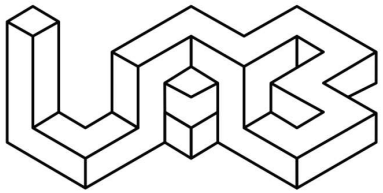
GPU TECHNOLOGY CONFERENCE 2019



Improving GPU utilization for multi-tenant deep learning workloads on DGX-2 and public GPU clouds

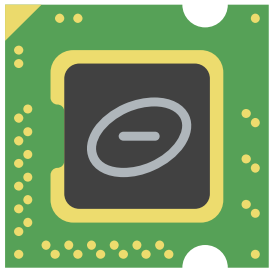
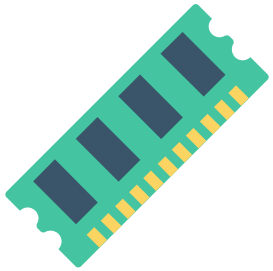
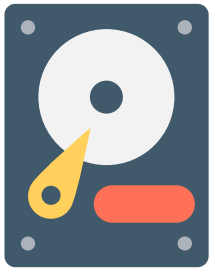
Jeongkyu Shin
Lablup Inc.
@inureyes

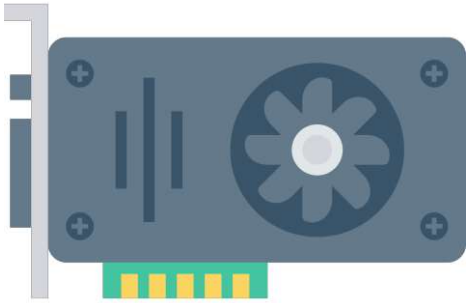
Joongi Kim
Lablup Inc.
@achimnol

Thursday, 3/21/2019 | 15:00 - 15:50 - SJCC Room

Contents

- GPU
- GPU as computational resource
- Issues on GPGPU computation
- Backend.AI
- Multi-tenant deep learning workloads
- Virtual GPU computation cluster with DGX-family
- Just model it event
  - Contributing to ML community with testing Backend.AI
  - Configuration & Characteristics
- Results and Insights from the event
- Closing

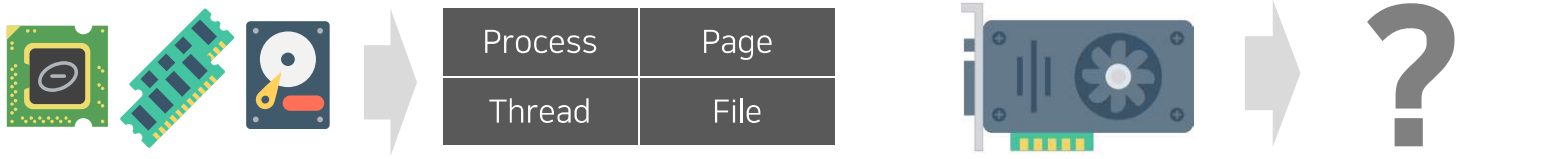






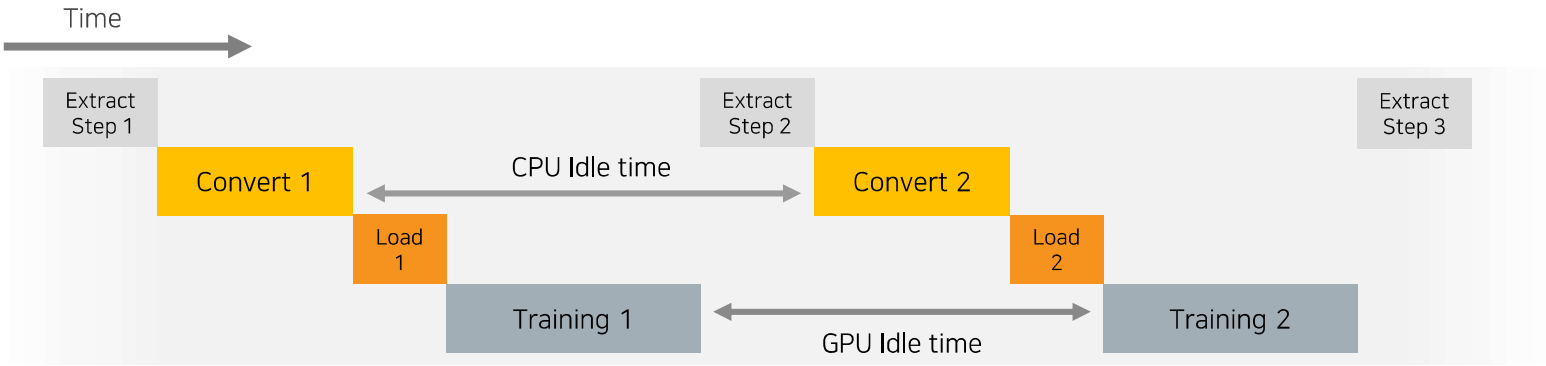
OS knows how to partition, share, and schedule
via standardized HW interfaces.

Lack of flexible GPU resource management

- Resource management / sharing technology is limited (as a peripheral device)

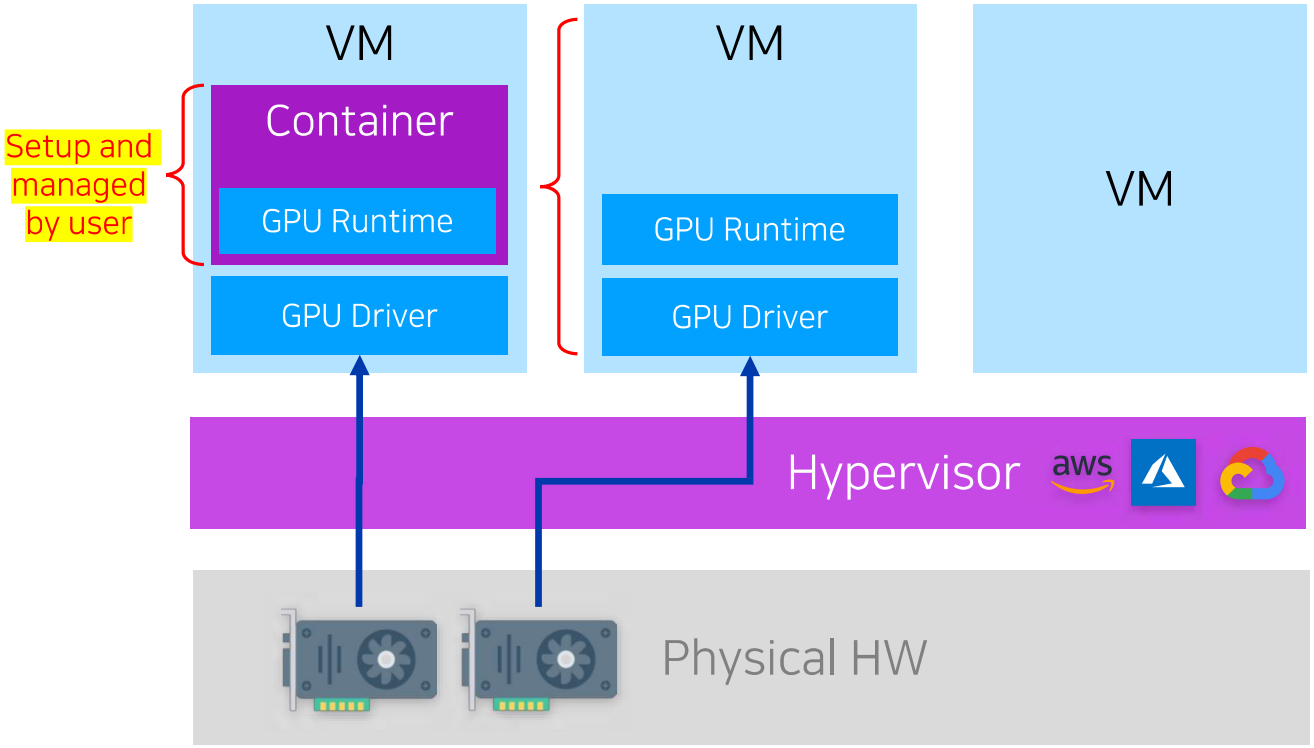


- Idle time from I/O latency



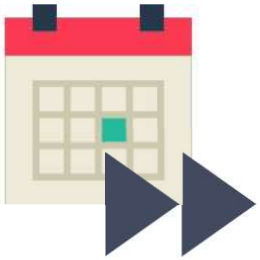
Why?

Because
GPU SW ecosystem
is **complex**.



Typical GPU cloud stack

Complexity of GPU Computing



Fast Release Cycles

Version Management

Framework Compatibility

Let

GPU computation

Be

Powerful and Easy

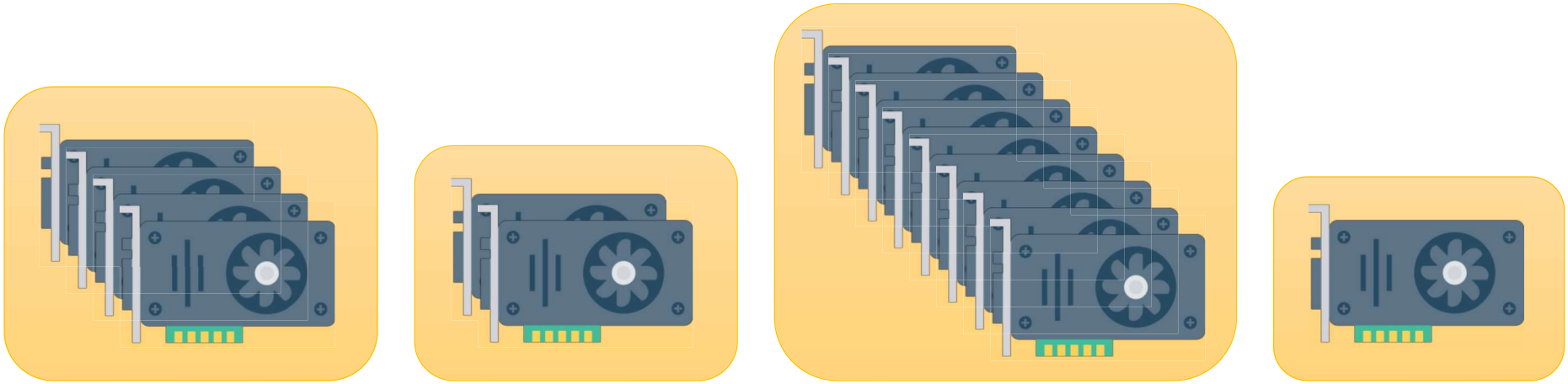
Let’s solve the issue.

By making a solution;;


…And, build what?
Let’s simplify the issues.

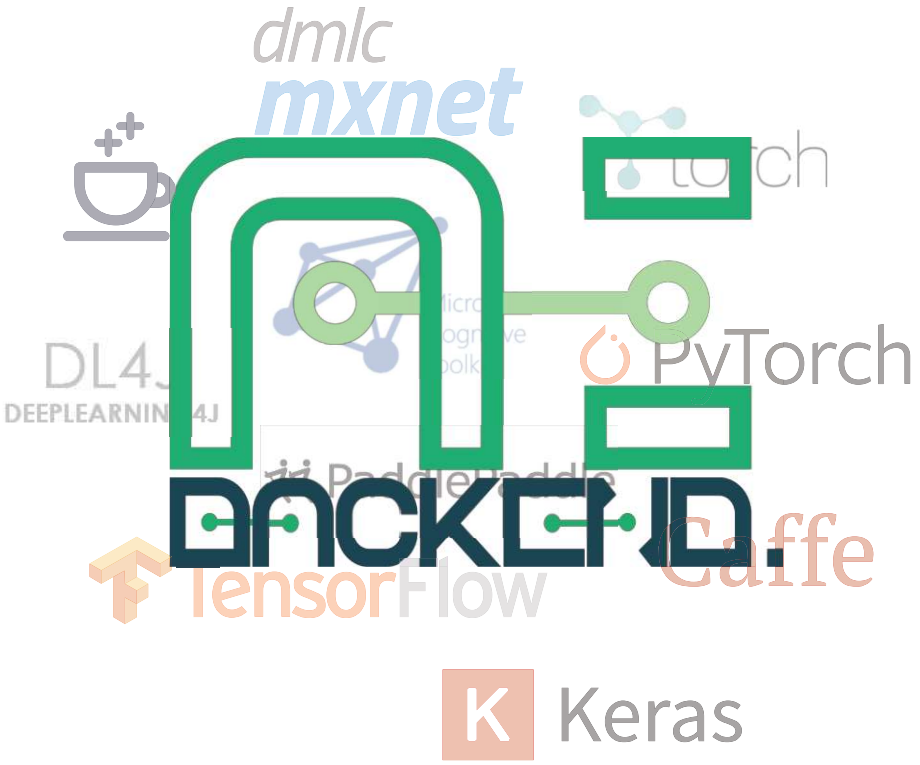
Problem:

1. How to **effectively manage** complex GPU resources?
2. How to **optimally use** various GPUs?
3. How to make it **easy**?



Backend.AI

https://github.com/lablup/backend.ai



Open-source

GPU computation

resource management platform

specialized in AI development

Provides:

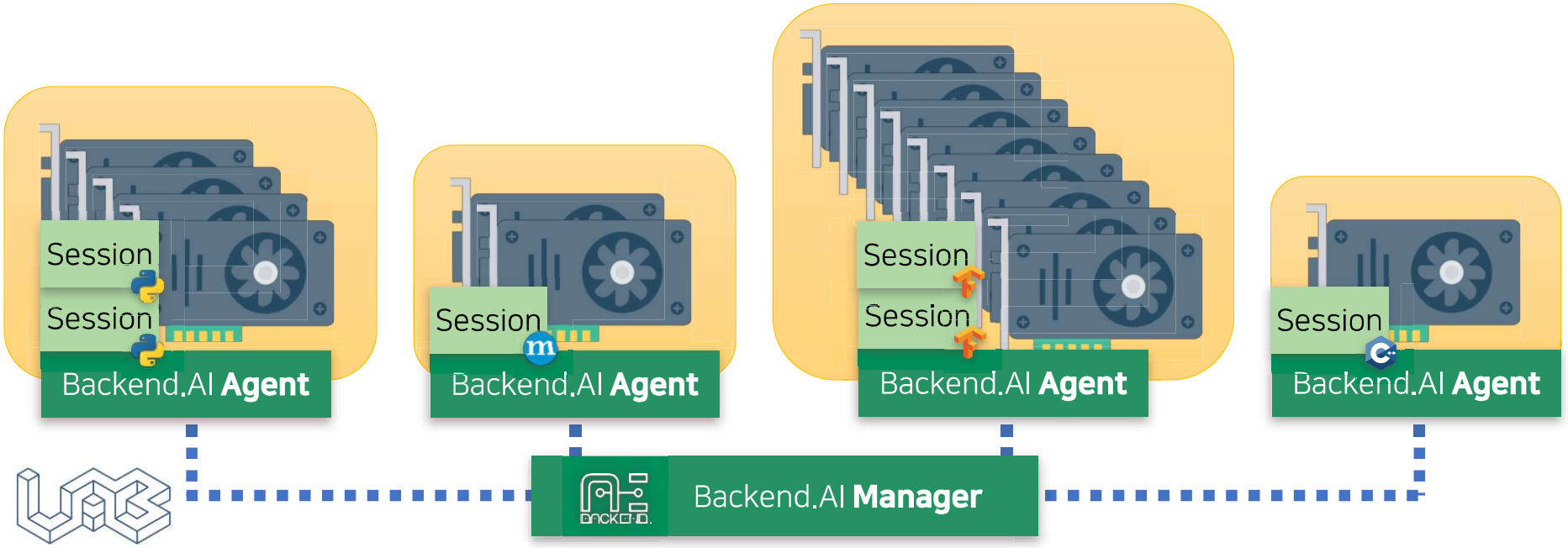
Fractional GPU resource scaling / sharing

Virtualizing GPUs at CUDA level

Good manageability and high utilization

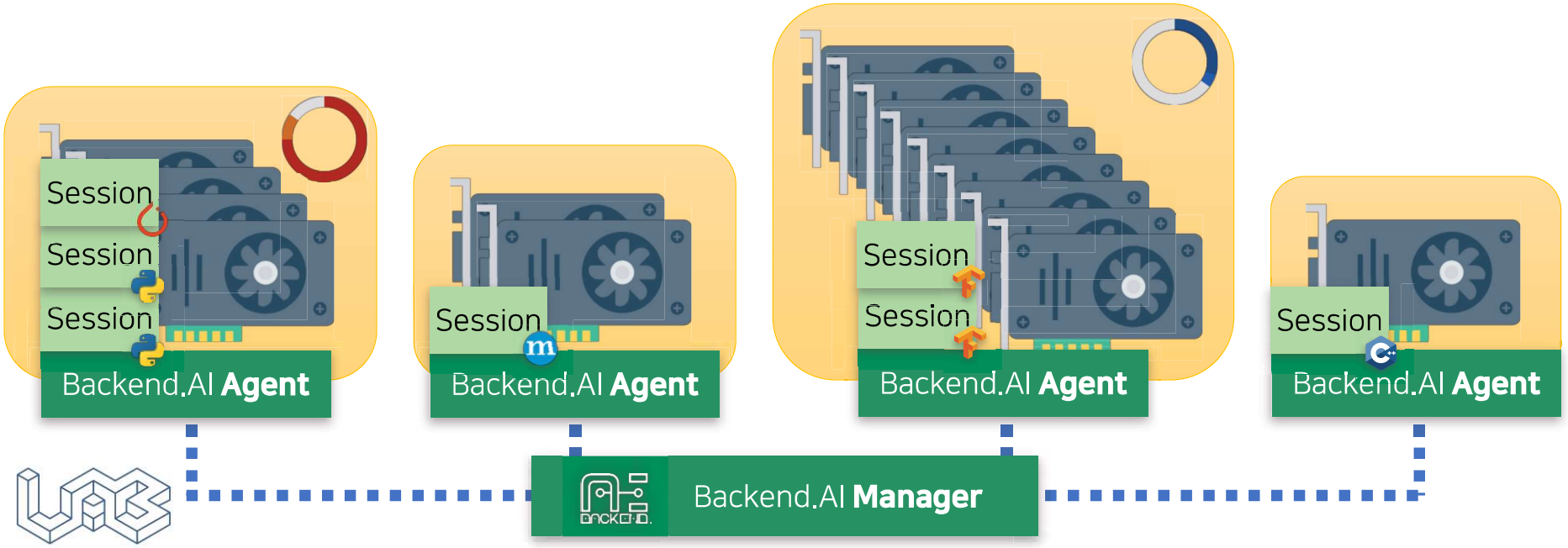
Problem:

1. How to **effectively manage** complex GPU resources?
2. How to **optimally use** various GPUs?
3. How to make it **easy**?



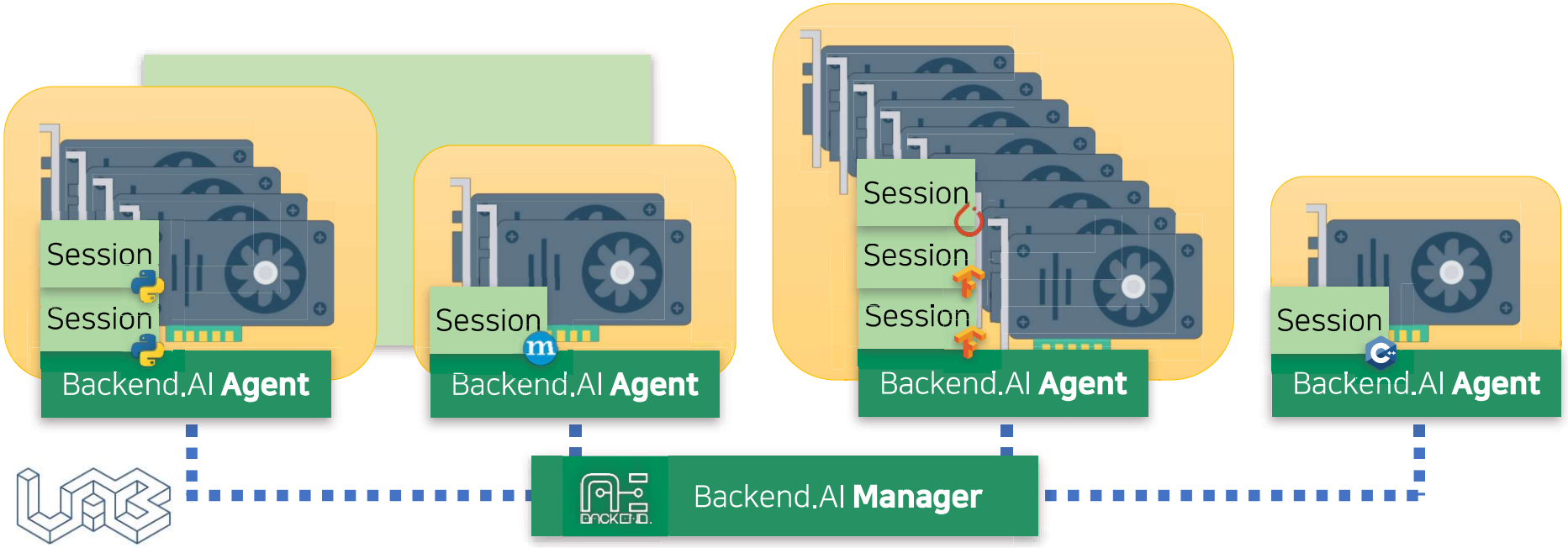
Problem:

1. How to **effectively manage** complex GPU resources?
2. How to **optimally use** various GPUs?
3. How to make it **easy**?



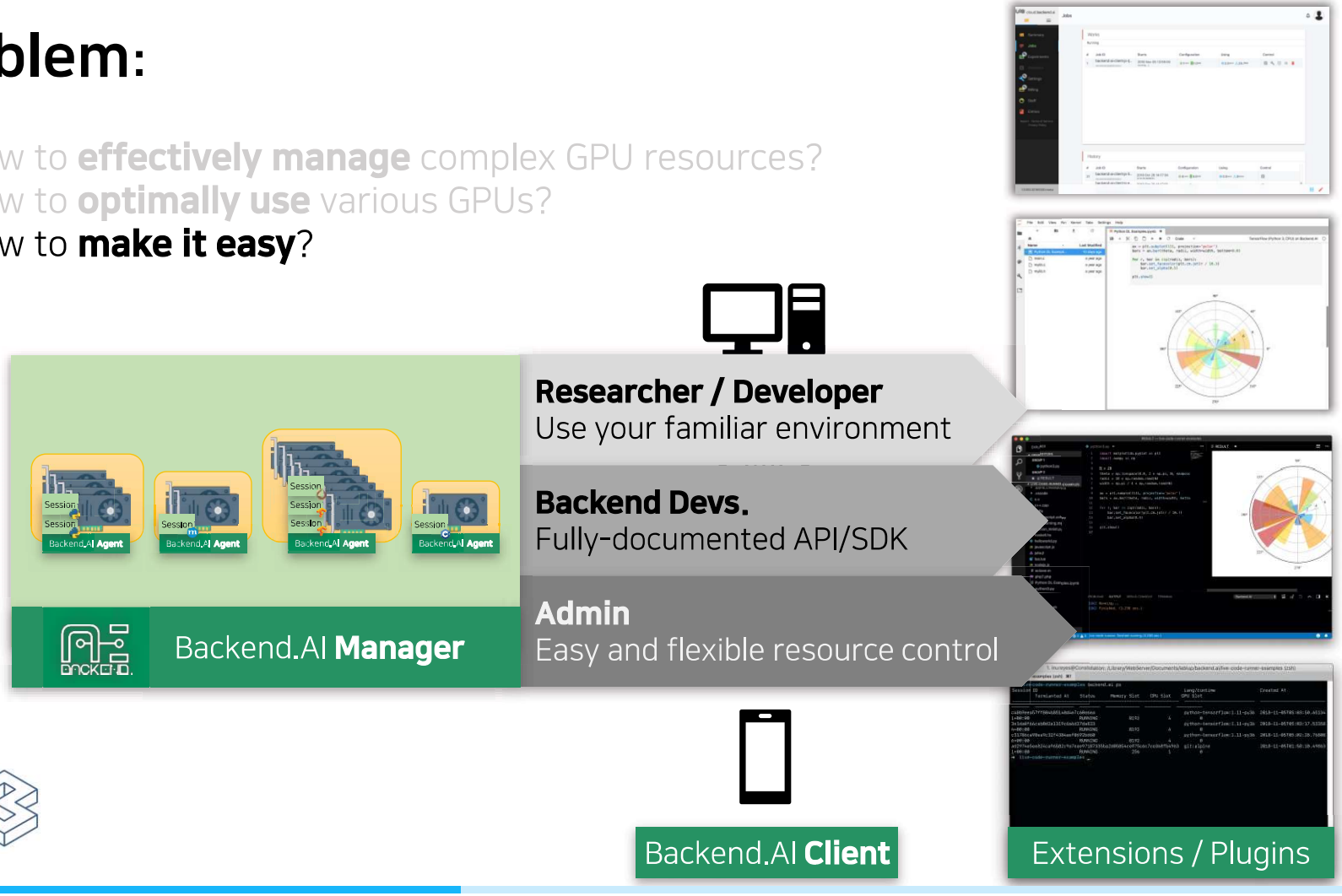
Problem:

1. How to **effectively manage** complex GPU resources?
2. How to **optimally use** various GPUs?
3. How to make it **easy**?

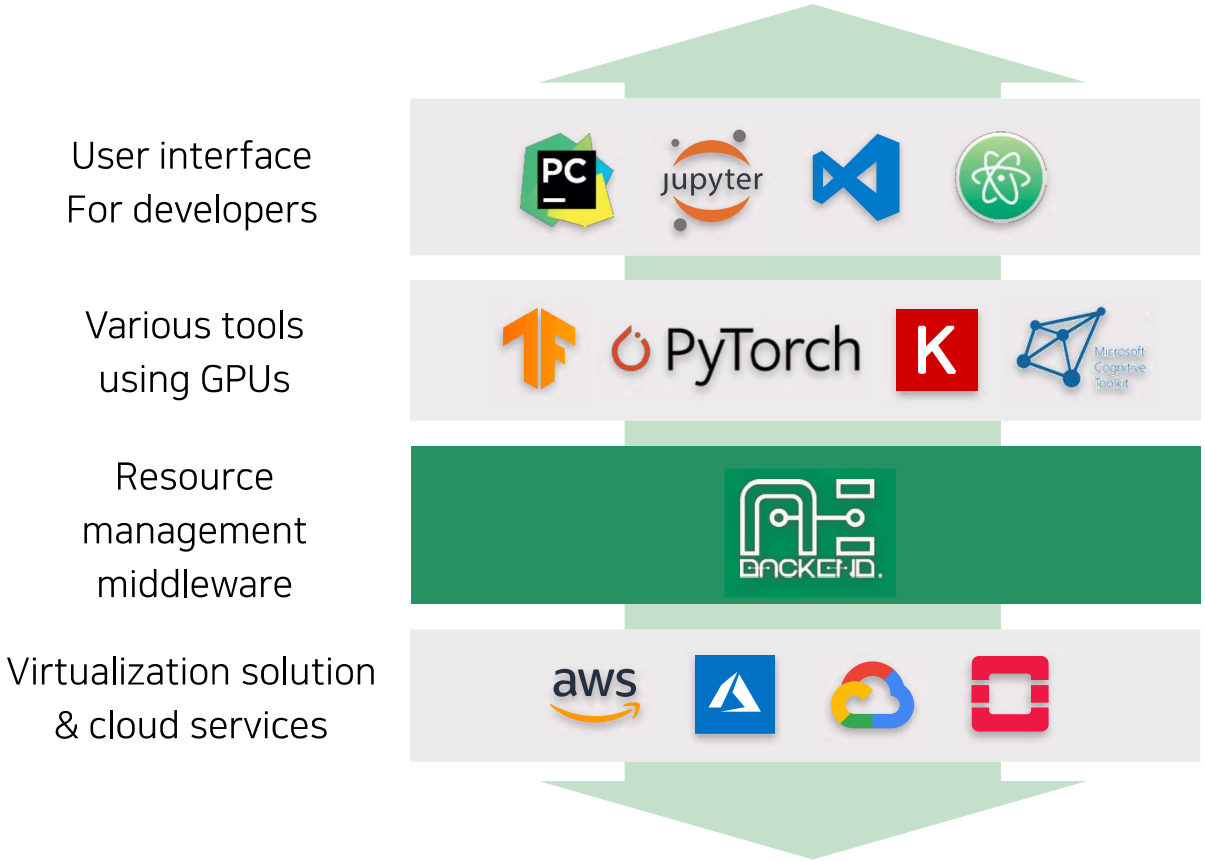


Problem:

1. How to **effectively manage** complex GPU resources?
2. How to **optimally use** various GPUs?
3. How to **make it easy**?



Backend.AI: Characteristics



Goals

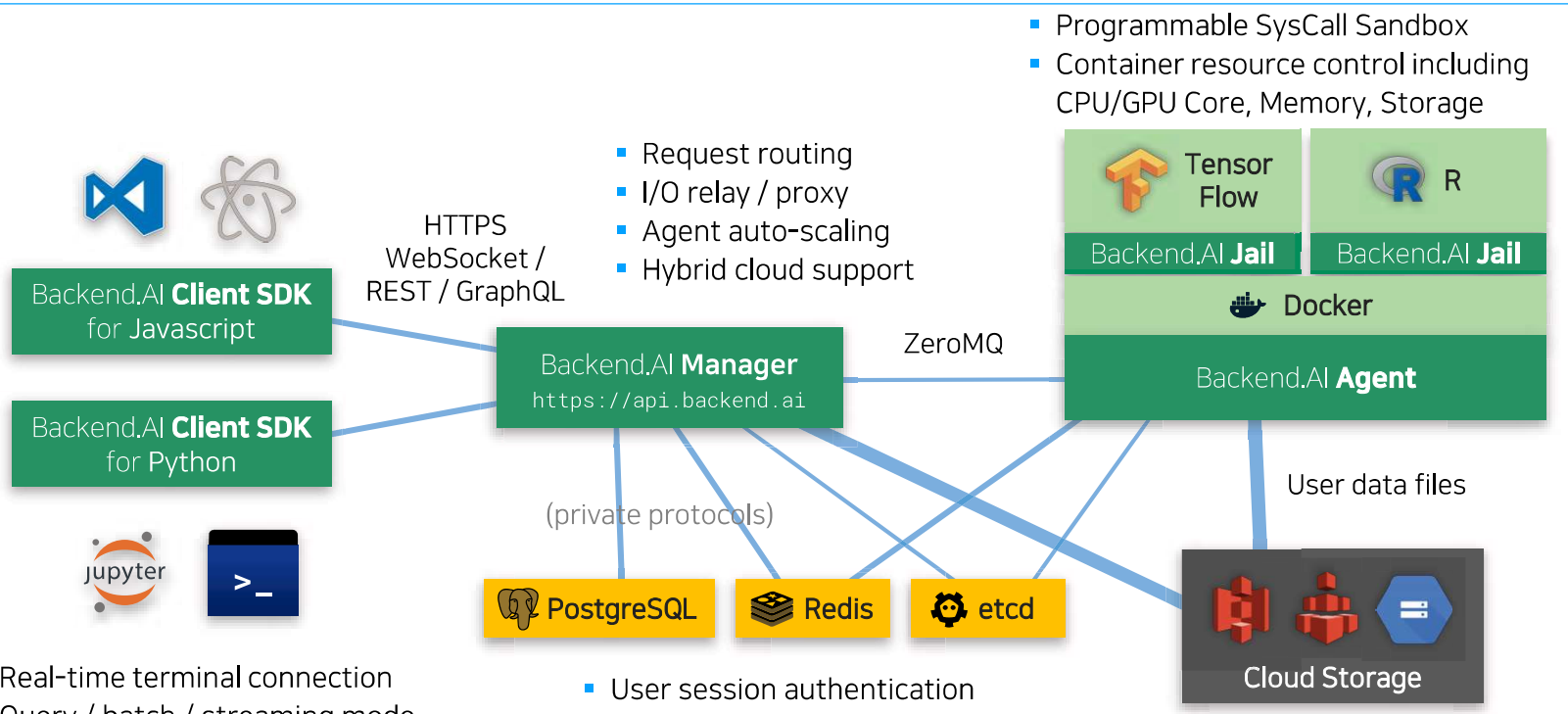
Easy **Together**

Fast **Everywhere**

Efficient

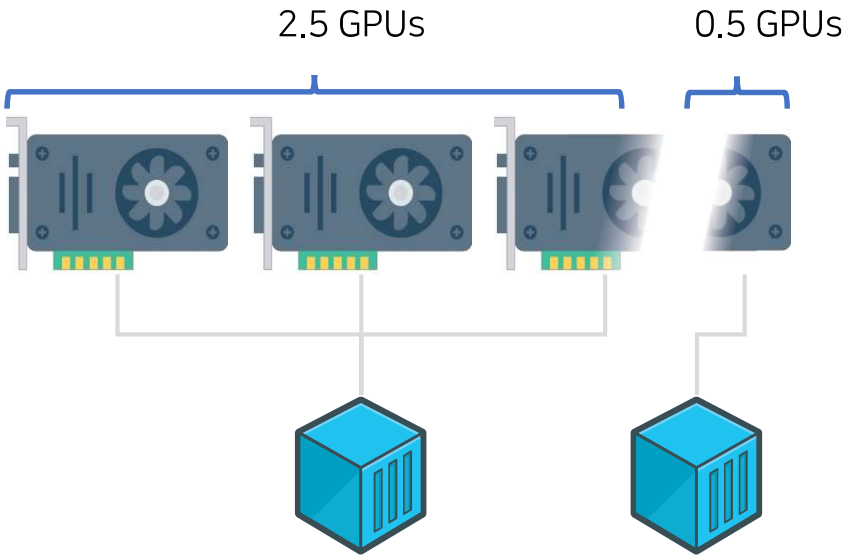
- High-density resource sharing by containers
- Programmable sandboxing by Backend.AI Jail
- Autoscaling & job scheduling
- GPUs as the first-class resource type
- Prebuilt and user-written container images
- High-fidelity resource usage tracking
- User-friendly integrations with Jupyter Notebook, Visual Studio Code, Atom and CLI/IDE
- Cloud service + open-source for on-premise

Backend.AI: Detail



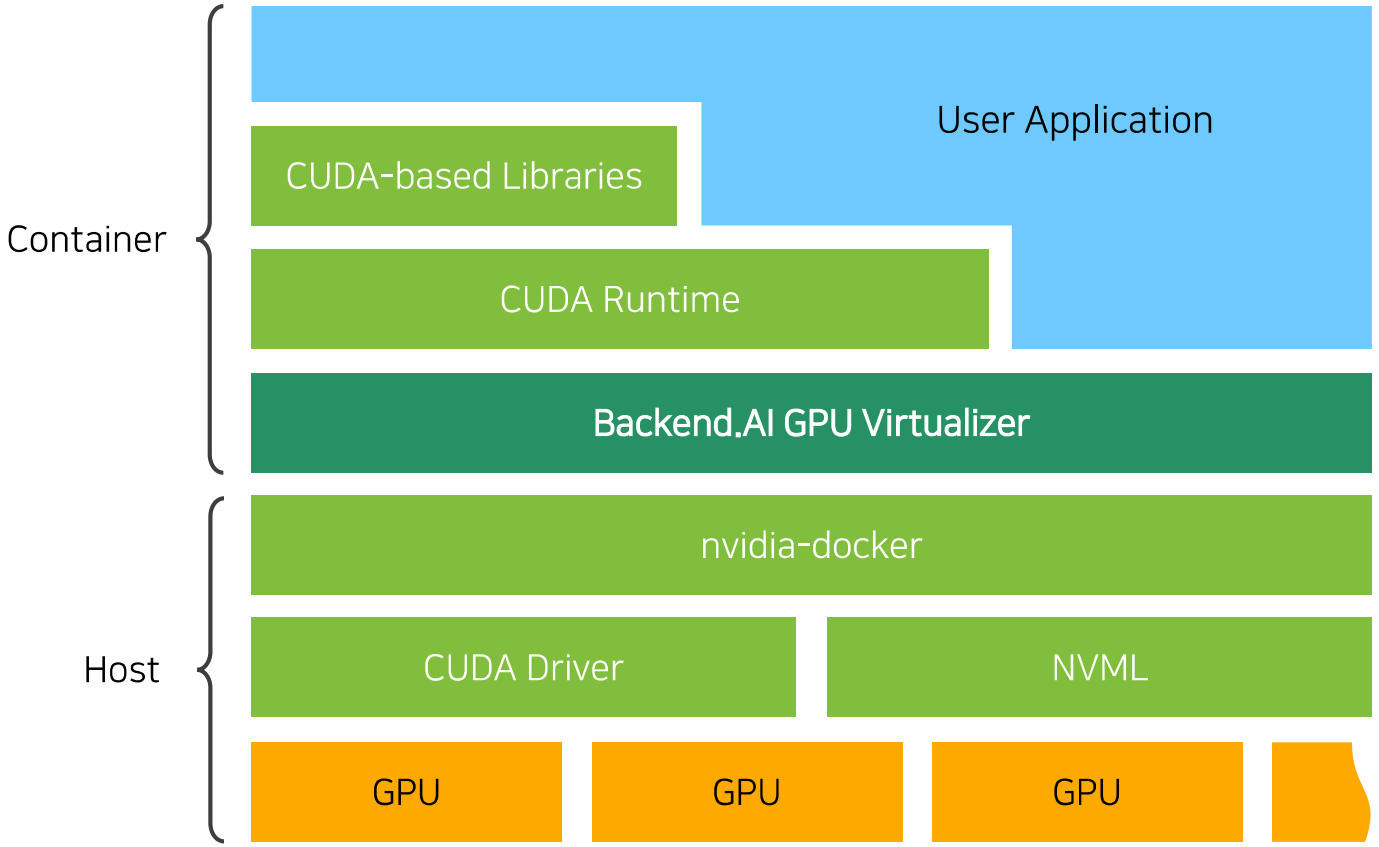
Backend.AI: GPU Features

- Container-level Fractional GPU scaling
  - Assign slices of SMP / RAM to containers
    - ✓ e.g.) can give **2.5 GPUs, 0.3 GPUs**
  - Shared GPUs : inference & education workloads
  - Multiple GPUs : model training workloads
- Virtual Folder sharing
  - Invite other users with various permissions
- GPU Plugin architecture
  - NVLink-optimized
- NVIDIA Platform integration
  - Supports NGC (for DL / HPC) integration
- Unified scheduling / monitoring
  - Console / GUI administration
  - Jupyter, Visual Studio Code, IntelliJ, Atom extensions/ plugins



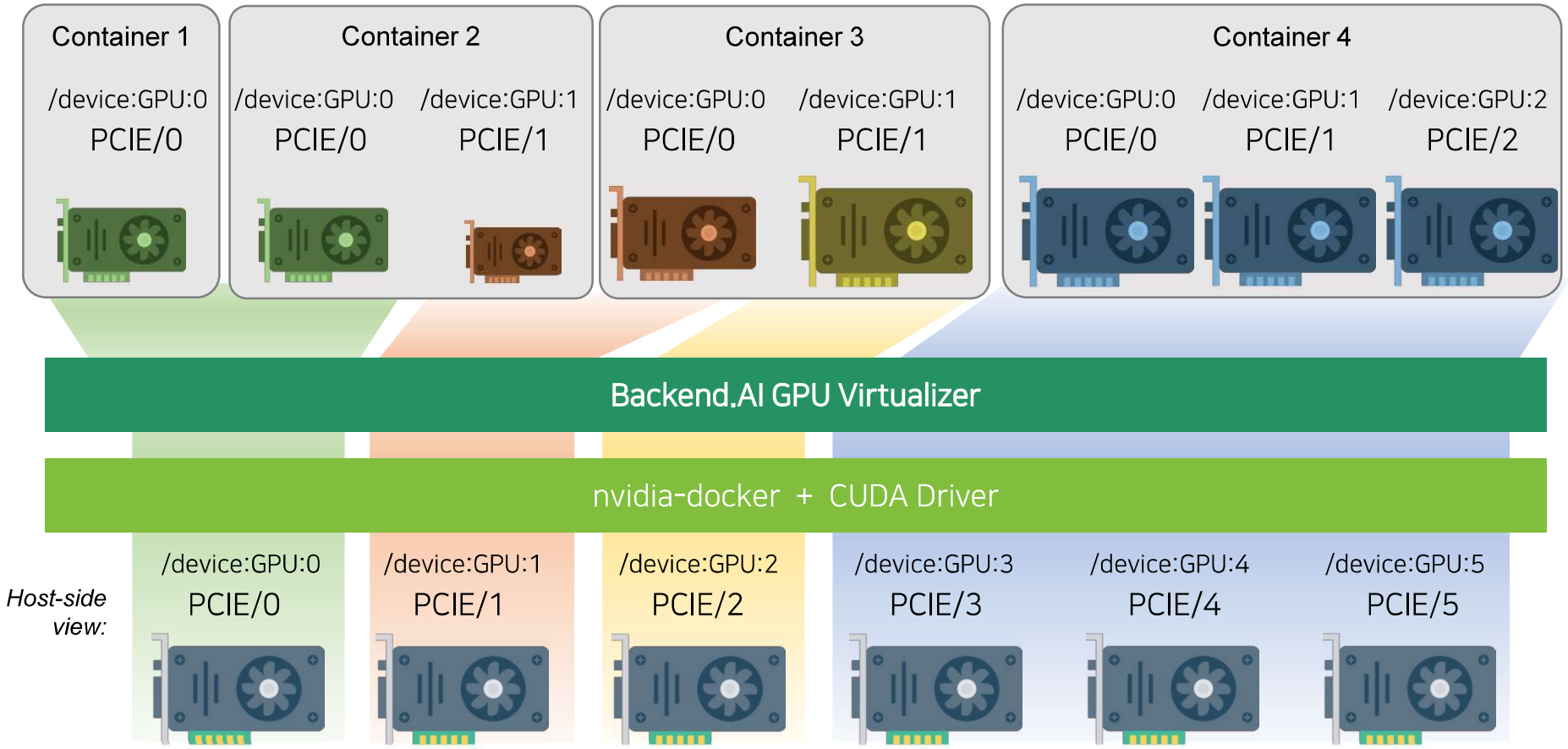
Example of GPU sharing / allocation
(2.5 / 0.5 slots)

CUDA API Virtualization



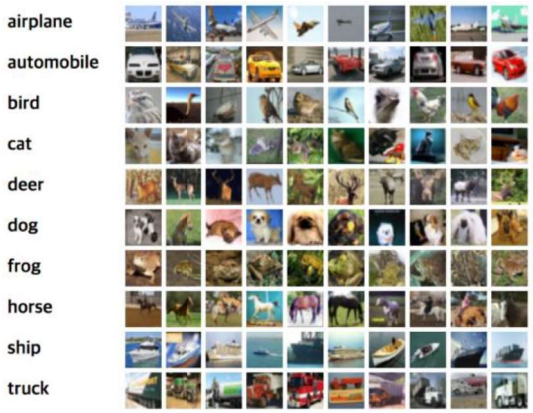
- ✓ Takes all benefits of nvidia-docker
- ✓ Requires no user code changes
- ✓ Supports all NGC containers and user-written CUDA apps
- ✓ Enforces per-container GPU resource limits

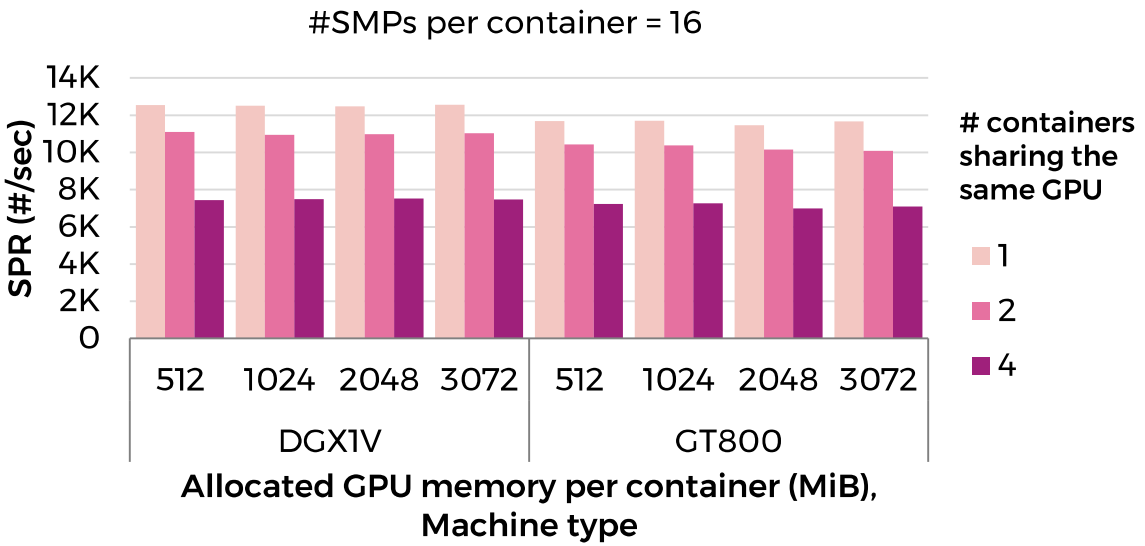
Fractional & Multi-GPU Scaling



Preliminary Performance Evaluation

- Benchmark: Sample processing rate of cifar-10 on a V100 GPU (16/32GB)

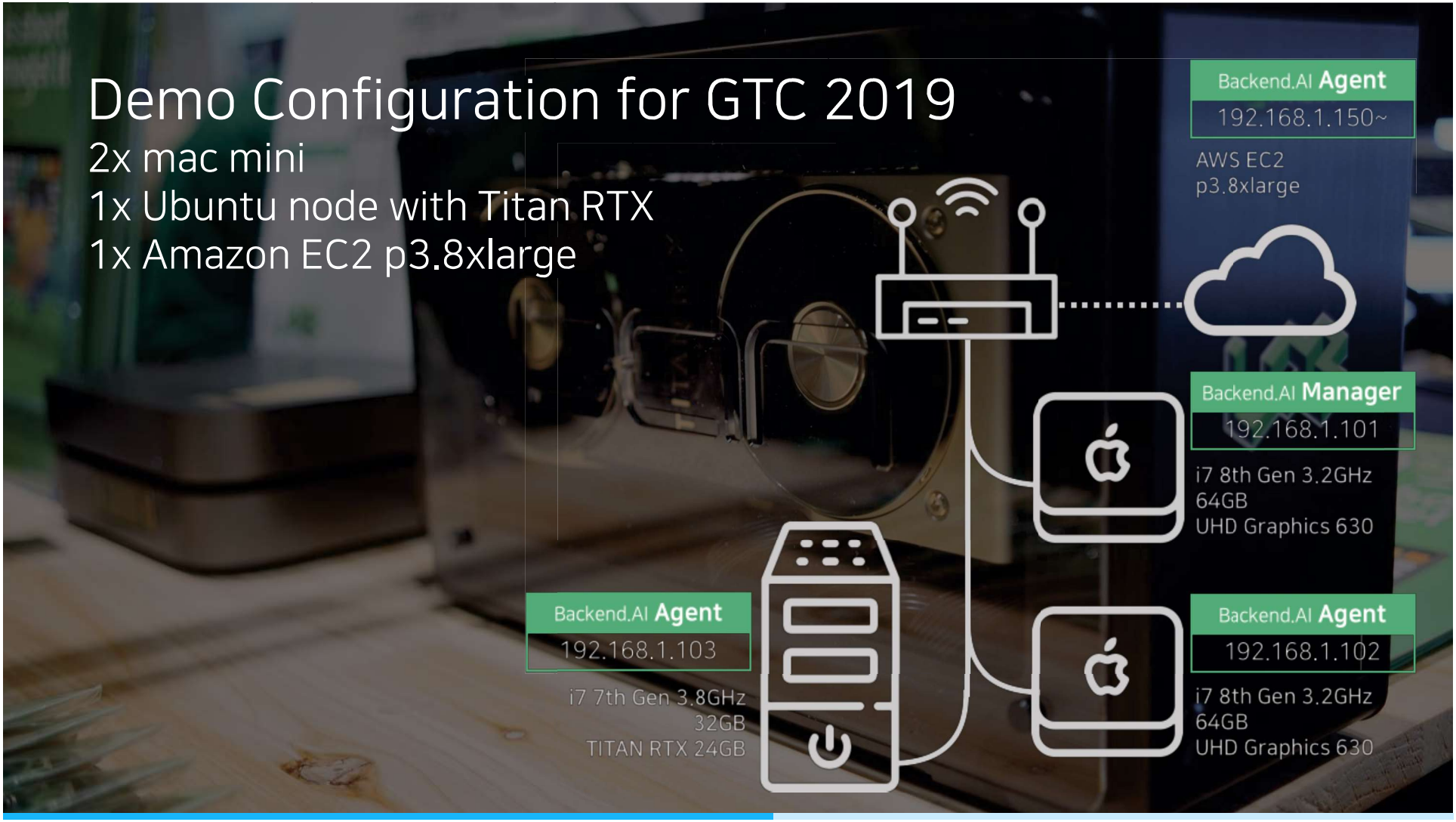


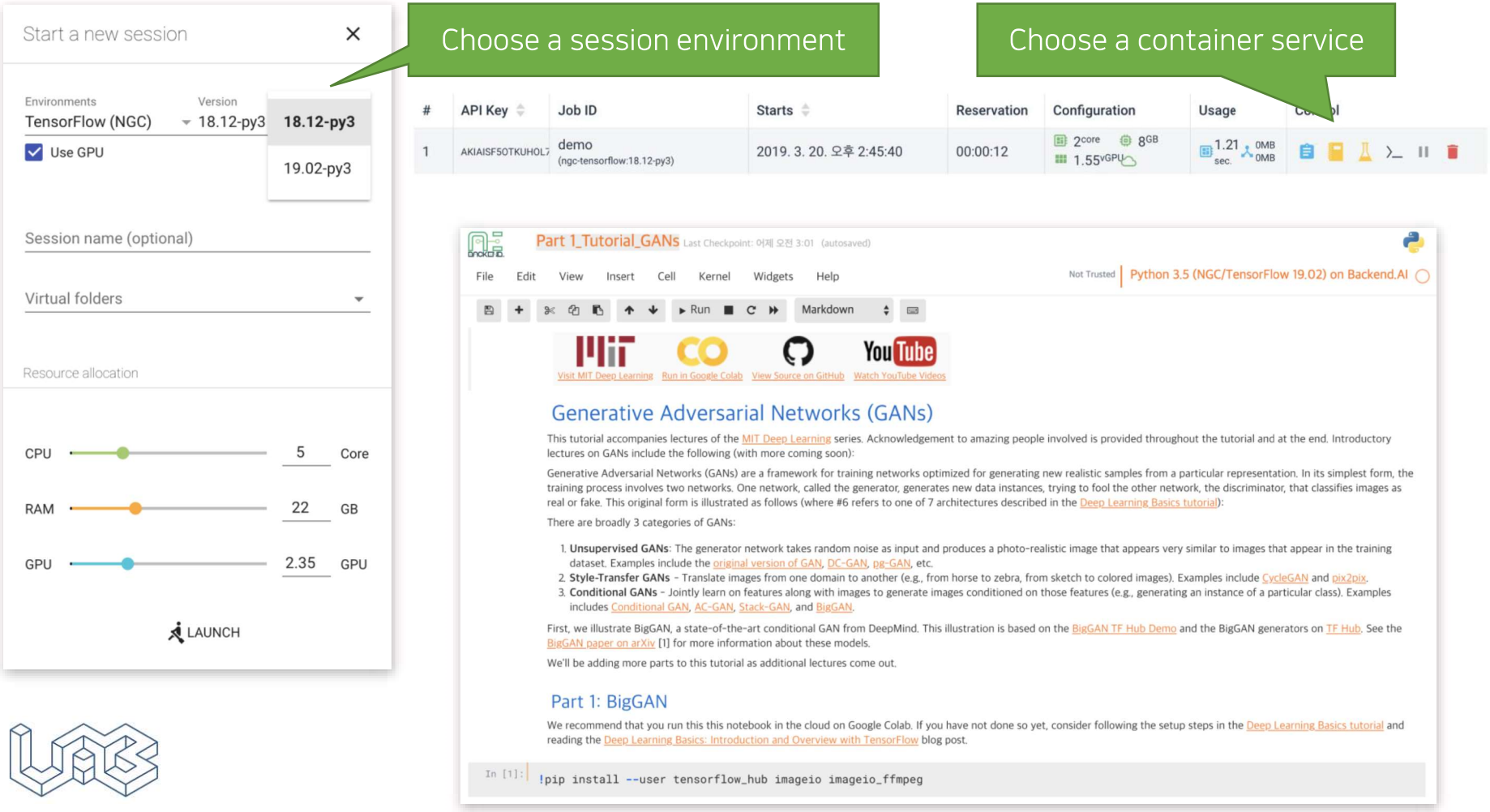


- Results
  - Sharing overhead : **-10% SPR** when a container is added to share the same GPU

Demo Configuration for GTC 2019

2x mac mini
1x Ubuntu node with Titan RTX
1x Amazon EC2 p3.8xlarge




Start a new session
Environments
TensorFlow (NGC)
Version
18.12-py3
18.12-py3
19.02-py3
Use GPU
Session name (optional)
Virtual folders
Resource allocation
CPU
5
Core
RAM
22
GB
GPU
2.35
GPU
LAUNCH
Choose a session environment
Choose a container service
API Key Job ID Starts Reservation Configuration Usage
1 AKIAISF5OTKUHOL7 demo (ngc-tensorflow:18.12-py3) 2019. 3. 20. 오후 2:45:40 00:00:12 2core 8GB 1.55vGPU 1.21 sec. 0MB 0MB
Part 1_Tutorial_GANs Last Checkpoint: 어제 오전 3:01 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted Python 3.5 (NGC/TensorFlow 19.02) on Backend.AI
Run Markdown
Visit MIT Deep Learning Run in Google Colab View Source on GitHub Watch YouTube Videos
Generative Adversarial Networks (GANs)
This tutorial accompanies lectures of the MIT Deep Learning series. Acknowledgement to amazing people involved is provided throughout the tutorial and at the end. Introductory lectures on GANs include the following (with more coming soon):
Generative Adversarial Networks (GANs) are a framework for training networks optimized for generating new realistic samples from a particular representation. In its simplest form, the training process involves two networks. One network, called the generator, generates new data instances, trying to fool the other network, the discriminator, that classifies images as real or fake. This original form is illustrated as follows (where #6 refers to one of 7 architectures described in the Deep Learning Basics tutorial):
There are broadly 3 categories of GANs:
1. Unsupervised GANs: The generator network takes random noise as input and produces a photo-realistic image that appears very similar to images that appear in the training dataset. Examples include the original version of GAN, DC-GAN, pg-GAN, etc.
2. Style-Transfer GANs - Translate images from one domain to another (e.g., from horse to zebra, from sketch to colored images). Examples include CycleGAN and pix2pix.
3. Conditional GANs - Jointly learn on features along with images to generate images conditioned on those features (e.g., generating an instance of a particular class). Examples includes Conditional GAN, AC-GAN, Stack-GAN, and BigGAN.
First, we illustrate BigGAN, a state-of-the-art conditional GAN from DeepMind. This illustration is based on the BigGAN TF Hub Demo and the BigGAN generators on TF Hub. See the BigGAN paper on arXiv [1] for more information about these models.
We'll be adding more parts to this tutorial as additional lectures come out.
Part 1: BigGAN
We recommend that you run this this notebook in the cloud on Google Colab. If you have not done so yet, consider following the setup steps in the Deep Learning Basics tutorial and reading the Deep Learning Basics: Introduction and Overview with TensorFlow blog post.
In [1]:
!pip install --user tensorflow_hub imageio imageio_ffmpeg

Live Demo

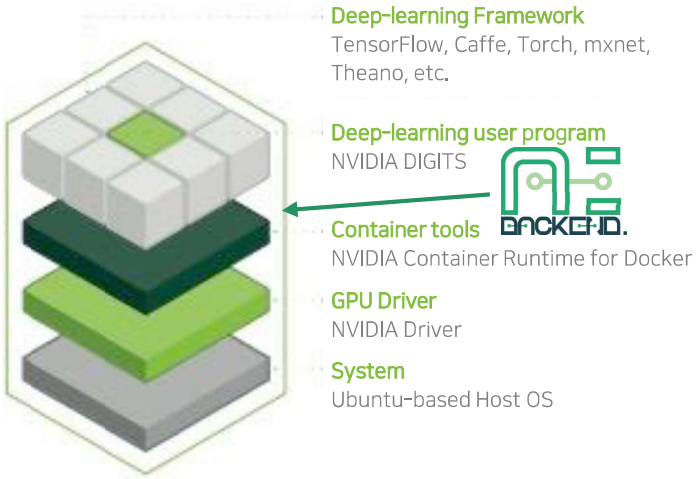

Let's
Scale and test it!

Dive into DGX

If we're going to do this,
let's use it for good.

NVIDIA DGX series

- NVIDIA DGX-1/DGX-2
  - Complete multi-GPU environment system
    - ✓ Ubuntu-based Host OS
    - ✓ NV Link / NV Switch
    - ✓ Great testbed for various load tests!
- Backend.AI on DGX-family
  - Complements NVIDIA Container Runtime
    - ✓ GPU sharing
    - ✓ Scheduling
    - ✓ Pipelining
    - ✓ Technical discussions via NVIDIA Inception Program



SYSTEM SPECIFICATIONS	
GPUs	**16X NVIDIA® Tesla V100**
GPU Memory	**512GB total**
Performance	**2 petaFLOPS**
NVIDIA CUDA® Cores	**81920**
NVIDIA Tensor Cores	**10240**
NVSwitches	**12**
Maximum Power Usage	**10 kW**
CPU	**Dual Intel Xeon Platinum 8168, 2.7 GHz, 24-cores**
System Memory	**1.5TB**

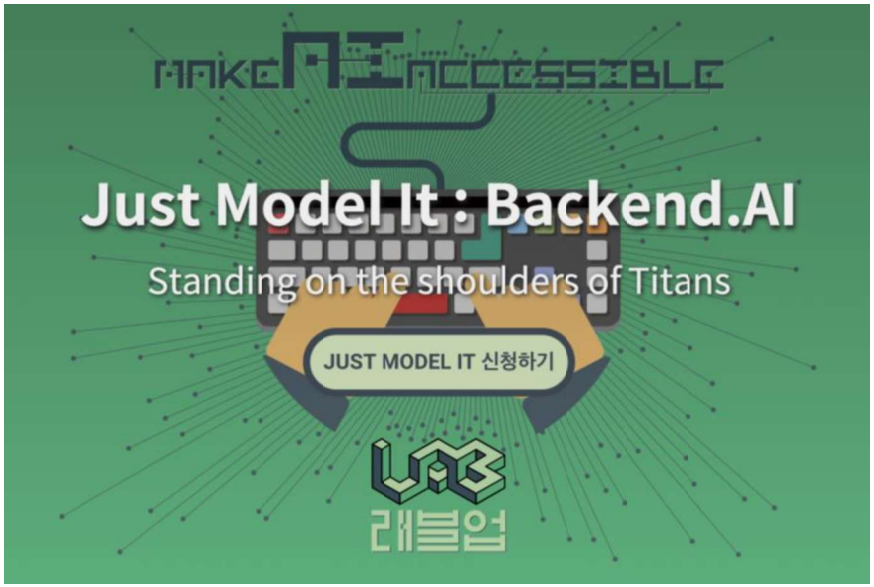
NGC Integration

- NGC: Optimal software stack for CUDA cluster
  - Up-to-date libraries, toolkits and frameworks
- WHY NGC containers?
  - Who can manage CUDA container images better than NVIDIA?
    - ✓ Ever-increasing complexity: NCCL, TensorRT, CUDA 10.1, RAPIDS, ...
  - DGX integration: keep up-to-date NVIDIA software stack for users
- NGC image support on Backend.AI
  - TensorFlow
  - DIGITS
  - PyTorch
  - Chainer

  } Every version since NGC 18.12
  - Others will be ready soon!

Just Model It Contest

- "Standing on the Shoulders of Titans"
- Jan.~Mar. 2019
  - https://events.backend.ai/just-model-it/
  - Provides GPU resources to ML scientists / developers for free!
  - **For us:** system validation & tests
  - **For participants:** chances to creating machine learning models without huddle



- How
  - Setup an virtual Backend.AI GPU cluster with many remote GPU servers / Cloud instances
  - Provide resources via Backend.AI client CLI / GUI app

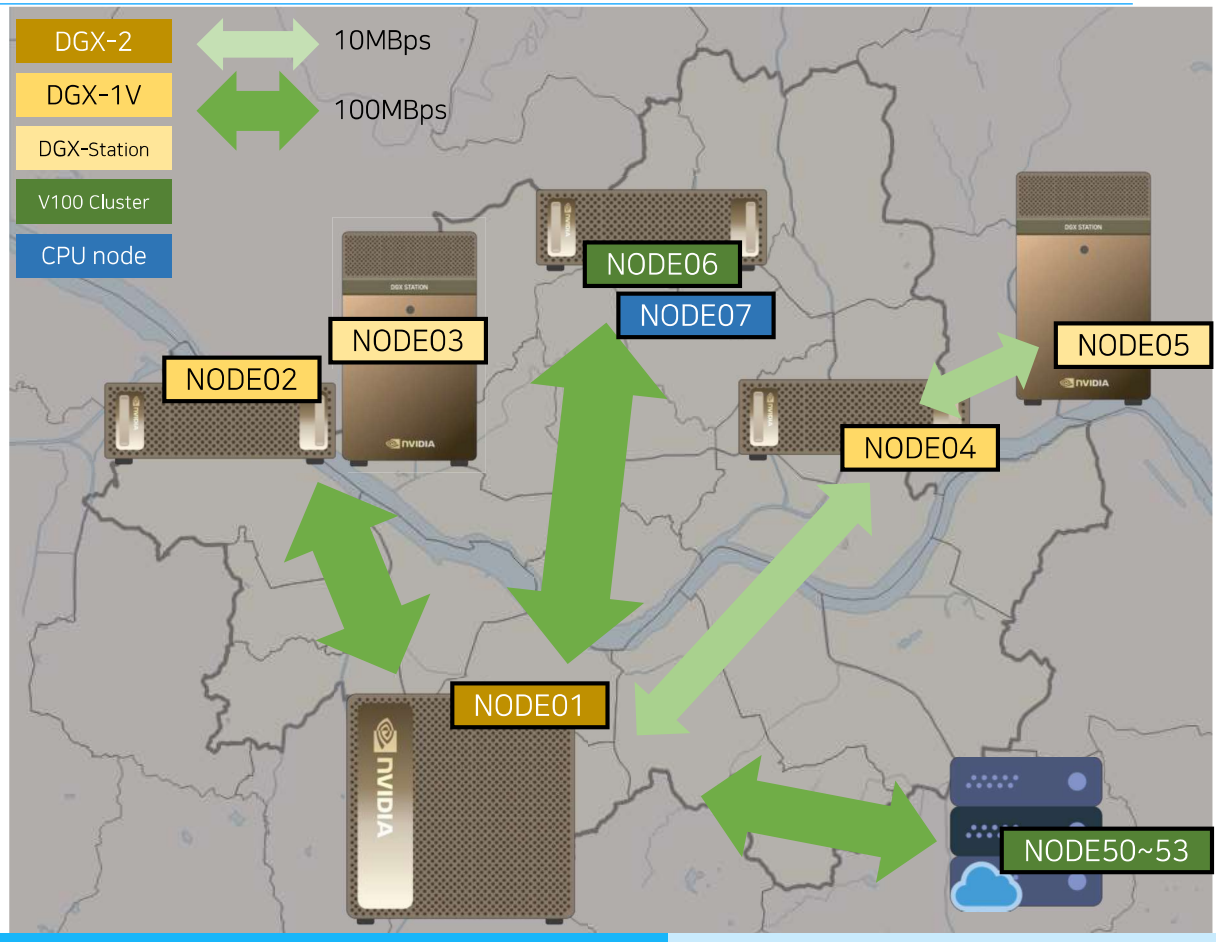
Creating virtual Backend.AI cloud with DGX series

- On-premise cluster for *Just model it* event
- 44 V100 on-premise GPUs + (8~32) V100 GPU instance on cloud
  - (16) 1 DGX-2 server NODE01
  - (4) 1 custom GPU server (with 4 V100 GPUs) NODE06
  - (16) 2 DGX-1V **(with support by Nvidia)** NODE02, NODE04
  - (8) 2 DGX Stations **(with support by Nvidia)** NODE03, NODE05
  - (8~32) Amazon EC2 instances (p3-8xlarge) as spot instances NODE50~NODE53
  - + CPU-only on-premise node (44-core Xeon) for compile / data preprocessing NODE07

- 4 geographically distant locations
  - DGX-2 + Custom GPU server (Lablup Inc.)
  - DGX-1V+DGX stations (Baynex , Local Nvidia Partner)
  - DGX-1V+DGX stations (Daebo, Local Nvidia Partner)
  - Amazon EC2 (ap-northeast-2)

Creating virtual Backend.AI cloud with DGX series

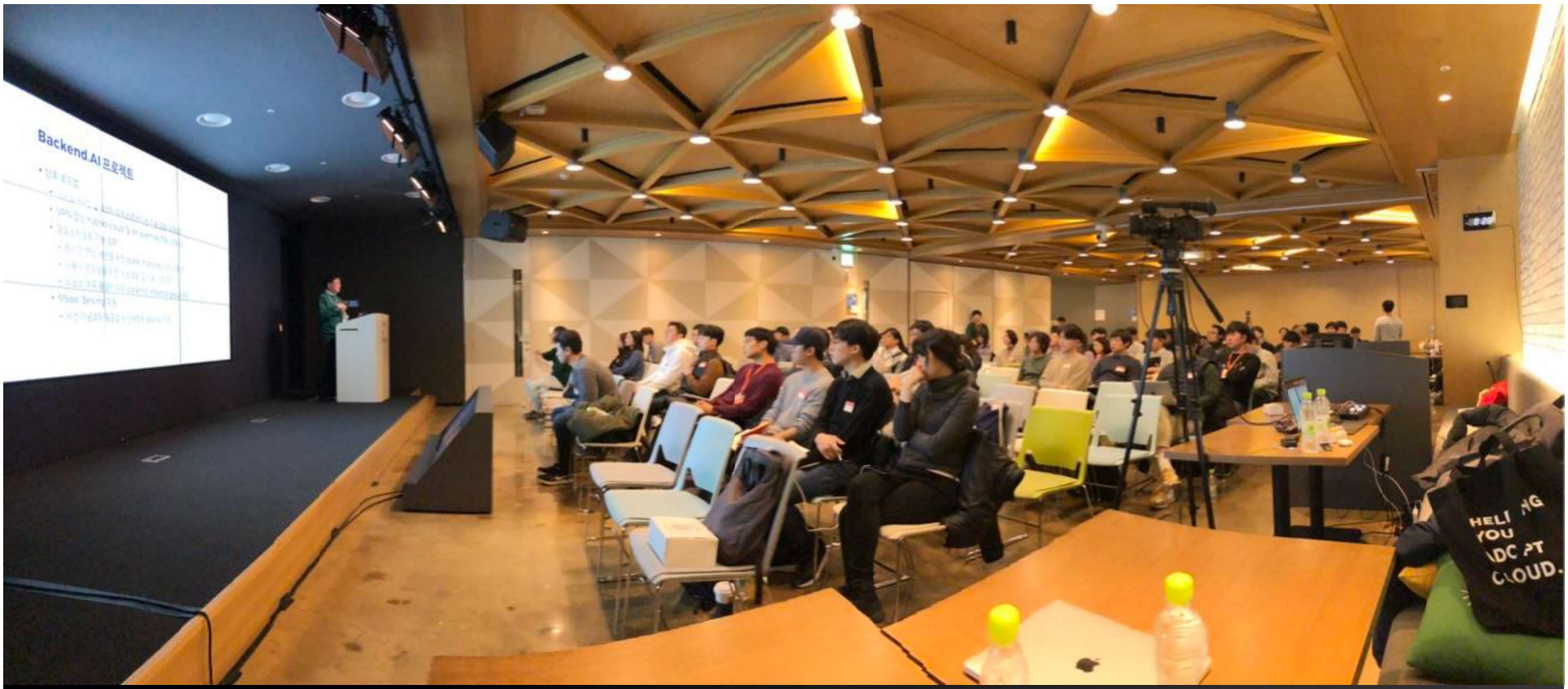
- Agent roles
  - NODE01
    - ✓ Backend.AI manager
  - NODE01~05
    - ✓ Active GPU Cluster
  - NODE06
    - ✓ Reserved / Staging area
  - NODE07
    - ✓ Image compilation / Julia
  - NODE50~53
    - ✓ Spot Instance on AWS
- Storage configuration
  - Scratch disk on each agent
  - Cachefilesd to each node
  - ~~RedHat Ceph Storage as Distributed Storage~~
    - ✓ Disabled due to the limited traffic bandwidth



Configurations

- 12 independent teams
  - Research teams / Independent developer / Startups

- Resource allocation (for each team)
  - CPU: 22 Cores (various clock, followed by host CPU)
  - RAM: 512GB
  - Storage: 3TB scratch (8 NVMe RAID-0) + ☒
  - GPU:64GB (**32x2** or **16x4** V100s)
    - ✓ 32x2: Text workloads (RNN / BERT projects)
    - ✓ 16x4: Image / video workloads (CNN / GAN projects)
    - ✓ Multi-GPU scaling mode



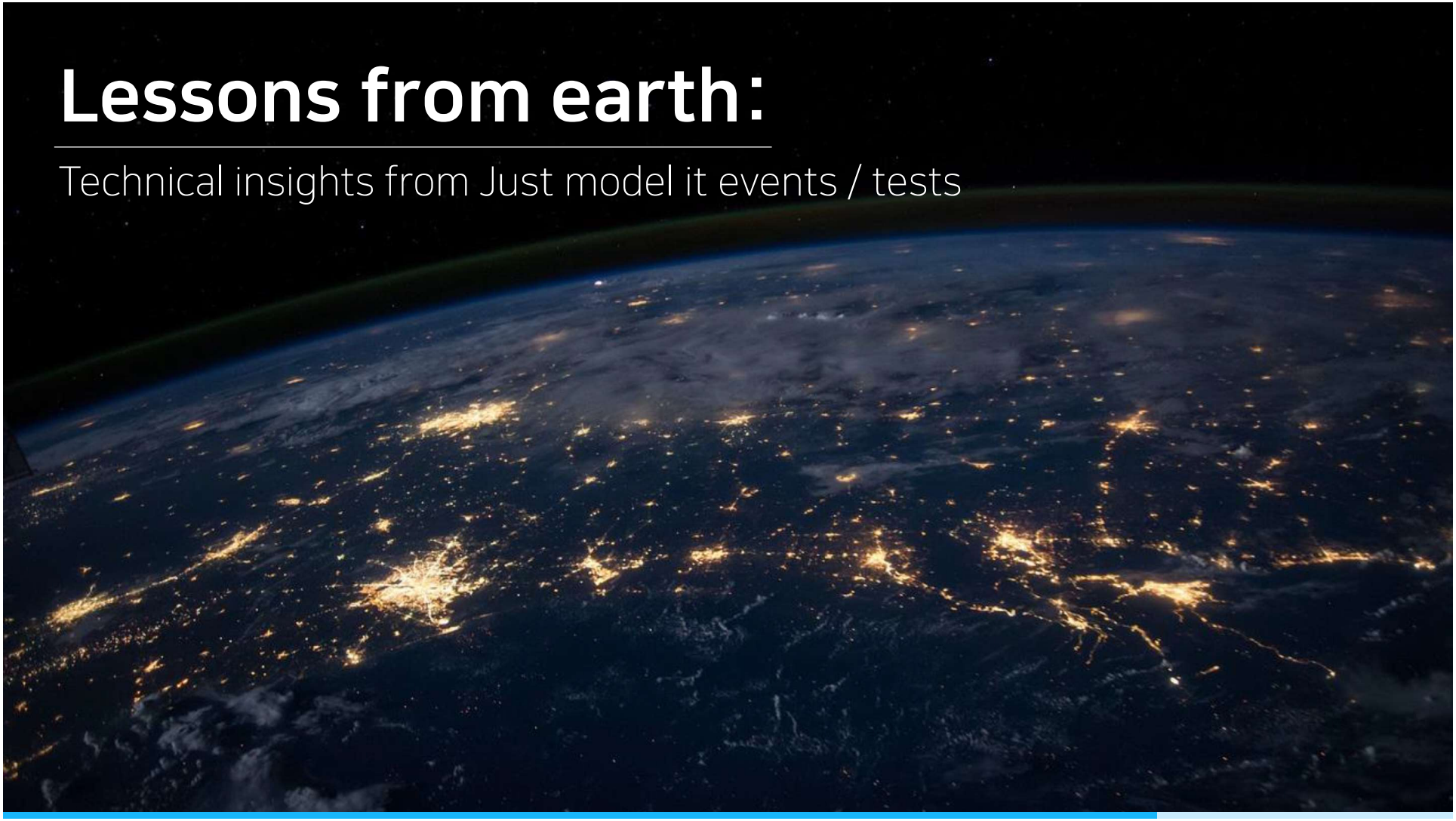
And Event Begins:

AI Tech Talk
21 Jan. 2019, Google Startup Campus


And one month passed.

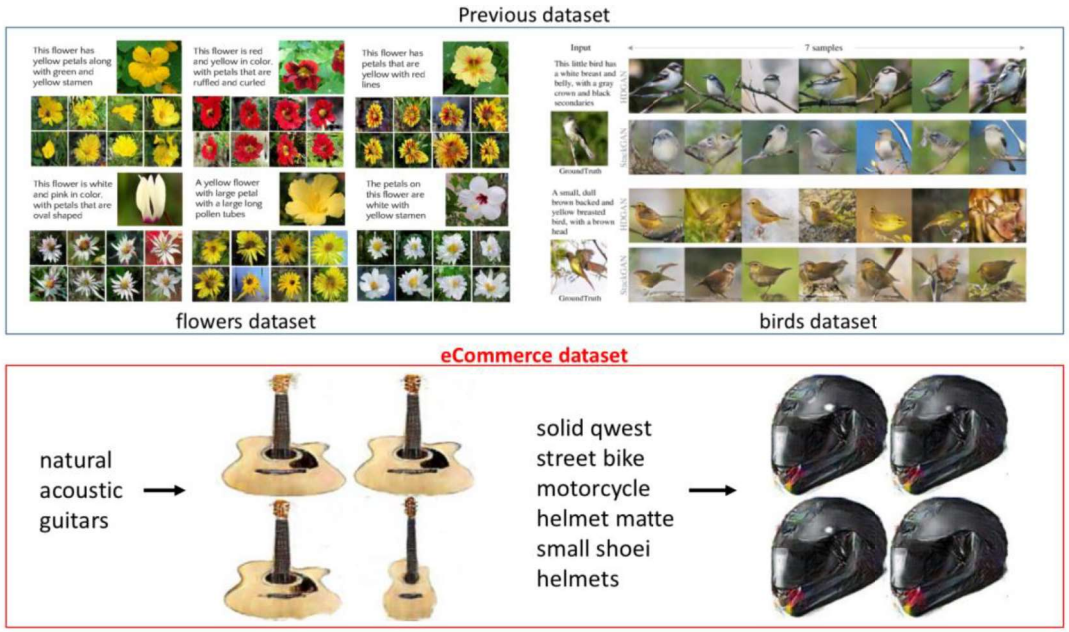
Lessons from earth:

Technical insights from Just model it events / tests



JMI Event Showcase: TAC-GAN-eCommerce

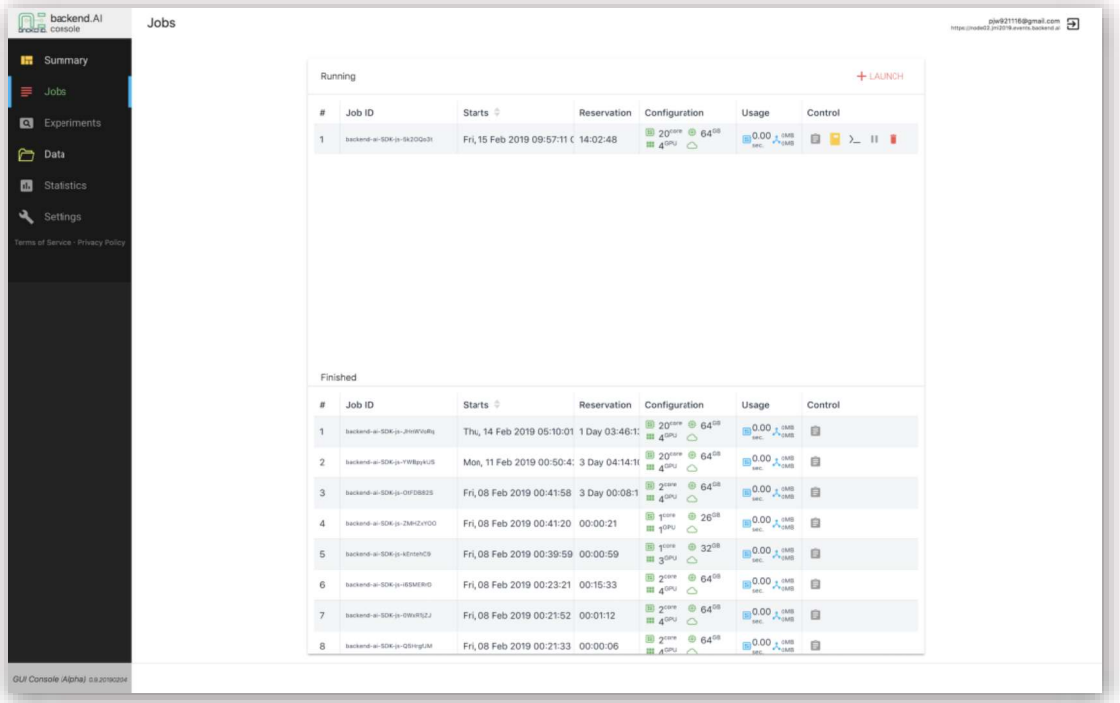
- Problem
  - 1. Missing image for product ad.
  - 2. Promotional text to product images → Generates unrelated meta data
- Solution: text to image synthesis
  - Meta data to product image
  - Prototyping TAC-GAN
  - 1. Creating production image using generator
  - 2. Judge abusing using discriminator



https://github.com/junwoopark92/TAC-GAN-eCommerce

JMI Event Showcase: TAC-GAN-eCommerce

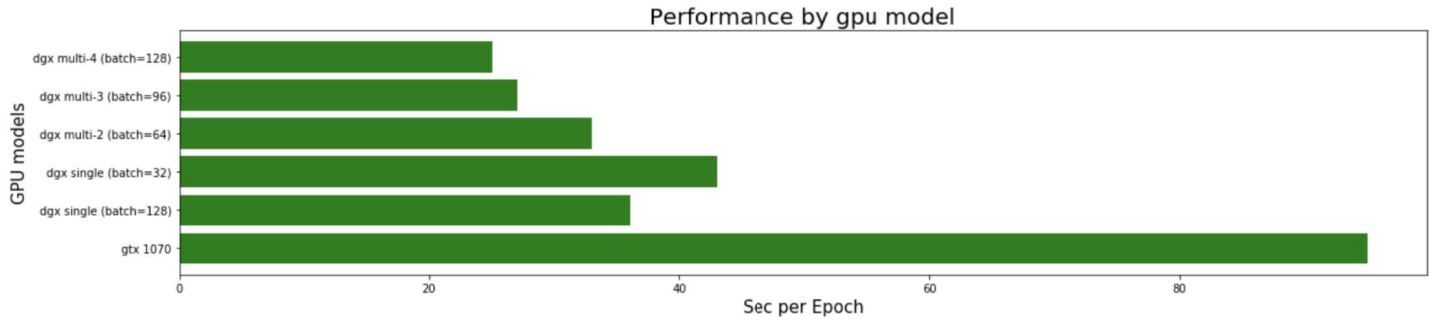
- Data specification
  - Amazon eCommerce Dataset
  - 9M products
  - 16,000 leaf categories
  - 260GB images
- Preprocessing Pipeline
  - Indexing using sntencepiece
  - Sentence embedding with doc2vec in genism
  - Data augmentation with label shuffling



https://github.com/junwoopark92/TAC-GAN-eCommerce

Event results: Benchmark (TAC-GAN-eCommerce)

- 1070 vs Tesla V100 16GB single (batch size = 128):
  - ~3X performance difference.
  - Adjusted the batch size until there was no performance degradation due to I/O.
  - Average load: 90~100 (1070), 80~90 (V100)
- Tesla V100 16GB (single ~ 4, batch size = 32 ~ 128)
  - Performance increases as the number of GPUs increases, but not linear.
  - TAC-GAN model size is small: Data feeding seems to be a bottleneck.
  - If the size of the batch is increased beyond a certain size, an error that exceeds the shared area of IO occurs.
  - Load average: Single GPU: 80~90, 4 GPUs: 40~50
    - ✓ May get additional performance as the model size increases.



Lessons

- **Backend.AI offers easier multi-tenancy management, as designed.**
  - nvidia-docker provides a consistent way of using GPUs *inside* containers.
  - Backend.AI provides a flexible and automated way of mapping GPUs *with* containers.
  - When JMI participants destroy/restart containers, they automatically get idle GPUs. Almost no intervention was required during the one-month period.

- **Unobtrusive upgrade is critical to keep long-running computations successful.**
  - We updated Backend.AI to keep containers running even when the manager/agent daemons restart completely.
  - Transparent websocket tunneling for in-container services (e.g., Jupyter) enables seamless reconnection upon Backend.AI upgrades.

Closing

- GPU
- GPU as computational resource
- Issues on GPGPU computation
- Backend.AI
- Multi-tenant deep learning workloads
- Virtual GPU computation cluster with DGX-family
- Just Model It (JMI) event
  - Contributing to ML community with testing Backend.AI
  - Configuration & Characteristics
- Results and Insights from the event
- Closing

Thank you!

If question, ask us via contact@lablup.com !

Also, visit booth **#240** for offline discussion.

Lablup Inc.	https://www.lablup.com
Backend.AI	https://www.backend.ai
Backend.AI GitHub	https://github.com/lablup/backend.ai
Backend.AI Cloud	https://cloud.backend.ai
CodeOnWeb	https://www.codeonweb.com

Credits & References

- Logos and names of companies and products are trademarks of respective owners and organizations.
- Illustrative icons are made by Freepik, Maxim Basinski, Smashicons, Srip from www.flaticon.com