KubeCon

CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

THE LINUX FOUNDATION

KubeCon | CloudNativeCon | OPEN SOURCE SUMMIT | AI_dev

China 2024

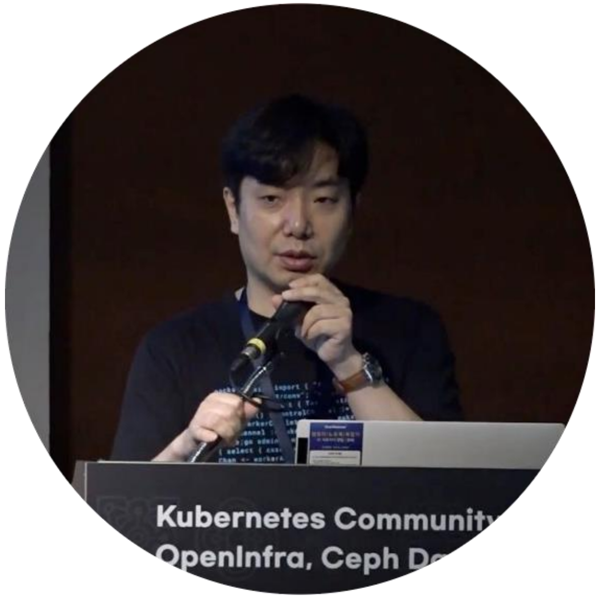
Unlocking the Power of Kubernetes (for AI)

AI-Driven Innovations for Next-Gen Infrastructure

Brandon Kang @ Akamai

Who am I?

- Brandon Kang (姜相鎭)
- Akamai Technologies, APJ CTG
- Principal Technical Solutions Architect



Back to Basics – Why Kubernetes?

- Scalability & Portability
- Self-Healing & Extensibility
- Resource Optimization
- Automated Rollouts and Rollbacks
- Service Discovery and Load Balancing
- Etc.

Why Deploy AI on Kubernetes?

- Dynamic Resource Scaling
- Burst Workloads
- Resource Isolation
- Shared Infrastructure
- GPU/TPU Management
- Efficient Utilization
- Consistency
- Observability

Why Deploy AI on Kubernetes?

- **Resource Management**
  Kubernetes can schedule and allocate resources efficiently, ensuring that AI workloads make optimal use of available hardware, such as GPUs and TPUs, which are often essential for AI tasks

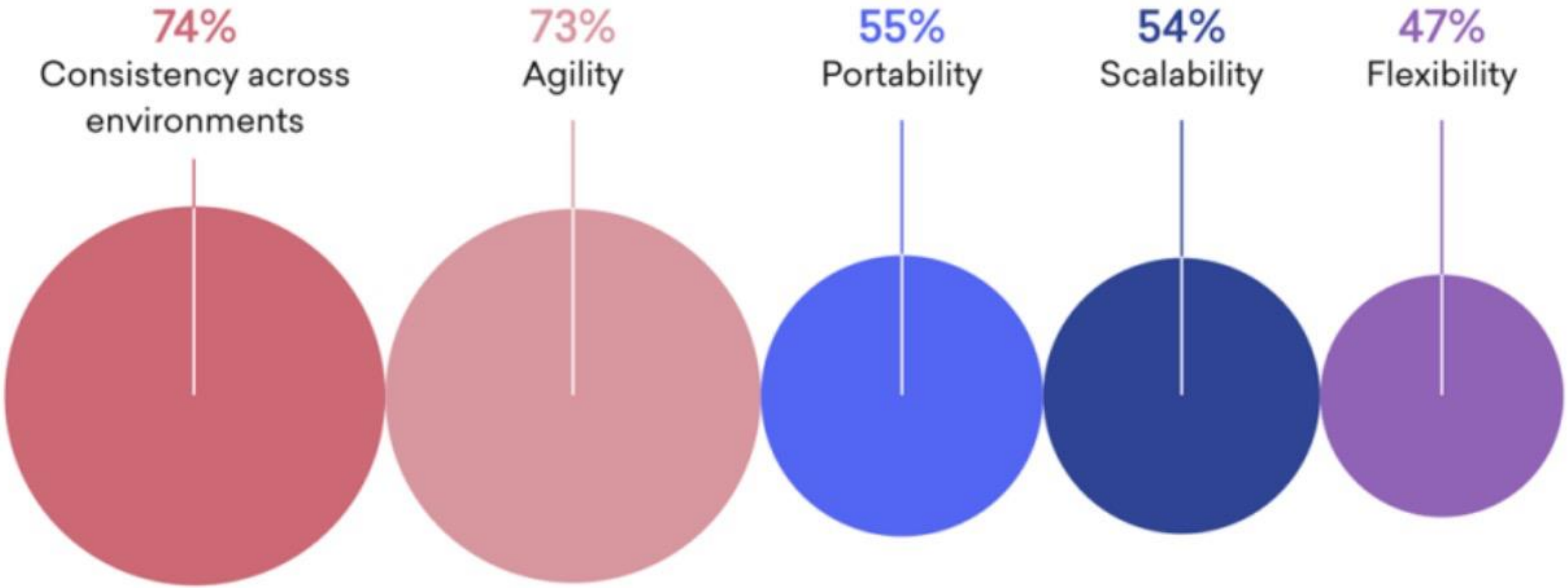
- **Cost Management**
  Efficient resource management helps in controlling costs, especially in cloud environments where resource usage directly impacts expenses.

Why Deploy AI on Kubernetes?

- **Custom Resource Definitions (CRDs)**
  Kubernetes allows for the creation of custom resources to support specialized AI workloads and requirements

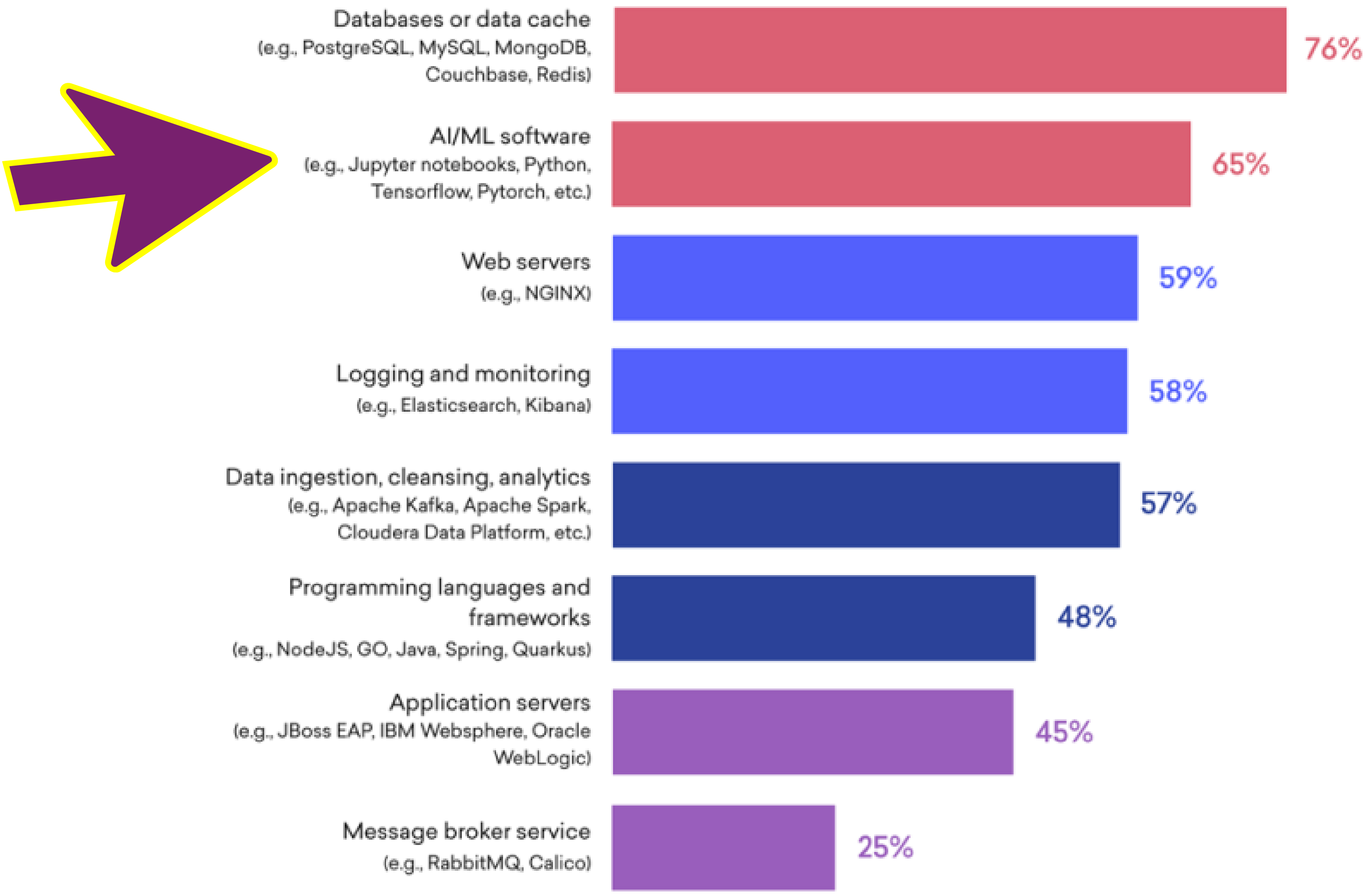
- **Integration with AI Tools**
  Kubernetes integrates well with various AI frameworks and tools, such as TensorFlow, PyTorch, and Nvidia's NeMo, facilitating seamless deployment and operation of AI models

Why Deploy AI on Kubernetes?



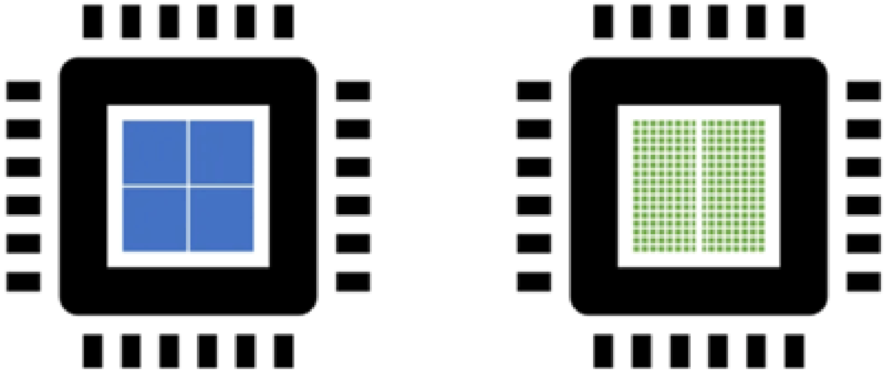
Why organizations deploy workloads on containers (Source: Red Hat)

Why Deploy AI on Kubernetes?



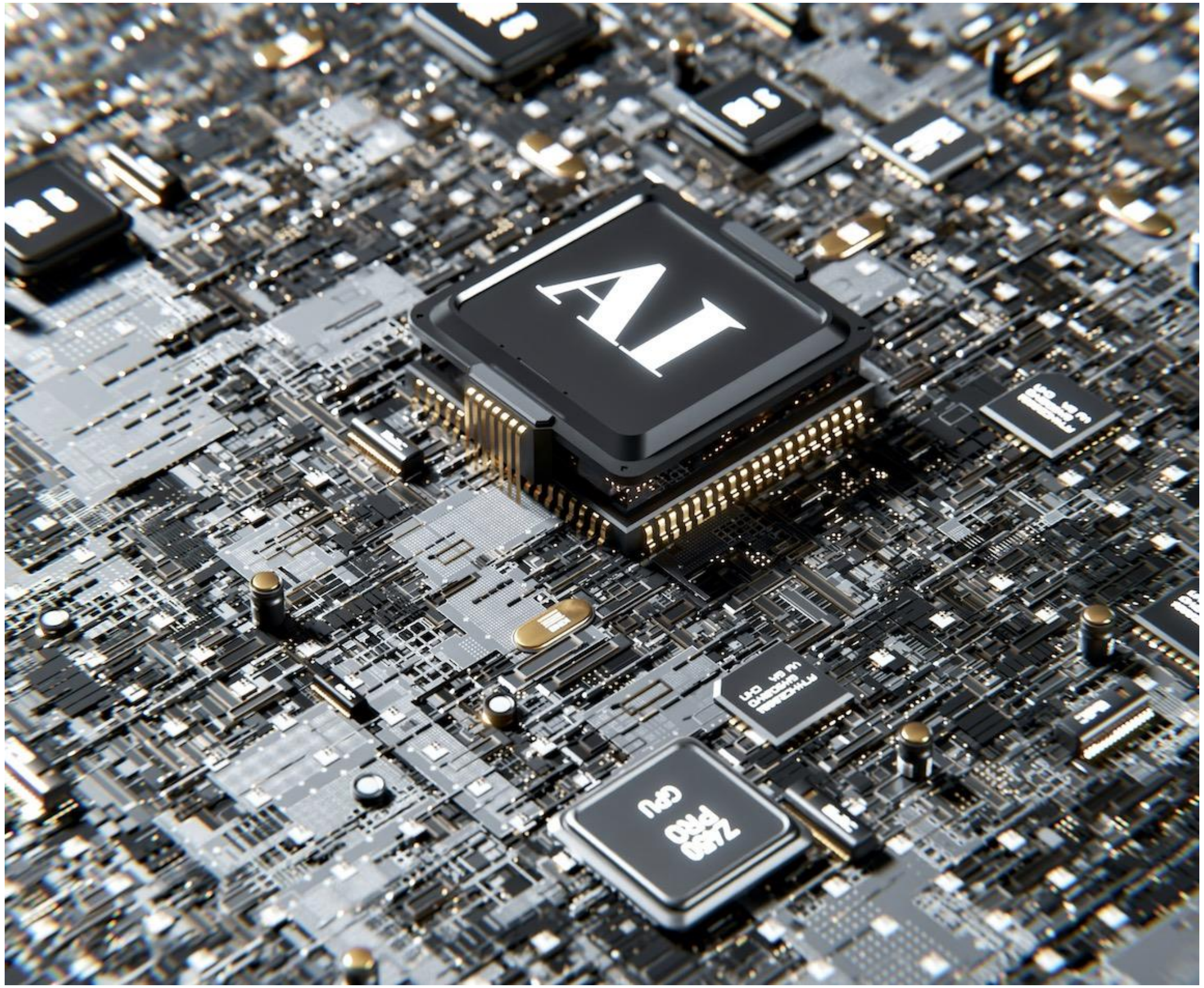
Major types of workloads and systems containerized with Kubernetes (Source: [Red Hat])

Using GPUs for AI Workloads



CPU	GPU
Central Processing Unit	Graphics Processing Unit
4-8 Cores	100s or 1000s of Cores
Low Latency	High Throughput
Good for Serial Processing	Good for Parallel Processing
Quickly Process Tasks That Require Interactivity	Breaks Jobs Into Separate Tasks To Process Simultaneously
Traditional Programming Are Written For CPU Sequential Execution	Requires Additional Software To Convert CPU Functions to GPU Functions for Parallel Execution

<GPU vs CPU processing. Source: towardsdatascience.com>



Using GPUs on Kubernetes Nodes

- **Dynamic Resource Allocation**
  Kubernetes can dynamically allocate GPU resources based on workload requirements, ensuring optimal utilization and minimizing idle resources
- **Cost Efficiency**
  By efficiently managing GPU resources, organizations can control costs, especially in cloud environments where GPU usage can be expensive
- **Multi-cloud Strategy**
  Organizations can leverage GPUs across multiple cloud providers without being locked into a single vendor, enhancing flexibility and reducing dependency risks

Using GPUs on Kubernetes Nodes

- **Device Plugins**
  Kubernetes supports GPU device plugins that can manage the lifecycle of GPU resources, ensuring efficient usage and management

- **GPU Vendor Integration**
  Kubernetes has strong integration with the GPU technologies like NVIDIA Kubernetes device plugin and CUDA, making it easier to leverage GPUs

GPU Device Plugin

- **Automatic GPU Discovery**
  The plugin automatically discovers GPUs on the Kubernetes nodes and makes them available to be scheduled by Kubernetes

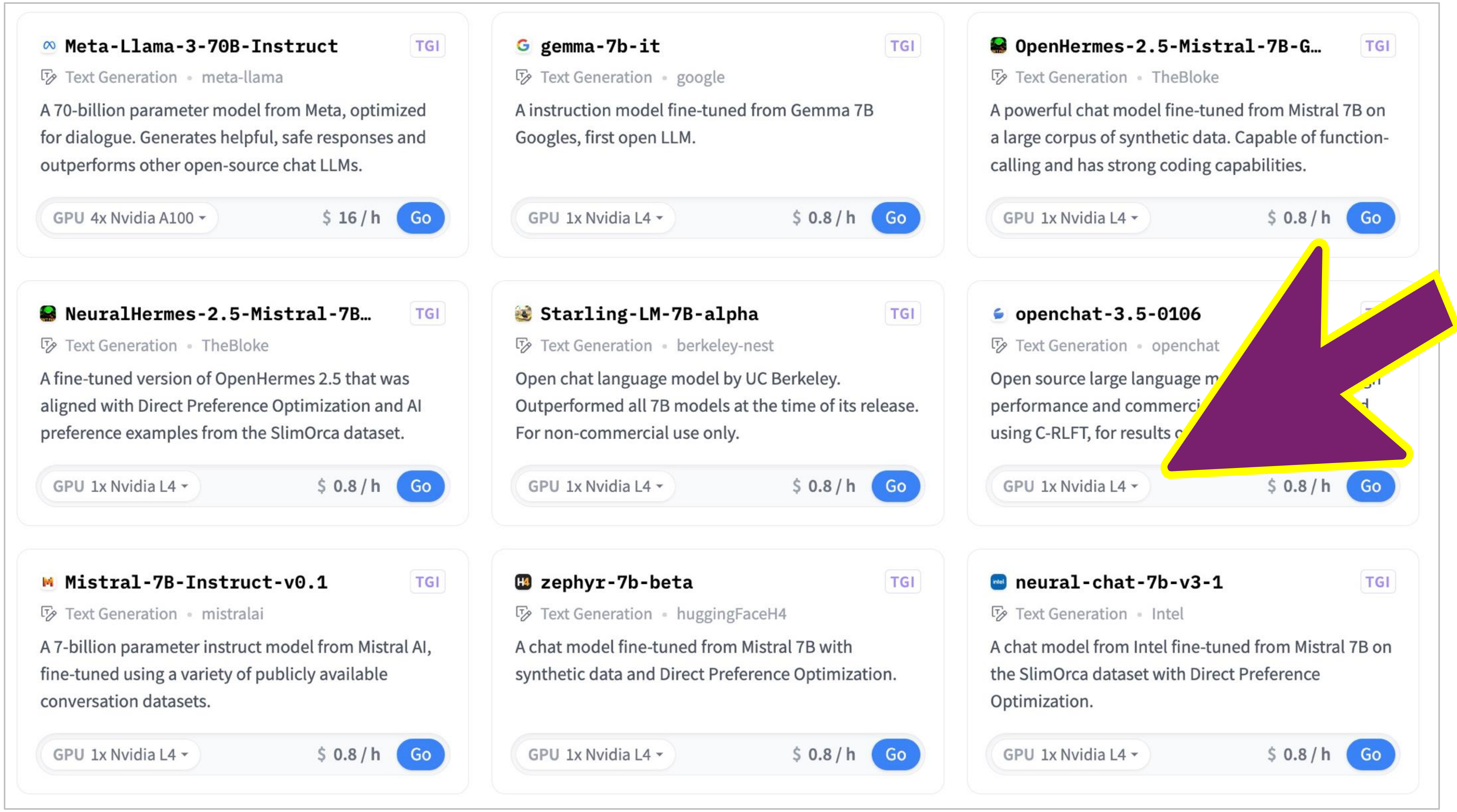
- **Resource Allocation**
  It allows users to request specific GPU resources in their pod specifications, enabling efficient utilization of GPU resources

- **Health Checks**
  The plugin includes mechanisms to monitor the health of the GPUs and ensure that only healthy GPUs are allocated to workloads

When Kubernetes is Not Ideal for AI

- Small-Scale Projects or Limited Resources
- High Initial Setup and Learning Curve
- Real-Time or Low-Latency Applications
- Specialized Hardware Requirements
- Development and Experimentation Phase
- Security and Compliance
- Cost Considerations

GPU Allocation is Important



Inference endpoints - Text Generation Models (Source: huggingface)

GPU Allocation Scenario

- AI Training
  This process is typically more resource-intensive and can take advantage of more powerful or multiple GPUs
  e.g.) FP64, FP32, FP16 supported,
  NVIDIA A100, NVIDIA V100, NVIDIA H100

- AI Inference
  This process is usually less resource-intensive compared to AI training and may only require a single GPU or a fraction of it
  e.g.) FP16, INT8 supported,
  NVIDIA T4, NVIDIA A10, NVIDIA Xavier

GPU Allocation Scenario


```python
# Model training function
def train_model(epochs):
    for epoch in range(epochs):
        train_loss = 0.0
        for data, label in train_data:
            data = data.as_in_context(ctx)
            label = label.as_in_context(ctx)
            with autograd.record():
                output = model(data)
                loss = loss_fn(output, label)
            loss.backward()
            trainer.step(batch_size=128)
            train_loss += loss.mean().asscalar()
        print(f"Epoch {epoch+1}, Loss: {train_loss/len(train_data):.3f}")

# Train the model
train_model(epochs=10)

# Save model parameters
model.save_parameters('cifar_resnet20_v1.params')
```

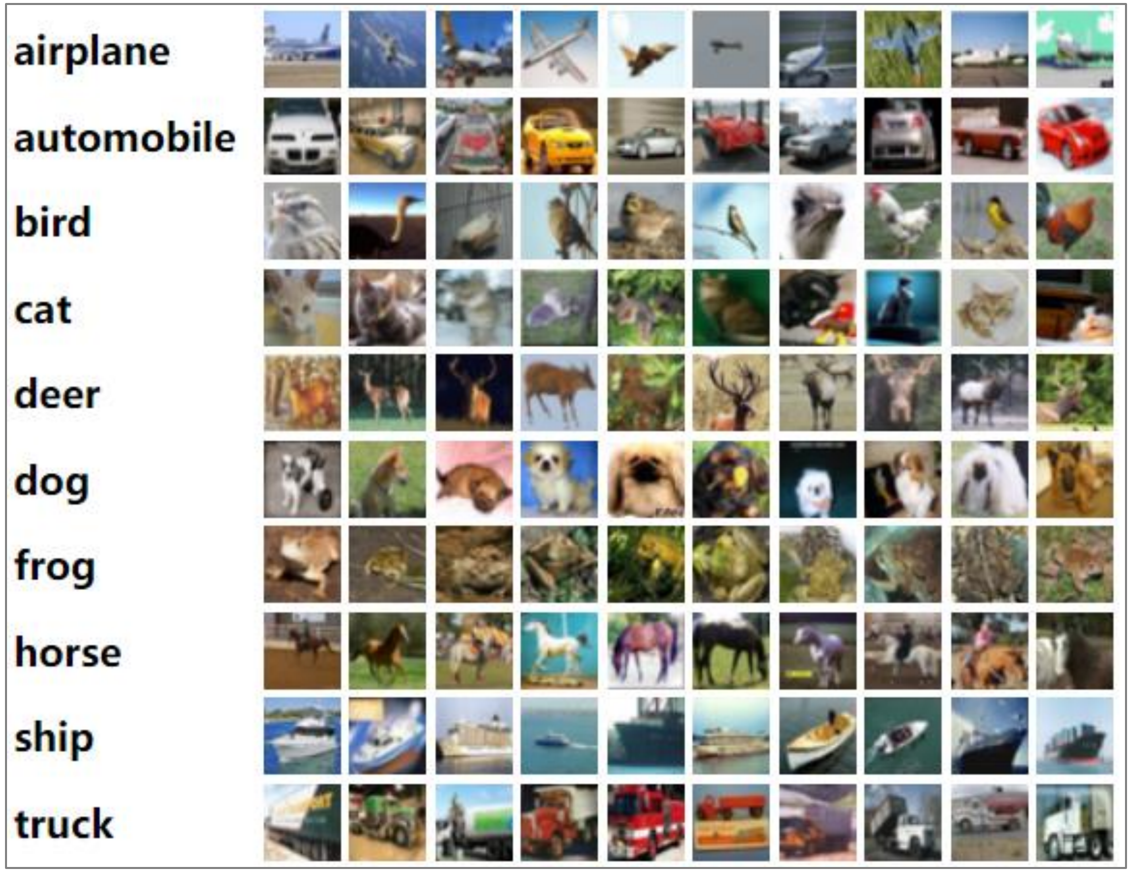
Training

VS.

```python
# Load the pre-trained model
model = get_model('cifar_resnet20_v1', classes=10, pretrained=True)

# CIFAR-10 class names list
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

Inference



The CIFAR-10 dataset

- Labeled subsets of the 80 million tiny images dataset
- Consists of 60,000 32x32 colour images in 10 classes
- 60,000 = 6,000 images per class (x10)
- 50,000 training images and 10,000 test images

GPU Allocation Scenario

ssh

```
root@localhost:~/ai# python3 ai_training.py
Epoch 1, Loss: 1.494
Epoch 2, Loss: 1.025
Epoch 3, Loss: 0.824
Epoch 4, Loss: 0.712
Epoch 5, Loss: 0.644
Epoch 6, Loss: 0.592
Epoch 7, Loss: 0.551
Epoch 8, Loss: 0.527
Epoch 9, Loss: 0.502
Epoch 10, Loss: 0.477
 * Serving Flask app 'ai_training'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.234.95.27:5000
Press CTRL+C to quit
127.0.0.1 - - [13/Aug/2024 08:38:34] "POST /predict HTTP/1.1" 200 -
```

ssh

```
root@localhost:~/ai# curl -X POST http://127.0.0.1:5000/predict -F "img=@animal.jpg"
{"prediction":"horse","probability":0.7999327182769775}
root@localhost:~/ai# curl -X POST http://127.0.0.1:5000/predict -F "img=@animal.jpg"
{"prediction":"horse","probability":0.9025683403015137}
root@localhost:~/ai#
```

Only Inferencing

After training(epoch 10, Loss 0.477)

GPU Allocation Scenario

- **GPU Operator**
  NVIDIA GPU Operator in Kubernetes to manage GPU resources effectively. This can help in dynamic allocation and better resource utilization

- **Automated Node Labeling**
  Automatically label nodes with GPU availability, allowing your scheduler to allocate training or inference workloads to the appropriate nodes

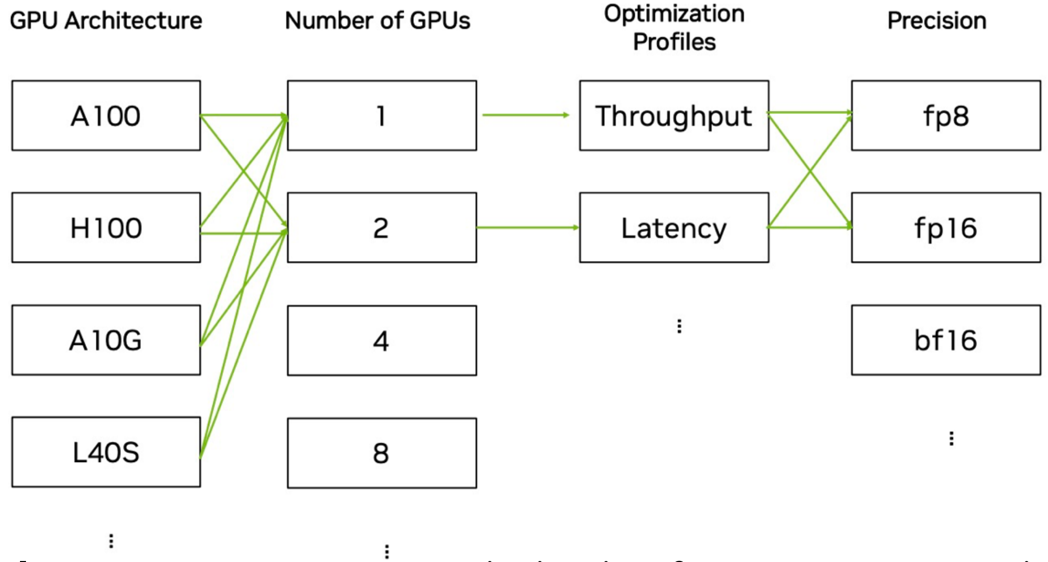
GPU Allocation Scenario

- ## Mixed Workloads on Same GPU
  For NVIDIA A100 GPUs, you can use MIG (Multi-Instance GPU) to partition a single GPU into multiple instances. This allows you to run training and inference workloads simultaneously on different GPU instances

- ## Dynamic Allocation Based on Load
  Implement a system that dynamically allocates GPUs based on the current load. For instance, during peak training periods, more GPUs can be allocated for training, while during high training or inference demand, GPUs can be reallocated accordingly

Example) GPU Allocation

Llama-3-8B-Instruct Optimization for NVIDIA GPUs with LoRA Support Option



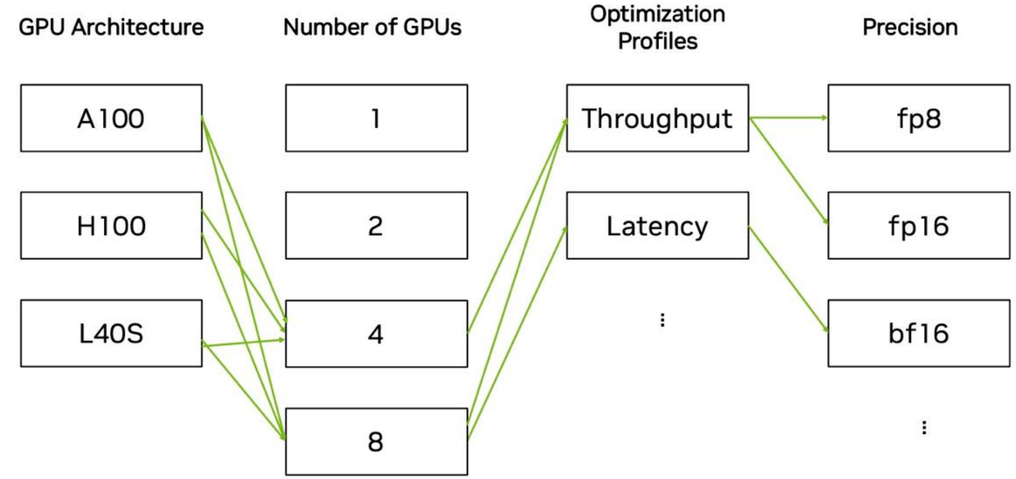
Optimized Models

llama3-8b-instruct:0.10.0+cbc614f5-**h100**x1-fp8-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**h100**x2-fp8-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**h100**x1-fp16-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**h100**x1-fp16-**lora**

llama3-8b-instruct:0.10.0+cbc614f5-**a100**x1-fp16-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**a100**x1-fp16-**lora**
llama3-8b-instruct:0.10.0+cbc614f5-**a100**x2-fp16-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**h100**x2-fp16-**latency**

llama3-8b-instruct:0.10.0+cbc614f5-**a10g**x2-fp16-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**a10g**x1-fp16-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**a10g**x1-fp16-**lora**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x2-fp8-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x2-fp16-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x1-fp8-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x1-fp16-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x1-fp16-**lora**

Llama-3-70B-Instruct Optimization for NVIDIA GPUs with LoRA Support Option



Optimized Models

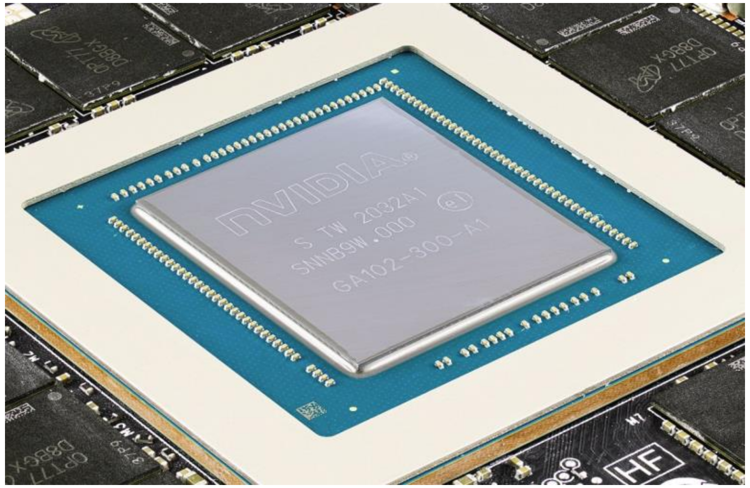
Illama3-70b-instruct:0.10.0+cbc614f5-**a100**x8-bf16-**latency**
llama3-70b-instruct:0.10.0+cbc614f5-**a100**x4-fp16-**throughput**
llama3-70b-instruct:0.10.0+cbc614f5-**a100**x4-fp16-**lora**

llama3-70b-instruct:0.10.0+cbc614f5-**h100**x8-fp8-**latency**
llama3-70b-instruct:0.10.0+cbc614f5-**h100**x8-fp16-**latency**
llama3-70b-instruct:0.10.0+cbc614f5-**h100**x4-fp8-**throughput**
llama3-70b-instruct:0.10.0+cbc614f5-**h100**x4-fp16-**throughput**
llama3-70b-instruct:0.10.0+cbc614f5-**h100**x4-fp16-**lora**

llama3-70b-instruct:0.10.0+cbc614f5-**l40s**x8-fp16-**throughput**
llama3-70b-instruct:0.10.0+cbc614f5-**l40s**x8-fp8-**latency**
llama3-70b-instruct:0.10.0+cbc614f5-**l40s**x4-fp8-**throughput**
llama3-70b-instruct:0.10.0+cbc614f5-**l40s**x8-fp16-**lora**

* Sourced from [NVIDIA NIM page]

VRAM(Video RAM)



<A GPU die surrounded by VRAM chips, from Wikipedia>

- Type of memory used specifically by the GPU
- Acts as the dedicated memory for the GPU
- Optimized for high bandwidth
- Allowing it to quickly supply the GPU with the data it needs

- GDDR (Graphics Double Data Rate)
- HBM (High Bandwidth Memory)

- Large Datasets / Complex Data Augmentation
- Real-time Inference / Multiple Models
- Parallel Processing / GPU Partitioning
- Fragmentation

Example)

Specifications	
GPU Memory	20GB GDDR6
Memory Interface	160 bit
Memory Bandwidth	360GB/s
Error Correcting Code (ECC)	Yes
NVIDIA Ada Lovelace Architecture-Based CUDA Cores	6,144
NVIDIA Fourth-Generation Tensor Cores	192
NVIDIA Third-Generation RT Cores	48
Single-Precision Performance	26.7 TFLOPS[3]
RT Core Performance	61.8 TFLOPS[3]
Tensor Performance	327.6 TFLOPS[4]
System Interface	PCIe 4.0 x16
Power Consumption	Total board power: 130W
Thermal Solution	Active
Form Factor	4.4" H x 9.5" L, single slot
Display Connectors	4x DisplayPort 1.4a[5]

VRAM

Schedule Resources with GPU

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  restartPolicy: Never
  containers:
    - name: cuda-container
      image: nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda10.2
      resources:
        limits:
          nvidia.com/gpu: 2 # requesting 2 GPUs
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

- A container using the NVIDIA CUDA sample image
- Requests 2 GPUs for its container
- tolerations for nodes with GPUs

Schedule Resources with GPU


```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: tensorflow-gpu-job
spec:
  template:
    spec:
      containers:
      - name: tensorflow-gpu-container
        image: tensorflow/tensorflow:latest-gpu
        command: ["python", "-c"]
        args: ["import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))"]
        resources:
          limits:
            nvidia.com/gpu: 1 # Request 1 GPU
      restartPolicy: OnFailure
```

Schedule Resources with GPU

- You can specify GPU limits without specifying requests, because Kubernetes will use the limit as the request value by default
- You can specify GPU in both limits and requests, but these two values must be equal
- You cannot specify GPU requests without specifying limits

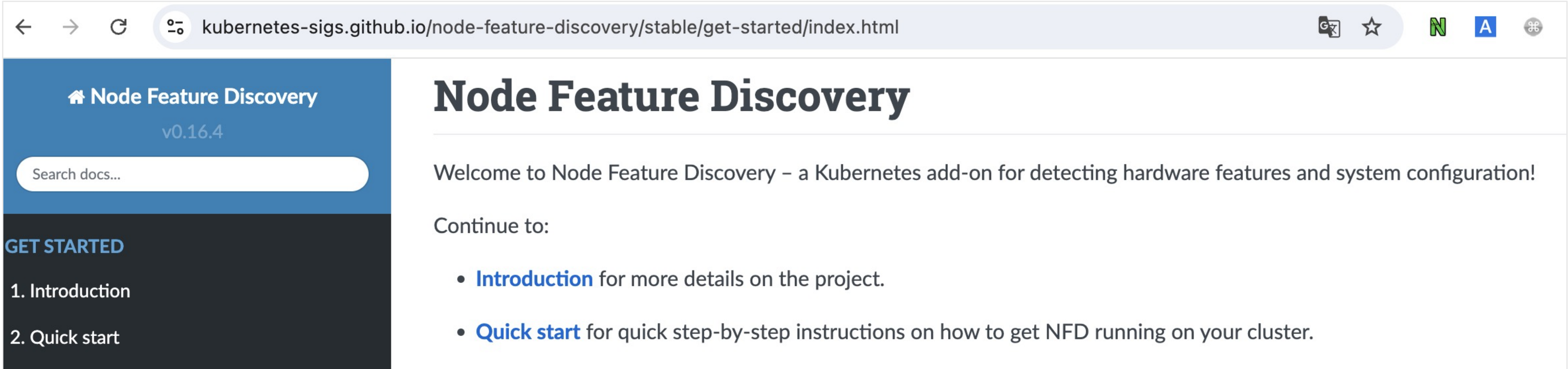
Various types of GPUs

```
# Label your nodes with the accelerator type they have
$ kubectl label nodes node1 accelerator=example-gpu-x100
$ kubectl label nodes node2 accelerator=other-gpu-k915
```

"**If** each node in your cluster have different types of GPUs, then you can use Node Labels and Node Selectors to schedule pods to appropriate nodes."

Automated Node Labeling

- Automatic discover and label GPU enabled nodes
- Deploying Kubernetes Node Feature Discovery (NFD)
- NFD detects the hardware features on each node
- NFD is configured to advertise those features as node labels
- NFD can add extended resources, annotations, and taints
- NFD creates the feature labels for the detected features
- Administrators can leverage NFD to also taint nodes



Automated Node Labeling


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
  - name: cuda-container
    image: nvidia/cuda:11.0-base
    resources:
      limits:
        nvidia.com/[illegible]: 1
  nodeSelector:
    feature.node.kubernetes.io/pci-10de.present: "true"
```

- Pre-defined NFD
- nodeSelector

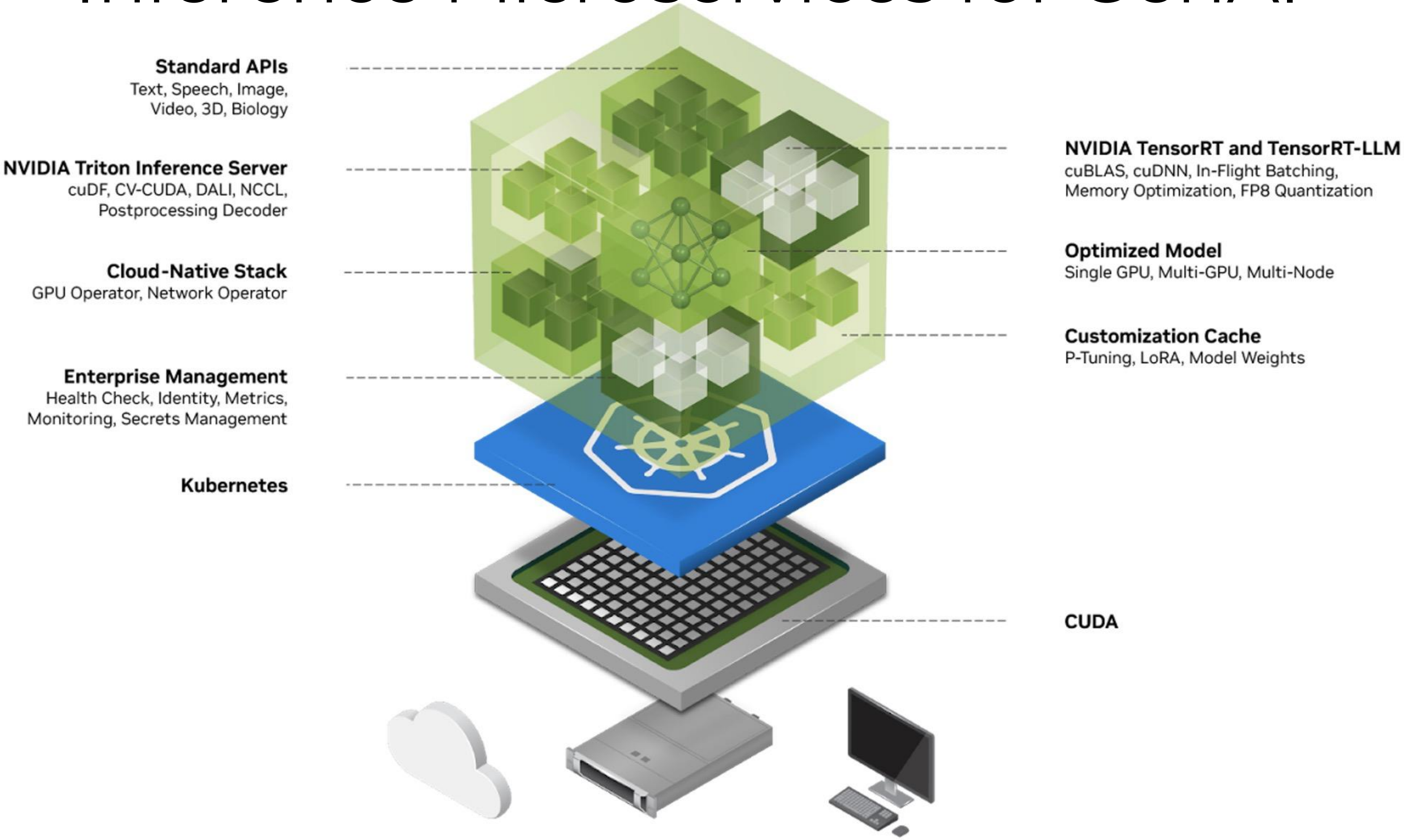
Automated Node Labeling


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-vector-add
spec:
  restartPolicy: OnFailure
affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "gpu.gpu-vendor.example/installed-memory"
            operator: Gt # (greater than)
            values: ["40535"]
          - key: "feature.node.kubernetes.io/pci-10.present" # NFD Feature label
            values: ["true"] # (optional) only schedule on nodes with PCI device 10
  containers:
    - name: example-vector-add
      image: "registry.example/example-vector-add:v42"
      resources:
        limits:
          gpu-vendor.example/example-gpu: 1 # requesting 1 GPU
```

- Pre-defined NFD
- NodeAffinity to schedule this Pod onto a node

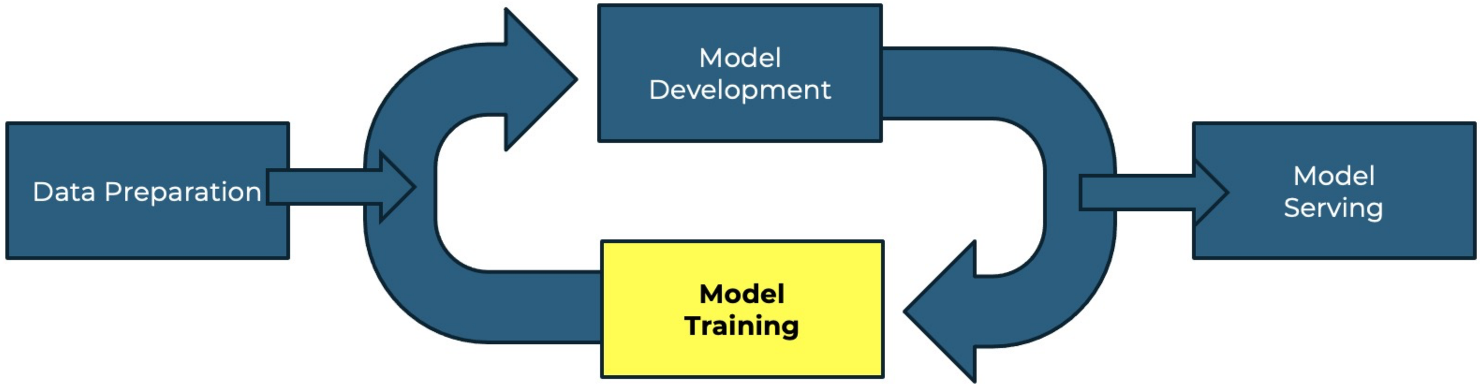
Industry Use Cases

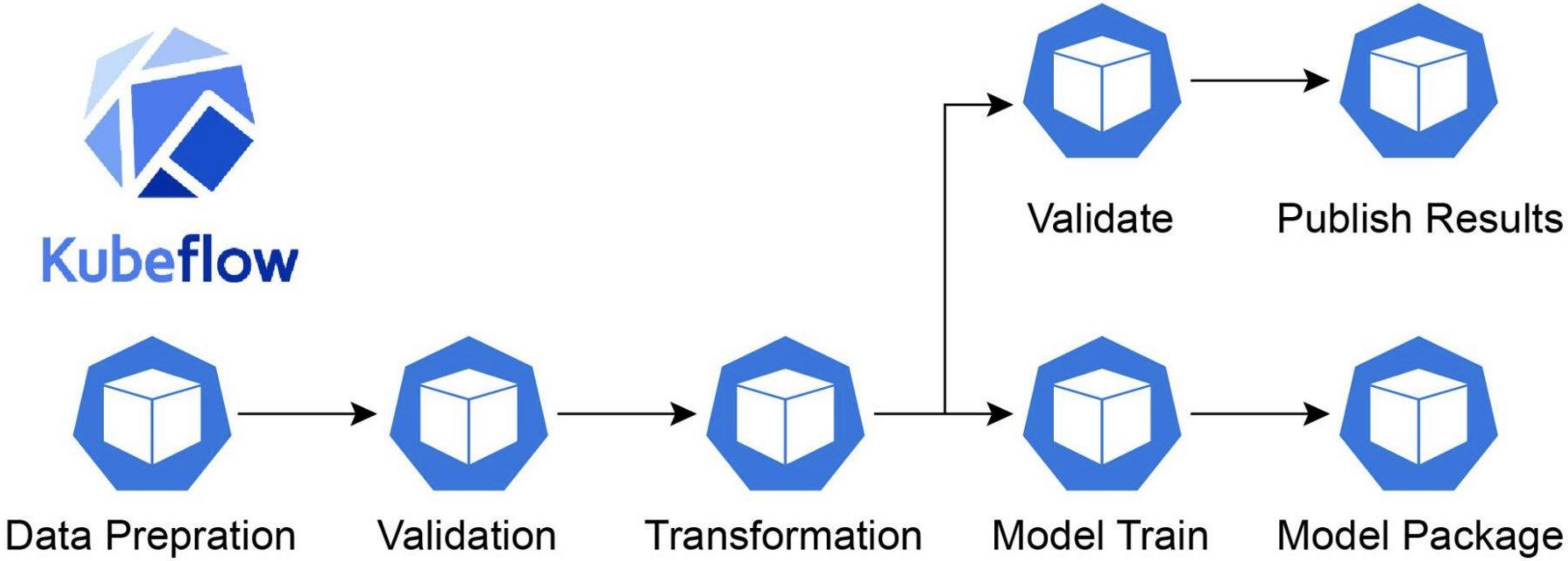
Inference Microservices for GenAI



Industry Use Cases

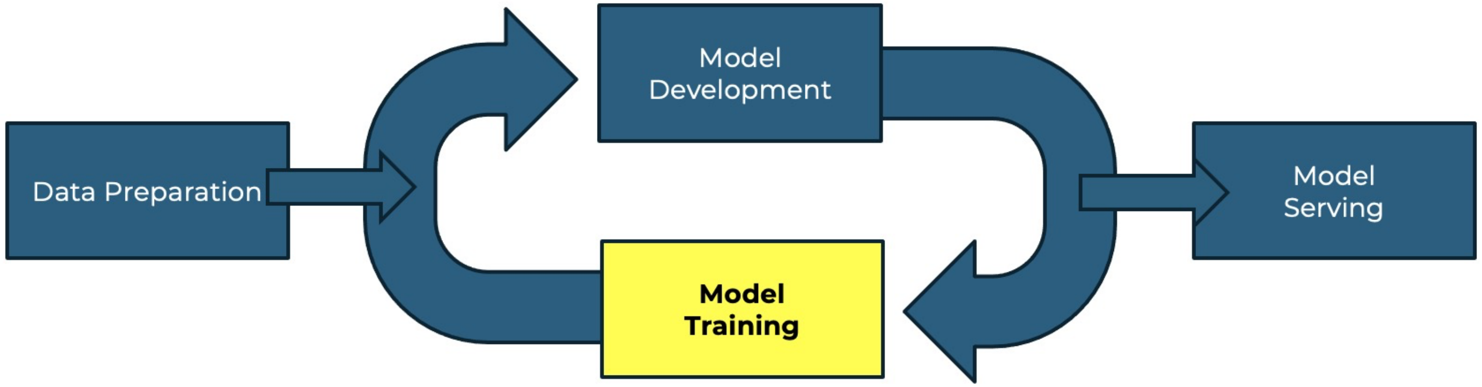
Model Training





Industry Use Cases

Model Training



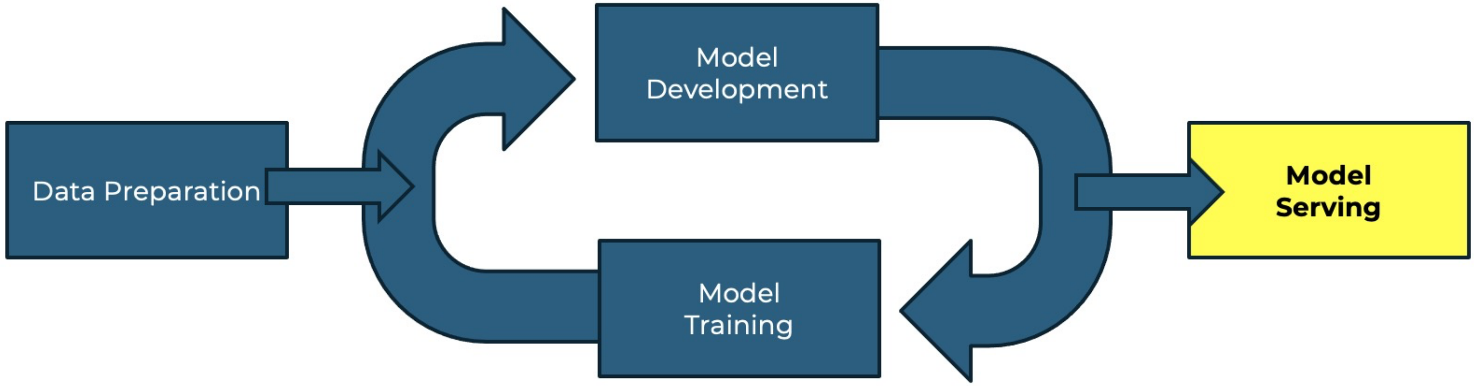
TensorFlow

Keras

PyTorch

Industry Use Cases

Model Serving

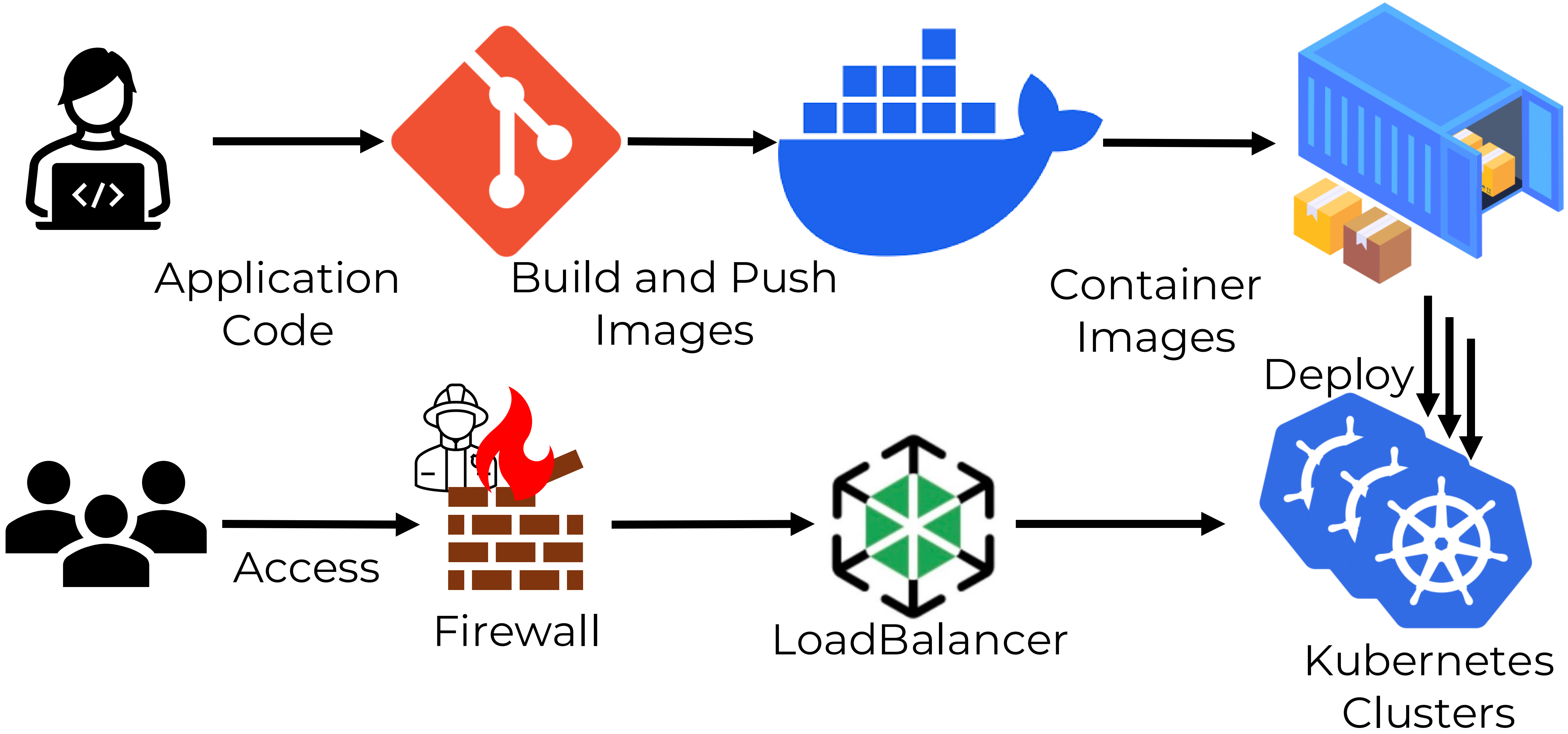


CORE

KServe

TensorFlow /serving

Deploying AI/ML on Kubernetes



Deploying AI/ML on Kubernetes


```
$ minikube start --driver docker --container-runtime docker --gpus all
😄  minikube v1.33.1 on Debian 11.10 (amd64)
✨  Using the docker driver based on user configuration
📌  Using Docker driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.44 ...
🔥  Creating docker container (CPUs=2, Memory=8000MB) ...
🐳  Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image nvcr.io/nvidia/k8s-device-plugin:v0.15.0
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, nvidia-device-plugin, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

$ kubectl get nodes -o yaml | grep -i gpu
      nvidia.com/gpu: "1"
      nvidia.com/gpu: "1"
```

Deploying AI/ML on Kubernetes


```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Load and preprocess the CIFAR-10 dataset
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck’)
....
....(skipped)
```

Deploying AI/ML on Kubernetes

sakang@localhost: ~

```
sakang@localhost:~$ python3 pytorch_example.py
Using device: cuda
Files already downloaded and verified
Files already downloaded and verified
[1,  2000] loss: 2.157
[1,  4000] loss: 1.895
[1,  6000] loss: 1.732
[1,  8000] loss: 1.611
[1, 10000] loss: 1.547
[1, 12000] loss: 1.505
[2,  2000] loss: 1.440
[2,  4000] loss: 1.414
```

```
root@localhost:~# nvidia-smi
Sun Aug 18 18:51:06 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.107.02             Driver Version: 550.107.02     CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Quadro RTX 6000                Off |   00000000:00:02.0 Off |                  Off |
| 33%   34C    P2             61W /  260W |     406MiB /  24576MiB |     10%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A     12186      C   python                                        170MiB |
|    0   N/A  N/A     19274      C   python3                                       232MiB |
+-----------------------------------------------------------------------------------------+
```

Deploying AI/ML on Kubernetes


```dockerfile
# Use the latest PyTorch image as the base image
FROM pytorch/pytorch:latest

# Set the working directory
WORKDIR /app

# Copy necessary files into the Docker image
COPY requirements.txt /app/requirements.txt
COPY pytorch_example.py /app/pytorch_example.py

# Install required packages
RUN pip install --no-cache-dir -r requirements.txt

# Execute the script
CMD ["python", "pytorch_example.py"]
```

e.g.) requirements.txt

```
torch==2.0.0
torchvision==0.15.0
numpy==1.24.0
pandas==2.0.3
matplotlib==3.7.1
scikit-learn==1.2.2
```

Deploying AI/ML on Kubernetes

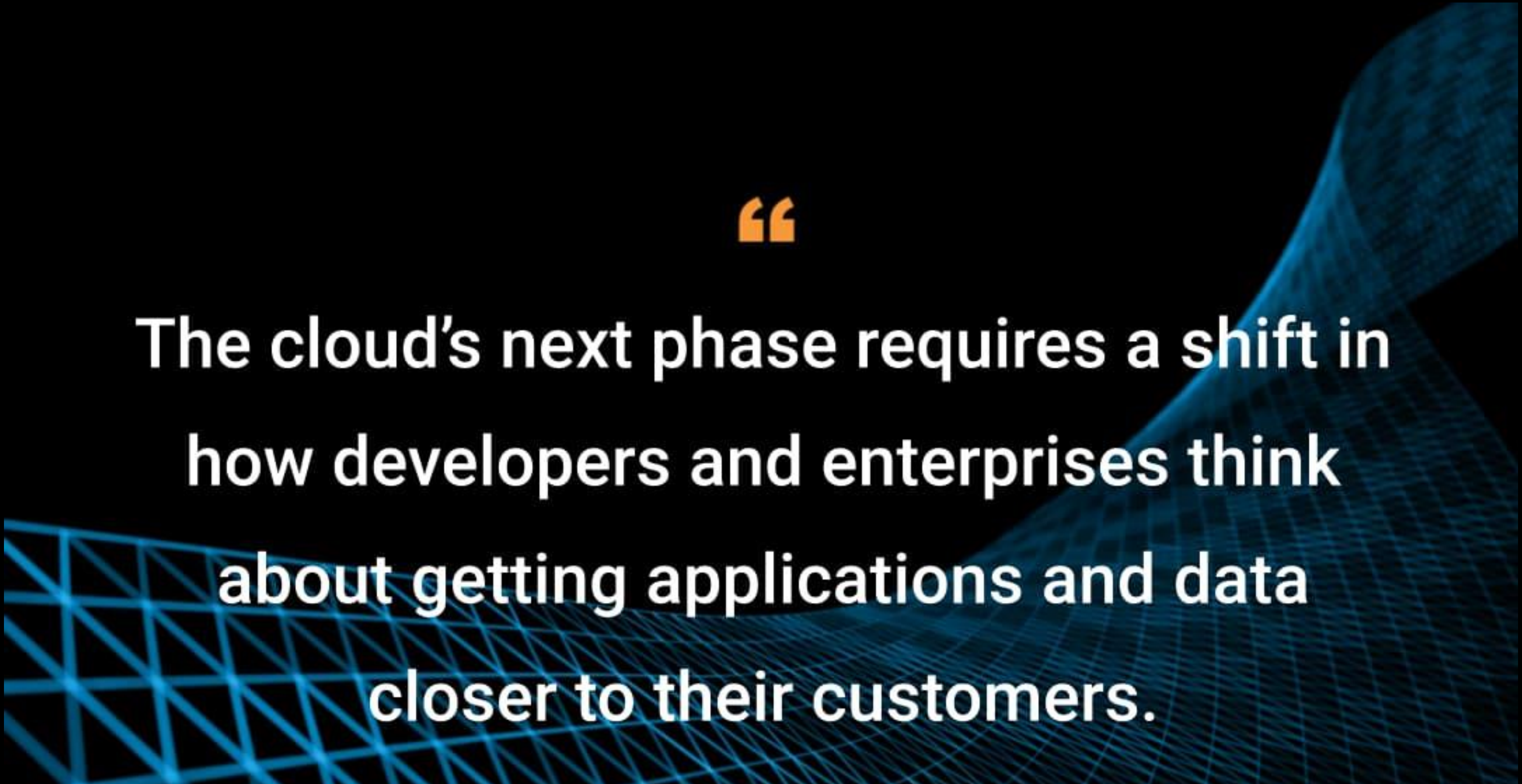
KubeCon CloudNativeCon OPEN SOURCE SUMMIT AI_dev China 2024

```yaml
#GPU_Pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pytorch-example
spec:
  restartPolicy: OnFailure
  containers:
    - name: pytorch-example
      image: localhost:5000/pytorch_example:latest
      imagePullPolicy: IfNotPresent
      resources:
        limits:
          nvidia.com/gpu: 1
  imagePullSecrets:
    - name: registrypullsecret
```

- Sets up a Docker container
- The latest PyTorch image
- Copies the necessary files into the container
- Finally runs the Python script

Demand for Distributed Cloud

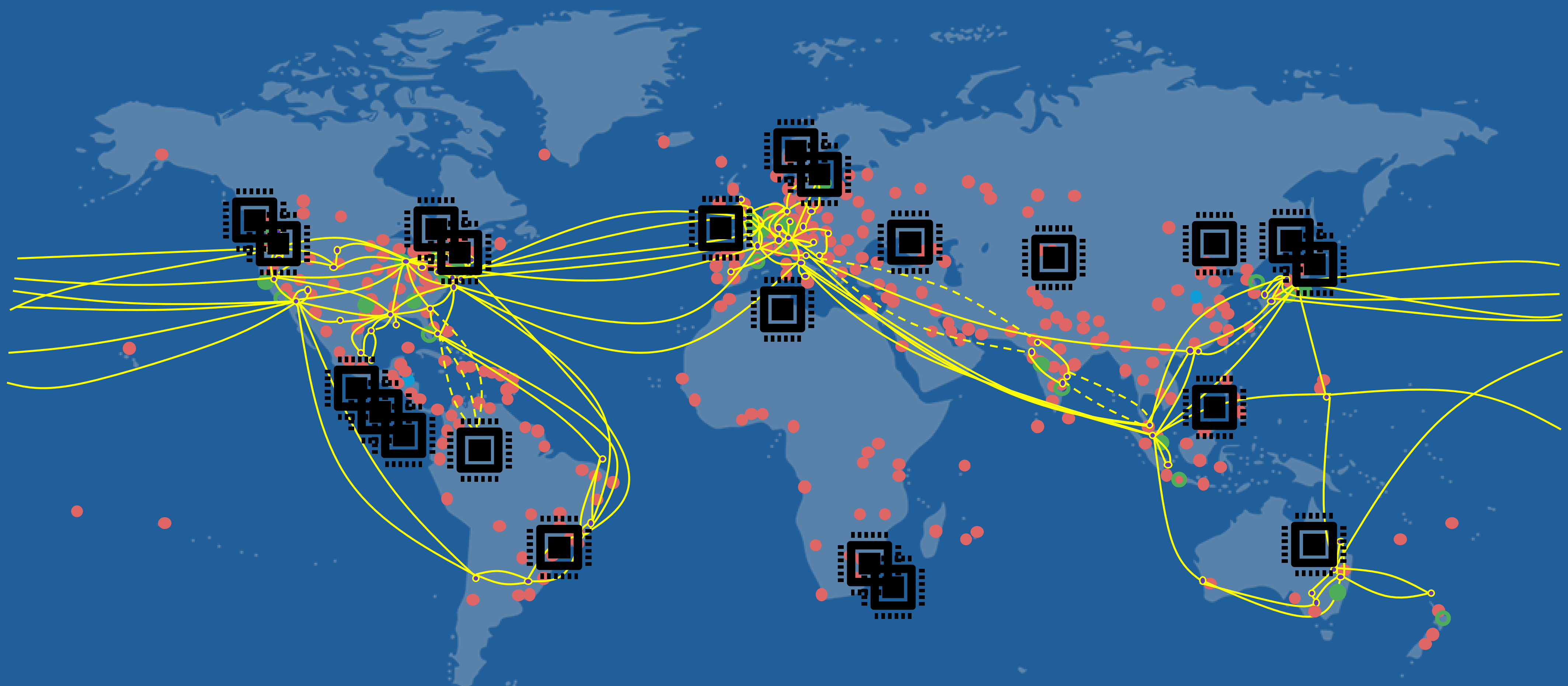
Demand for Distributed Cloud



Tom Leighton
CEO of Akamai Technologies

Distributed Cloud for AI on K8s

KubeCon CloudNativeCon OPEN SOURCE SUMMIT AI_dev China 2024



Core Computing Regions Distributed Regions / Edge Computing Sites / CDN PoPs

Source Codes & Scripts in the slide

- NFD(Node Feature Discovery) Script
  https://github.com/BrandonKang/k8s-Node-Feature-Discovery
- AI Training vs. AI Inference
  https://github.com/BrandonKang/ai_training_vs_inference
- Minikube & PyTorch ML case
  https://github.com/BrandonKang/Pytorch-on-Minikube

Thank You!