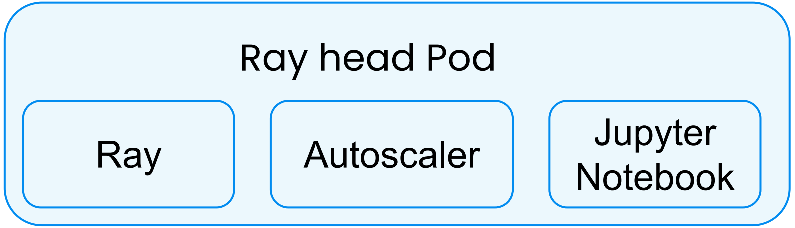
LangChain + Gradio

Demo: Infrastructure



Prototyping

- Run a Jupyter Notebook as a head Pod’s sidecar container.
- KubeRay injects an Autoscaler sidecar container into the head Pod if the autoscaling is enabled.

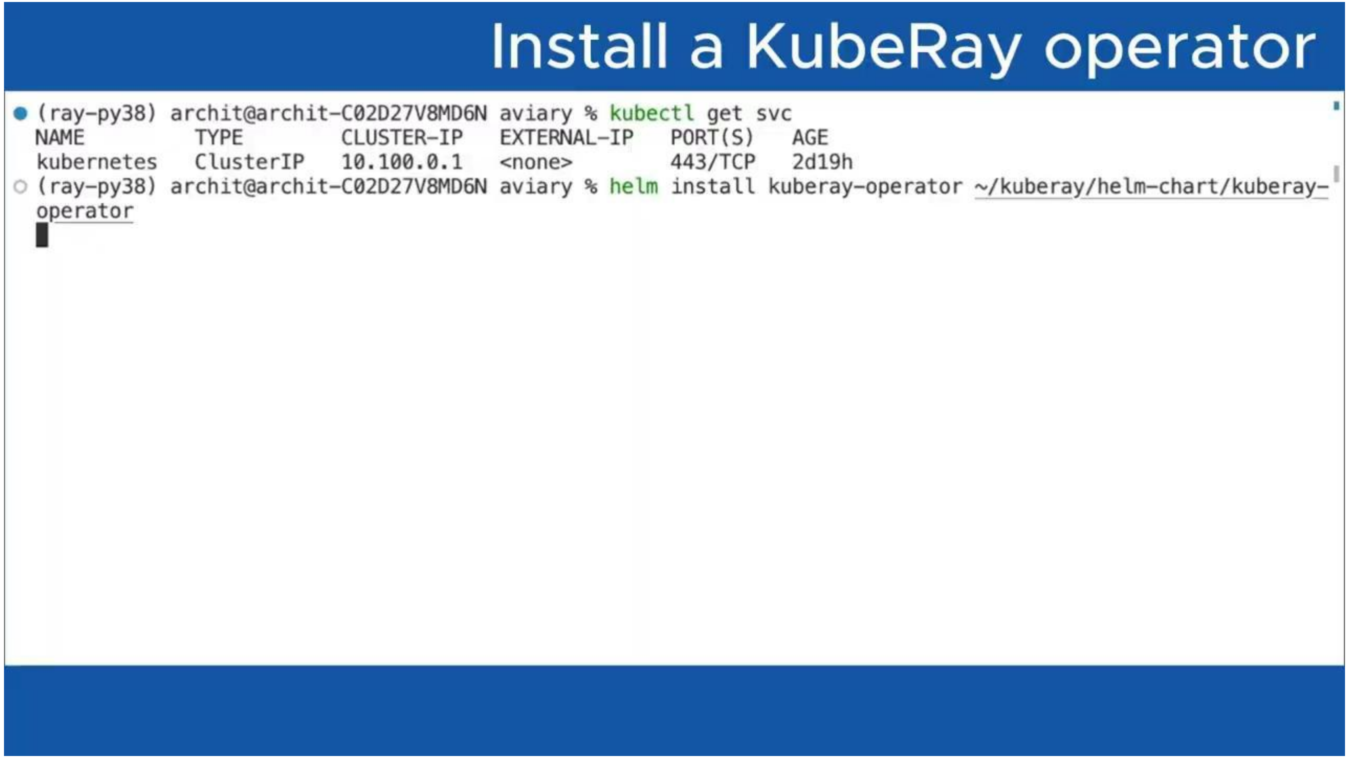


Fine-tuning with RayJob

- Fine-tune a Llama-2 7B on the **GSM8K** (8th grade math) dataset.
- Demo:
  - Use **Ray Train** + DeepSpeed. [1]
  - 16 NVIDIA A10 GPUs (24GB / GPU).
  - 1 epoch takes 22 minutes.

[1] https://github.com/ray-project/ray/tree/master/doc/source/templates/04_finetuning_llms_with_deepspeed

Demo: Fine-tuning with RayJob



Serving with RayService+RayLLM

- Demo:
  - Use **RayService** and **RayLLM** to deploy the fine-tuned LLama-2 7B.
  - RayLLM provides an OpenAI compatible API.

[1] https://github.com/ray-project/ray/tree/master/doc/source/templates/04_finetuning_llms_with_deepspeed

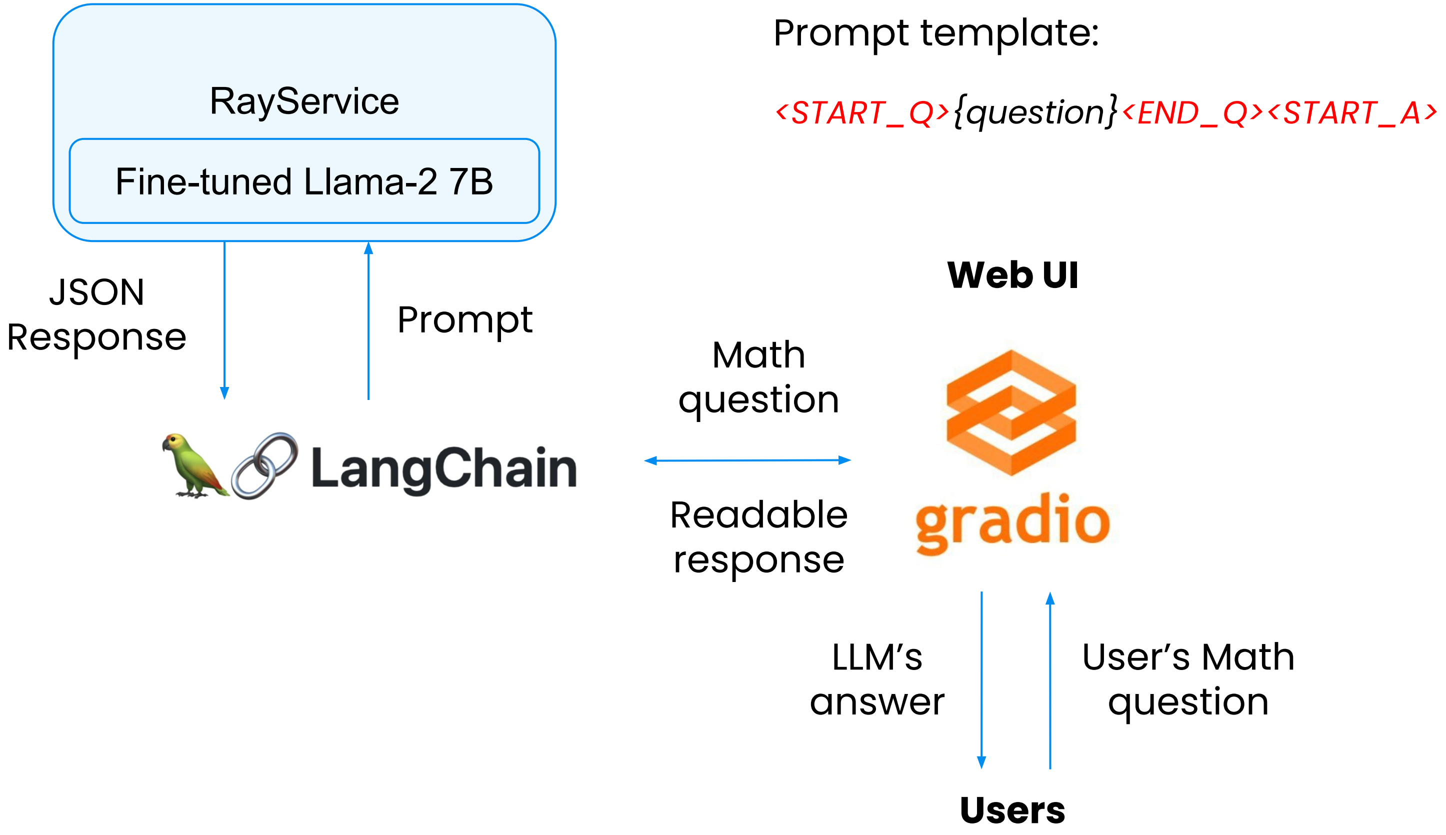
Demo: Serving with RayService+RayLLM

```
(ray-py38) archit@archit-C02D27V8MD6N aviary % kubectl get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1    <none>        443/TCP   20h
(ray-py38) archit@archit-C02D27V8MD6N aviary % helm install kuberay-operator ~/kuberay/helm-chart/kuberay-
operator
NAME: kuberay-operator
LAST DEPLOYED: Thu Sep 14 08:45:07 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
(ray-py38) archit@archit-C02D27V8MD6N aviary % kubectl apply -f rayjob-finetune.yaml
```

Build a LLM application with LangChain and Gradio

Prompt template:

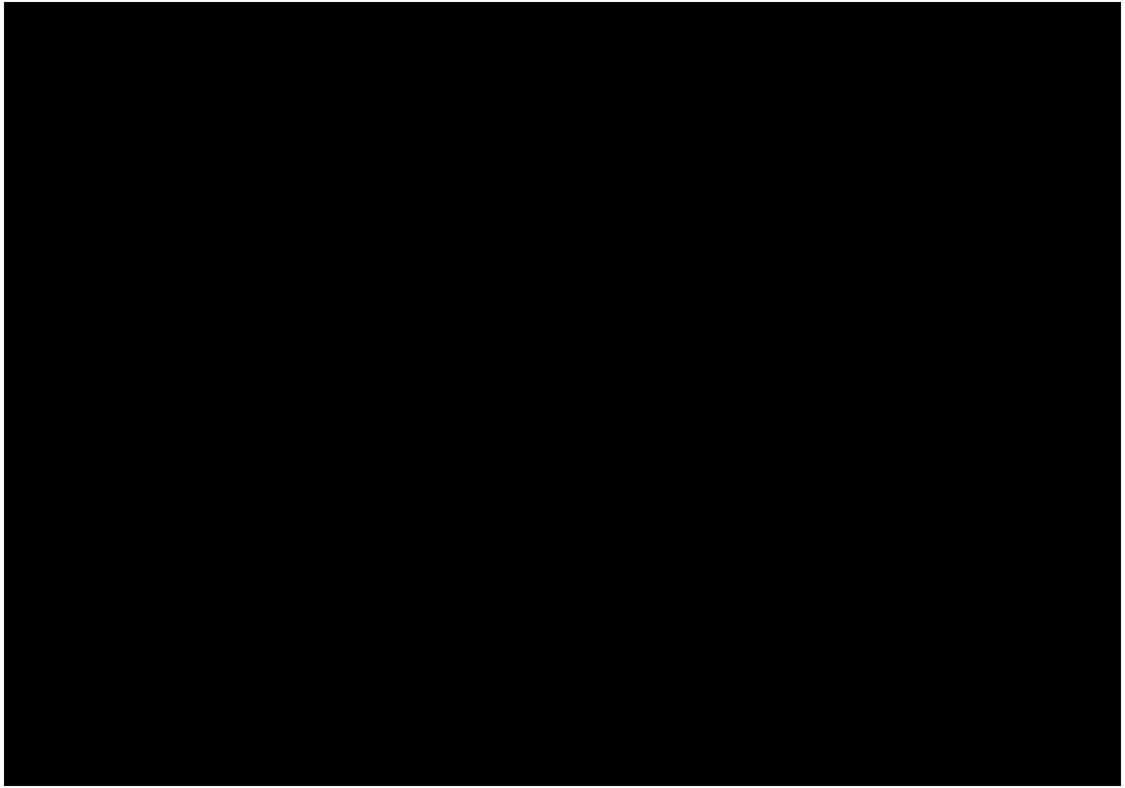
<START_Q>{question}<END_Q><START_A>



A math problem from GSM8K dataset

“Betty is saving money for a new wallet which costs $100. Betty has only half of the money she needs. Her parents decided to give her $15 for that purpose, and her grandparents twice as much as her parents. How much more money does Betty need to buy the wallet?”

Demo: Launch a LLM application



https://github.com/ray-project/llm-applications/blob/main/notebooks/rag.ipynb

Thank you.

Follow us! https://www.ray.io/community

Ray Slack channels:

#kuberay-discuss, #kuberay-questions

Ray on GKE:

Ray on GKE Github
ray-on-gke@google.com