ETH zürich

# Deep Learning for Autonomous Driving

## Ex 1: AWS Tutorial

CVL Computer Vision Lab

# AWS – Accounts and Ground Rules

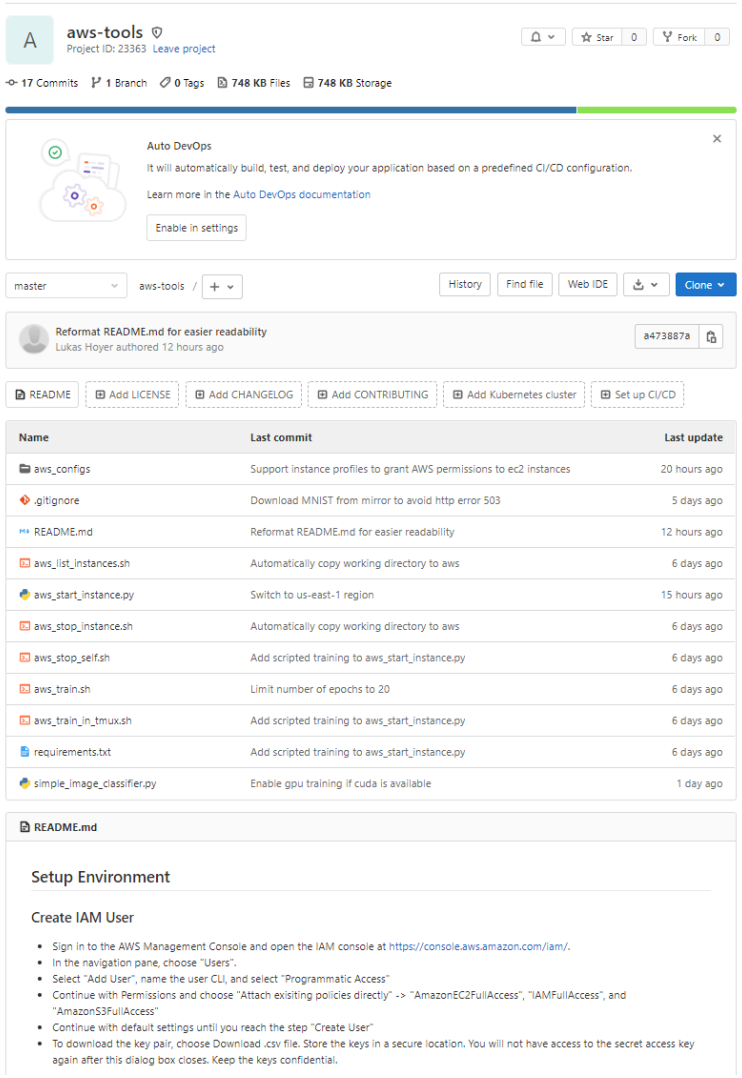
- Each team is getting one AWS account
- Email of the first user is used
- Reset the password to log in

- AWS account is only allowed to be used for the lecture
- The budget is computed to run **two** p2.xlarge using **Spot** Instances
- If a team is using two much credits, we will **delete** the account

- All our code and tutorials are using AWS EC2 with Pytorch
- We use an automated Terminal based job submission approach

# DLAD – AWS Tool

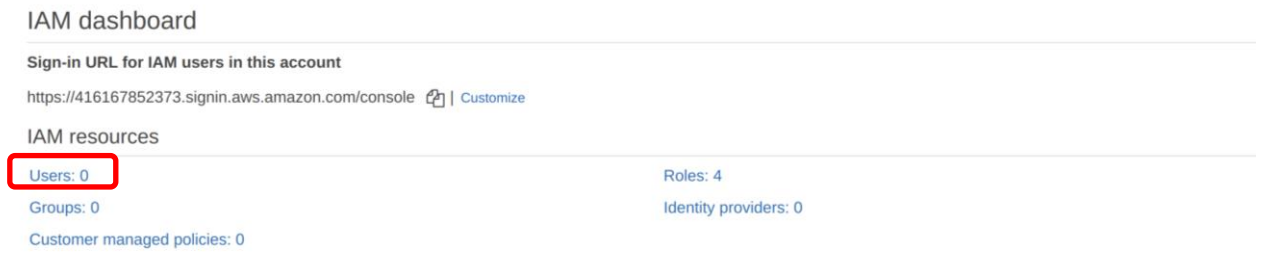
- All the instructions can be found at https://gitlab.ethz.ch/dlad21/aws-tools
- The following slides are a visualizations of these instructions
- Please use the git README.md for all the commands
- Do not copy from the slides!



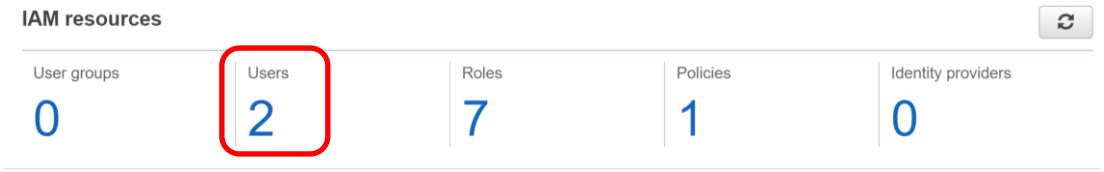
# First Steps – AWS IAM

- Sign into the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/

- In the navigation pane, choose "Users“

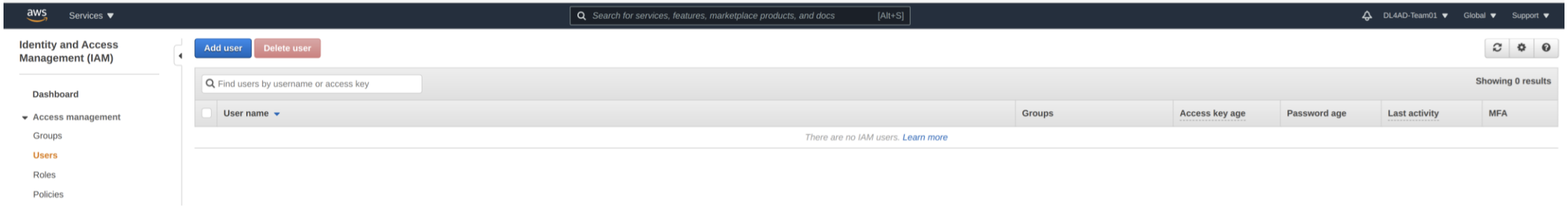



- May look slightly different

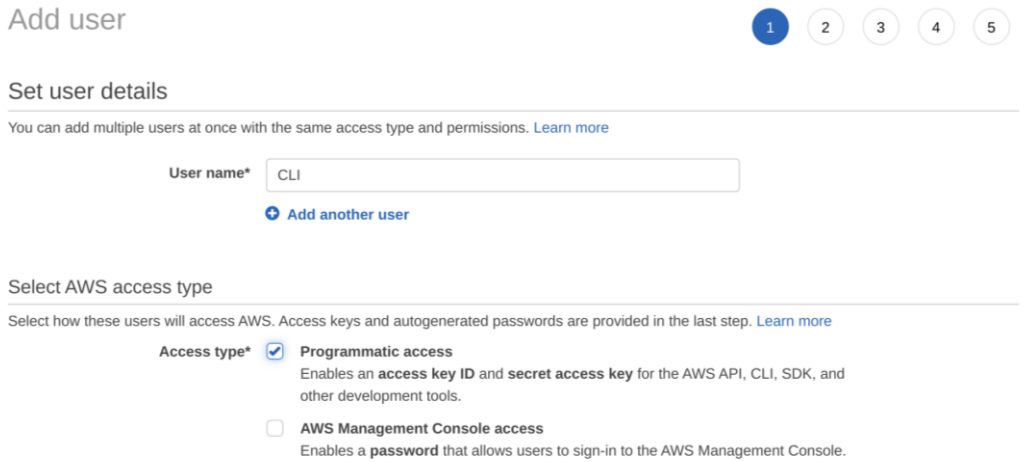


# First Steps – AWS IAM

- Select "Add User“

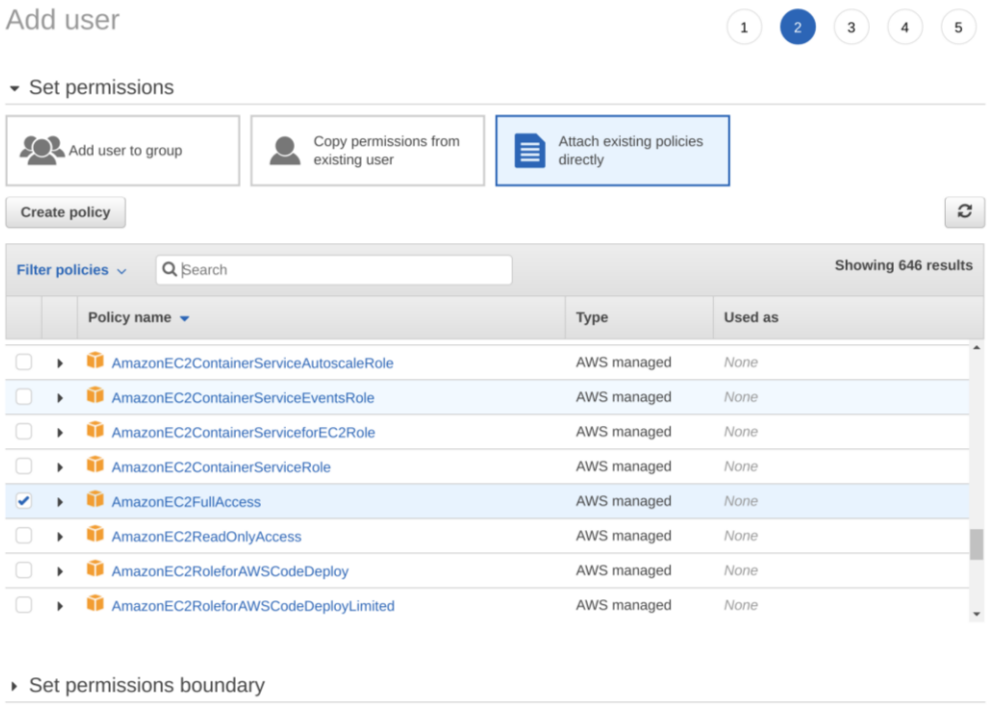



- Name the user CLI, and select "Programmatic Access"



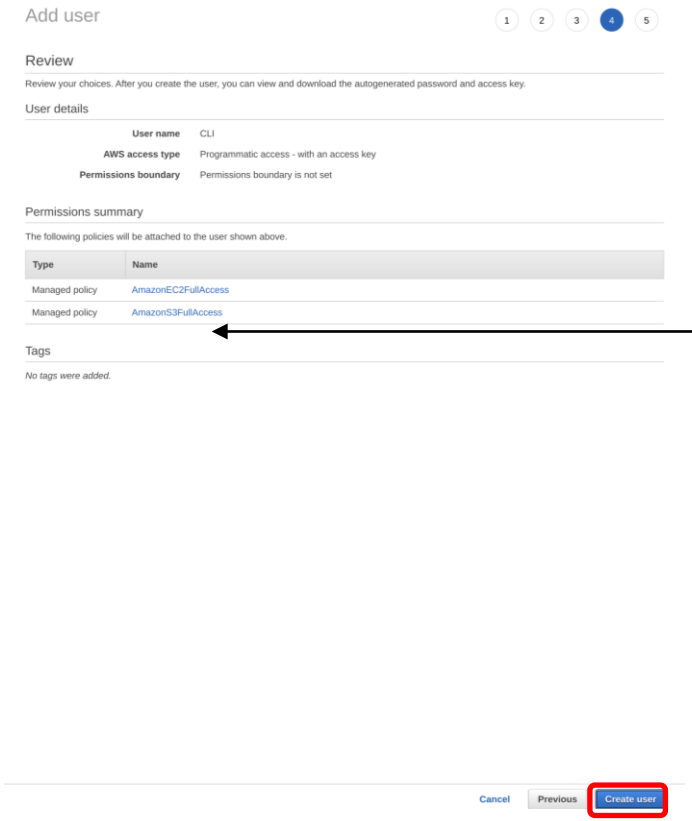
# First Steps – AWS IAM

- Continue with Permissions and choose "Attach existing policies directly”
  - "AmazonEC2FullAccess“
  - "IAMFullAccess“
  - "AmazonS3FullAccess"



# First Steps – AWS IAM

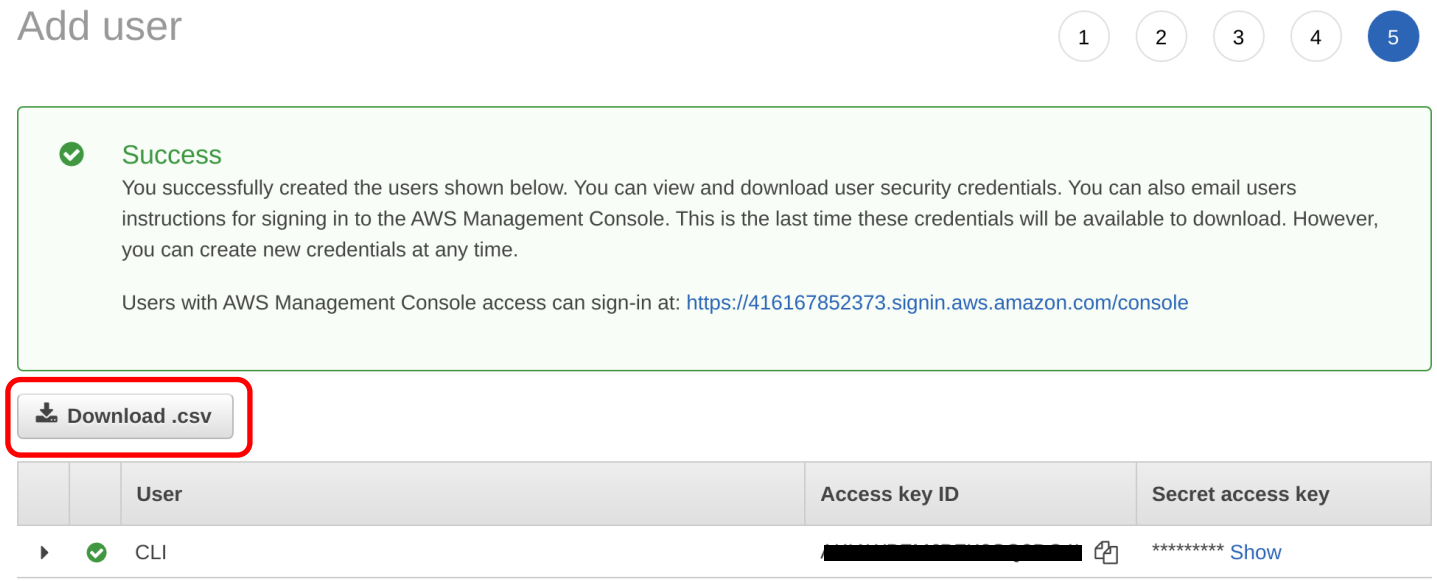
- Do not add any Tags
- Continue with default settings until you reach the step "Create User"




“IAMFullAccess” is missing here

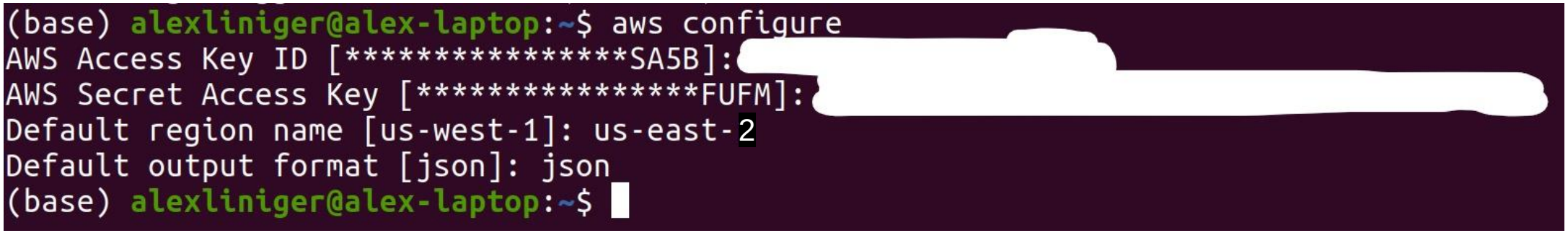
# First Steps – AWS IAM

- To download the key pair, choose **Download .csv** file.
- Store the keys in a **secure** location.
- You **will not have access to the secret access key again** after this dialog box closes. **Keep the keys confidential**.



# Second Steps – AWS CLI

- Install AWS-CLI on your computer using: **sudo apt install awscli**
  or follow https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html
- Configure AWS CLI: Run **aws configure** and provide the IAM credentials, choose us-east-2 as region and json as output format



# Second Steps – AWS CLI

- Create ssh key:
  - aws ec2 create-key-pair --key-name dlad-aws --query "KeyMaterial" --output text > ~/.ssh/dlad-aws.pem
  - chmod 400 ~/.ssh/dlad-aws.pem
- Create security group:
  - aws ec2 create-security-group --group-name dlad-sg --description "DLAD Security Group“
  - aws ec2 authorize-security-group-ingress --group-name dlad-sg --protocol tcp --port 22 --cidr 0.0.0.0/0

```
(base) alexliniger@alex-laptop:~$ aws ec2 create-key-pair --key-name dlad-aws --query "KeyMaterial" --output text > ~/.ssh/dlad-aws.pem
(base) alexliniger@alex-laptop:~$ chmod 400 ~/.ssh/dlad-aws.pem
(base) alexliniger@alex-laptop:~$ aws ec2 create-security-group --group-name dlad-sg --description "DLAD Security Group"
{
    "GroupId": "sg-0aeec420e617fc554"
}
(base) alexliniger@alex-laptop:~$ aws ec2 authorize-security-group-ingress --group-name dlad-sg --protocol tcp --port 22 --cidr 0.0.0.0/0
```

# Second Steps – AWS CLI

- Create policies, roles, and instance profile to grant permissions to ec2 instances
- This is necessary for aws_stop_self.sh and S3 access from the ec2 instance
  - aws iam create-role --role-name dlad-role --assume-role-policy-document file://aws_configs/ec2-role-trust-policy.json
  - aws iam put-role-policy --role-name dlad-role --policy-name EC2-Terminate-Permissions --policy-document file://aws_configs/ec2-terminate-policy.json
  - aws iam put-role-policy --role-name dlad-role --policy-name S3-Permissions --policy-document file://aws_configs/s3-access-policy.json
  - aws iam create-instance-profile --instance-profile-name dlad-instance-profile
  - aws iam add-role-to-instance-profile --instance-profile-name dlad-instance-profile --role-name dlad-role
- Different for second student – see README

# Work with Instances

- Clone the `aws-tool` git repository
- This gives you some functionality to start and stop AWS instances
  - `python aws_start_instance.py` → starts a new instance
  - `bash aws_list_instances.sh` → lists all instances
  - `bash aws_stop_instance.sh INSTANCE_ID` → stops instance with INSTANCE_ID
- But also allows for a fully automated approach

- Instances can also be started and stopped using the AWS homepage
- However, this makes automation harder (no automatic training and stopping)
- But can be convenient to see running instances and stop instances

# Work with Instances – Weights & Biases (wandb.ai)

- For all our logging we use **wandb** since it allows easy visualization and has a lot of tools convenient when working with remote instances
- Please open an account before you start working with instances

# Work with Instances – Manual

- Starts a new instance
  - python aws_start_instance.py
    - The first time this will result in an error, due to a too small spot instance quota with the error "MaxSpotInstanceCountExceeded”
    - This quota issue will be resolved automatically, and you will receive an email

```
(base) alexliniger@alex-laptop:~/Documents/Git/DLAD/aws-tools$ python aws_start_instance.py
Launch instance...
Wait for instance and copy files to AWS...
sending incremental file list
created directory /home/ubuntu/code
./
.gitignore
README.md
aws_list_instances.sh
aws_start_instance.py
aws_stop_instance.sh
aws_stop_self.sh
aws_train.sh
aws_train_in_tmux.sh
requirements.txt
simple_image_classifier.py
spot-options.json

sent 15,479 bytes  received 268 bytes  223.36 bytes/sec
total size is 14,625  speedup is 0.93
Sucessfully started instance i-0dea0343f358d8683 with tag 2021-03-18_15-44-11
Connect to instance using ssh:
ssh -q -o StrictHostKeyChecking=no -o LogLevel=ERROR -o ConnectTimeout=180 -i ~/.ssh/dlad-aws.pem -t ubuntu@ec2-3-237-4-79.compute-1.amazonaws.com
Rsync file updates:
rsync -av -e 'ssh -q -o StrictHostKeyChecking=no -o LogLevel=ERROR -o ConnectTimeout=180 -i ~/.ssh/dlad-aws.pem -t' . --exclude 'wandb/' --exclude '.git/' ubuntu@ec2-3-237-4-79.compute-1.amazonaws.com:~/code/
Connect to tmux session using ssh:
ssh -q -o StrictHostKeyChecking=no -o LogLevel=ERROR -o ConnectTimeout=180 -i ~/.ssh/dlad-aws.pem -t ubuntu@ec2-3-237-4-79.compute-1.amazonaws.com tmux attach-session -t dlad
(base) alexliniger@alex-laptop:~/Documents/Git/DLAD/aws-tools$ bash aws_list_instances.sh
[
    [
        {
            "Name": null,
            "Instance": "i-0dea0343f358d8683",
            "InstanceType": "m5.large",
            "InstanceLifecycle": null,
            "State": "running",
            "IP": "3.237.4.79",
            "DNS": "ec2-3-237-4-79.compute-1.amazonaws.com"
        }
    ]
]
```

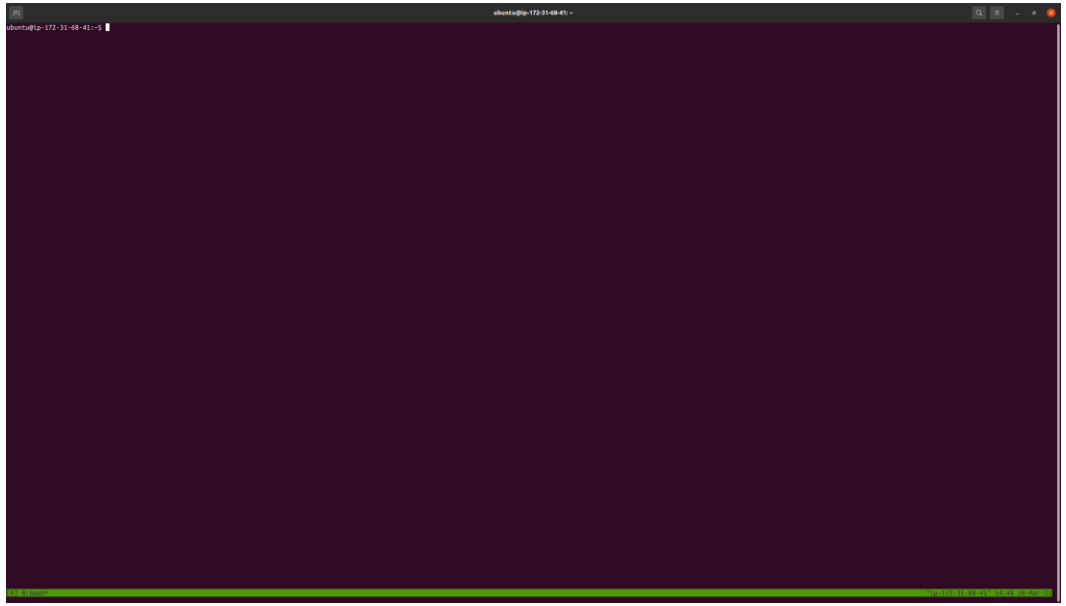
# Work with Instances – Manual

- Connect to a started instance
  - ssh -i ~/.ssh/dlad-aws.pem ubuntu@ec2-XXX.compute.amazonaws.com

```
(base) alexliniger@alex-laptop:~/Documents/Git/DLAD/aws-tools$ ssh -i ~/.ssh/dlad-aws.pem ubuntu@ec2-3-237-4-79.compute-1.amazonaws.com
=============================================================================
      __|  __|_  )
      _|  (     /   Deep Learning AMI (Ubuntu 18.04) Version 41.0
     ___|\___|___|
=============================================================================

Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1038-aws x86_64v)

Please use one of the following commands to start the required environment with the framework of your choice:
for AWS MX 1.7 (+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN) ________________________________ source activate mxnet_p36
for AWS MX 1.8 (+Keras2) with Python3 (CUDA + and Intel MKL-DNN) ________________________________ source activate mxnet_latest_p37
for AWS MX(+AWS Neuron) with Python3 ____________________________________________________ source activate aws_neuron_mxnet_p36
for AWS MX(+Amazon Elastic Inference) with Python3 ______________________________________ source activate amazonei_mxnet_p36
for TensorFlow(+Keras2) with Python3 (CUDA + and Intel MKL-DNN) ___________________________ source activate tensorflow_p37
for Tensorflow(+AWS Neuron) with Python3 ____________________________________________ source activate aws_neuron_tensorflow_p36
for TensorFlow 2(+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN) ____________________ source activate tensorflow2_p36
for TensorFlow 2.3 with Python3.7 (CUDA + and Intel MKL-DNN) ___________________________ source activate tensorflow2_latest_p37
for PyTorch 1.4 with Python3 (CUDA 10.1 and Intel MKL) _____________________________________ source activate pytorch_p36
for PyTorch 1.7.1 with Python3.7 (CUDA 11.1 and Intel MKL) ________________________________ source activate pytorch_latest_p37
for PyTorch (+AWS Neuron) with Python3 ____________________________________________ source activate aws_neuron_pytorch_p36
for base Python3 (CUDA 10.0) _____________________________________________________________ source activate python3

To automatically activate base conda environment upon login, run: 'conda config --set auto_activate_base true'
Official Conda User Guide: https://docs.conda.io/projects/conda/en/latest/user-guide/
AWS Deep Learning AMI Homepage: https://aws.amazon.com/machine-learning/amis/
Developer Guide and Release Notes: https://docs.aws.amazon.com/dlami/latest/devguide/what-is-dlami.html
Support: https://forums.aws.amazon.com/forum.jspa?forumID=263
For a fully managed experience, check out Amazon SageMaker at https://aws.amazon.com/sagemaker
When using INF1 type instances, please update regularly using the instructions at: https://github.com/aws/aws-neuron-sdk/tree/master/release-notes
=============================================================================

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Mar 18 14:47:19 UTC 2021

  System load:  1.28              Processes:             119
  Usage of /:   87.9% of 96.88GB  Users logged in:       0
  Memory usage: 4%                IP address for ens5:   172.31.68.41
  Swap usage:   0%                IP address for docker0: 172.17.0.1

  => / is using 87.9% of 96.88GB

 * Introducing self-healing high availability clusters in MicroK8s.
   Simple, hardened, Kubernetes for production, from RaspberryPi to DC.

     https://microk8s.io/high-availability

13 packages can be updated.
13 of these updates are security updates.
To see these additional updates run: apt list --upgradable

New release '20.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
```

# Work with Instances – Manual

- We use tmux to recover running jobs after the ssh connection is lost
  - tmux → “new” terminal



  - Detach from tmux → ctrl b + d
  - reattach to tmux → tmux attach -t 0

# Work with Instances – Manual

- Starting training on EC2 instance
  - Change to code dictionary: **cd ~/code**
  - Activate the conda pytorch environment: **source activate pytorch_latest_p37**
  - Install the project requirements: **pip install -r requirements.txt**

```
ubuntu@ip-172-31-68-41:~$ cd code
ubuntu@ip-172-31-68-41:~/code$ source activate pytorch_latest_p37
(pytorch_latest_p37) ubuntu@ip-172-31-68-41:~/code$ pip install -r requirements.txt
```

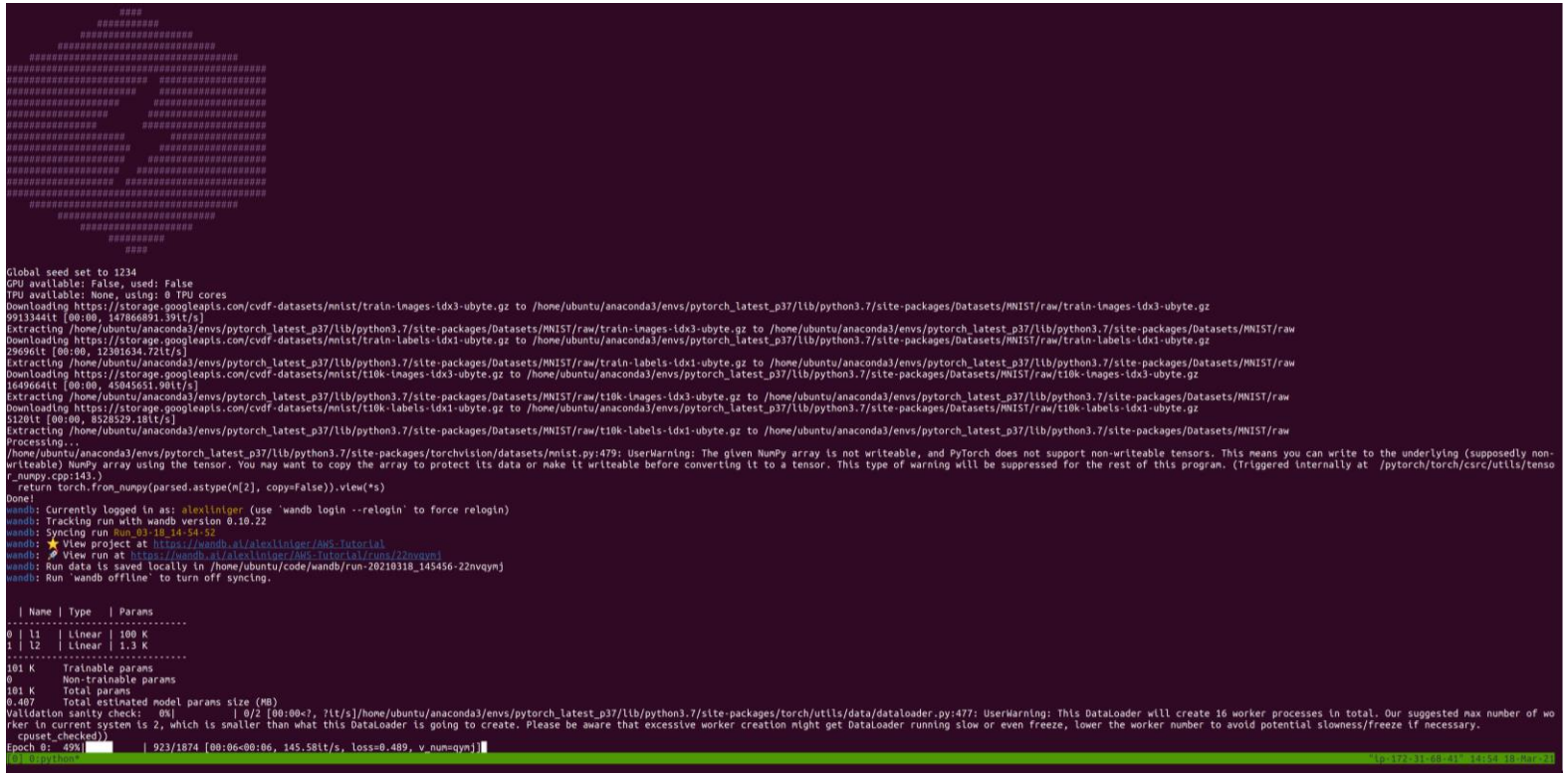
  - Login to wandb: **wandb login**

```
(pytorch_latest_p37) ubuntu@ip-172-31-68-41:~/code$ wandb login
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter:
```

  - API key you can find at: https://wandb.ai/authorize

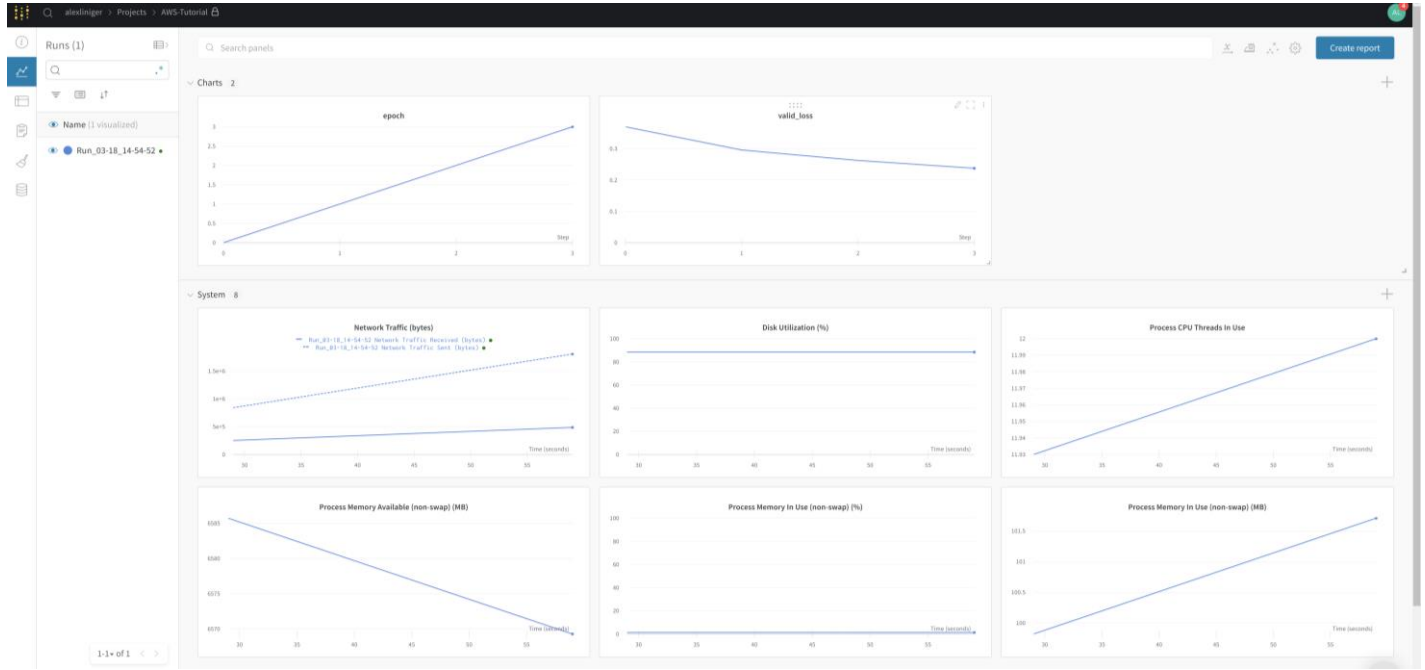
# Work with Instances – Manual

- Starting training on EC2 instance
  - We have now sourced the env and installed all requirements
  - We can now train the model: **python simple_image_classifier.py**



# Work with Instances – Manual

- Starting training on EC2 instance
  - We have now source the env and installed all requirements
  - We can now train the model: **python simple_image_classifier.py**
  - We can follow the training on wandb.ai



# Work with Instances – Manual

- Starting training on EC2 instance
  - We have now source the env and installed all requirements
  - We can now train the model: **python simple_image_classifier.py**
  - We can follow the training on wandb.ai
  - And to stop the instance: **bash aws_stop_self.sh**

```
--------------------------------------------------------------------------------
DATALOADER:0 TEST RESULTS
{'test_loss': 0.10556504875421524}
--------------------------------------------------------------------------------
[{'test_loss': 0.10556504875421524}]

wandb: Waiting for W&B process to finish, PID 19286
wandb: Program ended successfully.
wandb:
wandb: Find user logs for this run at: /home/ubuntu/code/wandb/run-20210318_145456-22nvqymj/logs/debug.log
wandb: Find internal logs for this run at: /home/ubuntu/code/wandb/run-20210318_145456-22nvqymj/logs/debug-internal.log
wandb: Run summary:
wandb:            valid_loss 0.10936
wandb:                 epoch 19
wandb:   trainer/global_step 34360
wandb:              _runtime 236
wandb:            _timestamp 1616079532
wandb:                 _step 20
wandb:             test_loss 0.10557
wandb: Run history:
wandb:            valid_loss ▁
wandb:                 epoch ▁
wandb:   trainer/global_step ▁
wandb:              _runtime ▁
wandb:            _timestamp ▁
wandb:                 _step ▁
wandb:             test_loss ▁
wandb:
wandb: Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 1 other file(s)
wandb:
wandb: Synced Run_03-18_14-54-52: https://wandb.ai/alexliniger/AWS-Tutorial/runs/22nvqymj
```

# Work with Instances – Manual

- Starting training on EC2 instance
  - We have now source the env and installed all requirements
  - We can now train the model: `python simple_image_classifier.py`
  - We can follow the training on wandb.ai
  - And to stop the instance: `bash aws_stop_self.sh`

- Note that the instance storage is temporary and local changes (including the environment setup) are lost after instance termination

# Work with Instances – Automatic

- Starting training on EC2 instance using our automated scripts
  - Create **wandb_key.txt** at the root of your local project
  - The file contains your API key: https://wandb.ai/authorize
  - To automate all the previous steps use: **python aws_start_instance.py --train**

```
(base) alexliniger@alex-laptop:~/Documents/Git/DLAD/aws-tools$ python aws_start_instance.py --train
Launch instance...
Wait for instance and copy files to AWS...
sending incremental file list
created directory /home/ubuntu/code
./
.gitignore
README.md
aws.log
aws_list_instances.sh
aws_start_instance.py
aws_stop_instance.sh
aws_stop_self.sh
aws_train.sh
aws_train_in_tmux.sh
requirements.txt
simple_image_classifier.py
spot-options.json

sent 15,717 bytes  received 287 bytes  230.27 bytes/sec
total size is 14,787  speedup is 0.92
Start training in tmux session...
Sucessfully started instance i-0b453979856fdbe2e with tag 2021-03-18_16-06-53
Connect to instance using ssh:
ssh -q -o StrictHostKeyChecking=no -o LogLevel=ERROR -o ConnectTimeout=180 -i ~/.ssh/dlad-aws.pem -t ubuntu@ec2-34-238-191-13.compute-1.amazonaws.com
Rsync file updates:
rsync -av -e 'ssh -q -o StrictHostKeyChecking=no -o LogLevel=ERROR -o ConnectTimeout=180 -i ~/.ssh/dlad-aws.pem -t' . --exclude 'wandb/' --exclude '.git/' ubuntu@ec2-34-238-191-13.compute-1.amazonaws.com:~/code/
Connect to tmux session using ssh:
ssh -q -o StrictHostKeyChecking=no -o LogLevel=ERROR -o ConnectTimeout=180 -i ~/.ssh/dlad-aws.pem -t ubuntu@ec2-34-238-191-13.compute-1.amazonaws.com tmux attach-session -t dlad
(base) alexliniger@alex-laptop:~/Documents/Git/DLAD/aws-tools$
```

  - Use last command to connect to tmux session using ssh

# Work with S3 Bucket

- S3 Buckets offer permanent storage which can be practical to safe training results, and such as models or test results
- Quick guide to work generate and use S3 Buckets
  - Create bucket: `aws s3 mb s3://BUCKET_NAME`
    - The bucket name has to be globally unique, can only use **small letters, number and dash**
  - Make it private: `aws s3api put-public-access-block --bucket BUCKET_NAME --public-access-block-configuration "BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets=true“`
  - Sync folder: `aws s3 sync local_path s3://BUCKET_NAME/bucket_path`