MPLS Japan 2024

生成AIの民主化を支える リファレンスアーキテクチャ

NTTコミュニケーションズ イノベーションセンター 技術顧問
山下 克司

自己紹介 ～ 山下克司

NTTコミュニケーションズ株式会社
イノベーションセンター 技術顧問

東京大学大学院 情報理工学系研究科 産学連携プロジェクト 技術アドバイザー

技術顧問 所属先

本日の内容は、いかなる所属先の見解を代表するものではなく、すべて登壇者個人の責任に基づくものですので、ご了承ください。

本日のアジェンダ

生成AIの民主化

LLMアプリケーション開発の

ユースケース

LLMサービスプラットフォームの
リファレンスアーキテクチャ

GPUスケジューリングとクラスタリング

これまでの人工知能の民主化の動き

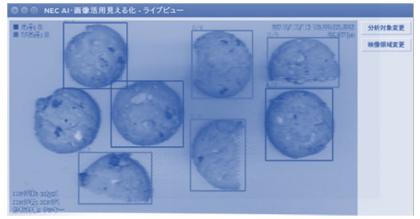
人工知能技術が高度な研究者の手を離れて、市井のエンジニアやプログラマーが通常のシステムと同様容易にシステムに組み込むことができるようになったこと。

APPLICATION
IoTデータ解析や画像認識による良品検査、音声認識と書き起こしなどのアプリケーション

Framework
オープンソースのフレームワークTensorFlow、Caffe、Caffe2、Microsoft Cognitive Toolkit（CNTK）、MXnet、Pytorchなどが自由に利用可能になった

Cloud
GPUインスタンスやFPGAインスタンスなど多様なタイプが従量課金で利用可能になった

Processor
GPUやTPUによる高速処理とSnapdragon/A11をはじめとするMPUが推論エンジンを搭載

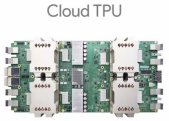


フェイフェイ・リは中国生まれのアメリカ人コンピューター科学者 AIの民主化を発見した

TensorFlow

Caffe2

PyTorch



Core MLは、CPU、GPU、ニューラルエンジンを活用して、オンデバイスのパフォーマンスを最適化しながら、メモリフットプリントと電力消費を最小限に抑えます。ユーザーのデバイスでのみモデルを実行することにより、ネットワーク接続の必要がなくなるため、ユーザーデータのプライバシーが保護され、Appのレスポンスも改善されます。

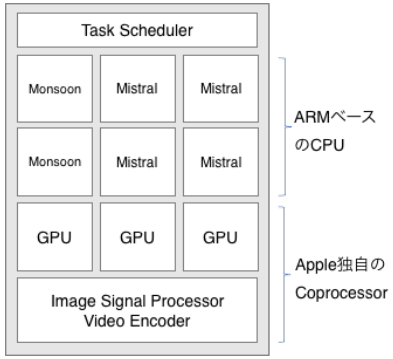


図3 A11 Bionic SOC構成

生成AIの民主化の動き

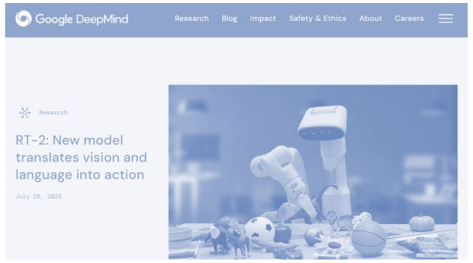
生成AIのモデル構築や推論に必要とされている巨大なGPUプールをプラットフォームがサービスとして提供することで、生成AIに対するアクセスが容易になり、あらゆるアプリケーションや知識検索に影響がおよぶ

APPLICATION
あらゆるアプリケーションのユーザーインターフェースに自然言語処理が用いられ知識検索がると同時にマルチモーダルな画像認識によってロボティクスなどに応用範囲が拡大した

Framework
Hugging Faceによるモデルの公開、GitHub Copilotによるプログラミング支援、LangChainやRAG等のユーザーアプリケーションフレームワークによりアプリケーション開発が容易に

Platform
大規模言語モデルを稼働させて、ユーザーセッションやAPIによるアクセスを実現するクラウドサービスとしてLLMアプリケーションのプラットフォームとして成長

Processor
NVDIA H100クラスのGPUやTensor Flow Processor V5eなどをクラスタリングするNVLINK等のOptical ネットワークによって巨大化したGPU資源プール



Google DeepMind RT2プロジェクト
「コーラの空き缶をゴミ箱に捨てて」

Google DeepMindのロボティクス責任者、ヴィンセント・ヴァンホーク博士は「RT-2はWebデータの大規模なコーパスから知識を転送できるため、ゴミが何であるかを理解しており、明示的なトレーニングなしで識別できる」と説明する。「ポテトチップスの袋やバナナは、食べた後にゴミになる。RT-2は視覚言語トレーニングデータからそれを理解できる」と述べている。

Hugging Face

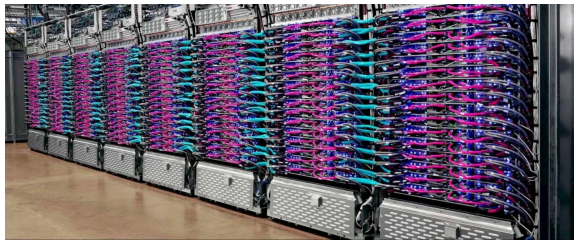
様々な開発スタイルによるモデルのオープン化

LangChain

GitHub Copilot

APIによるアプリケーション開発

巨大なGPU資源プール



Platform Service

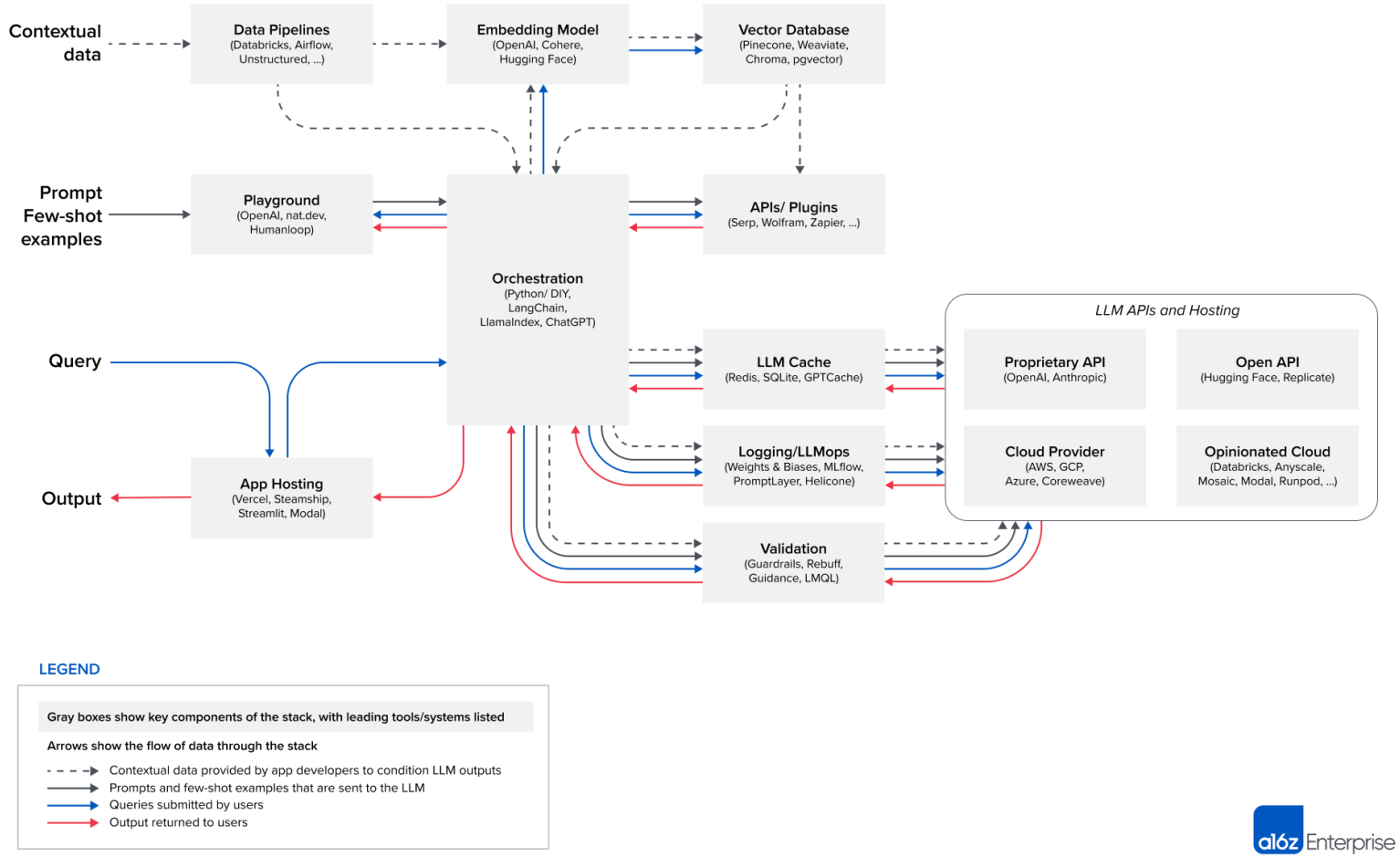
サービスとして動作している推論エンジン

Building LLM Applicationのリファレンス・ユースケース

LLMアプリケーション構築の3ステージ

- **Data preprocessing / embedding:** This stage involves storing private data (legal documents, in our example) to be retrieved later. Typically, the documents are broken into chunks, passed through an embedding model, then stored in a specialized database called a vector database.
- **Prompt construction / retrieval:** When a user submits a query (a legal question, in this case), the application constructs a series of prompts to submit to the language model. A compiled prompt typically combines a prompt template hard-coded by the developer; examples of valid outputs called few-shot examples; any necessary information retrieved from external APIs; and a set of relevant documents retrieved from the vector database.
- **Prompt execution / inference:** Once the prompts have been compiled, they are submitted to a pre-trained LLM for inference—including both proprietary model APIs and open-source or self-trained models. Some developers also add operational systems like logging, caching, and validation at this stage.

Emerging LLM App Stack



a16z Enterprise

https://a16z.com/emerging-architectures-for-llm-applications/

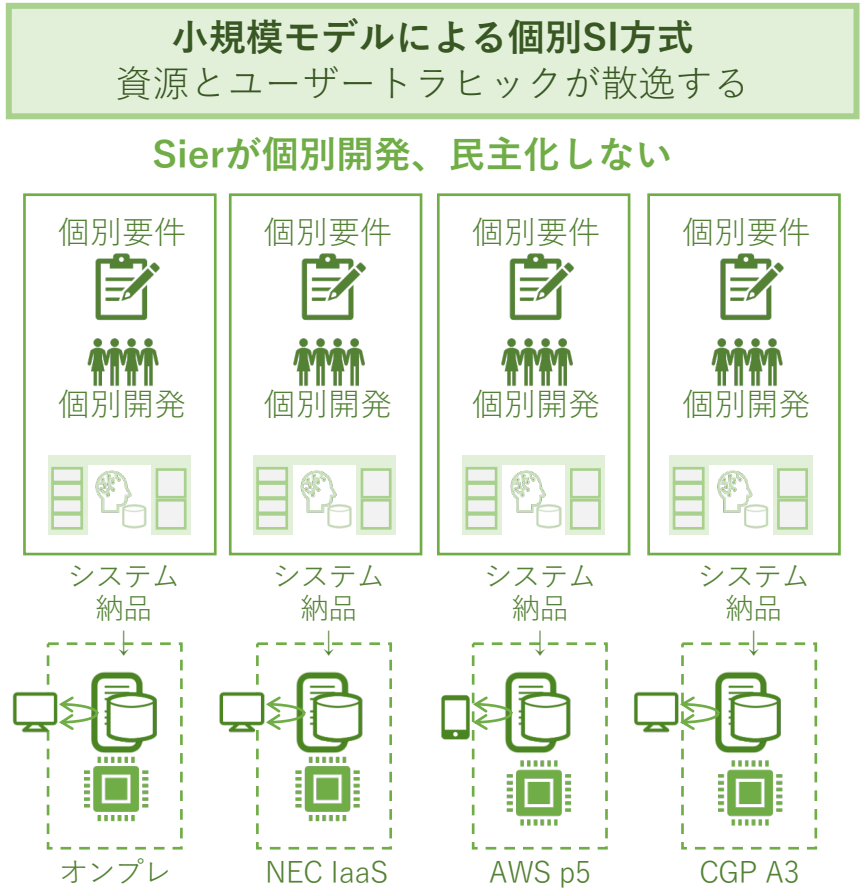
Component References

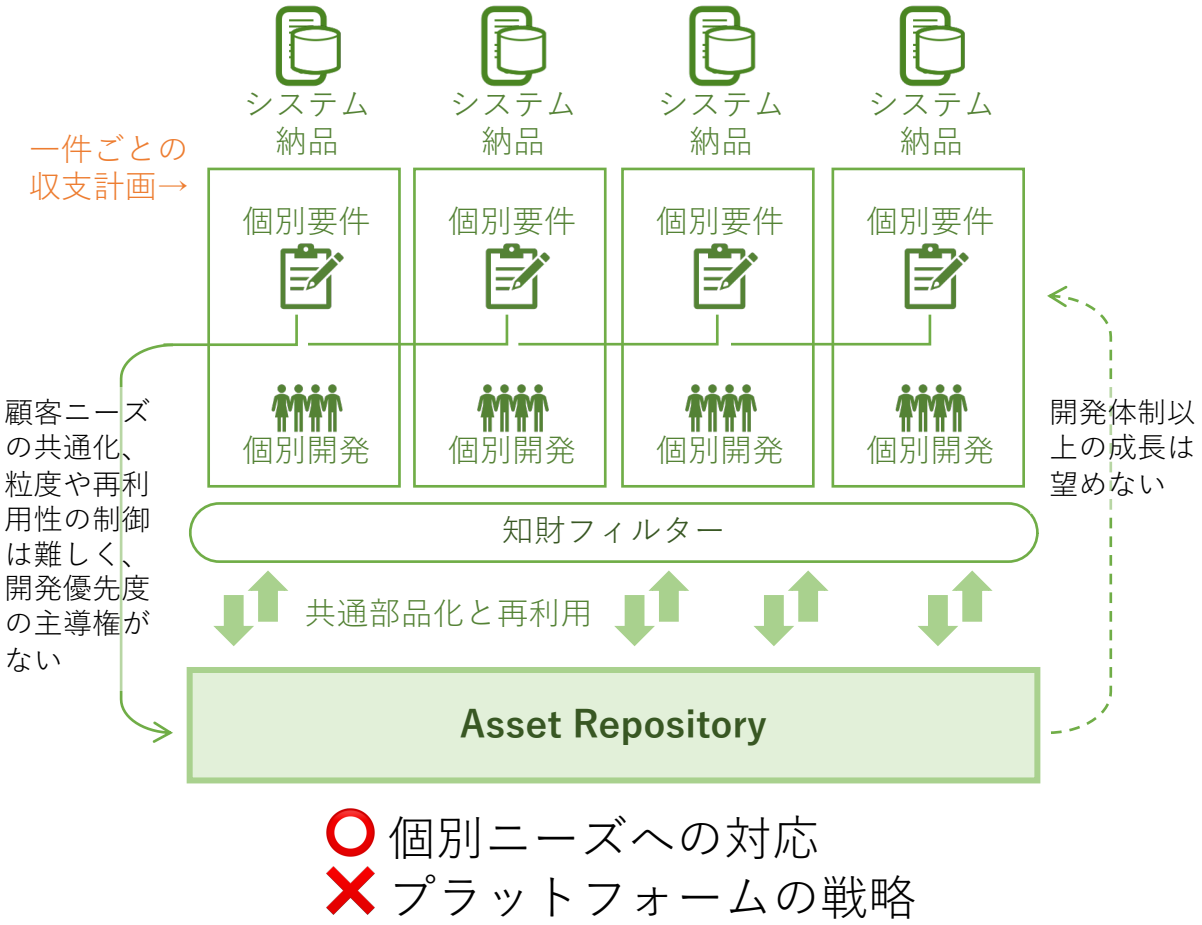
Data pipelines	Embedding model	Vector database	Playground	Orchestration	APIs/plugins	LLM cache
Databricks	OpenAI	Pinecone	OpenAI	Langchain	Serp	Redis
Airflow	Cohere	Weaviate	nat.dev	LlamaIndex	Wolfram	SQLite
Unstructured	Hugging Face	ChromaDB	Humanloop	ChatGPT	Zapier	GPTCache
		pgvector				

Logging / LLMops	Validation	App hosting	LLM APIs (proprietary)	LLM APIs (open)	Cloud providers	Opinionated clouds
Weights & Biases	Guardrails	Vercel	OpenAI	Hugging Face	AWS	Databricks
MLflow	Rebuff	Steamship	Anthropic	Replicate	GCP	Anyscale
PromptLayer	Microsoft Guidance	Streamlit			Azure	Mosaic
Helicone	LMQL	Modal			CoreWeave	Modal
						RunPod

個別SI

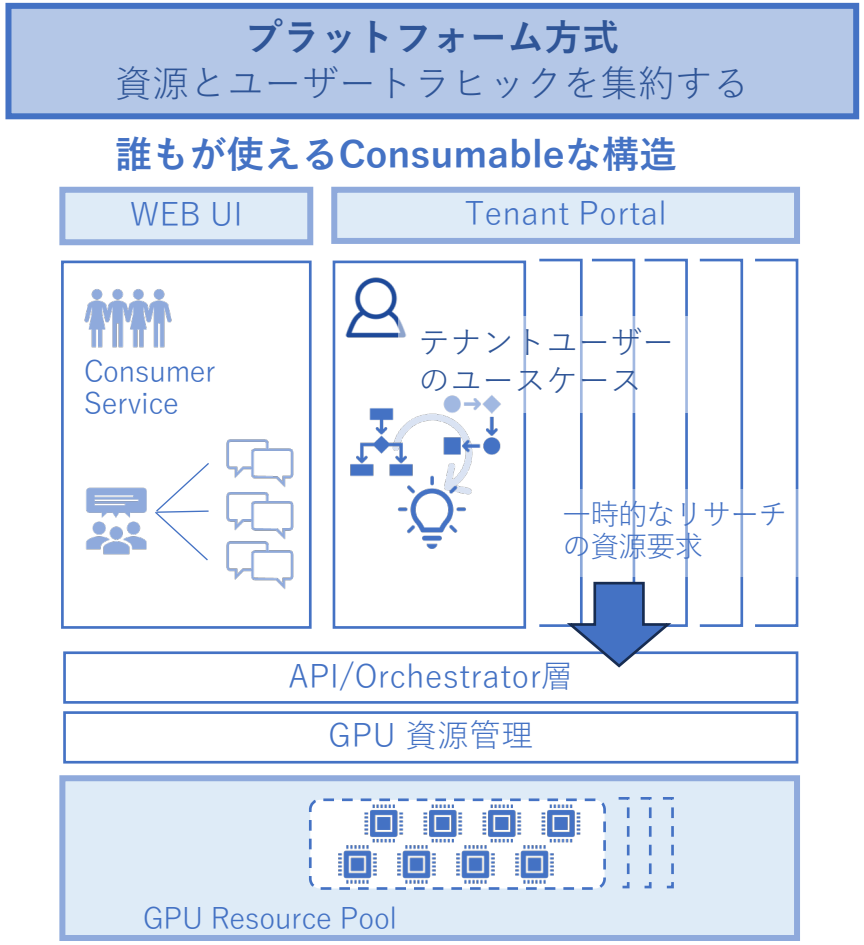
個別SI案件としての小規模モデルの活用を進める方式

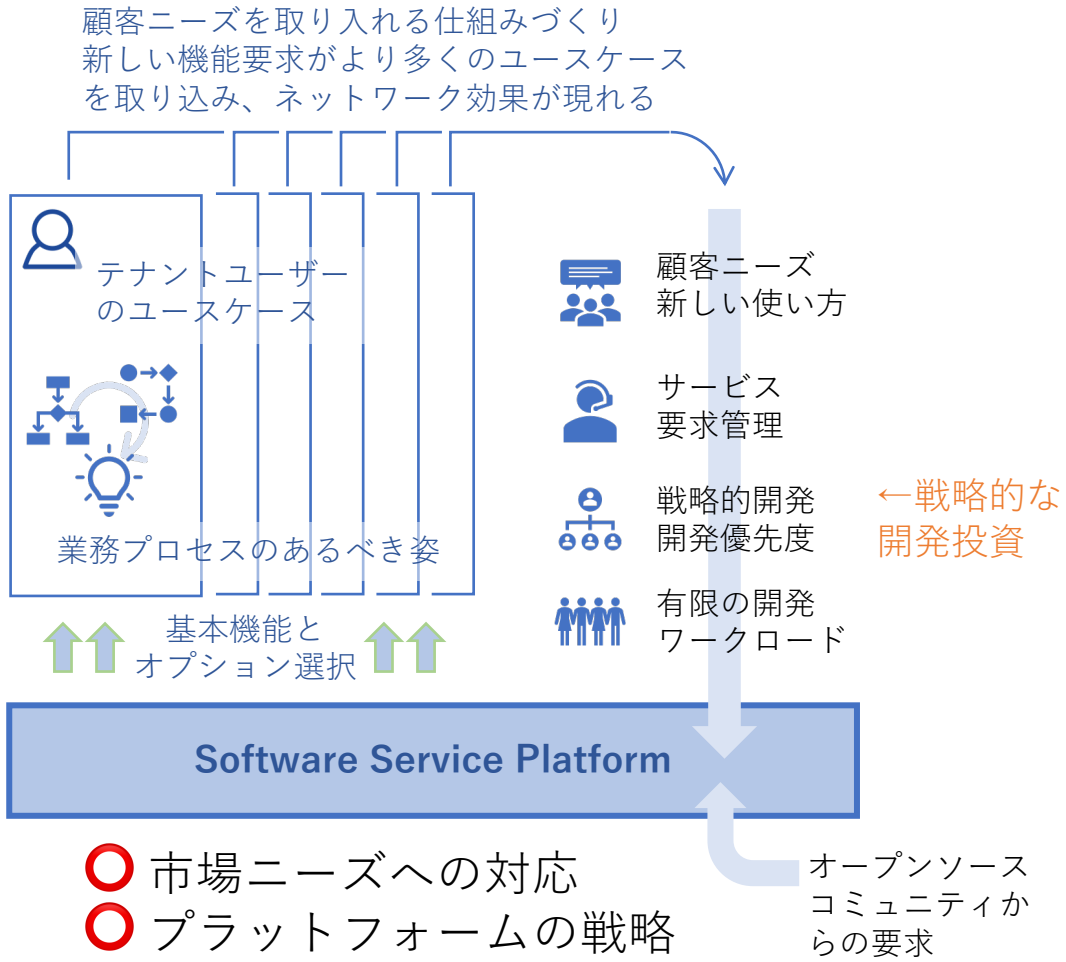




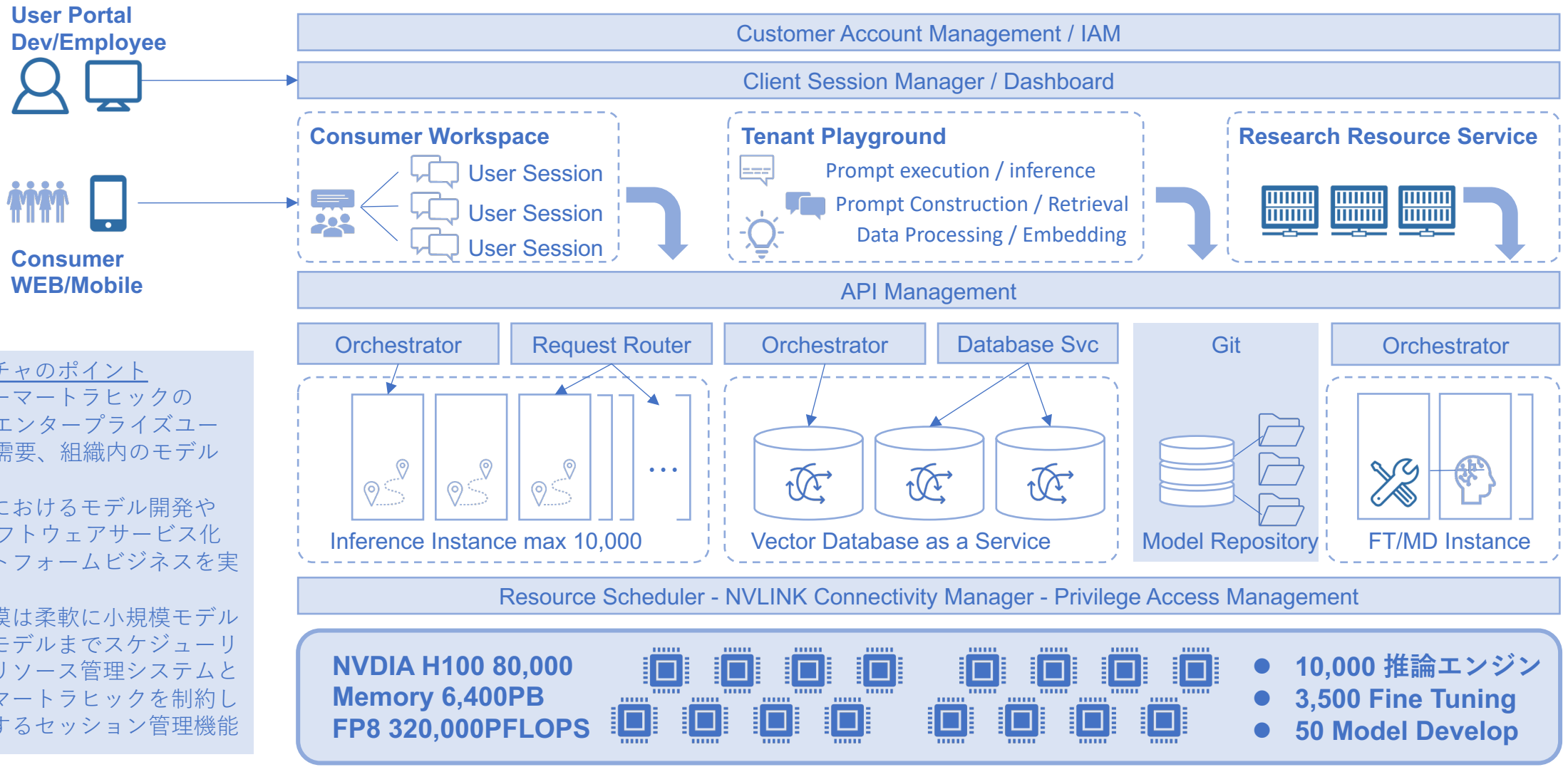
プラットフォームによる資源集約

コンシューマートラヒックを活用した柔軟な資源活用を進める方式





Building LLM Applicationをユースケースにしたプラットフォームのリファレンスアーキテクチャ



アーキテクチャのポイント

・コンシューマートラヒックのGPU資源とエンタープライズユーザーのGPU需要、組織内のモデル開発を共有

・テナントにおけるモデル開発やFT需要をソフトウェアサービス化してプラットフォームビジネスを実現

・モデル規模は柔軟に小規模モデルから大規模モデルまでスケジューリングできるリソース管理システムとコンシューマートラヒックを制約して資源確保するセッション管理機能

GPUサイズ

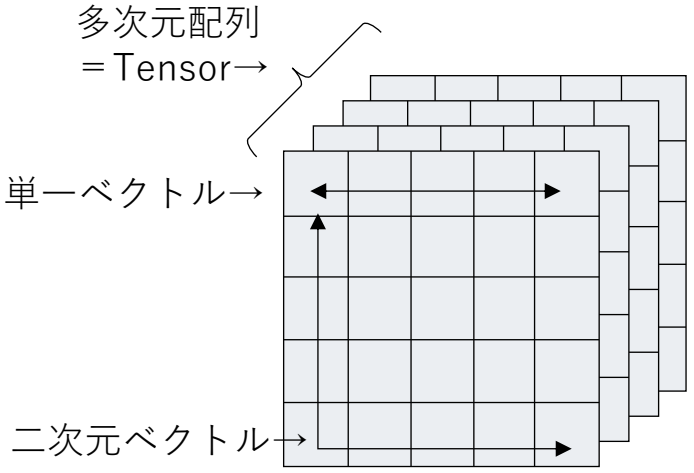
- 推論モデルサイズ 例えば16GB　←V100単体で対応できる範囲（NECやTuzumi）
  - LoRA Fine Tune*では 60GB　（←A100以上のモデル）
  - Full Parameter Fine Tuneでは180GB（←NVLINKで複数のGPUを束ねる）

*Hugging Face Parameter Efficient Fine-Tuning (PEFT) Library使用

GPUモデル	Tensor Cores数	CUDAコア数	メモリ容量	メモリバンド幅	TDP	価格
Tesla T4	320	2,560	16 GB GDDR6	320 GB/s	70 W	$2,500
Tesla V100		5,120	16 GB HBMe2	900 GB/s	250 W	$8,000
A100	432	6,912	40 GB HBMe2	1.6 TB/s	300 W	$11,000
H100	456	14,592	80 GB HBMe2	↑	350W	$30,000

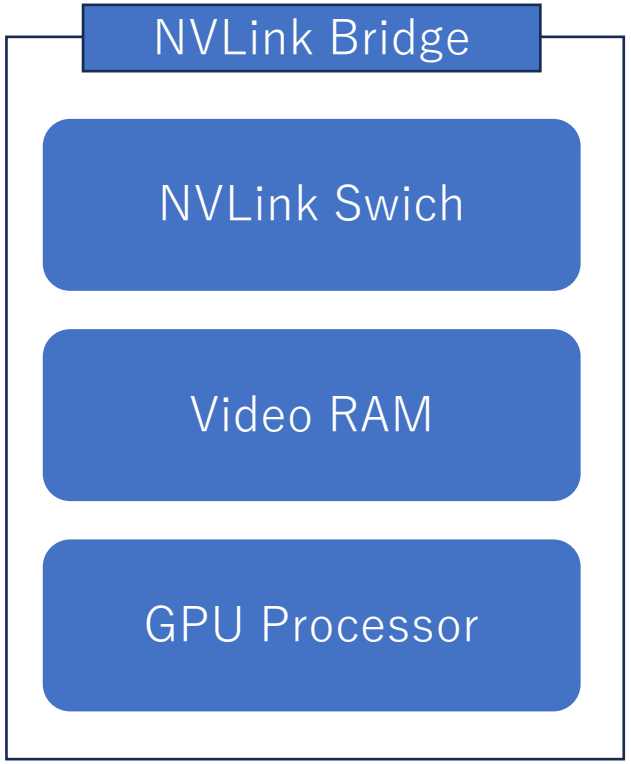
価格の情報は為替の関係で不確かです

GPUの基本的な構造



NVDIA Volta(2017)から採用

機械学習に使われる
↓
Tensor Core
画像→ **CUDA Core**
レイトレーシング→ **RT Core**



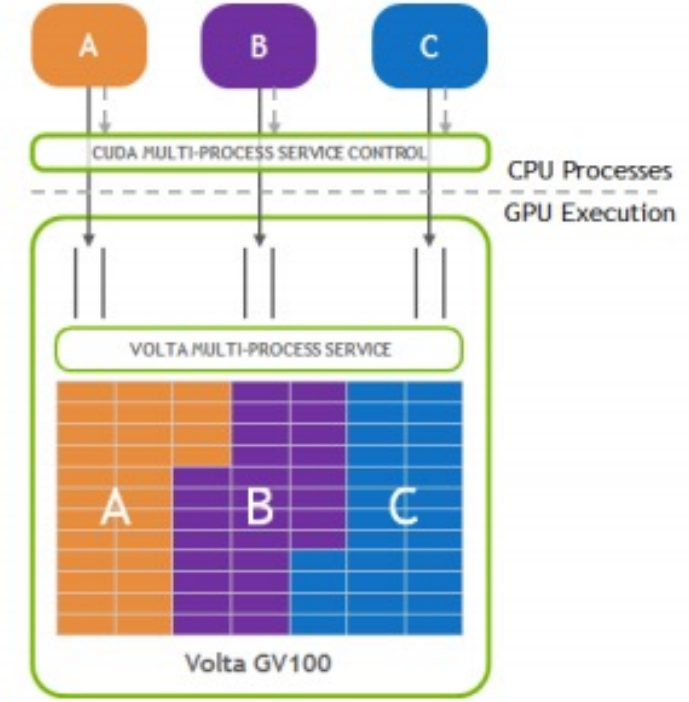
--- Application Toolkit ---
CUDA Toolkit
↕
--- runtime library---
User Mode Driver (libcuda.io)
Kernel Driver (nvdia.ko)
<NVDIA Display Driver Package>

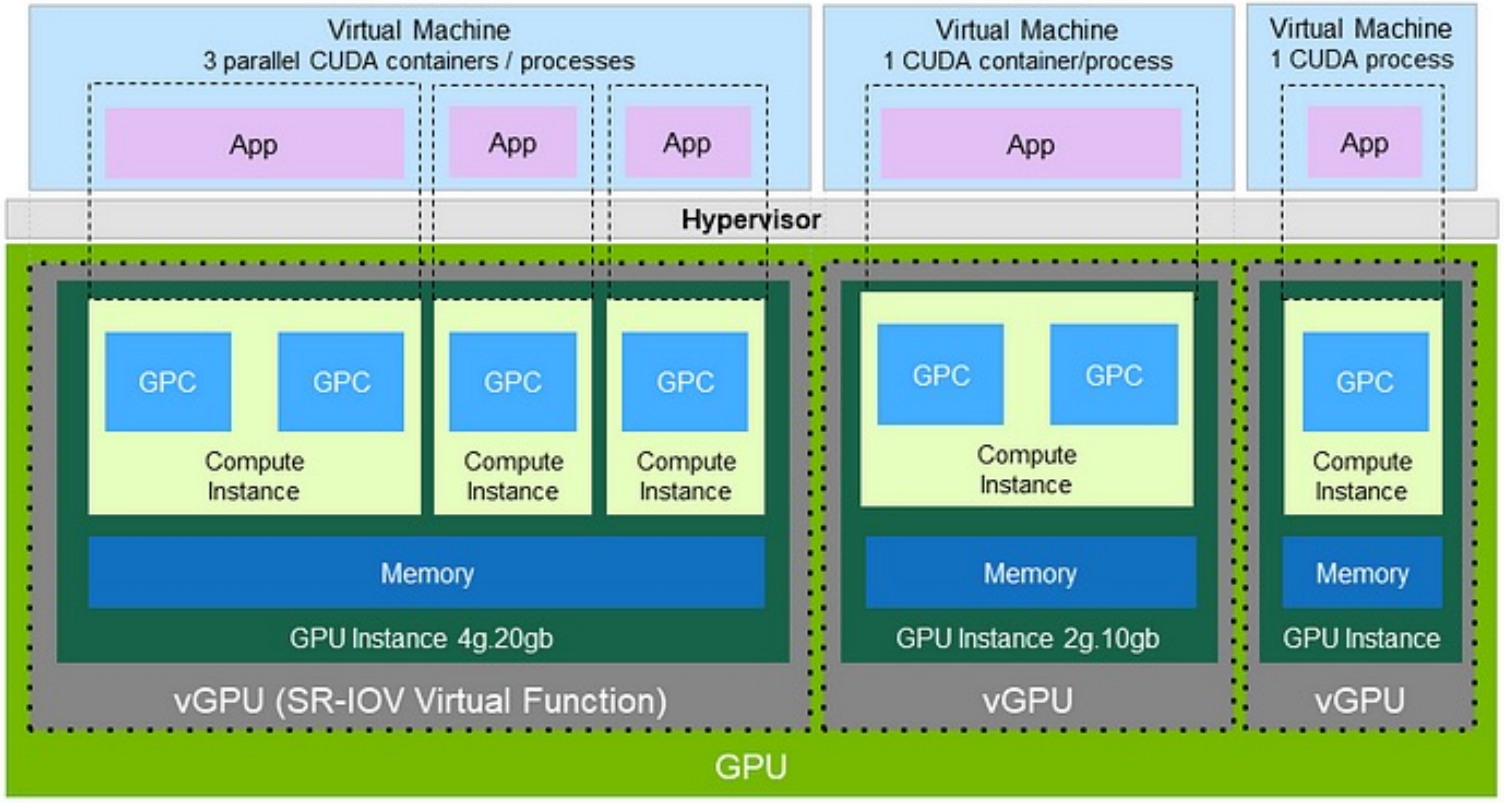
CUDA Toolkit

B200
NVDIA Blackwell Architecture
288billion Transistor
TSMC 4nm Process
10TB/s interchip communication
TensorRT-LLM Core
FP4 micro-tensor Scaling
TEE-I/O対応 Confidential Computing
NVLink Switch Chip (NVL72 130TB/s)

vGPUのスケジューリング

CUDA Multi Process Service
(nvdia-cuda-mps-serverがMPSごとに立つ)





- **mps-deamon**: mpsのcontrollerの役割。mps-clientのuidを見て、mps-serverを立ち上げたりといった管理をする。
- **mps-server**: GPU毎に立ち上がるserver。GPUのschedulerの役割。
- **mps-client**: mps-deamonおよびmps-serverとやり取りをするためのもの。1プロセスに対して立ち上がる。

https://docs.nvidia.com/vgpu/latest/grid-vgpu-user-guide/index.html
https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

ModelをGPUに載せる

リソースの大きさは操作者が理解している前提の命令セット
仮想GPUをスケジューリングに用いることで機械可読なリソースキャパシティを設定してアプリケーションに割り付けることが可能になる

```
if torch.cuda.is_available():
    model = model.to(“cuda:0")

model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype="auto",device_map="auto")
```

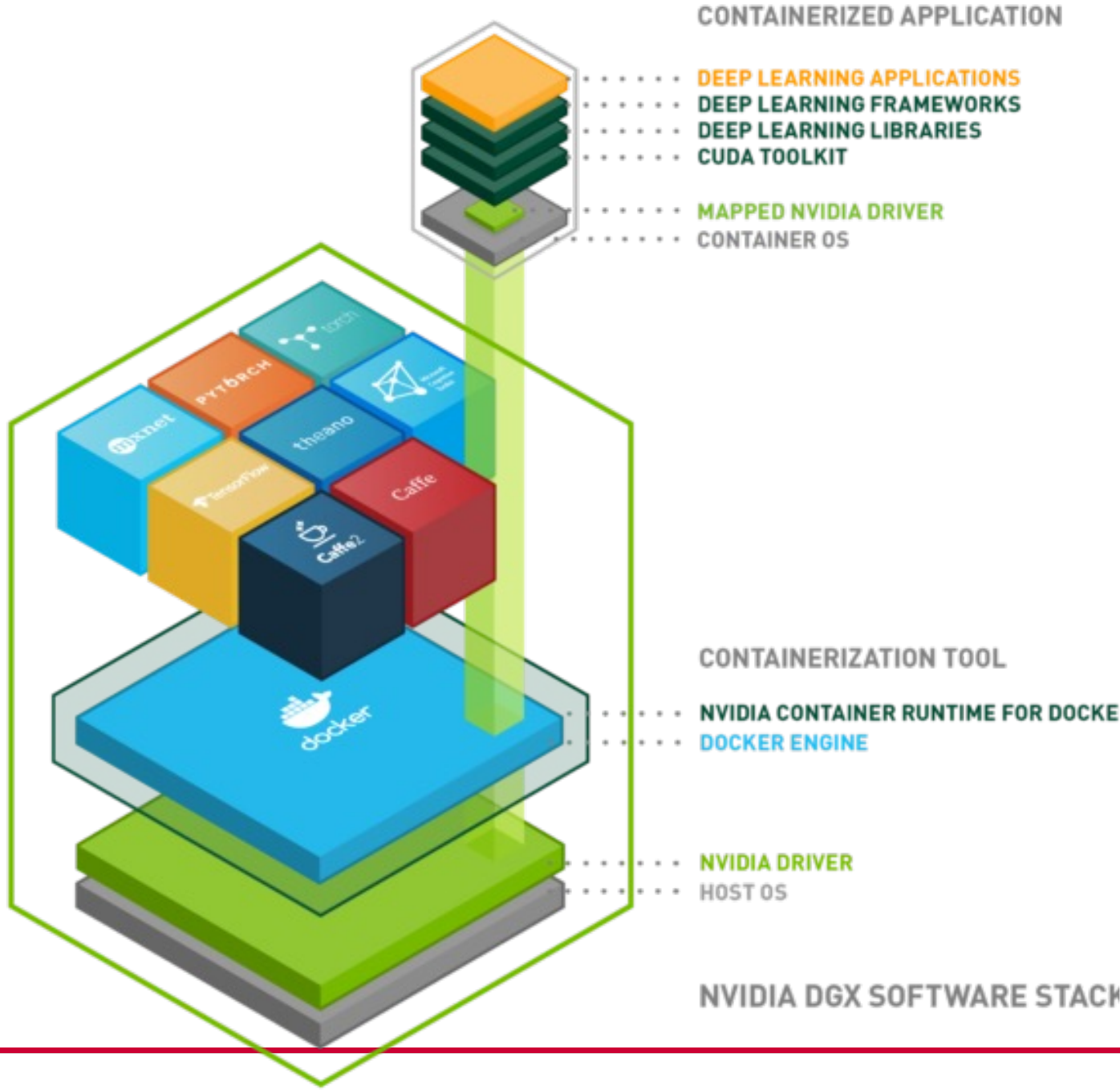
```
|=========================================+======================+======================|
|   0  NVIDIA A100 80GB PCIe          Off | 00000001:00:00.0 Off |                    0 |
| N/A   40C    P0              74W / 300W |   6997MiB / 81920MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA A100 80GB PCIe          Off | 00000002:00:00.0 Off |                    0 |
| N/A   38C    P0              74W / 300W |   6997MiB / 81920MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
```

```
export CUDA_VISIBLE_DEVICES=0,2
export CUDA_DEVICE_ORDER=FASTEST_FIRST
torchrun trainer-program.py ...
```

Enabling GPUs in the Container Runtime Ecosystem

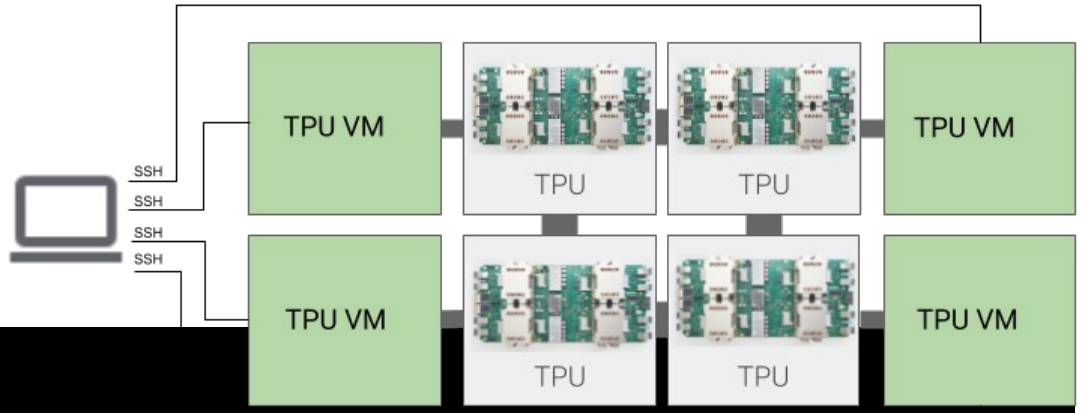
The NVIDIA Container Runtime introduced here is our next-generation GPU-aware container runtime. It is compatible with the Open Containers Initiative (OCI) specification used by Docker, CRI-O, and other popular container technologies.

You'll learn about the NVIDIA Container Runtime components and how it can be extended to support multiple container technologies. Let's examine the architecture and benefits of the new runtime, showcase some of the new features, and walk through some examples of deploying GPU accelerated applications using Docker and LXC.



Google TPUを使う (VMベース)

Dockerなどのコンテナは TPU VM上で動作させる
（TPU VM上のコンパイラーが必要）
→パブリックのコンテナオーケストレーションを用いることができない




```
Cloud TPU VMに接続する
$ gcloud compute tpus tpu-vm ssh tpu-name --zone=us-central2-b

PyTorch/XLAコンパイラーをインストールする
$ pip install torch~=2.4.0 torch_xla[tpu]~=2.4.0 torchvision -f https://storage.googleapis.com/libtpu-releases/index.html

スクリプト
$ export PJRT_DEVICE=TPU
$ import torch
$ import torch_xla.core.xla_model as xm
$ dev = xm.xla_device()
$ t1 = torch.randn(3,3,device=dev)
$ t2 = torch.randn(3,3,device=dev)
$ print(t1 + t2)

実行する
$ python3 tpu-test.py

tensor([[-0.2121, 1.5589, -0.6951],
[-0.7886, -0.2022, 0.9242],
[ 0.8555, -1.8698, 1.4333]], device='xla:1')
```

TPU で実行されるコードは、アクセラレータ線形代数（XLA）コンパイラによってコンパイルする必要があります。XLA は、ML フレームワーク アプリケーションによって出力されたグラフを受け取り、グラフの線形代数、損失、勾配のコンポーネントを TPU マシンコードにコンパイルするジャストインタイム コンパイラです。プログラムの残りの部分は、TPU ホストマシンで実行されます。XLA コンパイラは、TPU ホストマシンで実行される TPU ランタイムの一部です。

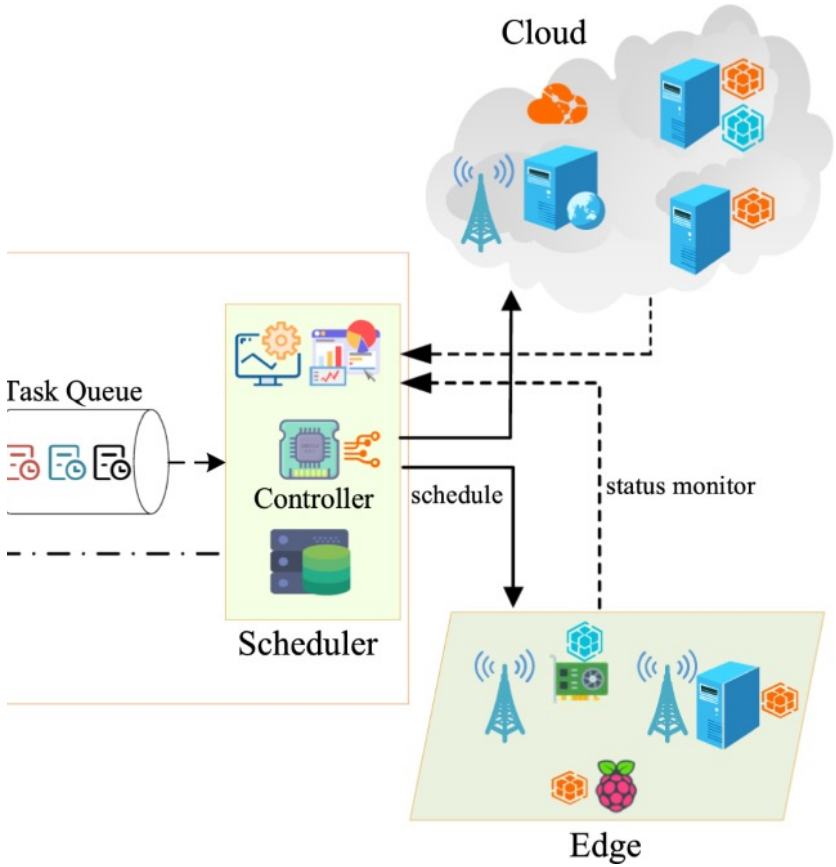
どんな箱にどのくらいのボールを入れるか

Bing Tang, Jincheng Luo, Mohammad S. Obaidat & Pandi Vijayakumar

”Container-based task scheduling in cloud-edge collaborative environment using priority-aware greedy strategy”

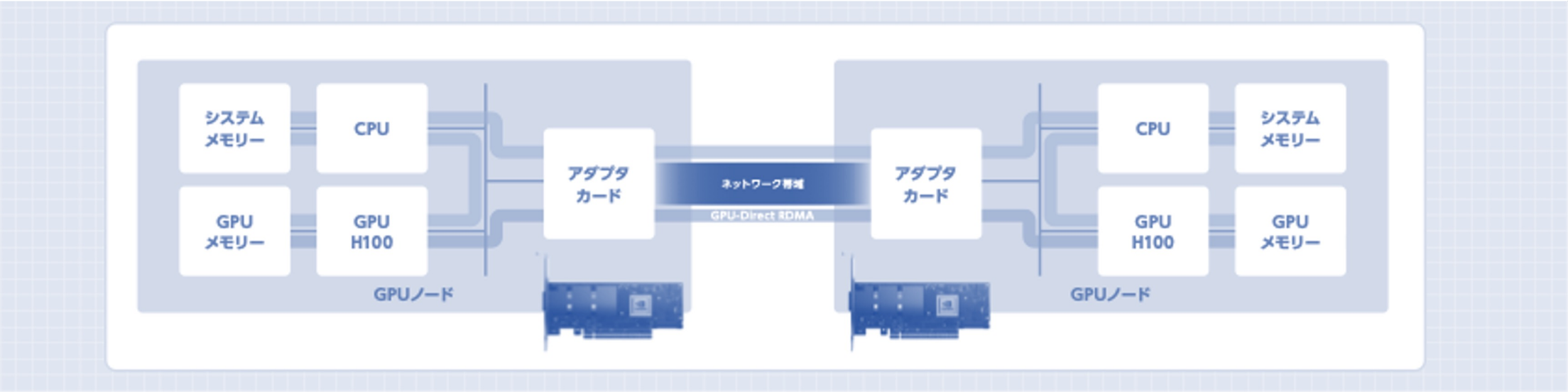
https://link.springer.com/article/10.1007/s10586-022-03765-2

Dockerに代表されるコンテナ仮想化技術は、軽量、高速なデプロイ、容易な移植性などの利点から、業界で広く利用されている。本稿では、クラウドエッジ協調環境におけるコンテナ技術に基づくAIベースのIoTアプリケーションのシナリオを考察し、**コンテナベースのタスクスケジューリングアルゴリズム**を提案する。優先順位を考慮した貪欲戦略を用いて、TOPSIS(Technique for Order Preference by Similarity to Ideal Solution)という多基準アプローチを採用したPGTという新しいスケジューリングアルゴリズムを提案する。クラウドサーバとエッジサーバのコンテナは統一されたプラットフォームで管理され、アプリケーションサービスはコンテナ内にデプロイされる。優先度が高いため、期限制約の小さいタスクが最初にスケジューリングされる。次に、タスクの応答時間、エネルギー消費量、タスク実行コストなど複数の指標を総合的に考慮し、タスク実行に最適なコンテナを見つける。エッジサーバの数とタスクの数を変化させたクラウド-エッジ協調環境におけるシミュレーションの結果、提案スケジューリング手法は、QoS満足度、エネルギー消費、ペナルティコスト、および総違反時間の改善において、4つのベースラインアルゴリズムを上回ることが示された。



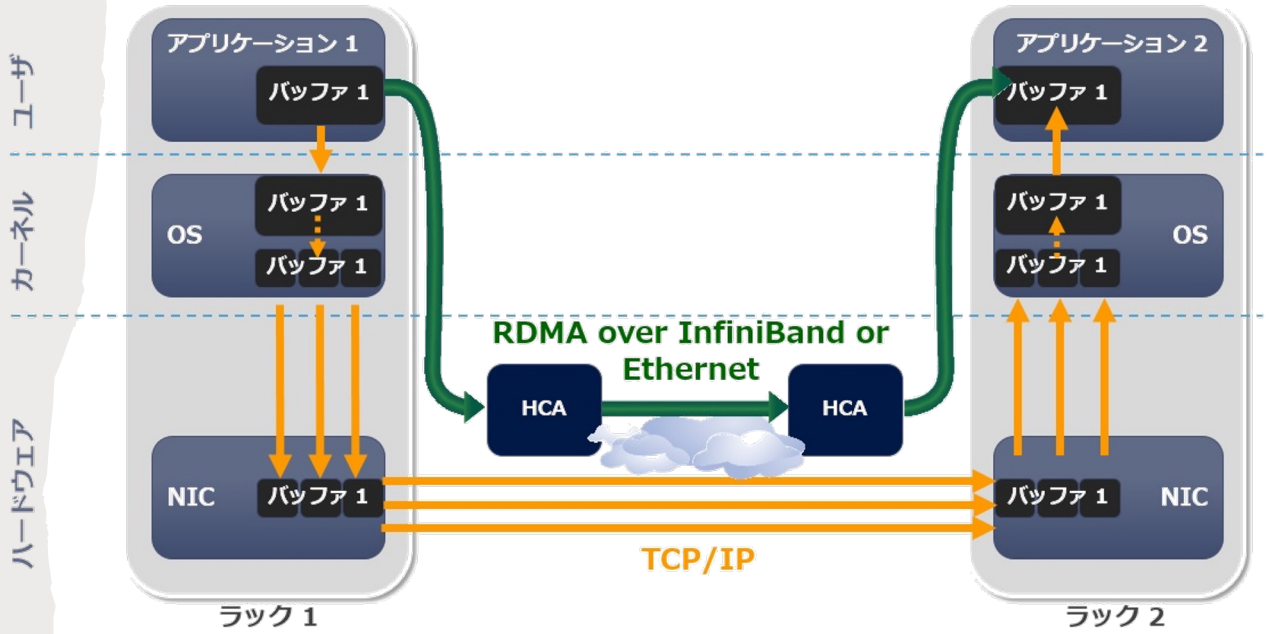
NVDIA DGX H100 GPU-Direct RDMA接続

GPUノード感通信でホストCPUを介することなく、GPUメモリー間で直接データ転送を実現
→ RDMAを実現する通信アダプターとCUDA ToolkitによってGPU-Direct RDMAを実現
☆ ノードを跨いだGPUメモリー間データ転送をオフロードできる

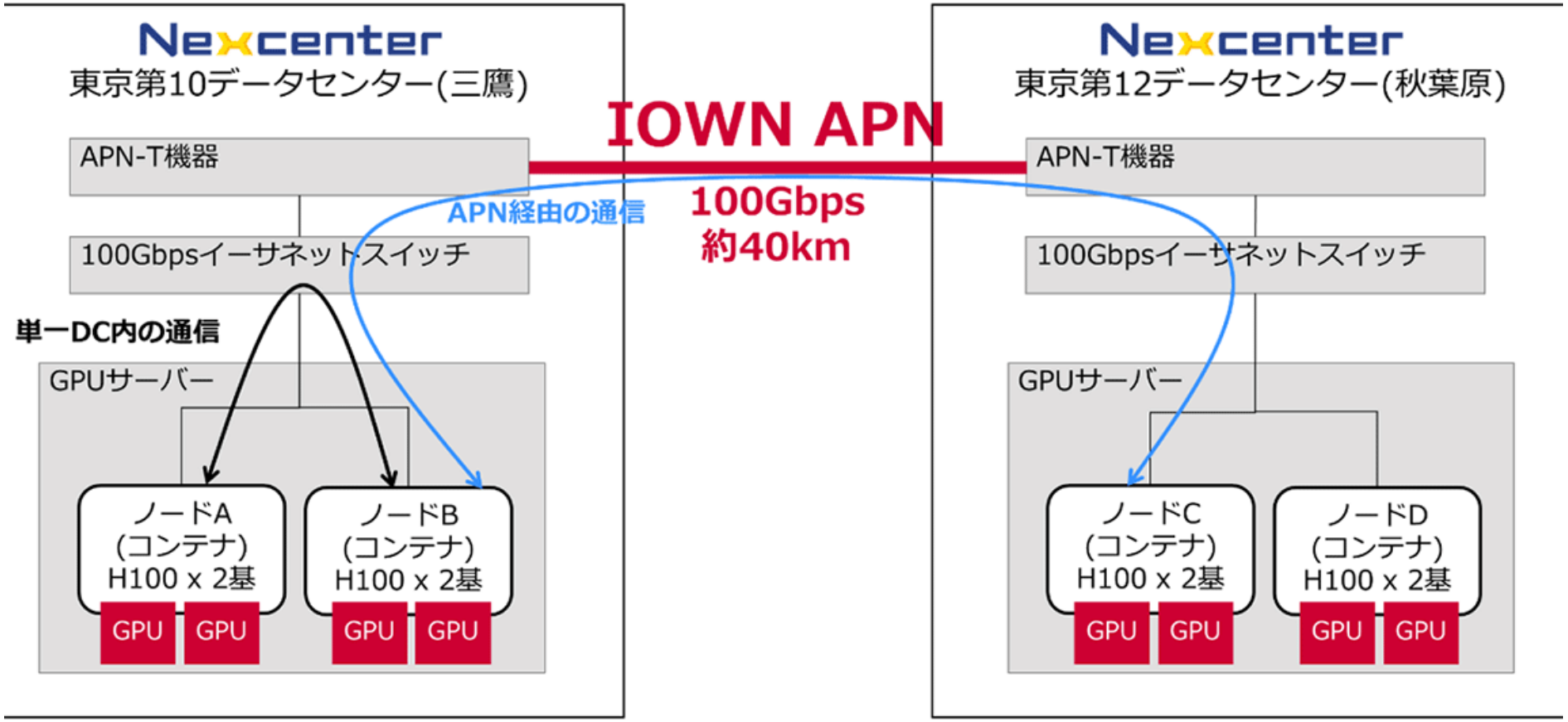


RDMAの仕組み

- 通常のネットワーク転送では、**CPUが**コンピュータ内のアプリケーションから、ネットワークのプロトコルスタック、ドライバそしてNICへとコピーすることでデータが転送され受け取る側もその逆の手順でデータを受け取ります。
- RDMA 転送では、対応のカード間で通信を行い**カードが直接送り元のコンピュータのアプリケーション内のメモリを読み**、通常のネットワークプロトコルをバイパスすることでメモリ間コピーを行わず、カードのハードウェアで転送を行い、直接転送先のコンピュータのアプリケーションメモリにデータを書き込むことで転送を完了します。このRDMA転送を広義のイーサネット環境に持ち込むものが、RoCE(RDMA over Converged Ethernet) と呼ばれる仕様になります。



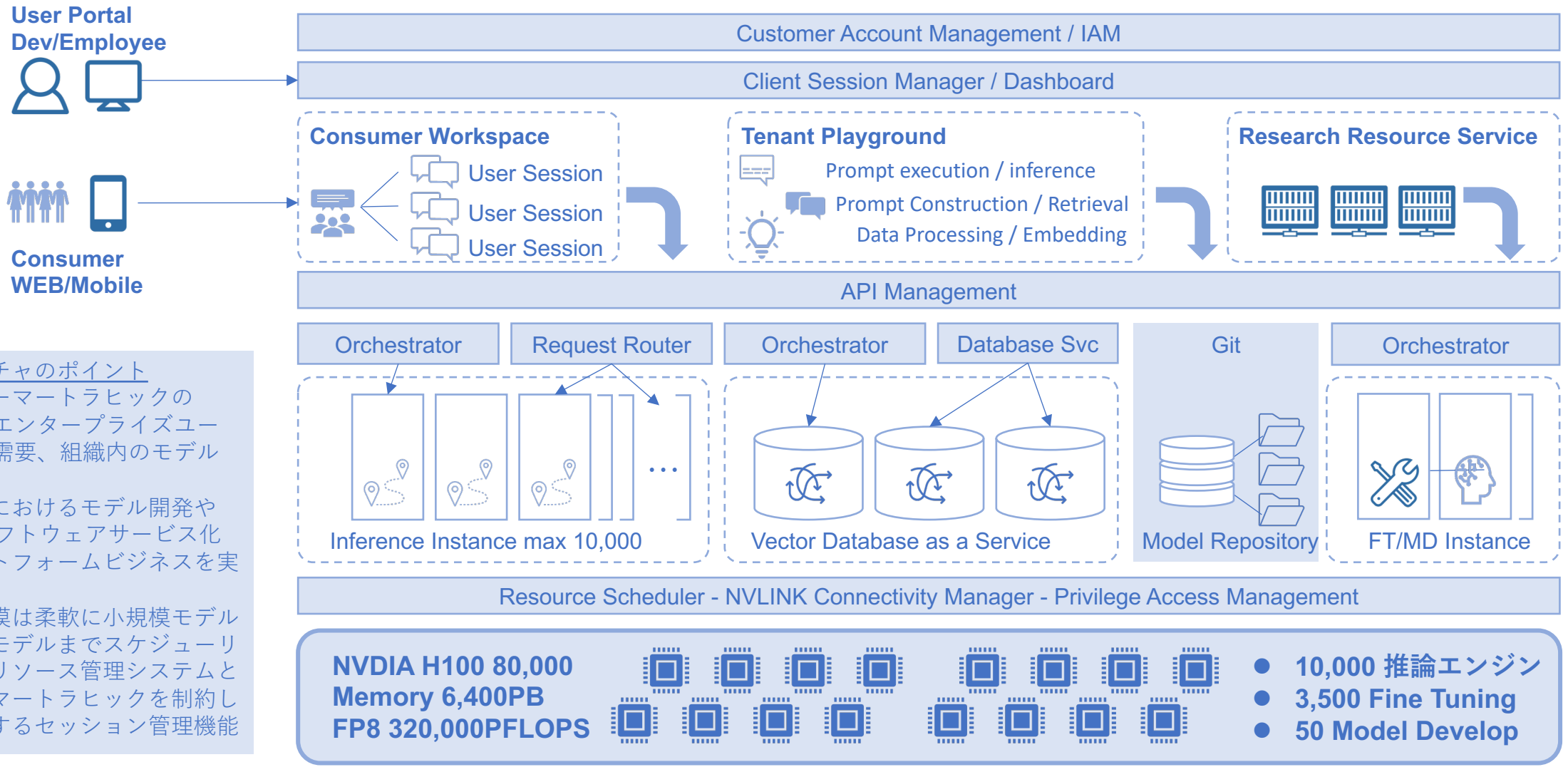
IOWN APNによるデータセンター間GPU接続



電源環境（A）

電源環境（B）

Building LLM Applicationをユースケースにしたプラットフォームのリファレンスアーキテクチャ



アーキテクチャのポイント

・コンシューマートラヒックのGPU資源とエンタープライズユーザーのGPU需要、組織内のモデル開発を共有

・テナントにおけるモデル開発やFT需要をソフトウェアサービス化してプラットフォームビジネスを実現

・モデル規模は柔軟に小規模モデルから大規模モデルまでスケジューリングできるリソース管理システムとコンシューマートラヒックを制約して資源確保するセッション管理機能