

How Salk Institute is Delivering 'Unlimited' GPU Compute to Their Research Teams

Salk centralized data science infrastructure while preserving data scientists' freedom and flexibility to use their preferred tools

Institution

The Salk Institute for Biological Studies is one of the world's preeminent research institutions, where scientists are using artificial intelligence to make groundbreaking contributions in cancer and aging research, Alzheimer's, diabetes, and cardiovascular disorders.

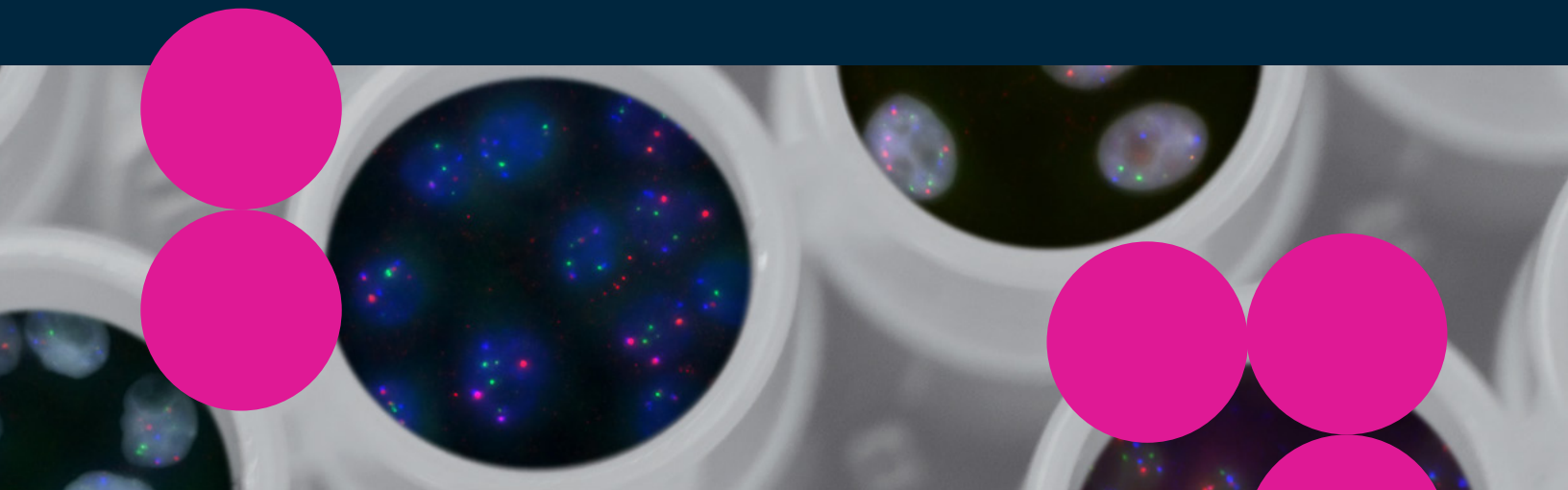
Challenge

The Salk Institute came to Run:ai with many of the same challenges that universities and research institutions share when it comes to efficient allocation of GPU resources. Siloed teams often purchase their own hardware, leaving IT without visibility into compute utilization or control of these independent systems. As teams begin to scale, the use of static quotas makes it difficult to manage researchers' variable compute demands, leading to frustrated researchers and limited ROI on the hardware. Anticipating these growing pains, Salk sought to centralize all of their GPU compute, for visibility into workloads and utilization metrics while ensuring fair allocation between all researchers.

Solution – Advanced Scheduling With Run:ai Atlas

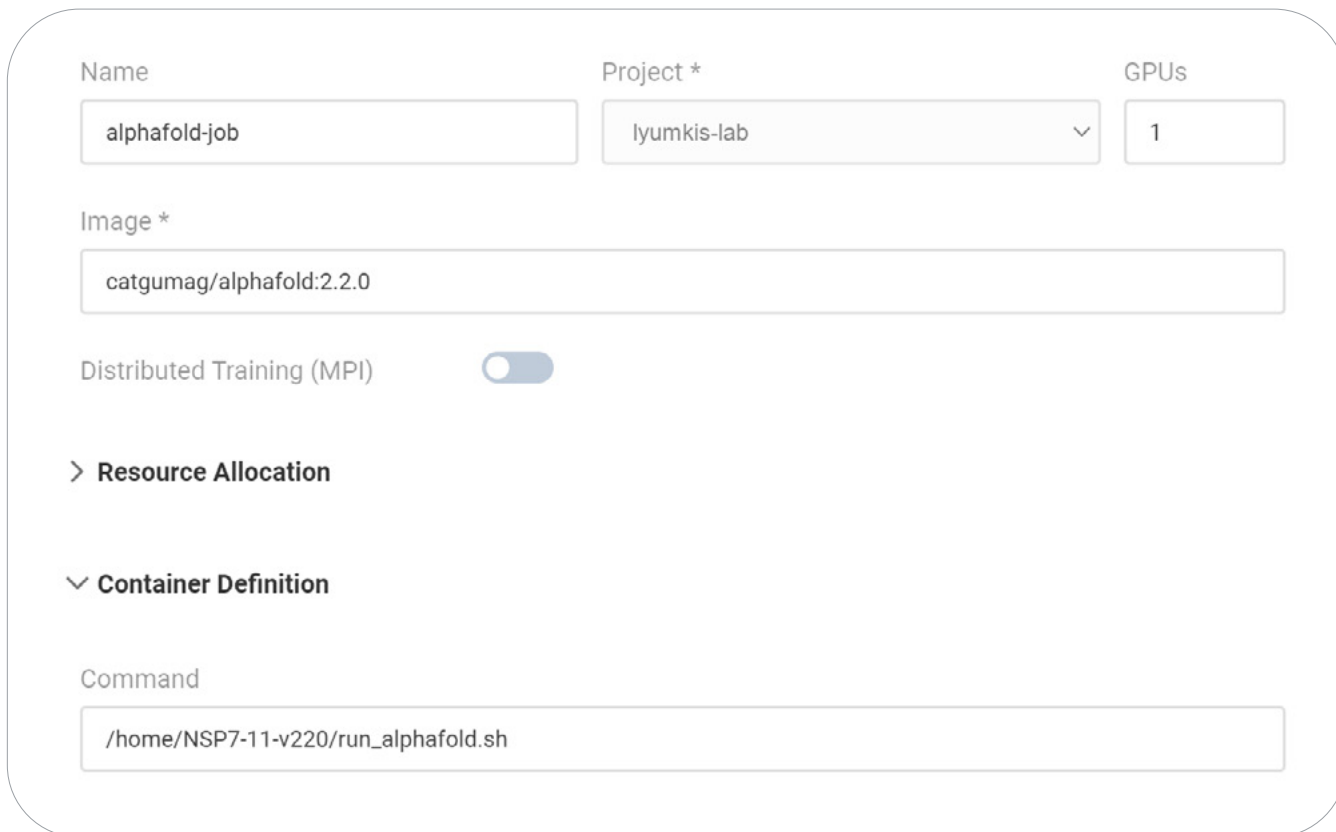
Run:ai Atlas unifies underlying compute hardware and infrastructure to drive changes that benefit IT as well as data scientists and researchers. Atlas uses Kubernetes as its backbone to effectively orchestrate pooled GPU compute. By pooling GPU resources, teams of data scientists and researchers unlock significantly more compute power to run their workloads, limiting the time that GPUs sit idle.

To ensure fair access to GPU resources, Run:ai leveraged a scheduling concept called 'projects' to provide guaranteed quotas to the research teams. Salk was able to create multiple projects, enabling multiple different lab groups at the university to all have GPU access. In this way, the lab groups always have preferential access to their guaranteed quota of GPUs, and they can also burst beyond their quota and utilize additional GPUs that are sitting idle within the cluster. Run:ai Atlas also has capabilities around Node Affinity to allow for context-based scheduling, specifying that certain workloads run on specific GPU types, or affording preferential access to teams that have contributed their hardware to the cluster to run their workloads on their own hardware.



With enterprise-ready features such as single sign on (SSO), Run:ai Atlas helps IT quickly onboard and provision GPU resources for new researchers. The platform also delivers historical metrics which help IT understand the performance of the cluster over time, giving actionable data for capacity planning or building a business case for purchase of additional hardware.

Finally, the extensibility of the platform unlocks the potential for many workloads to run in parallel on a single cluster. Researchers at Salk are able to quickly spin up and launch GPU-accelerated Jupyter notebooks which creates a simple development environment for their AI models. Additionally, Run:ai enables Salk to leverage advanced research tools including Cryosparc and Alphafold with no integration hassles, as seen in the images below.



The image shows a web form for submitting a job. It includes the following fields and controls:

- Name:** A text input field containing "alphafold-job".
- Project *:** A dropdown menu with "lyumkis-lab" selected.
- GPUs:** A text input field containing "1".
- Image *:** A text input field containing "catgumag/alphafold:2.2.0".
- Distributed Training (MPI):** A toggle switch that is currently turned off.
- Resource Allocation:** A section header with a right-pointing chevron.
- Container Definition:** A section header with a downward-pointing chevron.
- Command:** A text input field containing the bash script: `/home/NSP7-11-v220/run_alphafold.sh`.

Pictured: Salk's Jupyter notebook for Alphafold and the Run:ai job submission form. They use the Jupyter notebook to create a bash script with all of the parameters that interest them.

```

* [19]: #Define options as a variable in python environment
#This example will run the monomer preset with the full_dbs option
options = ""
--fasta_paths=/home/RORg-v220/sequence.fasta \
--output_dir=/home/RORg-v220/ \
--model_preset=multimer \
--use_gpu_relax=true \
--run_relax=true \
--data_dir=/home/dbs \
--db_preset=full_dbs \
--max_template_date=2021-12-31 \
--bfd_database_path=/home/dbs/bfd/bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt \
--mgnify_database_path=/home/dbs/mgnify/mgy_clusters_2018_12.fa \
--obsolete_pdbs_path=/home/dbs/pdb_mmcif/obsolete.dat \
--uniprot_database_path=/home/dbs/uniprot/uniprot.fasta \
--uniref90_database_path=/home/dbs/uniref90/uniref90.fasta \
--uniclust30_database_path=/home/dbs/uniclust30/uniclust30_2018_08/uniclust30_2018_08 \
--template_mmcif_dir=/home/dbs/pdb_mmcif/mmcif_files \
--pdb_seqres_database_path=/home/dbs/pdb_seqres/pdb_seqres.txt

...

[21]: #create the bash script file and open it with the python console
!touch run_alphafold.sh
ofile = open("run_alphafold.sh", "w")
#write in the file to add the options after the python script (.py)
ofile.write("#!/bin/bash" + "\n" + "ldconfig" + "\n" + "python /app/alphafold/run_alphafold.py " + options)
ofile.close()

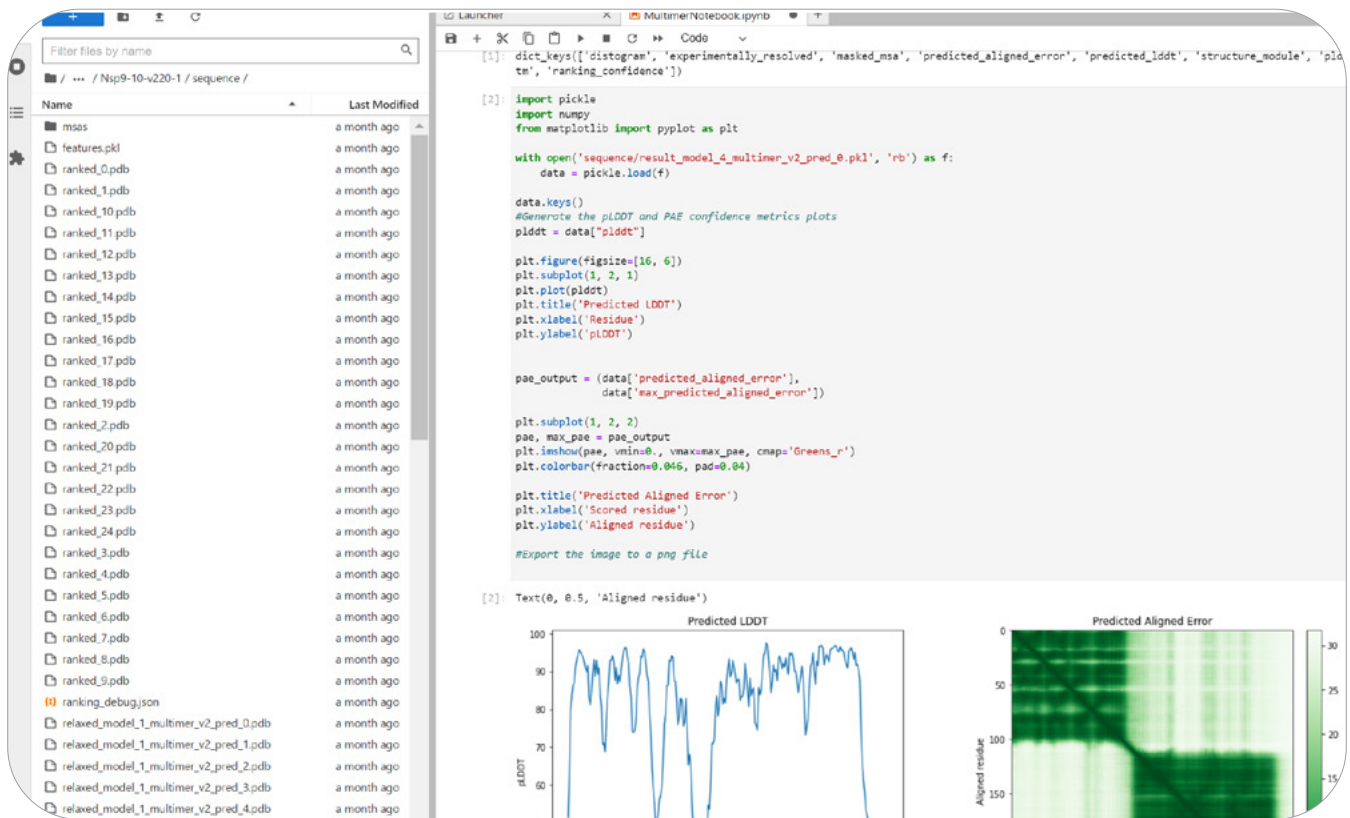
#add execute permission to the script so that alphafold will be allowed to use it.
!chmod +x run_alphafold.sh
#read back the bash script to make sure that it looks correct
!cat run_alphafold.sh

#!/bin/bash
ldconfig
python /app/alphafold/run_alphafold.py --fasta_paths=/home/RORg-v220/sequence.fasta --output_dir=/home/RORg-v220/ --model_preset=multimer --use_gpu_relax=true --run_relax=false --data_dir=/home/dbs --db_preset=full_dbs --max_template_date=2021-12-31 --bfd_database_path=/home/dbs/bfd/bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt --mgnify_database_path=/home/dbs/mgnify/mgy_clusters_2018_12.fa --obsolete_pdbs_path=/home/dbs/pdb_mmcif/obsolete.dat --uniprot_database_path=/home/dbs/uniprot/uniprot.fasta --uniref90_database_path=/home/dbs/uniref90/uniref90.fasta --uniclust30_database_path=/home/dbs/uniclust30/uniclust30_2018_08/uniclust30_2018_08 --template_mmcif_dir=/home/dbs/pdb_mmcif/mmcif_files --pdb_seqres_database_path=/home/dbs/pdb_seqres/pdb_seqres.txt

```

Pictured: Once the script is created, it's easy to run the latest deepmind AlphaFold container and pull the proper parameters into the workload.





Pictured: When the run is completed, the team goes back to the Jupyter notebook to generate diagnostic plots and to retrieve the predicted structures. Overall, Salk has found it easy to implement and use Alphafold to analyze many custom sequences. Shown in these example screenshots are coronavirus polyproteins that undergo maturation to drive viral particle formation.



Run:ai is a singularly transformative solution for us. As biomedical researchers, our goal is to make breakthroughs that are key to understanding and curing disease, not developing and maintaining computational infrastructure. Run:ai enables us to harness the power of technologies like deep learning, ensuring that we can continue to innovate through the use of next generation computational tools for uncovering insights hidden in biological data.

- Talmo Pereira, Salk Fellow & Principal Investigator

