KubeCon

CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

KubeCon | CloudNativeCon

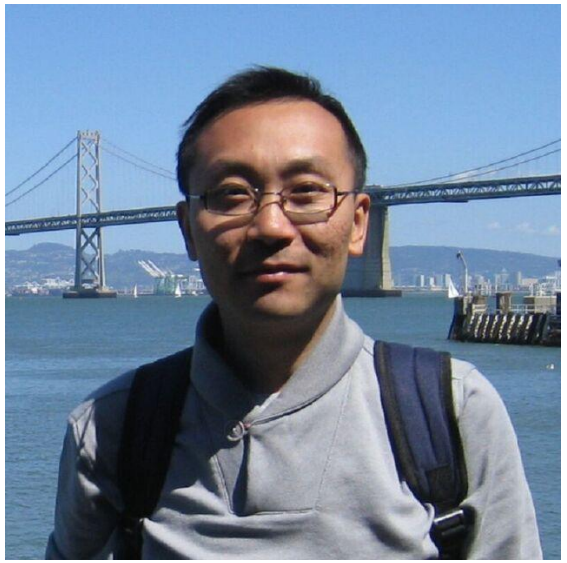
THE LINUX FOUNDATION OPEN SOURCE SUMMIT

AI_dev Open Source GenAI & ML Summit

China 2024

Boundaryless Computing: Optimizing LLM Performance, Cost, and Efficiency in Multi-cloud Architecture

Who we are?



Kai Zhang (wsxiaozhang@gmail.com)
Senior Staff Engineer,
Alibaba Cloud Intelligence

Jian Zhu (jiazhu@redhat.com)
Senior Software Engineer,
RedHat

Agenda

- Challenges and solution of running LLM cross clouds/regions
- Accelerates LLM from the data perspective - Fluid
- Manages multiple clusters in the K8s way - OCM
- Demo - Deploy and scale LLM inference service crossing clouds quickly and easily
- Future works

The Challenges & Solution

Challenges to infrastructure brought by LLM

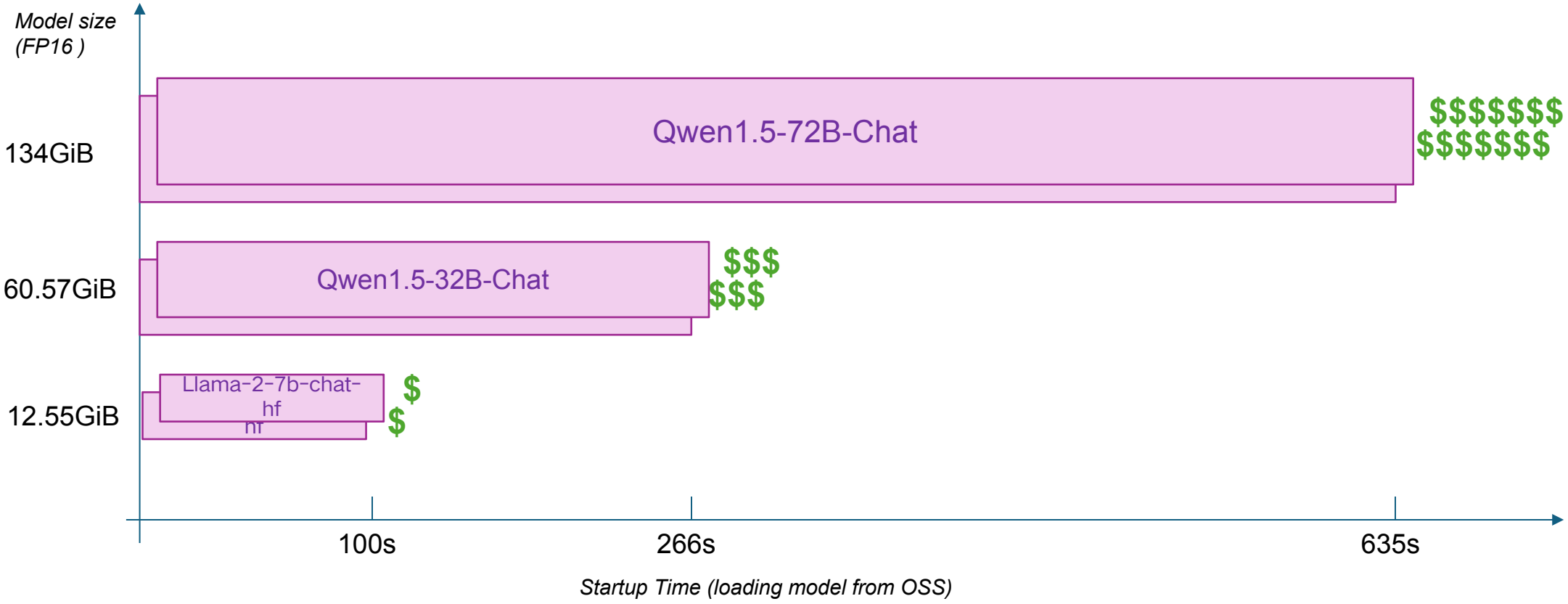
- The emergence of AIGC/LLM has led to a significant increase in GPU resource consumption, especially during the pre-training phase of foundation models.
- Microsoft has hundreds of thousands of GPUs deployed in more than 60 data centers in Azure cloud for serving ChatGPT

Model	Parameters	GPU counts	Training days
Llama	7B	80 * A100	42
GPT3	175B	1K * A100	30
Llama 3.1	405B	16K * H100	54

- GPU resources in a single data center or cloud region cannot meet LLM workloads resource requirements
- Distributing, synchronizing, and managing model consistency and data security across multiple geographies is a challenge of efficiency and complexity

Challenges to infrastructure brought by LLM

- The large model causes the inference service to start very slowly, which seriously affects the elasticity and user experience
- Regional inference services, repeatedly pulling models from remote storage, rapidly driving up bandwidth costs



Optimization of LLM efficiency in multi-clouds and multi-regions

Optimize GPU resources scheduling

1. Schedule GPU resources cross multiple Kubernetes clusters and dynamically adjust AI tasks distribution

Optimize data/model access performance

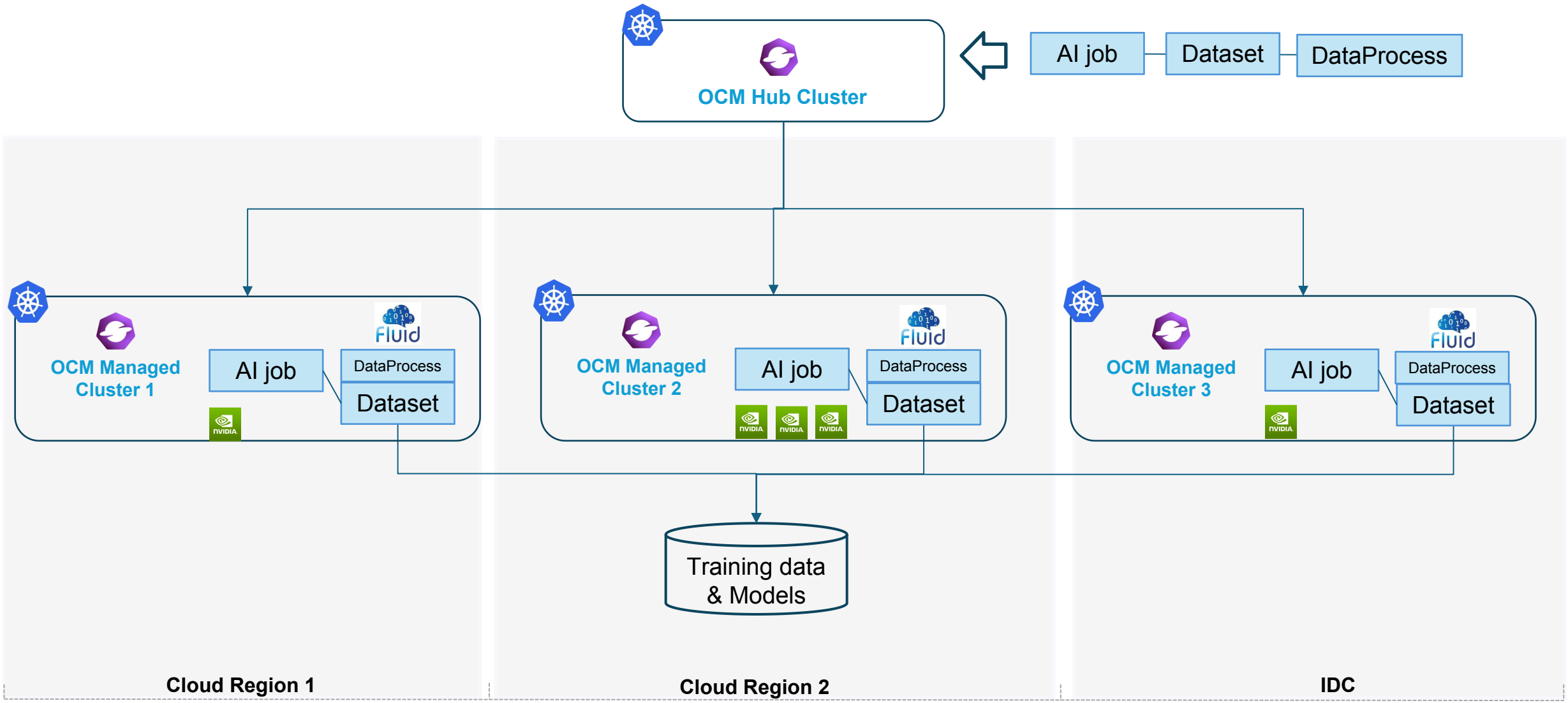
2. Automatic optimization of large model file loading process, accelerate LLM inference service startup and elastic scaling

Optimize the ease of use of multi-geographic model services

3. Cross-geographic models and data management and access acceleration, simplify the user experience

+ Fluid

Solution architecture： OCM + Fluid



Fluid – Accelerates LLM from the data perspective

What is Fluid

Defines the standard API for Kubernetes to access and manage data.

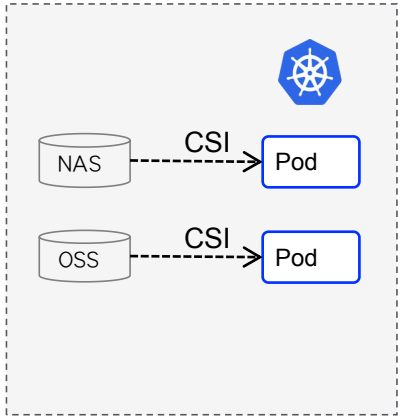
CLOUD NATIVE SANDBOX

Dataset Abstraction **Dataset Acceleration** **Dataset Process**

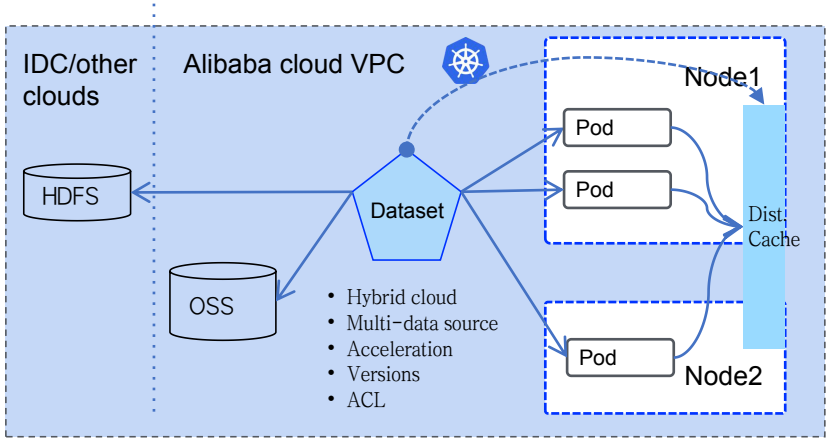
Fluid

https://fluid-cloudnative.github.io/

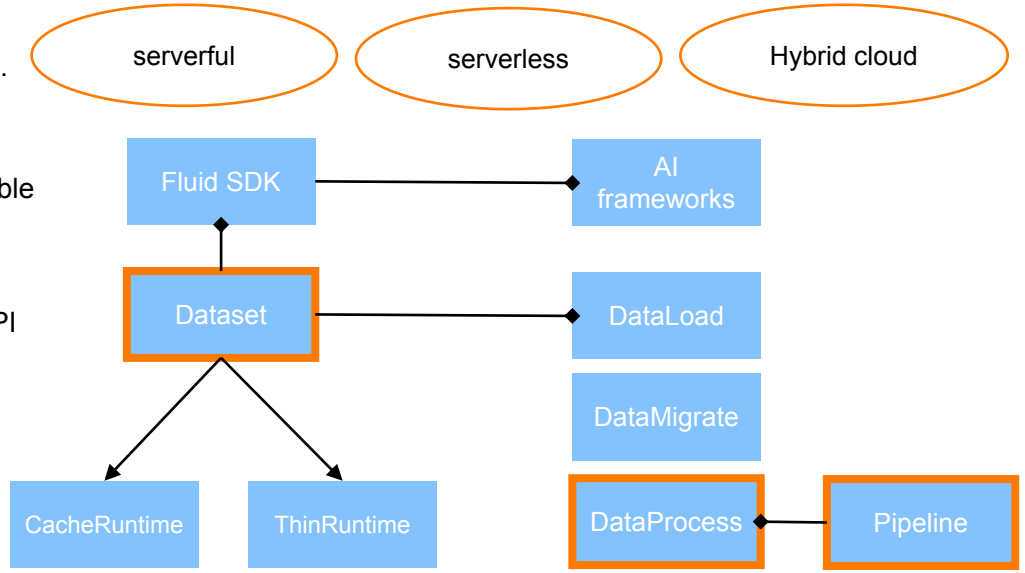
Storage perspective of K8s



Data usage perspective of Fluid



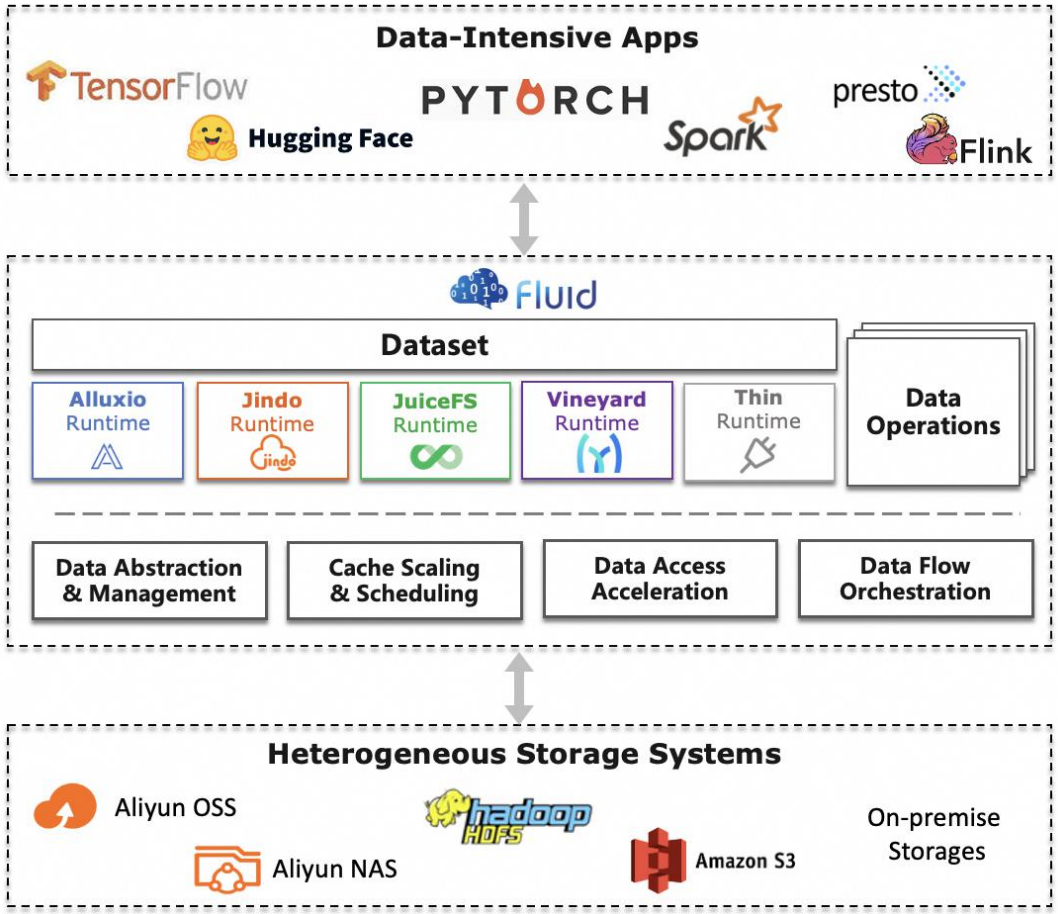
- Unified arch.
- Programmable
- Standard API
- Extensible



- Jindofs
- Juicefs
- Alluxio
- EFC

- CubeFS
- ChuboFS
- Cefs
- Proprietary storages

Fluid: Data and Task Orchestrator in K8s

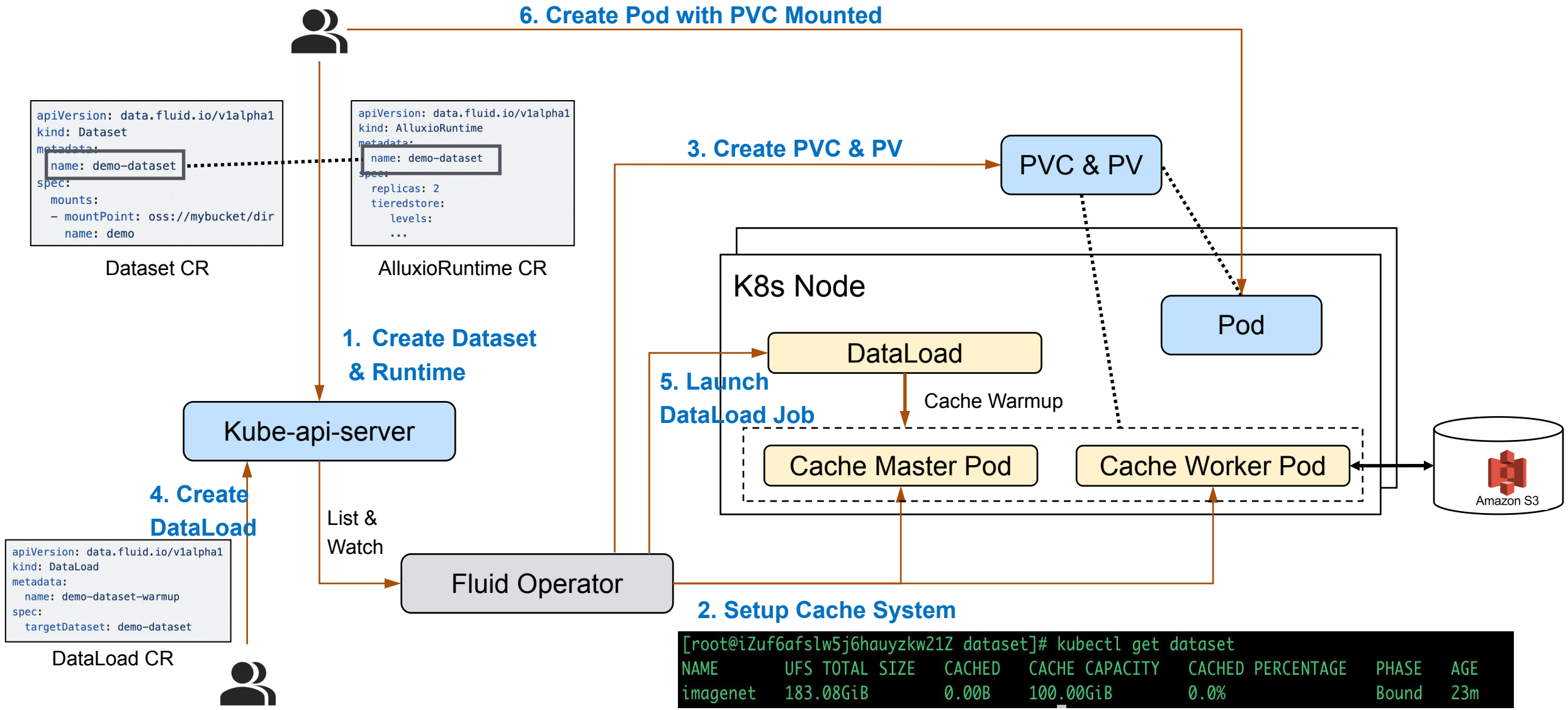


Joint launched by Nanjing University, Alibaba Cloud and Alluxio

https://github.com/fluid-cloudnative/fluid

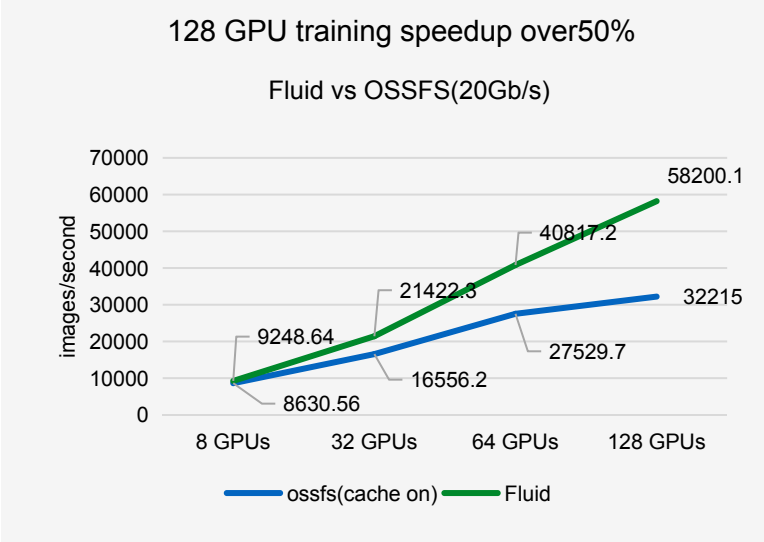
- **Standardized:** K8s Native APIs for data access and distributed cache management.
- **Extensible**: Runtime plugins for different distributed cache and storage backends.
- **Elasticity**: Scale out and in the distributed cache on demand.
- **Performance:** Accelerate data access via elastic distributed cache
- **Automation**: Operation for Data like. *prefetching processing*, *migration* and *cache scaling*
- **Orchestration**: Data and task co-aware scheduling

Out-of-the-Box Distributed Cache

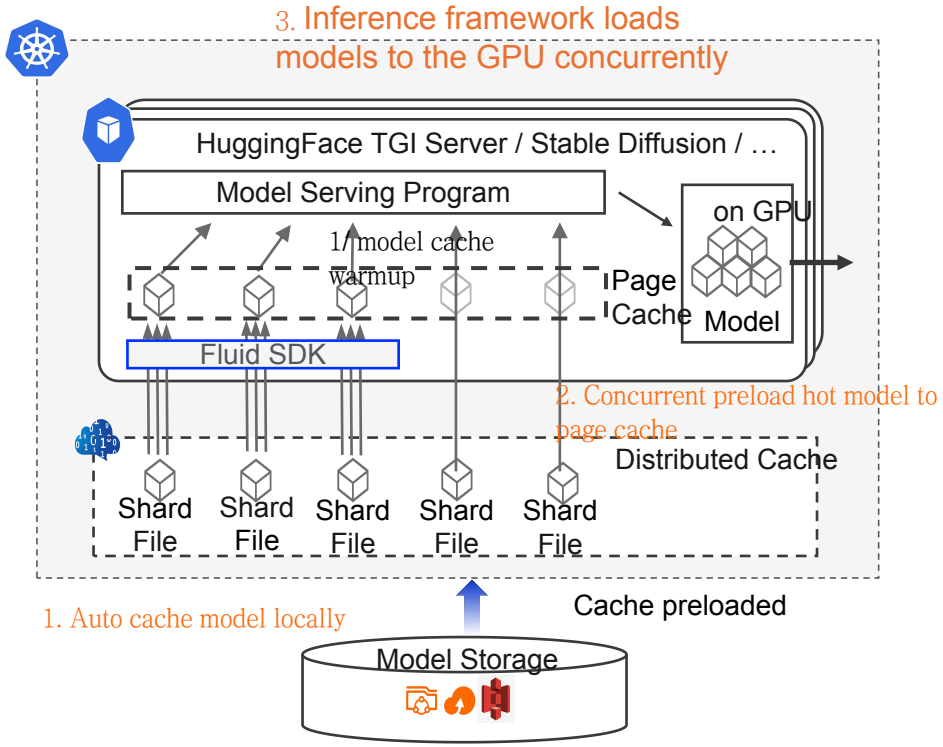


Fluid user's scenarios

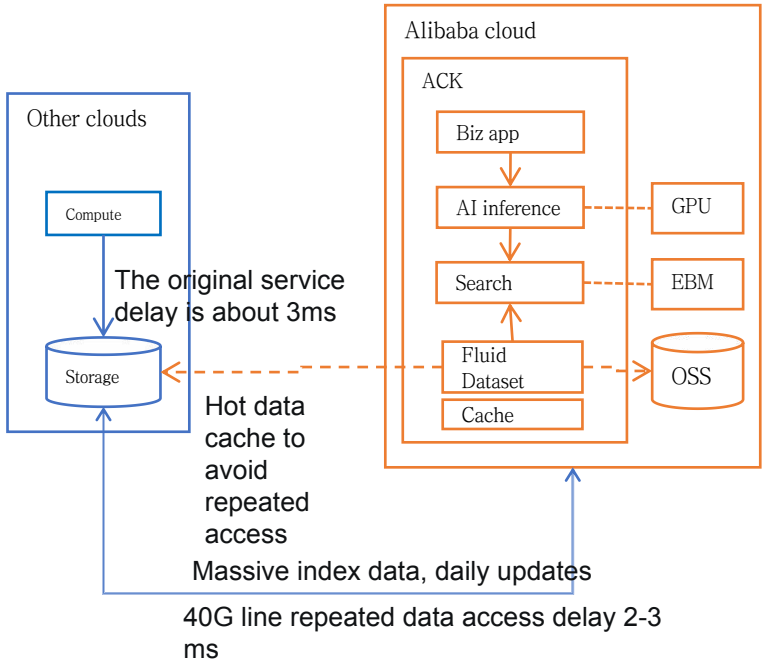
1. Accelerate AI distributed training and optimize scalability



2. Accelerate LLM inference service loading models



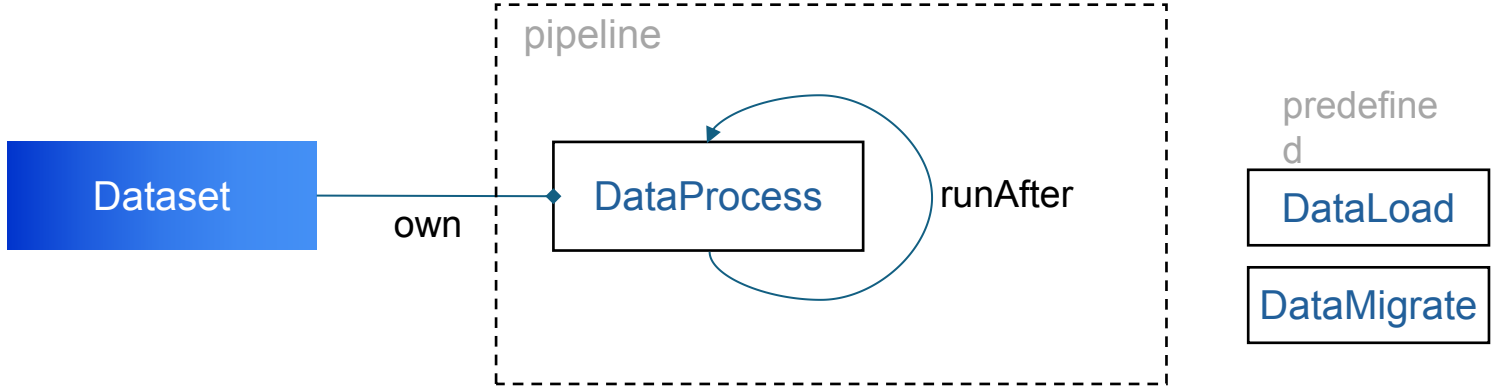
3. Hybrid cloud/multi-cloud data scheduling to accelerate access and optimize bandwidth costs

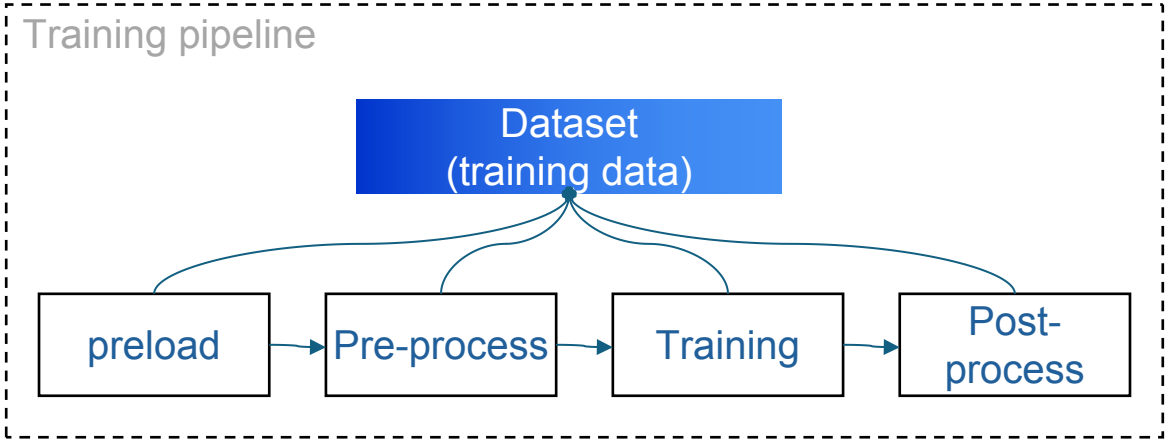


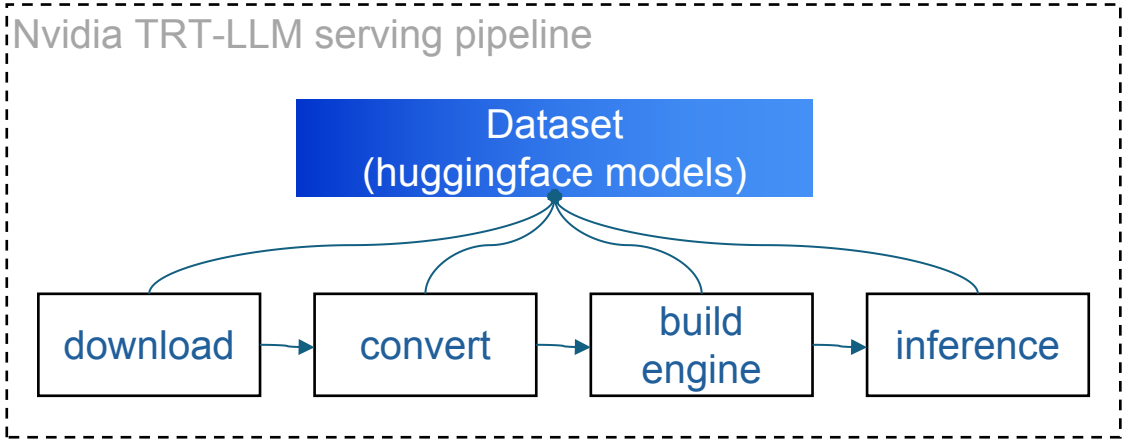
- Accelerate AI training by more than 30% and reduce the cold start delay of large model reasoning by 85%

Fluid DataProcess

Dataset centric processing pipeline







OCM - Manages multiple clusters in the K8s way

What is OCM

https://open-cluster-management.io

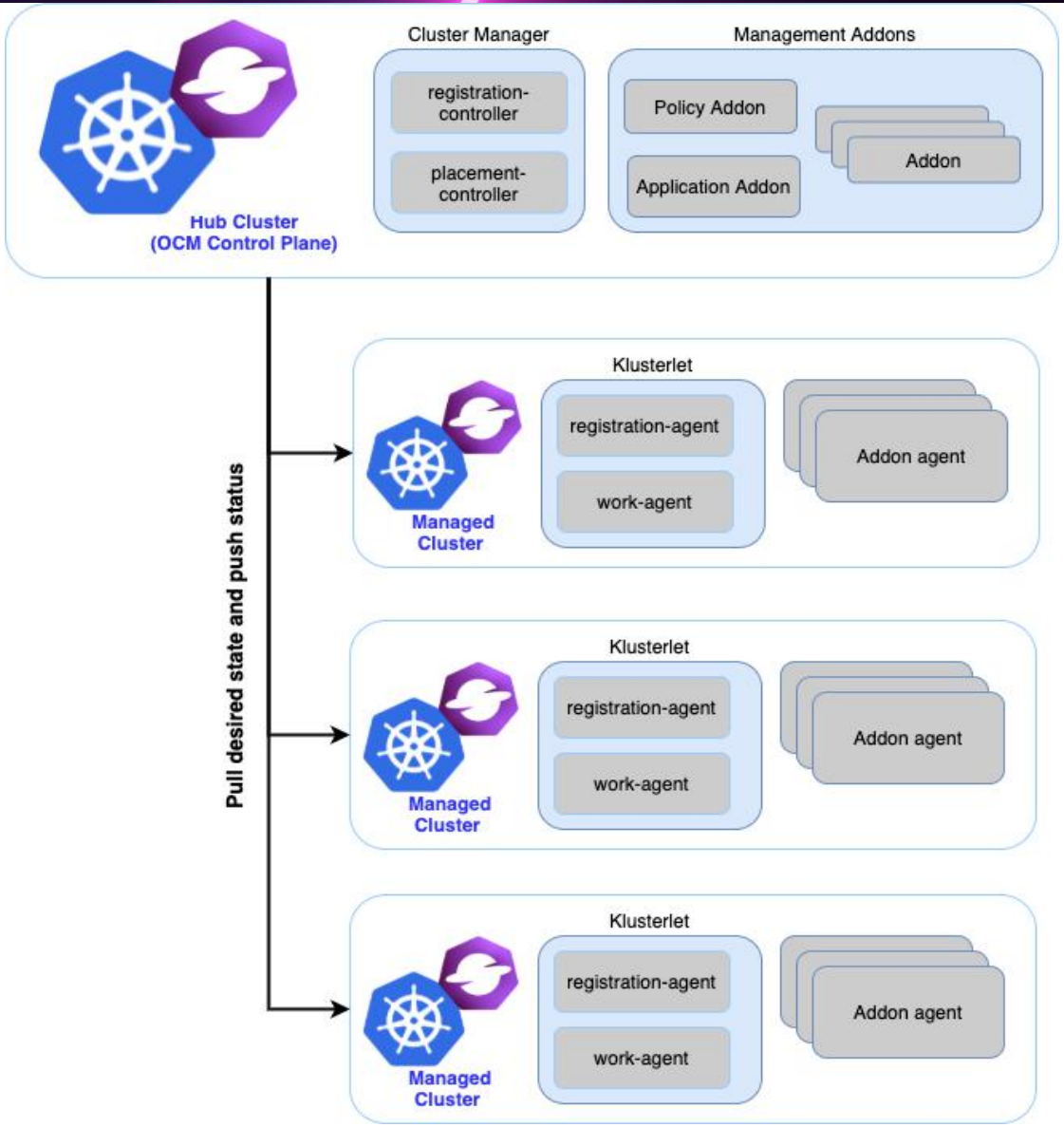
- An open-source CNCF Sandbox project
- Multi-cluster, multi-cloud Kubernetes orchestration, vendor neutral APIs
- Hub, spoke architecture with a centralized view of your entire fleet
- Modular and extensible
- Integration point for making Kubernetes capabilities multi-cluster aware

CLOUD NATIVE COMPUTING FOUNDATION

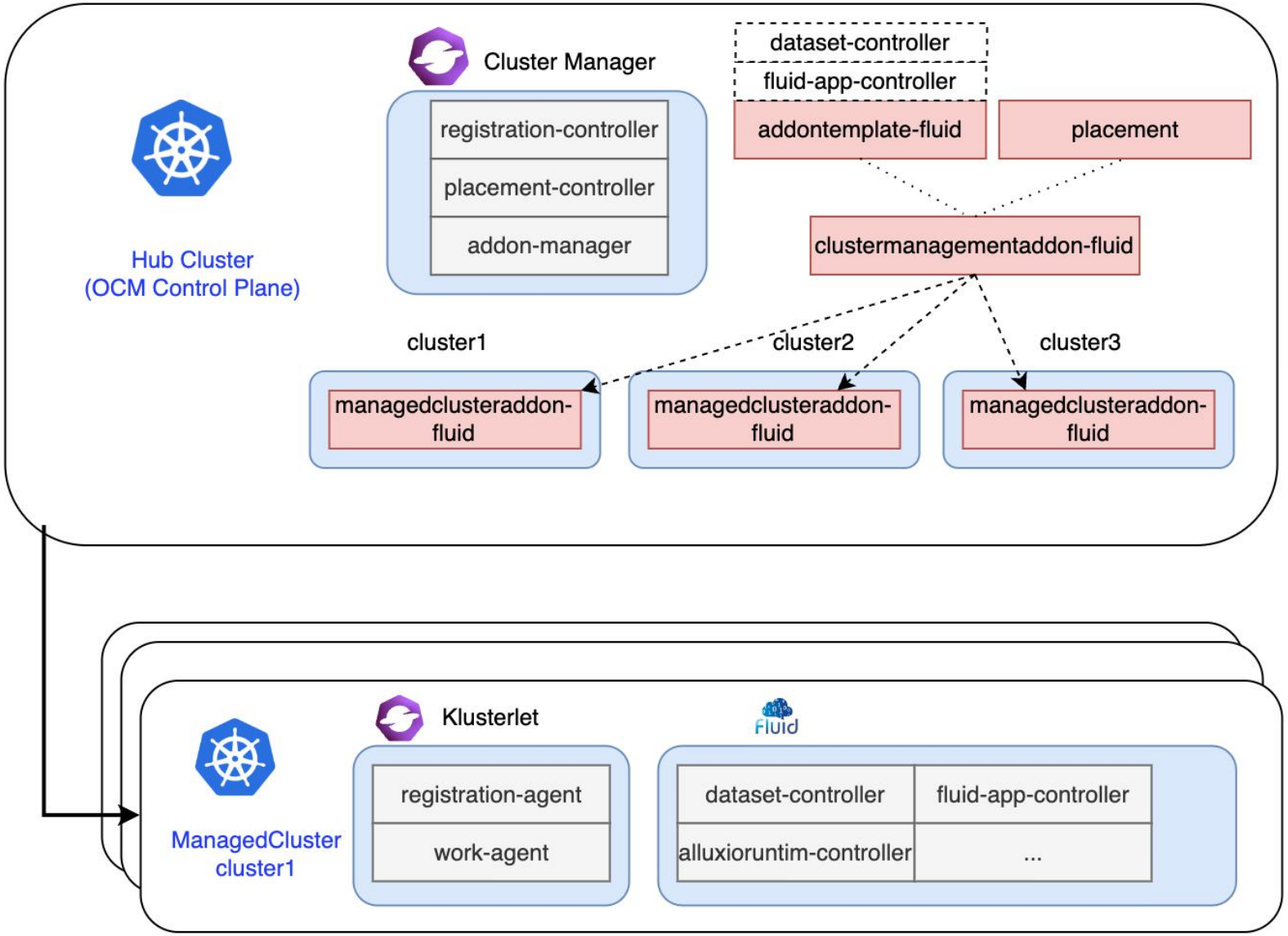
CLOUD NATIVE SANDBOX

OCM Architecture

- Registration
  - Managed cluster lifecycle manager
  - Initial double opt-in handshake
- ManifestWork
  - Deliver resources to one managed cluster
- Placement
  - Provide groupings of managed clusters based on cluster labels or claims
  - Extensible scheduling (CPU, GPU, etc) by the AddOnplacementScore
- ManifestWorkReplicaSet
  - Deliver resources to managed clusters selected by placements
- Add-ons
  - Policy Framework
  - Application lifecycle
  - Cluster Proxy
  - Managed serviceaccount
  - Fluid



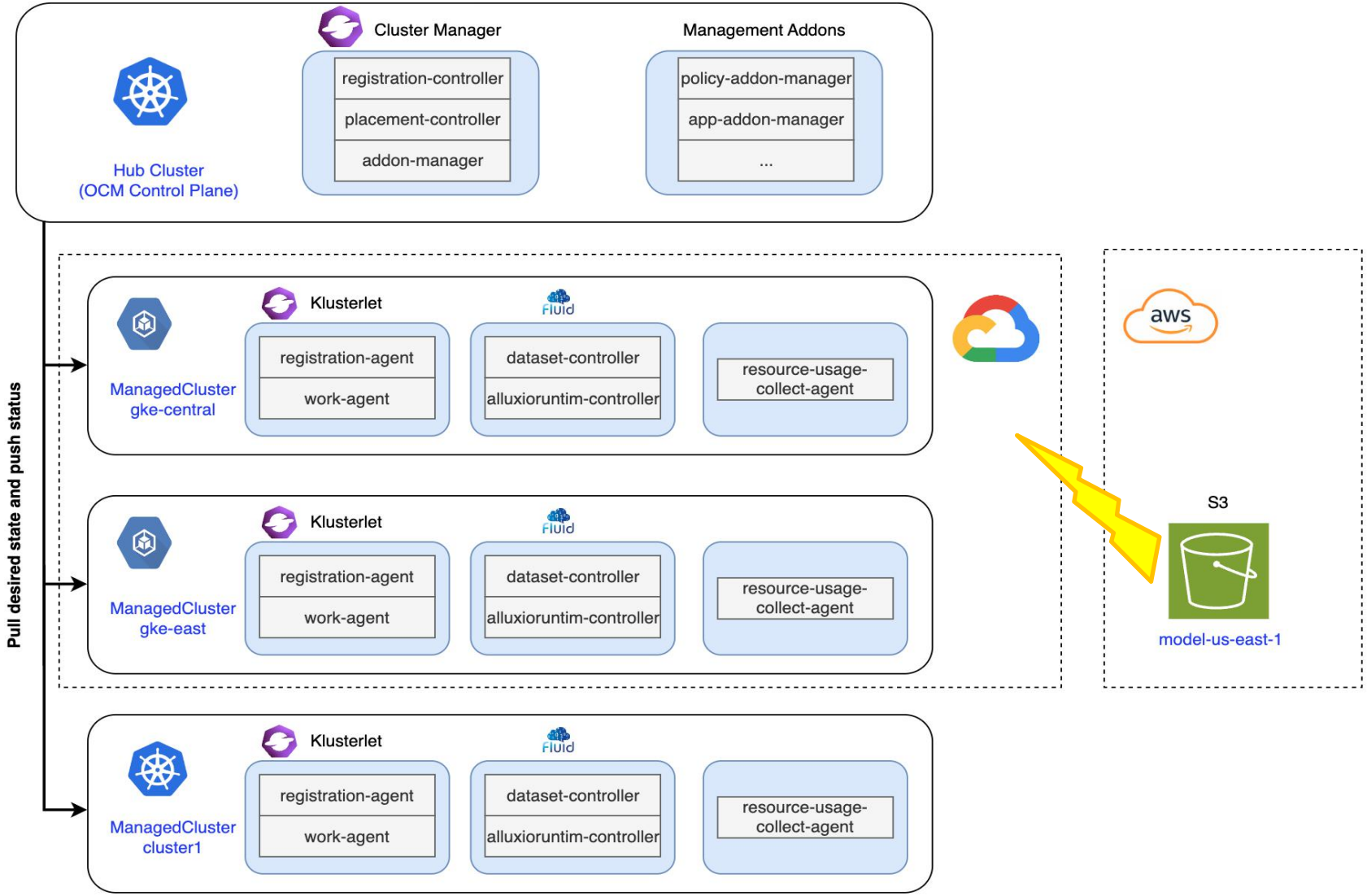
Integrate fluid into OCM



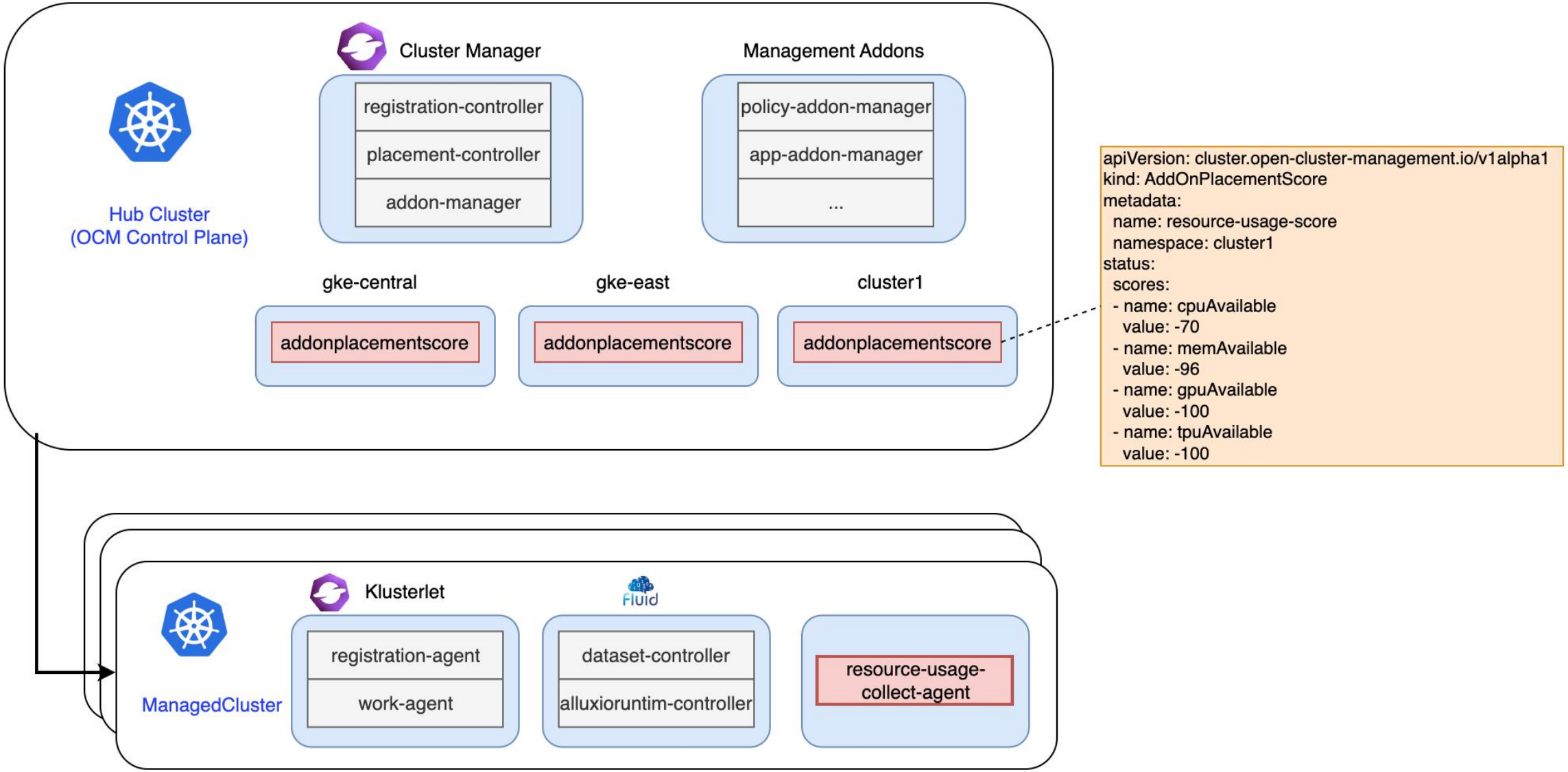
Demo:

Deploy and scale LLM inference service crossing clouds quickly and easily

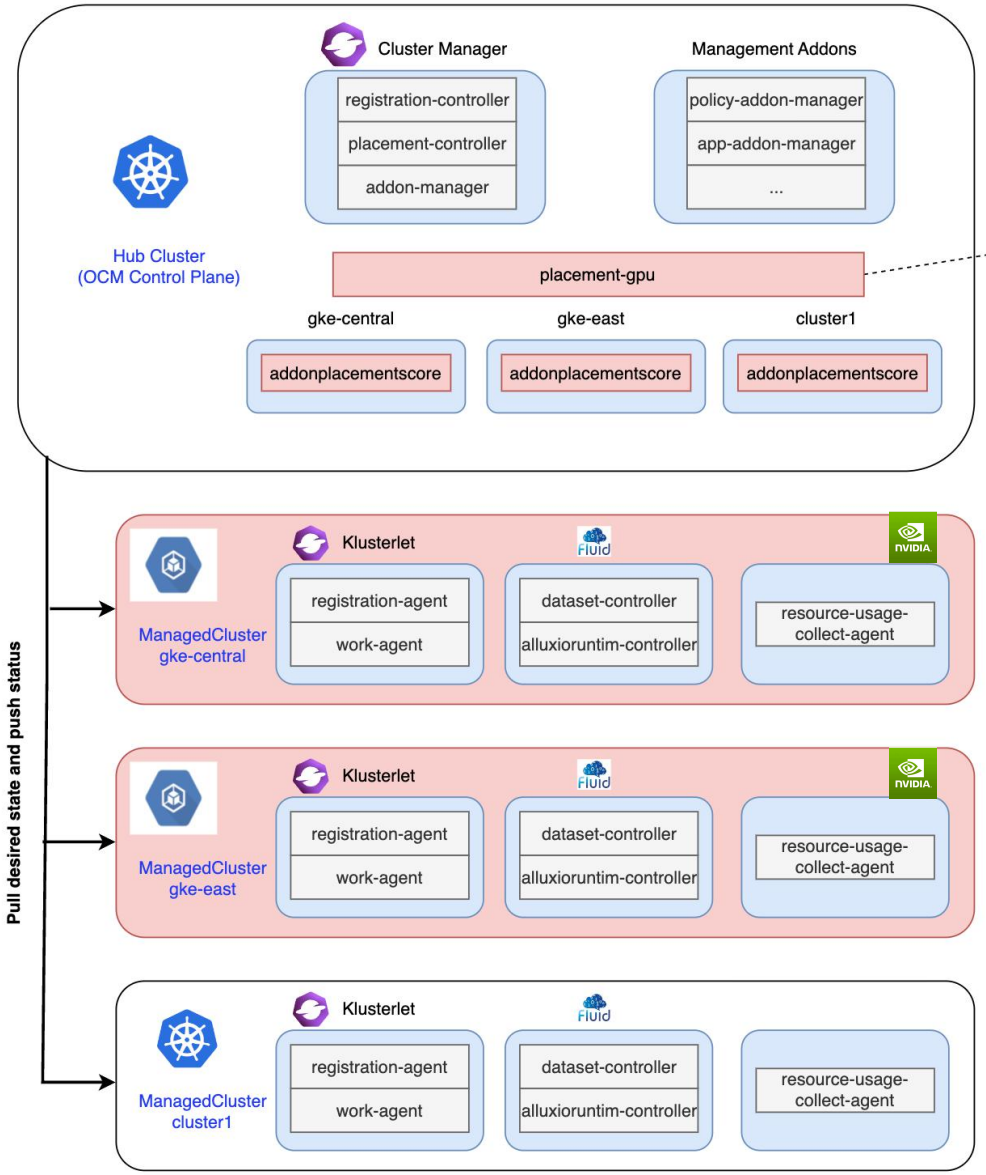
Demo Scenario



Demo Scenario

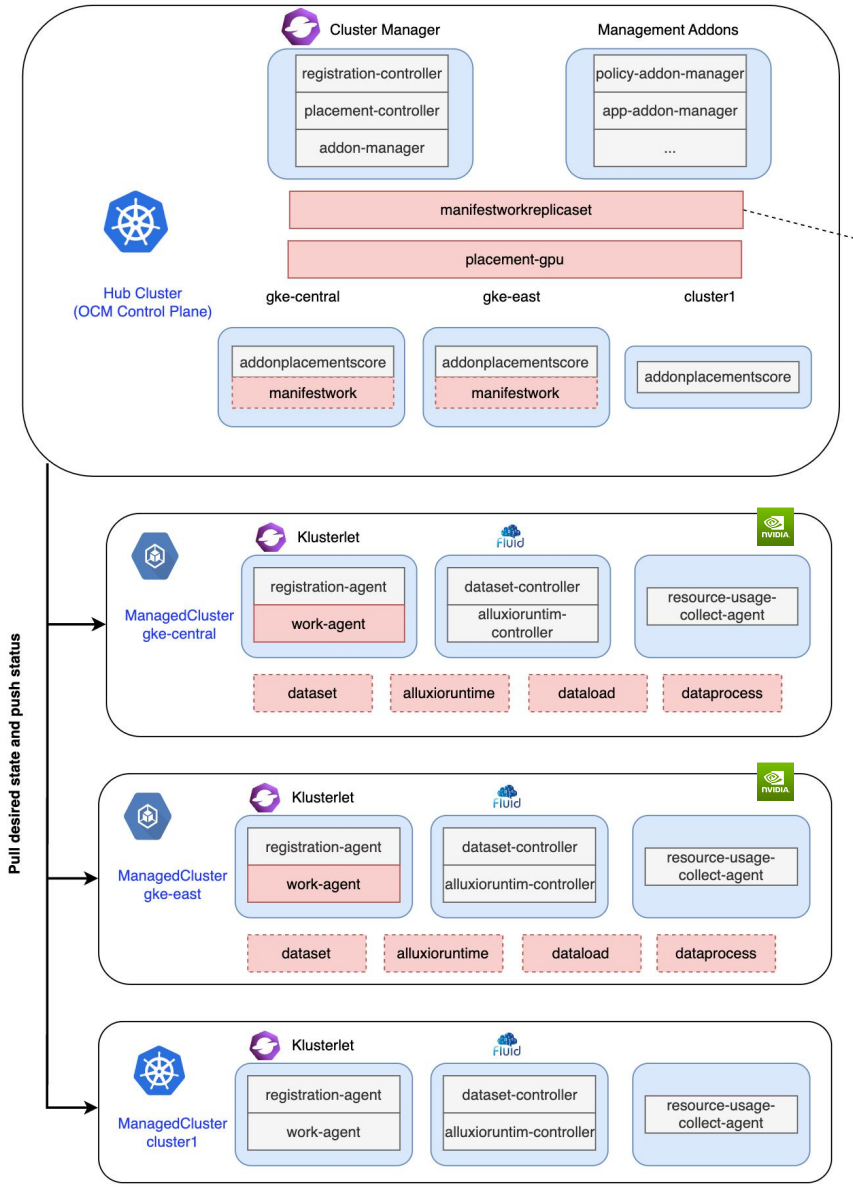


Demo Scenario



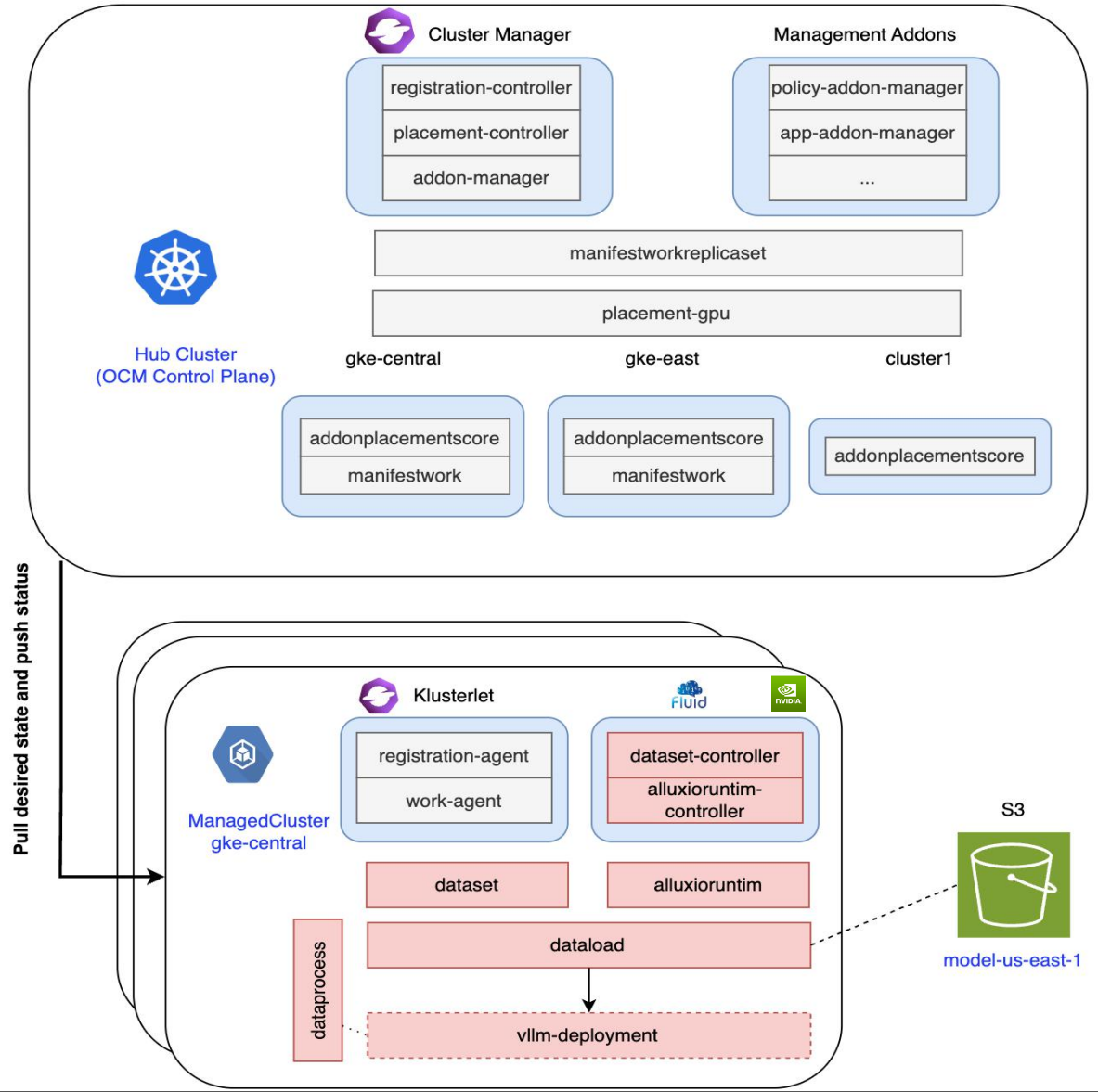

```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement-gpu
  namespace: default
spec:
  numberOfClusters: 2
  prioritizerPolicy:
    configurations:
    - scoreCoordinate:
        addOn:
          resourceName: resource-usage-score
          scoreName: gpuAvailable
        type: AddOn
      weight: 5
    - scoreCoordinate:
        builtIn: Steady
        type: BuiltIn
      weight: 1
    mode: Exact
```

Demo Scenario




```
apiVersion: work.open-cluster-management.io/v1alpha1
kind: ManifestWorkReplicaSet
metadata:
  name: my-dataflow
  namespace: default
spec:
  manifestWorkTemplate:
    workload:
      manifests:
      - apiVersion: data.fluid.io/v1alpha1
        kind: DataLoad
        metadata:
          name: model-dataload
          namespace: default
        spec:
          dataset:
            name: model-data
            namespace: default
          loadMetadata: true
          target:
          - path: /models
            replicas: 1
      - apiVersion: data.fluid.io/v1alpha1
        kind: DataProcess
        metadata:
          name: apply-my-gpu-app
          namespace: default
        spec:
          dataset:
            name: model-data
            namespace: default
            mountPath: ""
          runAfter:
            kind: DataLoad
            name: model-dataload
            namespace: default
          processor:
            script:
              image: bitnami/kubectl
              imageTag: 1.28.12
              command:
              - kubectl apply -f
              source: |
                apiVersion: apps/v1
                kind: Deployment
                metadata:
                  labels:
                    app: qwen
                  name: qwen-vllm
                  namespace: default
                spec:
                  replicas: 1
                  selector:
                    matchLabels:
                      app: qwen
                  strategy:
                    type: Recreate
                  template:
                    metadata:
                      labels:
                        app: qwen
                    spec:
                      containers:
                      - args:
                        - -c
                        - exec python3 -m
vllm.entrypoints.openai.api_server --trust-remote-code
                          --model /model/models/ --gpu-memory-
utilization 0.95 --max-model-len 8192
                          --dtype half --enforce-eager
                      command:
                      - /bin/bash
                      image: quay.io/zhujian/kube-ai-vllm:0.4.1
                      name: qwen
  placementRefs:
  - name: placement-gpu
    rolloutStrategy:
      type: All
```

Demo Scenario



Results Comparison

1. When starting the vLLM application, using fluid to preload model from S3 takes only 50% time of loading with s3fs directly
2. When scaling the application, vLLM loads the fluid cached model locally, getting 5 times faster than the cold starting

	dataload(s)	model loading(s)	server ready(s) (including model loading time)	sum(s)
s3fs	n/a	132.6	147	147
fluid dataprocess	50	2.1	23.8	50+23.8=73.8
auto scale	n/a	2.1	14.5	14.5

Future Works

- GPU cost priority scheduling in multi-region and multi-cluster – Skypilot
- Multi-cluster training task scheduling based on priority queue– Kueue， Kube-queue
- Unified traffic control and load balancing for LLM inference services across clusters – Service mesh， LLM Gateway
- Assign dynamic tuning/rescheduling of inference service instances across clusters

Thank you!

&

Question?