No More Runtime Setup!

Let's Bundle, Distribute, Deploy, Scale LLMs Seamlessly with Ollama Operator

DaoCloud Fanshi Zhang


DaoCloud
Kubernetes

Fanshi Zhang

Senior software engineer

nekomeowww

The Challenge

Deploying and scaling LLMs is complex

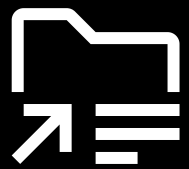
Model Distributing 101

Overview of steps

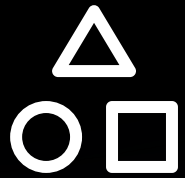
1	2	3	4
Train pre-trained models	Train LoRA	Merge weights	Quantize

Model Distributing 101

Ways to deploy models



Mount with Volumes



Bundle into images

Model Distributing 101

Challenges and complexities

Weights

Nodes

Weights are large

LLAMA 2 has roughly 83GB of weight & parameter files

Distribute weights across deploying worker nodes

Inference server is distributed, each of worker node requires dedicated copy of weights

Caching and cold boot for serverless and edge scenarios

For serverless scenarios like WasmEdge, IoT, Ray, rolling update models is a challenge

Model Serving 101

Bringing models to production

Complex dependencies

Managing dependencies across environments can be tedious and error-prone.

Environment setup

Setting up environments with Python, CUDA, and more is complex and time-consuming.

Distribution overhead

Distributing large models efficiently remains a significant challenge.

Model Serving 101

NVIDIA Triton

Bringing models to production

This is how Triton Inference Server can be used to serve models:

```
# Step 1: Create the example model repository
git clone -b r24.07 https://github.com/triton-inference-server/server.git
cd server/docs/examples
./fetch_models.sh

# Step 2: Launch triton from the NGC Triton container
docker run --gpus=1 --rm --net=host -v ${PWD}/model_repository:/models nvcr.io/nvidia/tritonserver:24.07-py3 tritonserv

# Step 3: Sending an Inference Request
# In a separate console, launch the image_client example from the NGC Triton SDK container
docker run -it --rm --net=host nvcr.io/nvidia/tritonserver:24.07-py3-sdk
/workspace/install/bin/image_client -m densenet_onnx -c 3 -s INCEPTION /workspace/images/mug.jpg
```

Model Serving 101

Bringing models to production

TorchServe

This is how Triton Inference Server can be used to serve models:

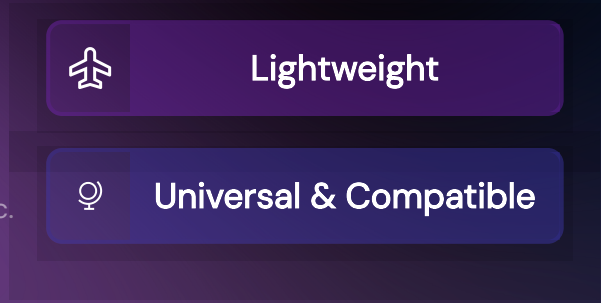
```
# Step 1: Create the example model repository
git clone -b r24.07 https://github.com/triton-inference-server/server.git
cd server/docs/examples
./fetch_models.sh

# Step 2: Launch triton from the NGC Triton container
docker run --gpus=1 --rm --net=host -v ${PWD}/model_repository:/models nvcr.io/nvidia/tritonserver:24.07-py3 tritonserv

# Step 3: Sending an Inference Request
# In a separate console, launch the image_client example from the NGC Triton SDK container
docker run -it --rm --net=host nvcr.io/nvidia/tritonserver:24.07-py3-sdk
/workspace/install/bin/image_client -m densenet_onnx -c 3 -s INCEPTION /workspace/images/mug.jpg
```

Ollama

Universal solution to model bundling, distributing, serving, etc.



Ollama – Bundling Models

Universal bundling

```
# Create a new model
ollama create mymodel -f ./Modelfile

# Pull an existing model
ollama pull llama2

# List available models
ollama list

# Run a model
ollama run llama2 "Tell me a joke about programming"
```

Ollam- LoRA, Customizing, Prompting

Integrating LoRA for training

```
# Modelfile with LoRA
FROM llama2
ADAPTER ./path/to/lora/weights.bin

# Create the model with LoRA
ollama create mylora -f ./Modelfile

# Run the model with custom prompts
ollama run mylora "Translate this to French: Hello, world!"
```

Ollama - Distributing

Just like OCI images

`ollama push [model name]`

OCI Registry

`ollama push [model name]`

OCI ready

OCI-Compatible Distribution

Ollama uses OCI-compatible formats for easy integration with existing container workflows

Ollama - Serving

Just like OCI images

One simple command, across all platforms, environments.

```
ollama run [model name]
```

So what have we done?

So what have we done?

Ollama Operator

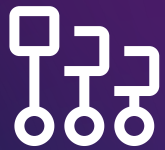
One simple intall-to-go plugin solution that brings Ollama to your Kubernetes clusters

Features

Key capabilities

Model caching

Model Preloading



Distributed replicas

Resource limit

Operator automation

Any Kubernetes clusters

No extra plugins, no extra CRDs, no worries!

Kuberneted

Works with any certified Kubernetes cluster

Cloud Agnostic

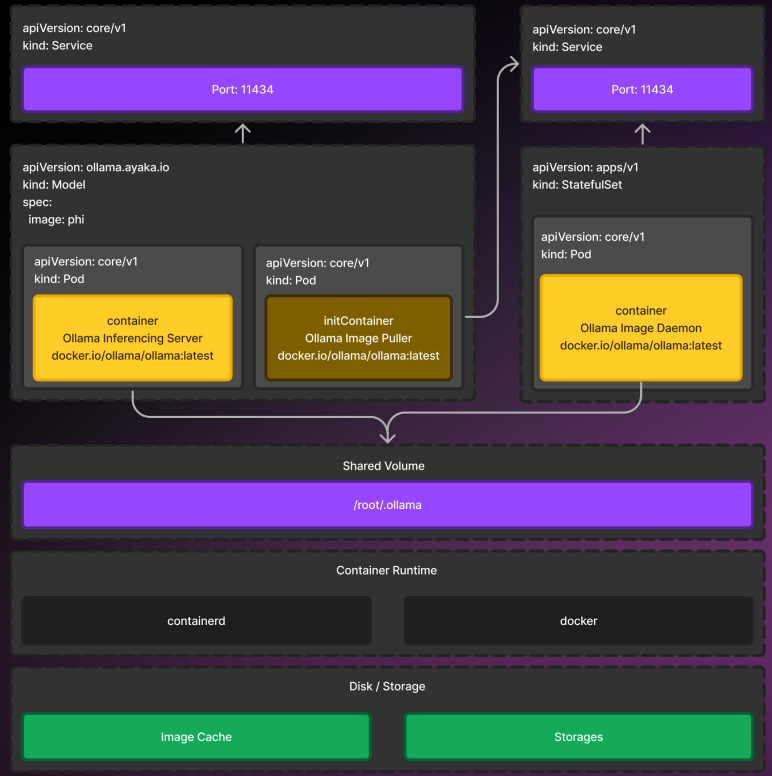
Deploy on any cloud provider or on-premises, even Raspberry Pi

Plug and Play

No additional plugins or CRDs required. Dependency...free?

How It Works

Deploying models with Ollama Operator



Watch it in live

Light! Shot! Action!

Scaling with Ease

Effortless scalability

1. Update the `replicas` field in the Model CRD
2. Automatic load balancing
3. Resource optimization
4. Horizontal Pod Autoscaler (HPA) support

```yaml
apiVersion: ollama.ayaka.io/v1
kind: Model
metadata:
  name: llama2
spec:
  image: llama2
  replicas: 5  # Scale to 5 replicas
```

Future

Foresight from our perspective

1 Documentations & CLI

- Use cases
- Explain how to configure for advanced use cases
- Improve CLI to directly interact with `ollama`

2 Automation

- Better model pooling
- Predictive maintenance for model performance
- Real-time model health monitoring

3 Beyond

- Seamless integration with popular ML frameworks
- Gateway, routing and load balancing support
- Cluster inference

Let’s Build Together

Join us in shaping the future of LLM deployment

GitHub

Documentation

KubeCon CloudNativeCon THE LINUX FOUNDATION OPEN SOURCE SUMMIT AI_dev

To community

Let’s improve it together

KubeCon

CloudNativeCon

Thank You!

Let’s revolutionize LLM deployment together