

# AI from Space using Azure

Christos Charmatzis

@christoscharmatzis

<https://tageoforce.com>

*Center of the Milky Way Galaxy*

Source: [Wikipedia](#)



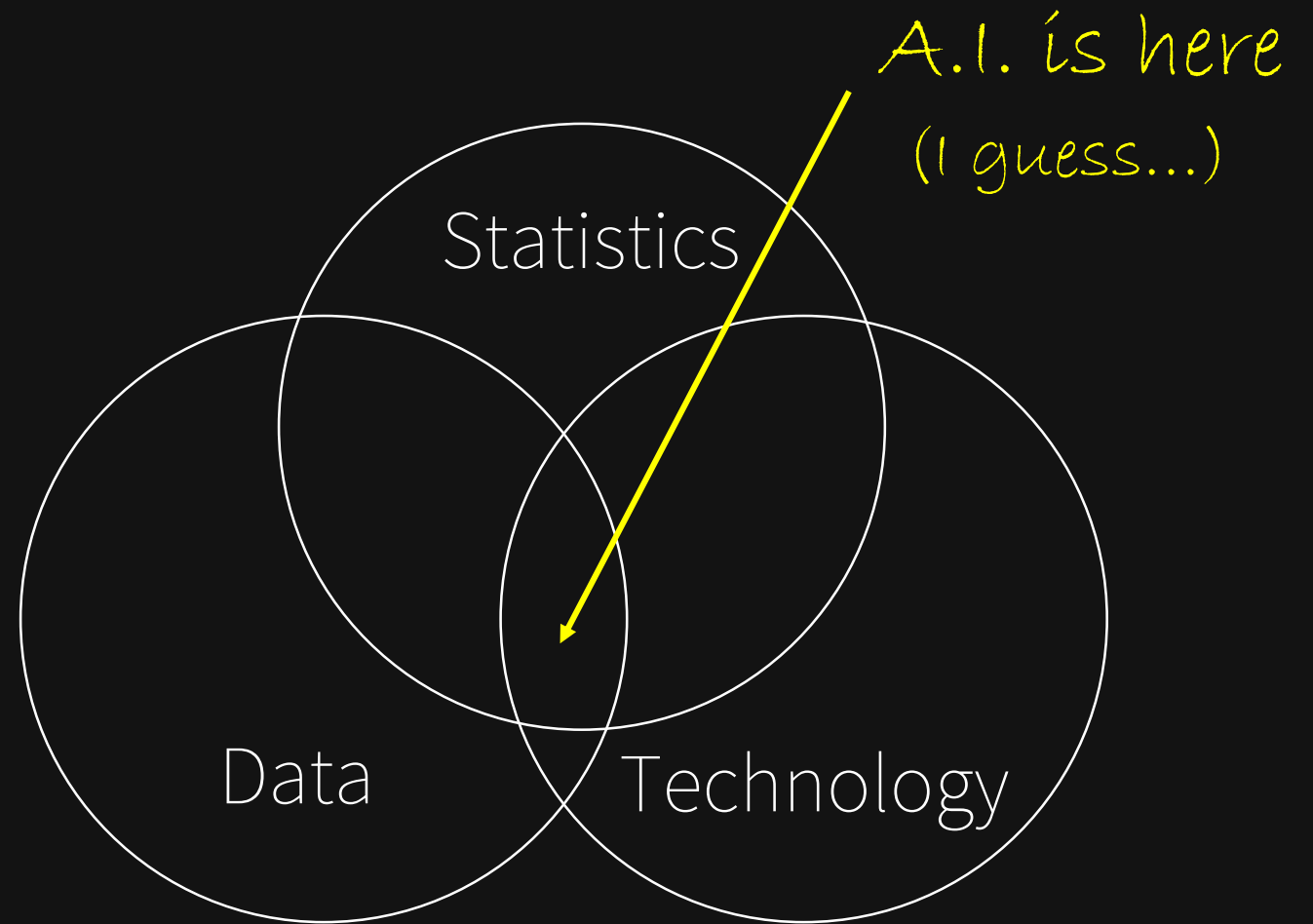
# Few things about me

- Project manager @TA-Geoforce
- GIS Specialist 10+ years
- AI professional
- Open Source enthusiasm
- Piano player

*Chopin – Heroic Polonaise*  
*Source: [youtube.com/Rouseau](https://youtube.com/Rouseau)*



What they  
show us



What's AI for  
me



AI  $\equiv$  Knowledge

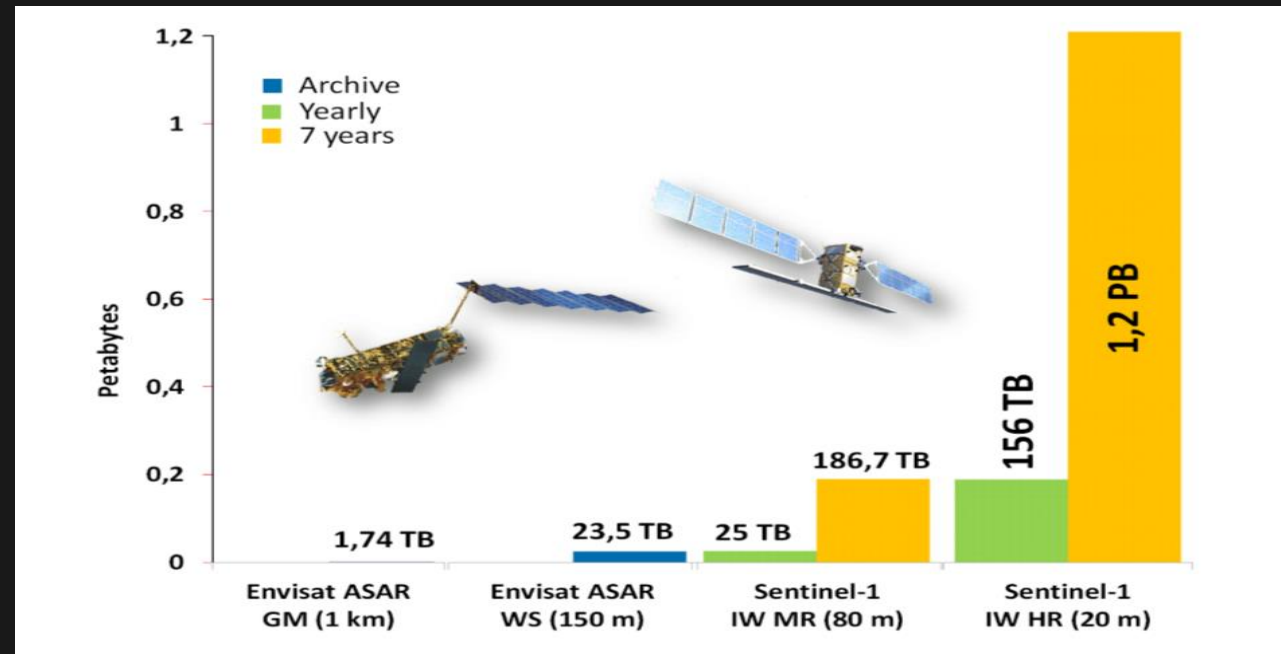
Data

All of us

# Earth Observation Data

Only ESA satellites produces around 150 terabytes per day!!

(source:[https://www.esa.int/Applications/Observing\\_the\\_Earth/Working\\_towards\\_AI\\_and\\_Earth\\_observation](https://www.esa.int/Applications/Observing_the_Earth/Working_towards_AI_and_Earth_observation))



Growth of data volume from ENVISAT ASAR to Sentinel-1.

Source: Big Data Infrastructures for Processing Sentinel Data - Wolfgang Wagner, Vienna - 2015



# Did you know that Azure has an Open Data Catalogue?

- MODIS
- NAIP
- NOAA Global Forecast System (GFS)
- Harmonized Landsat Sentinel-2
- NOAA Integrated Surface Data (ISD)
- Daymet

And the best part is that are **FREE OF CHARGE!**

<https://azure.microsoft.com/en-us/services/open-datasets/catalog>

## Let's get started

1. Talk with the client about the goal of the AI project.
2. Split the question that needs to be answered in small questions.
3. Search for datasets
4. Create the project structure.



You just hit  
the wall

The problem in every single AI project is ONE (1)  
WRANGLING with the DATA.

Visualize your data

# Use ready examples

---

## Azure Notebooks

Package: blob storage ▾

Language: Python

Download Notebook

### Demo notebook for accessing MODIS data on Azure

This notebook provides an example of accessing MODIS data from blob storage on Azure, including (1) finding the MODIS tile corresponding to a lat/lon coordinate, (2) retrieving that tile from blob storage, and (3) displaying that tile using the [rasterio](#) library.

This notebook uses the MODIS surface reflectance product as an example, but data structure and access will be the same for other MODIS product.

MODIS data are stored in the East US data center, so this notebook will run most efficiently on Azure compute located in East US. We recommend that substantial computation depending on MODIS data also be situated in East US. You don't want to download hundreds of terabytes to your laptop! If you are using MODIS data for environmental science applications, consider applying for an [AI for Earth grant](#) to support your compute requirements.

### Imports and environment

```
In [3]: import os
import tempfile
```

Spatial data  
are special  
data?

Tensorflow and Pytorch are specialized Deep Learning frameworks, that are developed for specific needs. E.g. Image recognition

Things don't go well when you try to use them outside their comfort zone.

# My favorite deep learning framework

## Raster-Vision

<https://rastervision.io/>

Raster Vision is an open source framework for Python developers building computer vision models on satellite, aerial, and other large imagery sets (including oblique drone imagery)



Chip Classification



Object Detection



Semantic Segmentation

# Spatial Data vs Big Data

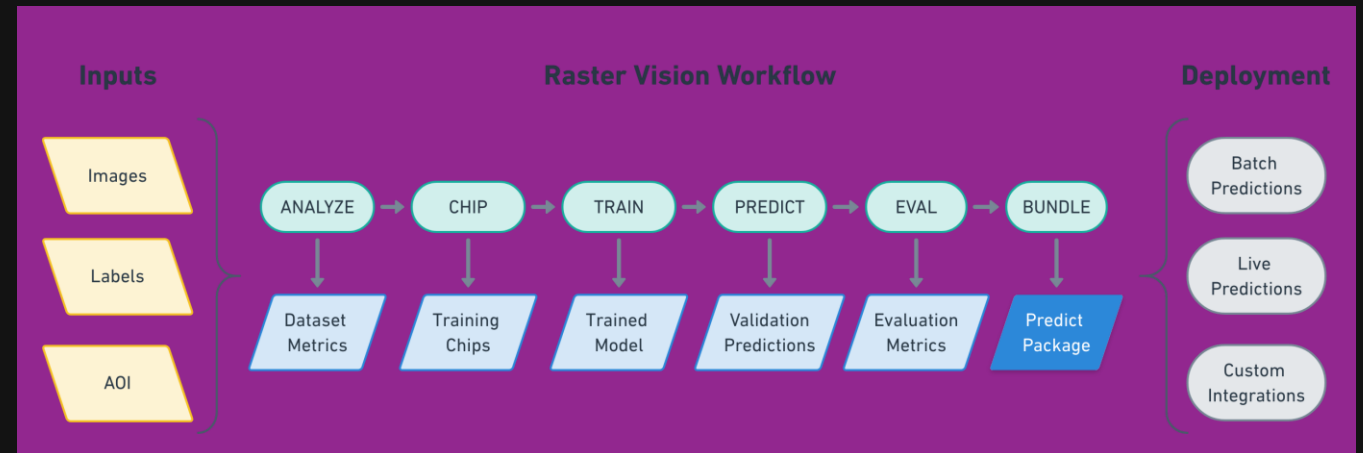
Since everything is relative:

- If you are studying (labeling) small features (e.g. roofs, cars, parking places) you are OK!!! There is nothing to worry about Big Data

- If you are studying (labeling) large features (e.g. lakes, oil spills, forests)

You are in Big (Trouble) Data!!!!

# Raster-Vision workflow



# Dockerize everything

```
//requirements.txt
```

```
azure  
azure-storage  
azure-storage-blob
```

```
#Dockerfile
```

```
FROM quay.io/azavea/raster-vision:pytorch-0.10
```

```
COPY requirements.txt /  
RUN pip install -r /requirements.txt
```

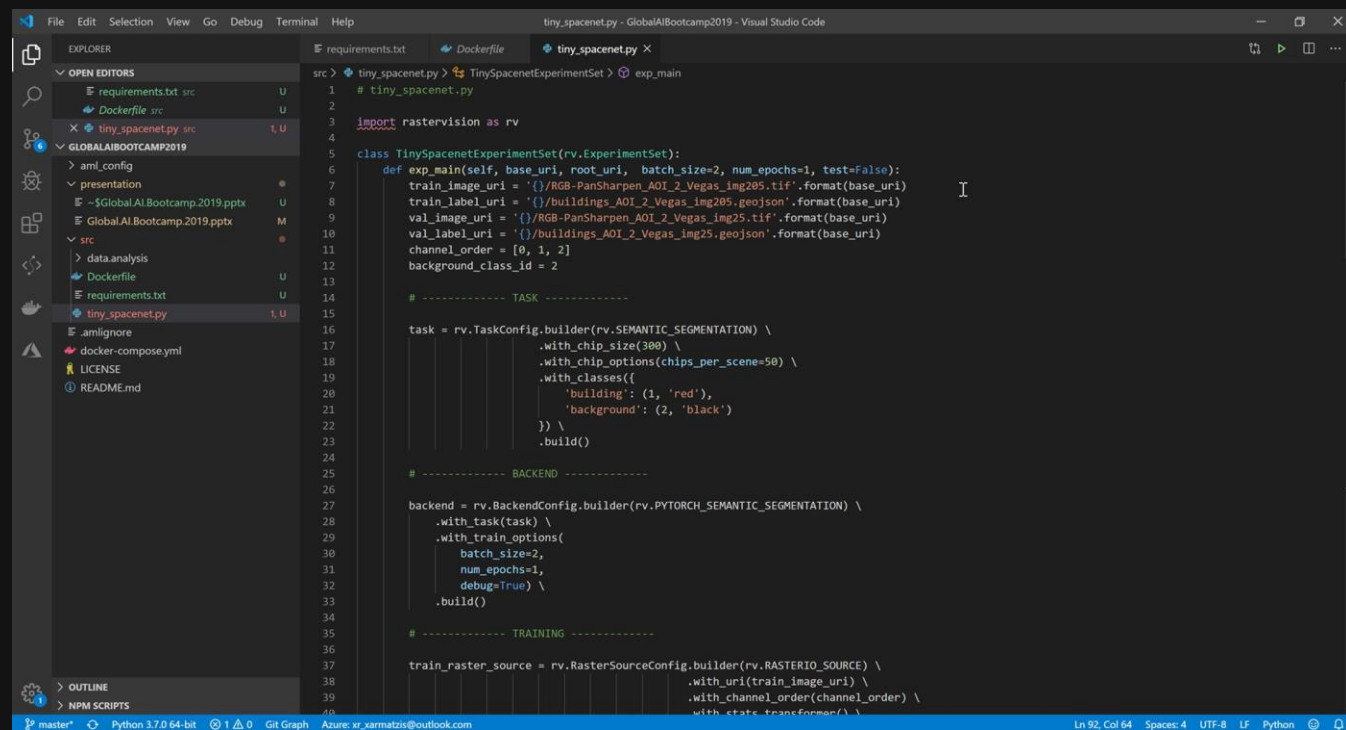
```
COPY tiny_spacenet.py
```

```
ENV PATH=$PATH:/src  
ENV PYTHONPATH /src
```

```
ADD ./ /src  
WORKDIR /src/
```



Write  
experiments



The screenshot shows a Visual Studio Code editor window titled "tiny\_spacenet.py - GlobalAIBootcamp2019 - Visual Studio Code". The Explorer panel on the left shows the project structure with files like requirements.txt, Dockerfile, and tiny\_spacenet.py. The main editor displays the code for tiny\_spacenet.py, which includes imports, class definitions, and configuration builders for task, backend, and training.

```
src > tiny_spacenet.py > TinySpacenetExperimentSet > exp_main
1 # tiny_spacenet.py
2
3 import rastervision as rv
4
5 class TinySpacenetExperimentSet(rv.ExperimentSet):
6
7     def exp_main(self, base_uri, root_uri, batch_size=2, num_epochs=1, test=False):
8         train_image_uri = '{} /RGB-PanSharpen_AOI_2_Vegas_img205.tif'.format(base_uri)
9         train_label_uri = '{} /buildings_AOI_2_Vegas_img205.geojson'.format(base_uri)
10        val_image_uri = '{} /RGB-PanSharpen_AOI_2_Vegas_img25.tif'.format(base_uri)
11        val_label_uri = '{} /buildings_AOI_2_Vegas_img25.geojson'.format(base_uri)
12        channel_order = [0, 1, 2]
13        background_class_id = 2
14
15        # ----- TASK -----
16
17        task = rv.TaskConfig.builder(rv.SEMANTIC_SEGMENTATION) \
18            .with_chip_size(300) \
19            .with_chip_options(chips_per_scene=50) \
20            .with_classes({
21                'building': (1, 'red'),
22                'background': (2, 'black')
23            }) \
24            .build()
25
26        # ----- BACKEND -----
27
28        backend = rv.BackendConfig.builder(rv.PYTORCH_SEMANTIC_SEGMENTATION) \
29            .with_task(task) \
30            .with_train_options(
31                batch_size=2,
32                num_epochs=1,
33                debug=True) \
34            .build()
35
36        # ----- TRAINING -----
37
38        train_raster_source = rv.RasterSourceConfig.builder(rv.RASTERIO_SOURCE) \
39            .with_uri(train_image_uri) \
40            .with_channel_order(channel_order) \
41            .with_stats_transformer() \
42            .build()
```

## Build & run

//Build docker image

```
docker build -t  
charmatzis/raster_vision_azure_batch_demo.
```

//Run it

```
docker run  
charmatzis/raster_vision_azure_batch_demo  
python /src/tiny_spacenet.py -- base_uri
```

```
wasbs://demo@charmatzis.blob.core.windows  
.net/ --root_uri
```

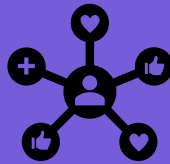
```
wasbs://demo@charmatzisdata.blob.core.win  
dows.net/results
```



If not local,  
then what?



Good choice (generally) Azure Machine Learning and good match with VS Code.

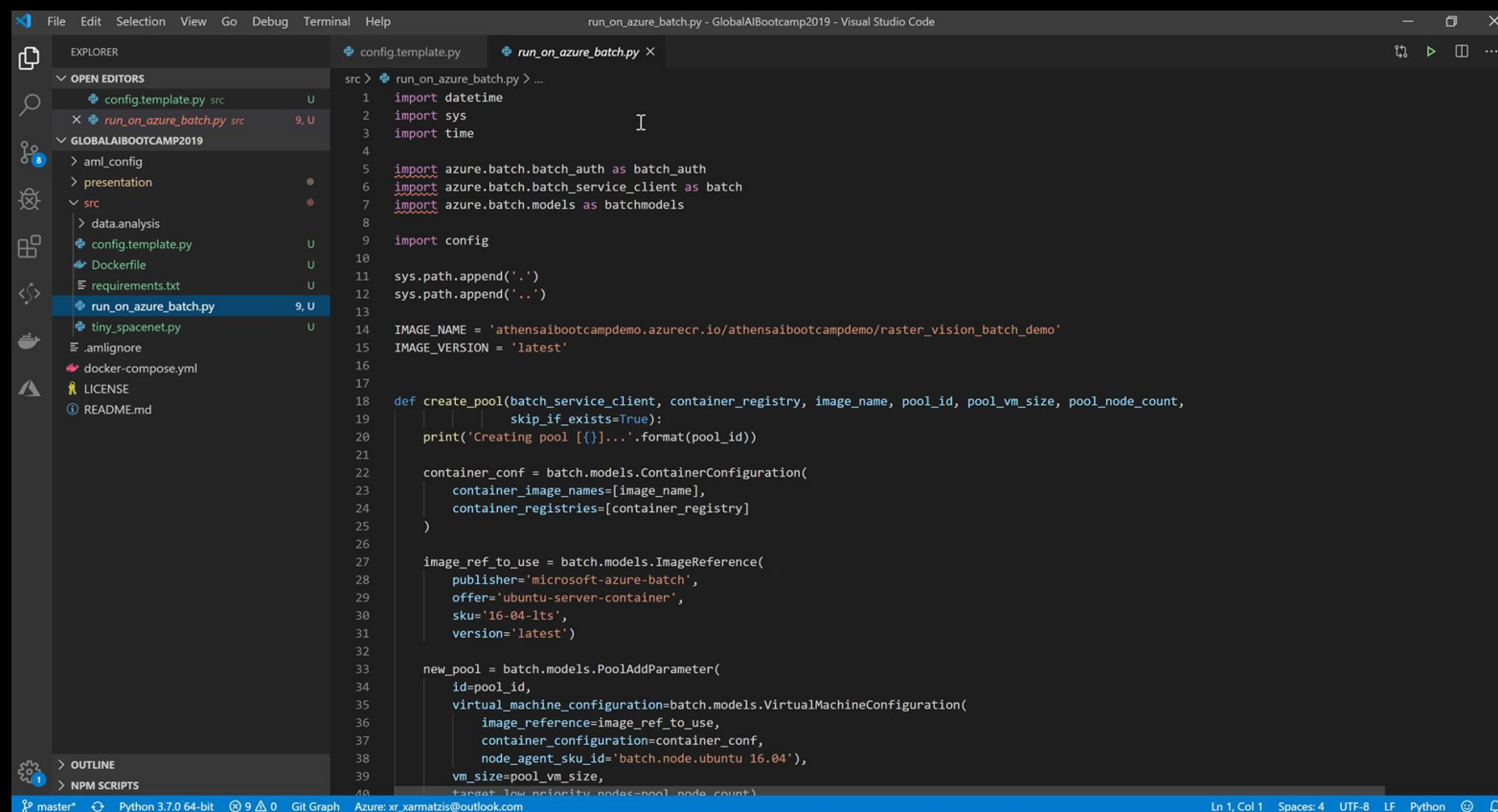


If you are working in special stuff (as we always do) use Azure Batch.

# Move images to Azure with 3 simple moves

- Azure Container Registry  
`docker login athensaibootcampdemo.azurecr.io`
- Tag your docker container  
`docker tag  
charmatzis/raster_vision_azure_batch_demo:latest  
athensaibootcampdemo.azurecr.io/charmatzis/  
raster_vision_azure_batch_demo :latest`
- Upload it to ACR  
`docker push athensaibootcampdemo.azurecr.io/  
athensaibootcampdemo /  
raster_vision_azure_batch_demo :latest`

# Run it on Azure Batch



```
src > run_on_azure_batch.py > ...
1 import datetime
2 import sys
3 import time
4
5 import azure.batch.batch_auth as batch_auth
6 import azure.batch.batch_service_client as batch
7 import azure.batch.models as batchmodels
8
9 import config
10
11 sys.path.append('.')
12 sys.path.append('..')
13
14 IMAGE_NAME = 'athensaiibootcampdemo.azurecr.io/athensaiibootcampdemo/raster_vision_batch_demo'
15 IMAGE_VERSION = 'latest'
16
17
18 def create_pool(batch_service_client, container_registry, image_name, pool_id, pool_vm_size, pool_node_count,
19                 skip_if_exists=True):
20     print('Creating pool [{}]...'.format(pool_id))
21
22     container_conf = batch.models.ContainerConfiguration(
23         container_image_names=[image_name],
24         container_registries=[container_registry]
25     )
26
27     image_ref_to_use = batch.models.ImageReference(
28         publisher='microsoft-azure-batch',
29         offer='ubuntu-server-container',
30         sku='16-04-lts',
31         version='latest')
32
33     new_pool = batch.models.PoolAddParameter(
34         id=pool_id,
35         virtual_machine_configuration=batch.models.VirtualMachineConfiguration(
36             image_reference=image_ref_to_use,
37             container_configuration=container_conf,
38             node_agent_sku_id='batch.node.ubuntu 16.04'),
39         vm_size=pool_vm_size,
```

# Run it on Azure Batch...

## But how?

- It connects to your container registry and uses those docker images
- Create a Pool if it doesn't exist yet. Here, you can configure which kind of VMs and how many of them you want in your pool. And more importantly, you can specify that it are Low Prio VMs, which are cheap.
- Create a Job within the Pool
- Create a separate task to process each year of data. In a real-life situation, you would have a task for each day of data.



# \$Pricing\$

## NC-series

Add to estimate	Instance	Core	RAM	Temporary storage	GPU	Pay as you go (Low priority)	Pay as you go (normal priority)	1 year reserved (% Savings)	3 year reserved (% Savings)	Spot (% Savings)
+	NC6	6	56 GiB	340 GiB	1X K80	\$0.18/hour	\$0.90/hour	\$0.5733/hour (~36%)	\$0.3996/hour (~56%)	\$0.18/hour (~80%)
+	NC12	12	112 GiB	680 GiB	2X K80	\$0.36/hour	\$1.80/hour	\$1.1466/hour (~36%)	\$0.7991/hour (~56%)	\$0.36/hour (~80%)
+	NC24r	24	224 GiB	1,440 GiB	4X K80	\$0.792/hour	\$3.96/hour	\$2.5224/hour (~36%)	\$1.7578/hour (~56%)	\$0.792/hour (~80%)
+	NC24	24	224 GiB	1,440 GiB	4X K80	\$0.72/hour	\$3.60/hour	\$2.2932/hour (~36%)	\$1.5981/hour (~56%)	\$0.72/hour (~80%)

## NCsv2-series

Add to estimate	Instance	Core	RAM	Temporary storage	GPU	Pay as you go (Low priority)	Pay as you go (normal priority)	1 year reserved (% Savings)	3 year reserved (% Savings)	Spot (% Savings)
+	NC6s v2	6	112 GiB	736 GiB	1X P100	\$0.36/hour	\$2.07/hour	\$1.3187/hour (~36%)	\$0.9189/hour (~56%)	\$0.36/hour (~83%)
+	NC12s v2	12	224 GiB	1,474 GiB	2X P100	\$0.72/hour	\$4.14/hour	\$2.6371/hour (~36%)	\$1.8378/hour (~56%)	\$0.72/hour (~83%)
+	NC24rs v2	24	448 GiB	2,948 GiB	4X P100	\$1.584/hour	\$9.108/hour	\$5.8015/hour (~36%)	\$4.0430/hour (~56%)	\$1.584/hour (~83%)
+	NC24s v2	24	448 GiB	2,948 GiB	4X P100	\$1.44/hour	\$8.28/hour	\$5.2742/hour (~36%)	\$3.6755/hour (~56%)	\$1.44/hour (~83%)

<https://azure.microsoft.com/en-us/pricing/details/batch/>

How can I  
monitor my  
Batch?

The screenshot displays the Batch Labs interface. On the left, a sidebar lists various jobs and pools, with 'test-many-tasks' selected. The main panel shows a table of tasks for 'test-many-tasks', all of which are in a 'completed' state. To the right of the table, there are job statistics including a gauge for 'Running tasks' (0) and a bar chart for 'Succeeded' (20000) and 'Failed' (0) tasks. The bottom status bar indicates 'No current background tasks'.

Id	State	Created	Started	Completed	Exit code
task_0_0	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_1	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_10	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_11	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_12	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_13	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_14	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_15	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_16	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_17	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_18	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_19	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_2	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_20	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_21	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_22	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_23	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_24	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_25	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_26	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_27	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_28	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_29	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_3	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0
task_0_30	completed	Nov 17, 2017	Nov 17, 2017	Nov 17, 2017	0

# Conclusions

- If you have normal experiments, use Azure Machine Learning
- If you are working in some crazy stuff go straight to Azure Batch using containers.
- Never, use your laptop for deep learning...

Thank U

&

Questions