

How to Run Tasks with dstack on Vultr

Learn how to efficiently run machine learning and data science tasks with dstack on Vultr cloud. Step-by-step guide for seamless deployment and execution.

Contents

01	Introduction	3
02	Create a Virtual Environment	3
03	Install dstack	4
04	Run a Fine-tuning Task	5
05	Conclusion	8

Introduction

[dstack](#) runs AI and machine learning workloads efficiently. Each task defines a specific job, such as training a model or processing data, and runs in a cloud or on-premise environment. Users can define tasks using a simple configuration file, specifying resources like GPUs, dependencies, and scripts. dstack handles scheduling, execution, and resource allocation, ensuring that tasks run smoothly without manual intervention. This makes it easier for teams to manage AI workloads, scale operations, and optimize performance without dealing with complex infrastructure setups.

In this guide, you'll fine-tune the DeepSeek R1 Distill Qwen 1.5B model using dstack on Vultr Cloud GPUs. dstack automates workload management, leveraging Vultr's high-performance GPUs for efficient training with LoRA optimization and W&B logging.

Create a Virtual Environment

In this section, you are going to create virtual environment on your machine and prepare the environment for the dstack dev environment deployment.

1. Install the `venv` package.

CONSOLE

```
$ apt install python3-venv -y
```

2. Create a virtual environment.

CONSOLE

```
$ python3 -m venv dstack-env
```

3. Activate the virtual environment.

```
CONSOLE
```

```
$ source dstack-env/bin/activate
```

Install dstack

In this section, you are going to install all the necessary dependencies for dstack and activate the dstack server for the dev environment deployment in the later section.

1. Create a directory and navigate into it to store the backend file.

```
CONSOLE
```

```
$ mkdir -p ~/.dstack/server
```

2. Create a backend `yml` file to declare Vultr as the provider.

```
CONSOLE
```

```
$ nano ~/.dstack/server/config.yml
```

3. Copy and paste the below configuration.

```
YAML
```

```
projects:  
- name: main  
  backends:  
  - type: vultr  
    creds:  
      type: api_key  
      api_key: <vultr-account-api-key>
```

[Retrieve your Vultr API Key.](#)

Save and close the file.

4. Install dstack.

CONSOLE

```
$ pip install "dstack[all]" -U
```

5. Activate the dstack server.

CONSOLE

```
$ dstack server
```

Note down the URL on which the dstack server is running and token provided in the output.

6. Point the CLI to the dstack server.

CONSOLE

```
$ dstack config --url <URL> \  
  --project main \  
  --token <TOKEN>
```

Run a Fine-tuning Task

In this section, you will configure and run a training task using dstack on [Vultr Cloud GPUs](#). You will define the task, set up environment variables, and execute the fine-tuning process for the `DeepSeek-R1-Distill-Qwen-1.5B model`.

1. Continue in the `dstack-env` virtual environment, and create a directory and navigate into it.

CONSOLE

```
$ mkdir quickstart && cd quickstart
```

2. Initialize the directory.

```
CONSOLE
```

```
$ dstack init
```

3. Create a YAML file to define the dstack dev environment configuration.

```
CONSOLE
```

```
$ nano .dstack.yaml
```

4. Copy and paste the below configuration.

```
YAML
```

```
type: task
# The name is optional, if not specified, generated randomly
name: trl-train

python: "3.10"

nvcc: true
# Required environment variables
env:
- WANDB_API_KEY
- WANDB_PROJECT
- MODEL_ID=deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
# Commands of the task
commands:
- git clone https://github.com/huggingface/trl.git
- pip install transformers
- pip install trl
- pip install peft
- pip install wandb
- cd trl/trl/scripts
- python sft.py
  --model_name_or_path $MODEL_ID
  --dataset_name trl-lib/Capybara
  --learning_rate 2.0e-4
  --num_train_epochs 1
  --packing
  --per_device_train_batch_size 2
  --gradient_accumulation_steps 8
  --gradient_checkpointing
```

```
--logging_steps 25
--eval_strategy steps
--eval_steps 100
--use_peft
--lora_r 32
--lora_alpha 16
--report_to wandb
--output_dir DeepSeek-R1-Distill-Qwen-1.5B-SFT
```

resources:**gpu:**

```
# 24GB or more vRAM
```

```
memory: 24GB..
```

Save and close the file.

The YAML file defines a dstack task to fine-tune `DeepSeek-R1-Distill-Qwen-1.5B` using Hugging Face's TRL with LoRA optimization. It pulls the TRL repository, installs dependencies, and runs supervised fine-tuning on `trl-lib/Capybara` with gradient checkpointing and accumulation for memory efficiency. Training progress is logged to Weights & Biases (W&B). The task requests a 24GB+ GPU, ensuring efficient model fine-tuning on dstack.

Note

The configuration uses the `WANDB_API_KEY` and `WANDB_PROJECT` environment variables for logging training metrics with Weights & Biases (W&B). To use W&B, [create an account](#) and [retrieve your API key](#) for `WANDB_API_KEY`. The `WANDB_PROJECT` variable can be set to any preferred project name for organizing experiment logs.

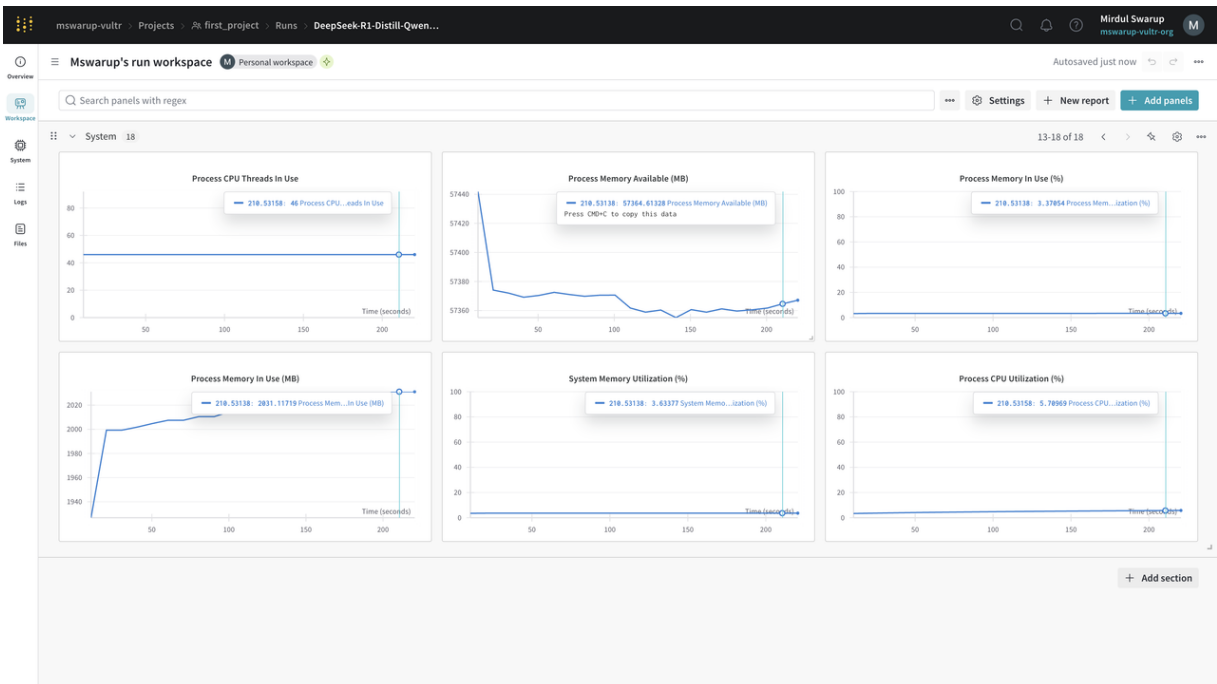
5. Apply the configuration.

CONSOLE

```
$ dstack apply -f .dstack.yaml
```

The configuration may take up to several minutes to start (depending on which machine you're using; VM start under 2 minutes while baremetals may take 30 min to provision). The fine-tuning process may further take additional time to complete the process and store the model.

6. Visit WandDB Dashboard by accessing the URL in terminal output to track model fine-tuning process.



Conclusion

In this guide, you successfully fine-tuned the DeepSeek R1 Distill Qwen 1.5B model using dstack on Vultr Cloud GPUs. You set up a virtual environment, installed dstack, and configured Vultr as your cloud provider, streamlining the training process. With this setup, you can now efficiently train and optimize AI models at scale, ensuring high-performance and reliable machine learning workflows.

For more information, refer to the following documentation:

- [Understand dstack Services](#)
- [Deploy Dev Environments with dstack on Vultr](#)
- [How to Deploy Services with dstack on Vultr](#)



VULTR

