cloud-GC Documentation

Release 0.1

Jiawei Zhuang

Mar 12, 2018

Contents

GEOSChem-on-cloud project aims to build a cloud computing capability for GEOS-Chem that can be easily accessed by researchers worldwide.

See *Why move to the cloud* for the motivation of this project. See *Quick start guide for new users* to start your first GEOS-Chem simulation on the Amazon Web Services (AWS) cloud within 10 minutes (and within seconds for the next time).

This project is supported by the AWS Public Data Set Program and the NASA Atmospheric Composition Modeling and Analysis Program (ACMAP).

Warning: *GEOSChem-on-cloud* is at initial development and things are moving very fast. Please use the GitHub issue tracker to request new funtionalities, report bugs, or just discuss general issues.

CHAPTER 1

Table of Contents

1.1 Overview

This chapter provides a high-level overview of cloud computing.

1.1.1 Why move to the cloud

Remove technical barriers

Atmospheric scientists often need to waste time on non-science tasks: installing software libraries, making models compile and run without bugs, preparing model input data, or even setting up a Linux server.

Those technical tasks are getting more and more challenging – as atmospheric models evolve to incorporate more scientific understandings and better computational technologies, they also need more complicated software, more computing power, and much more data.

Cloud computing can largely alleviate those problems. **The goal of this project is to allow researchers to fully focus on scientific analysis, not fighting with software and hardware problems.**

Software

On the cloud, you can launch a server with everything configured correctly. Once I have built the model and saved it as an Amazon Machine Image (AMI), anyone can replicate exactly the same software environment and start using the model immediately (see *Quick start guide for new users*). You will never see compile errors anymore.

This has more implications in the age of High-Performance Computing (HPC). Modern atmospheric models are often built with complicated software frameworks, notably the Earth System Modeling Framework (ESMF). Those frameworks allow model developers to utilize HPC technologies without writting tons of boilerplate MPI code, but they add extra burdens on model users – installing and configuring those frameworks is daunting, if not impossible, for a typical graduate student without a CS background. Fortunately, no matter how difficult it is to install those libraries, there only needs to be one person to build it once on the cloud. Then, no one needs to redo this labor again.

Note: This software dependency hell can also be solved by containers such as Docker and Singularity (e.g. Docker-WRF). But the cloud also solves compute and data problems, as discussed below. You can combine containers and cloud to have a consistent environment across local machines and cloud platforms.

Compute

Local machines need up-front investment and have fixed capability. Right before AGU, everyone is running models and jobs are pending forever in the queue. During Christmas, no one is working and machines are just idle but still incur maintenance cost.

Clouds are elastic. You can request an HPC cluster with 1000 cores for just 1 hour, and only pay for exactly that hour. If you have powerful local machines, you can still use the cloud to boost computing power temporarily.

Data

GEOS-Chem currently have 30 TB of GEOS-FP/MERRA2 meteorological input data. With a bandwidth of 1 MB/s, it takes two weeks to download a 1-TB subset and a year to download the full 30 TB. To set up a high-resolution nested simulation, one often needs to spend long time getting the corresponding meteorological fields. GCHP can ingest global high-resolution data and will further push the data size to increase.

The new paradigm to solve this big data challenge is to "move compute to data", i.e. perform computing directly in the cloud environment where data is already available. (also see *Massive Earth observation data*). AWS has agreed to host all GEOS-Chem input data for free under the Public Data Set Program. By having all the data already available in the cloud environment, you can perform simulations over any periods with any configurations.

Open new research opportunities

Cloud not only makes model simulations much easier, but also opens many new research opportunities in Earth science.

Massive Earth observation data

Massive amounts of satellite and other Earth science data are being moved to the cloud. One success story is the migration of NOAA's NEXRAD data to AWS (Ansari et al., 2017, BAMS) – it is reported that "data access that previously took 3+ years to complete now requires only a few days" (NAS, 2018, Chapter "Data and Computation in the Cloud"). By learning cloud computing you can get access to massive Earth science datasets on AWS, without having to spend long time downloading them to local machines.

The most exciting project is perhaps the cloud migration of NASA's Earth Observing System Data and Information System (EOSDIS). It will open new opportunities such as ultra-high-resolution inversion of satellite data, leveraging massive data and computing power available on the cloud. This kind of analysis is hard to imagine on traditional platforms.

Machine learning and deep learning

There is a growing interest in applying machine learning in Earth science, as illustrated clearly by the AGU 2017 Fall meeting (H082, A028) and AMS 2018 meeting (AMS-AI, and its summary).

Cloud platforms are the go-to choice for training machine learning models, especially deep neural networks. There are massive amounts of GPUs on the cloud, which can offer ~50x performance than CPUs for training neural nets. Pre-configured environment on the cloud (e.g. AWS Deep Learning AMI) allows users to run the program immediately without wasting time configuring GPU libraries.

Instructions on using cloud are often included in the official documentations of ML/DL frameworks:

- Keras on AWS GPU. Keras is the most popular high-level deep learning library, built on top of TensorFlow.
- XGBoost on AWS cluster. XGBoost is the most popular library for gradient boosting, and is also the most widely used tool in Kaggle.

... and in deep learning textbooks and course materials:

- Stanford CS231n: Convolutional Neural Networks for Visual Recognition. See Google Cloud Tutorial and AWS Tutorial. (CS231n should be one of the most popular deep learning courses, with all videos and materials freely available online)
- Deep learning with Python. by François Chollet, the author of Keras. See Appendix B. Running Jupyter notebooks on AWS GPU. (This book got full 5-star on Amazon)
- Deep Learning - The Straight Dope. It is a very nice interactive textbook on deep learning. Its official Chinese version has an instruction on using AWS. See AWS official docs for the equivalent instruction in English.

1.1.2 Status of cloud for scientific computing

Cloud was originally invented for web applications, not for scientific computing. But the interest in using cloud platforms for science is growing rapidly, especially in the recent 2~3 years. Technical tests on whether cloud is suitable for science have been done over 10 years, tracing back to Evangelinos and Hill (2008) who tested MITgcm on AWS. Now we start to see mature applications for daily research work – the democratization of cloud computing!

Atmospheric models

The Modeling Research in the Cloud Workshop was hosted by NCAR in 2017. Most presentation slides are available online.

Relavant applications

- OpenFOAM on cloud. OpenFOAM is a popular library for computational fluid dynamics (CFD). The team started cloud migration in 2015 and the project is now very mature. CFD simulations are very similar to atmospheric simulations from a computational perspective – they both solve variants of Navier–Stokes equations and use MPI-based domain decomposition.

University Classes

Scientific computing classes start to teach and use cloud platforms, mostly AWS:

- Harvard CS205, 2018 Spring, Computing Foundations for Computational Science
- MIT 18.337/6.338, 2017 Fall, Modern Numerical Computing in Julia
- Duke STA663, 2017, Computational Statistics in Python
- Also see *Machine learning and deep learning*

(If you know more about them please let me know!)

1.1.3 External resources on cloud computing for science

The majority of cloud computing textbooks and AWS documentations are written for web developers and system architects, NOT for domain scientists who just want to do scientific computing and data analysis. As a researcher with limited IT background, it is crucial the pick up the correct tutorial when learning cloud computing. Here are my recommendations:

[1] **University of Washington** has a very nice high-level overview and technical documentation about cloud computing for scientific research.

[2] **Cloud Computing for Science and Engineering** (Foster and Gannon 2017) is the first textbook I am aware of that provides hands-on tutorials for domain scientists. The book is free available online.

[3] **Cloud Computing in Ocean and Atmospheric Sciences** (Vance et al. 2016) gives a nice overview of various cloud computing applications in our field. It doesn't tell you how to actually use the cloud, though.

[4] **Researcher's Handbook by AWS** is the most useful AWS material for you (as a scientist, not an IT person). You will need to sign-up the AWS Research Cloud Program to download the PDF file.

1.2 Beginner tutorials

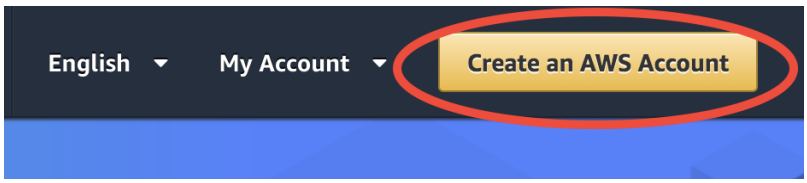
This chapter provides hands-on tutorials for absolute beginners with no cloud computing experience.

The beginner tutorial is aimed to be read in-order!

1.2.1 Quick start guide for new users

Step 1: Sign up an Amazon Web Service(AWS) account

Go to http://aws.amazon.com, click on "Create an AWS account" on the upper-right corner:



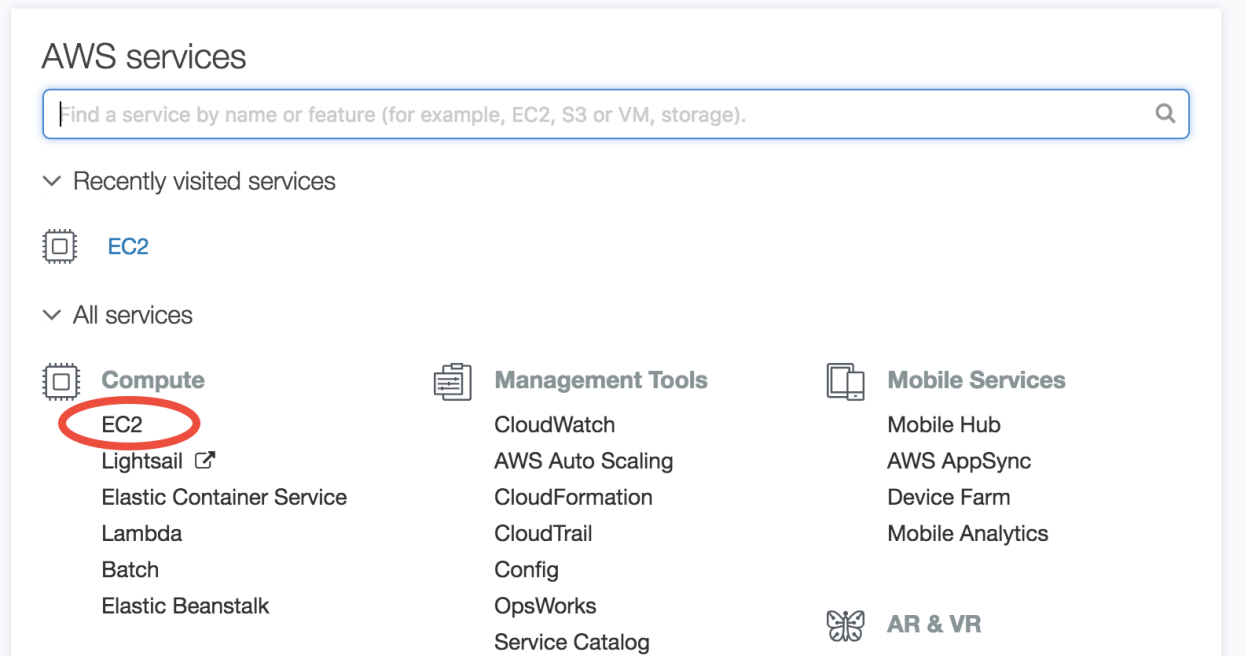
(the button will become "Sign In to the Console" for the next time)

After entering some basic information, you will be required to enter your credit card number. Don't worry, this beginner tutorial will only cost you $0.1.

Note: If you are a student, check out the $100 educational credit (can be renewed every year!) at https://aws.amazon.com/education/awseducate/. I haven't used up my credit for after playing with AWS for a whole year, so haven't actually paid any money to them

---

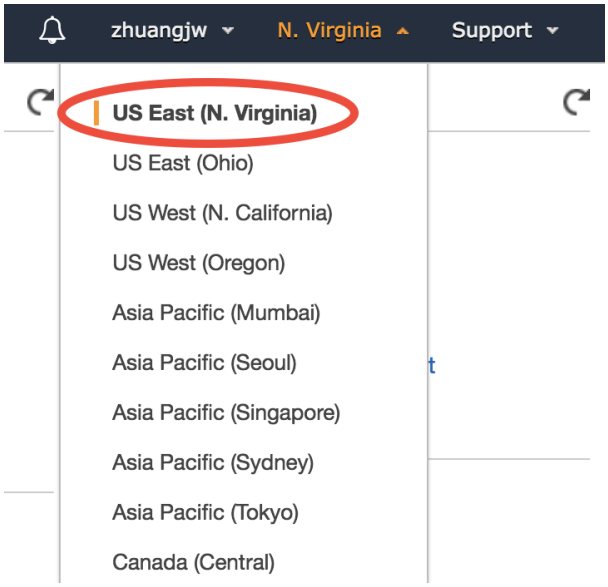
Now you should have an AWS account! It's time to run the model in cloud. (You can skip Step 1 for the next time, of course)



Step 2: Launch a server with GEOS-Chem pre-installed

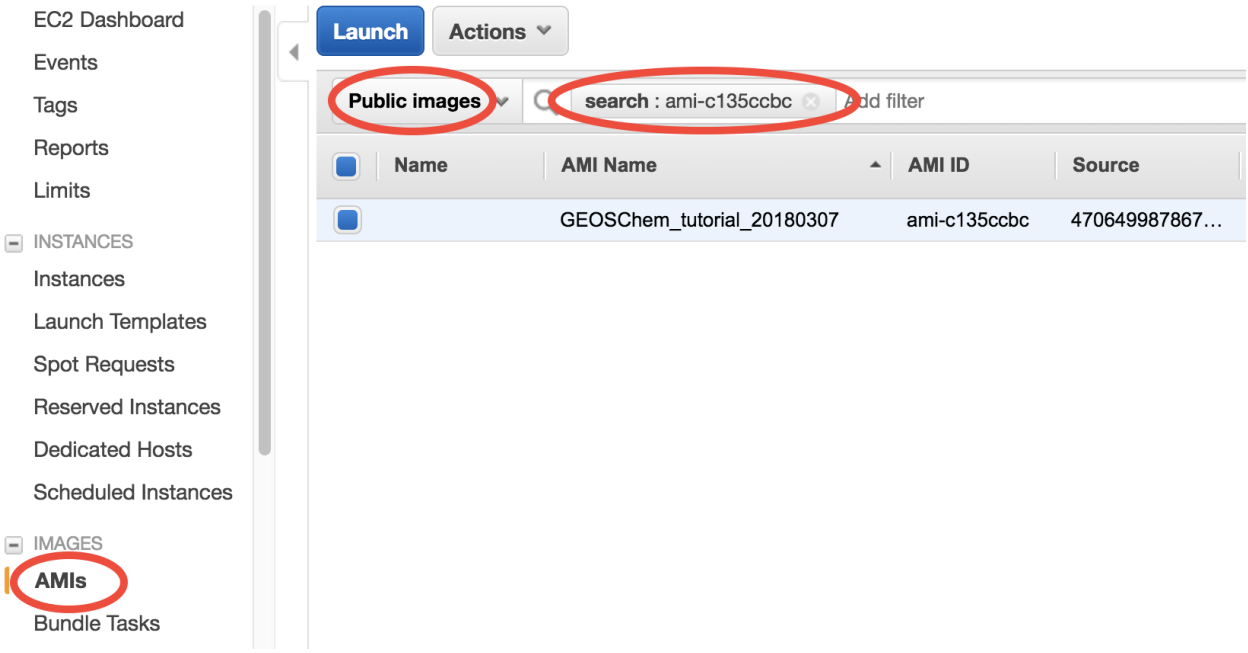
Log in to AWS console, and click on EC2 (Elastic Compute Cloud), which is the most basic cloud computing service.

In the EC2 console, make sure you are in the **US East (N. Virginia)** region as shown in the upper-right corner of your console. Choosing a region closer to your physical location will give you better network. To keep this tutorial minimal, I built the system in only one region. But working across regions is not hard.



In the EC2 console, click on "AMI" (Amazon Machine Image) under "IMAGES" on the left navigation bar. Then select "Public images" and search for **ami-c135ccbc** or **GEOSChem_tutorial_20180307** – that's the system with GEOS-Chem installed. Select it and click on "Launch".

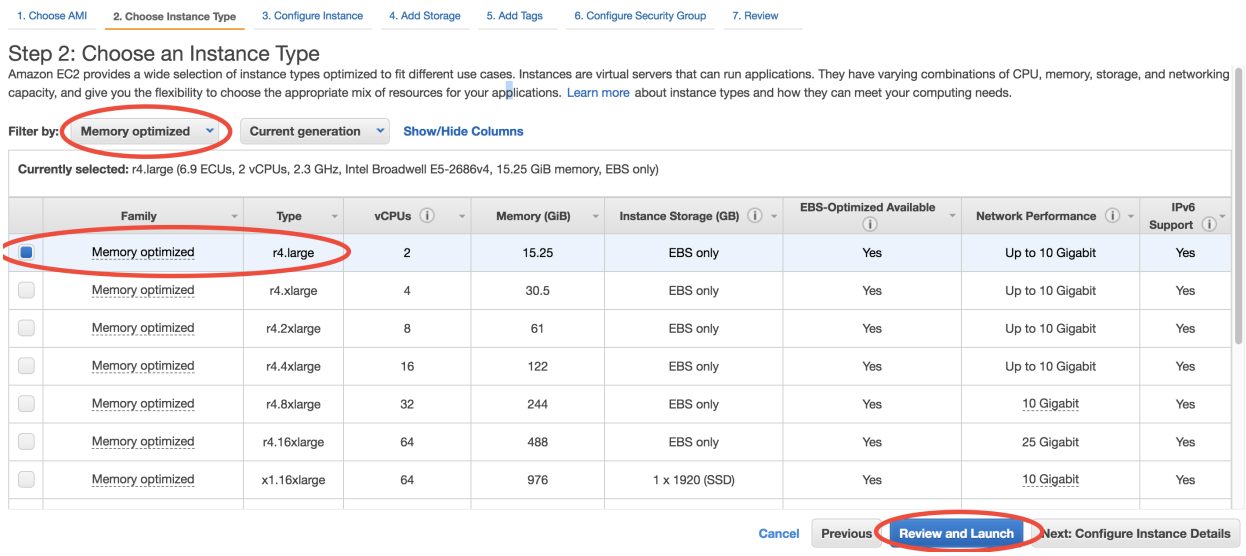
This is one of the game-changing features of cloud computing. An AMI means a saved system. I started with a brand new Linux system and built GEOS-Chem on it. After that, everyone is able to get a perfect clone of my



system, with everything installed correctly. This magic can hardly happen on traditional machines! You can make any modifications you like to your copy, such as changing the model code, downloading more data or installing additional software. If you screw things up (e.g. install some bad software, delete important files...), you can simply launch again and start over.

You have already specified your operating system, or the "software" side of the virtual server. Then it's time to specify the "hardware" side, mostly about CPUs.

In this toy example, choose "Memory optimized"-"r4.large" to test GEOS-Chem with the minimum fee.



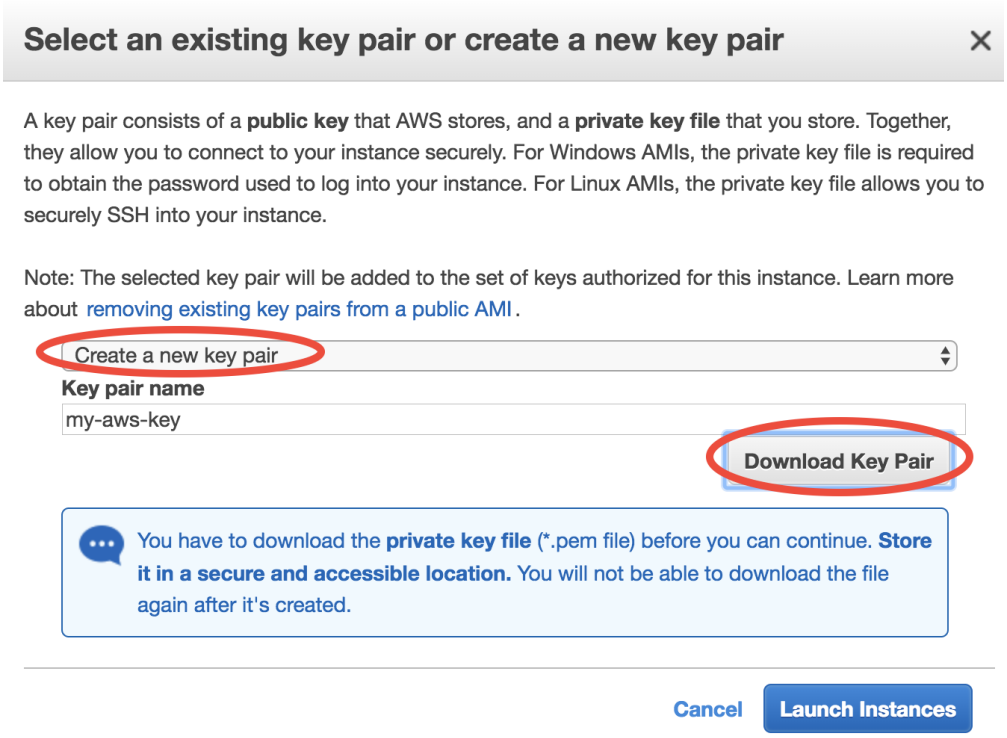
There are many CPU options, including numbers and types. AWS free tier also gives you 750 free hours of "t2.micro", which is the tiniest CPU. Its memory is too small to run GEOS-Chem, but it is good for testing software installation if you need to.

Then, just click on "Review and Launch". You don't need to touch other options this time. This brings you to "Step

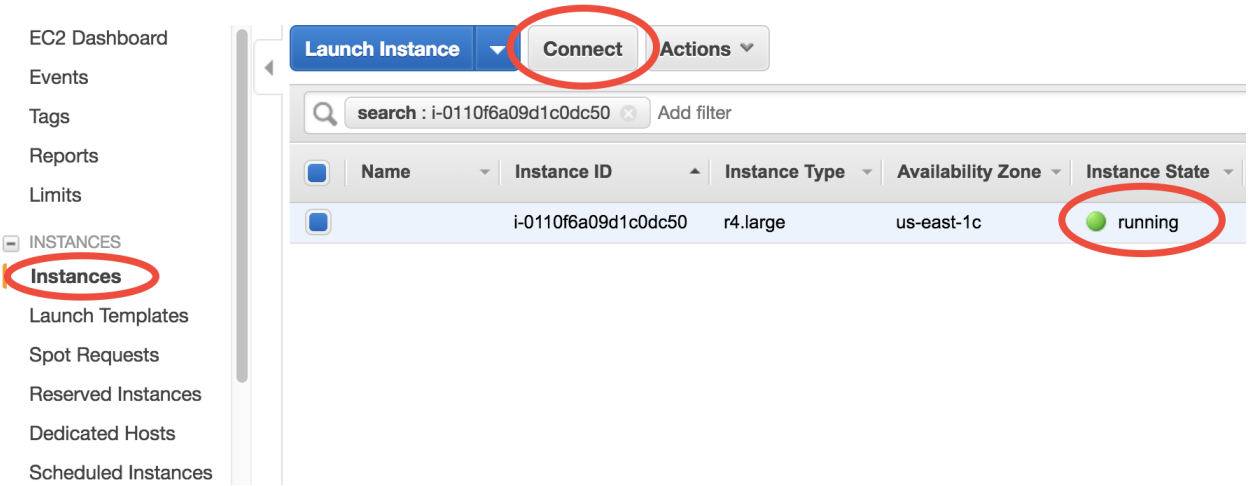
7: Review Instance Launch". Simply click on the Launch button again.

For the first time of using EC2, you need to create and download a "Key Pair". This is equivalent to the password you enter to ssh to your local server, but much safer than a normal password. Here, such "password" is a file, being stored in your own computer. The only way to share your server password with others is to share that file.

Give your KeyPair a name, click on "Download Key Pair", and finally click on "Launch Instances". (for the next time, you can simply select "Choose an existing Key Pair" and launch).



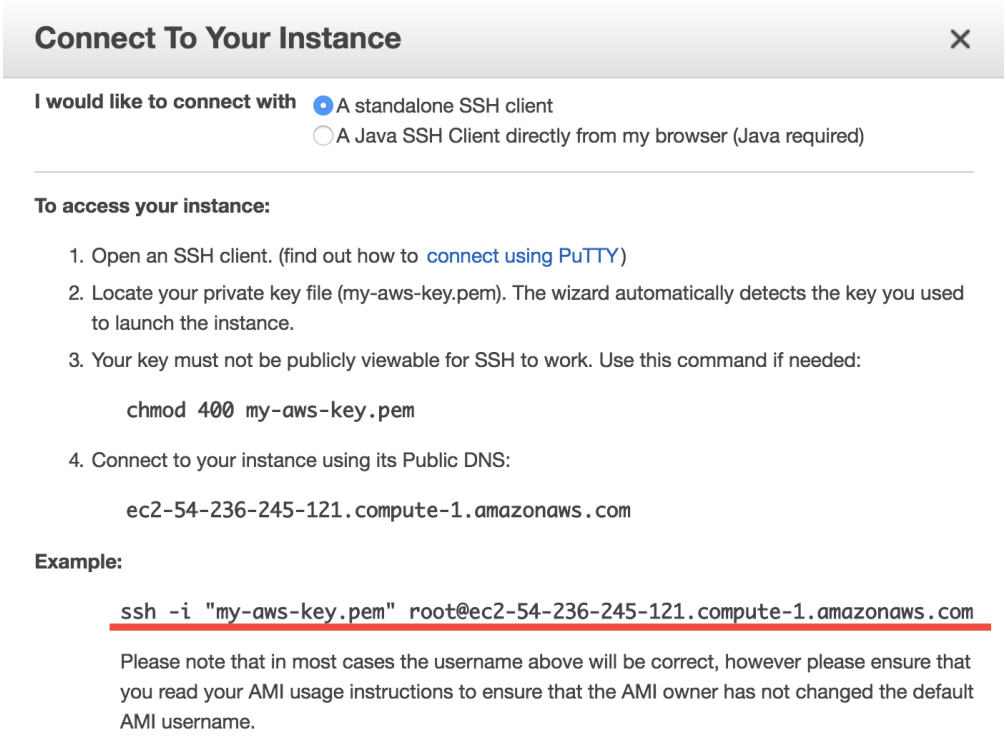
You can monitor your server in the EC2-Instance console. Within < 1min of initialization, "Instance State" should become "running":



You now have your own server running on the cloud!

Step 3: Log into the server and run GEOS-Chem

Select your instance, click on the "Connect" button near the blue "Launch Instance" button, then you should see this page:



- On Mac or Linux, copy the `ssh -i "xx.pem" root@xxx.com` command under "Example". Before using that command to ssh to your server, do some minor stuff:
  1. `cd` to the directory where store your Key Pair (preferably `$HOME/.ssh`)
  2. Use `chmod 400 xx.pem` to change the key pair's permission (also mentioned in the above figure; only need to do this at the first time).
  3. Change the user name in that command from `root` to `ubuntu`. (You'll be asked to use `ubuntu` if you keep `root`).
- On Windows, please refer to the guide for MobaXterm and Putty (Your life would probably be easier with MobaXterm).

Your terminal should look like this:

That's a system with GEOS-Chem already built!

Note: Trouble shooting: if you have trouble `ssh` to the server, please *make sure you don't mess-up the "security group" configuration*.

Go to the pre-generated run directory:

```
$ cd geosfp_4x5_standard
```

Just run the pre-compiled the model by:

```
.ssh $chmod 400 my-aws-key.pem
.ssh $ssh -i "my-aws-key.pem" ubuntu@ec2-54-236-245-121.compute-1.amazonaws.com
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-1049-aws x86_64)

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

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

37 packages can be updated.
0 updates are security updates.


*** System restart required ***
Last login: Thu Mar  8 05:16:24 2018 from 50.234.189.5
ubuntu@ip-172-31-36-170:~$ ls
gc_bleeding_edge  gcdata  geosfp_4x5_standard  miniconda  python_example  ut_bleeding_edge
ubuntu@ip-172-31-36-170:~$
```

```
$ ./geos.mp
```

Or you can re-compile the model on your own:

```
$ make clean
$ make -j4 mpbuild NC_DIAG=y BPCH_DIAG=n TIMERS=1
```

Congratulations! You've just done a GEOS-Chem simulation on the cloud, without spending any time on setting up your own server, configuring software environment, and preparing model input data!

The default simulation length is only 20 minutes, for demonstration purpose. The "r4.large" instance type we chose has only a single, slow core (so it is cheap, just ~$0.1/hour), while its memory is large enough for GEOS-Chem to start. For serious simulations, it is recommended to use "Compute Optimized" instance types with multiple cores such as "c5.4xlarge".

Note: The first simulation on a new server will have slow I/O and library loading because the disk needs "warm-up". Subsequent simulations will be much faster.

Step 4: Analyze output data with Python (Optional)

If you wait the simulation to finish (takes 5~10 min), it will produce NetCDF diagnostics called `GEOSChem.inst.20130701.nc4`. There is also a pre-generated `GEOSChem.inst.20130701_backup.nc4` ready for you to analyze:

```
ubuntu@ip-172-31-36-170:~/geosfp_4x5_standard$ ncdump -h GEOSChem.inst.20130701_
↪backup.nc4
netcdf GEOSChem.inst.20130701_backup {
dimensions:
      time = UNLIMITED ; // (1 currently)
      lev = 72 ;
      ilev = 73 ;
      lat = 46 ;
```

(continues on next page)

(continued from previous page)

```
      lon = 72 ;
variables:
      double time(time) ;
              time:long_name = "Time" ;
              time:units = "minutes since 2013-07-01 00:00:00 UTC" ;
              time:calendar = "gregorian" ;
              time:axis = "T" ;
```

Anaconda Python and xarray are already installed on the server for analyzing all kinds of NetCDF files. If you are not familiar with Python and xarray, checkout my tutorial on xarray for GEOS-Chem. You can simply use `ipython` from the command line:

```
ubuntu@ip-172-31-36-170:~/geosfp_4x5_standard$ ipython
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 18:10:19)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import xarray as xr

In [2]: ds = xr.open_dataset("GEOSChem.inst.20130701_backup.nc4")

In [3]: ds
Out[3]:
<xarray.Dataset>
Dimensions:          (ilev: 73, lat: 46, lev: 72, lon: 72, time: 1)
...
    SpeciesConc_CO   (time, lev, lat, lon) float32 ...
    SpeciesConc_O3   (time, lev, lat, lon) float32 ...
    SpeciesConc_NO   (time, lev, lat, lon) float32 ...
```

A much better data-analysis environment is Jupyter notebooks. If you have been using Jupyter on your local machine, the user experience on the cloud would be exactly the same.

To use Jupyter on remote servers, re-login to the server with port-forwarding option `-L 8999:localhost:8999`:

```
$ ssh -i "xx.pem" ubuntu@xxx.com -L 8999:localhost:8999
```

Then simply run `jupyter notebook --NotebookApp.token='' --no-browser --port=8999`:

```
ubuntu@ip-172-31-36-170:~$ jupyter notebook --NotebookApp.token='' --no-browser --
↪port=8999
[I 21:11:41.503 NotebookApp] Writing notebook server cookie secret to /run/user/1000/
↪jupyter/notebook_cookie_secret
[W 21:11:41.986 NotebookApp] All authentication is disabled.  Anyone who can connect␣
↪to this server will be able to run code.
[I 21:11:42.046 NotebookApp] Serving notebooks from local directory: /home/ubuntu
[I 21:11:42.046 NotebookApp] 0 active kernels
[I 21:11:42.046 NotebookApp] The Jupyter Notebook is running at:
[I 21:11:42.046 NotebookApp] http://localhost:8999/
[I 21:11:42.046 NotebookApp] Use Control-C to stop this server and shut down all␣
↪kernels (twice to skip confirmation).
```

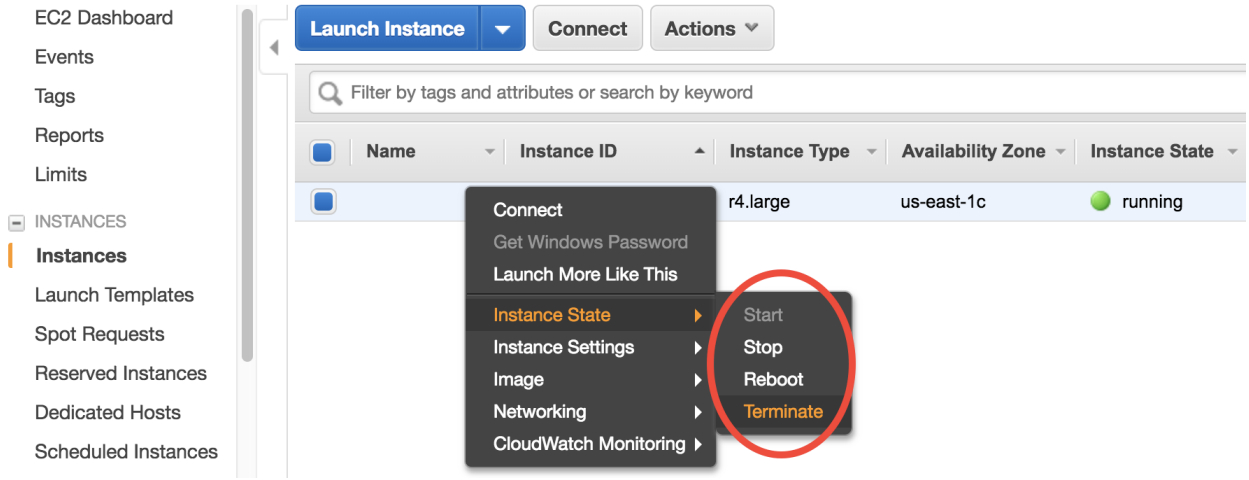
Visit `http://localhost:8999/` in your browser, you should see a Jupyter environment just like on local machines. The server contains an *example notebook* `python_example/plot_GC_data.ipynb` that you can just execute.

Note: There are many ways to use Jupyter on remote servers. Port-forwarding is the easiest way, and is the only way

that also works on local HPC clusters (which has much stricter firewalls than cloud platforms). The port number 8999 is just my random choice, to distinguish from the default port number 8888 for local Jupyter. You can use whatever number you like as long as it doesn't conflict with existing port numbers.

Step 5: Shut down the server (Very important!!)

Right-click on the instance in your console to get this menu:



There are two different ways to stop being charged:

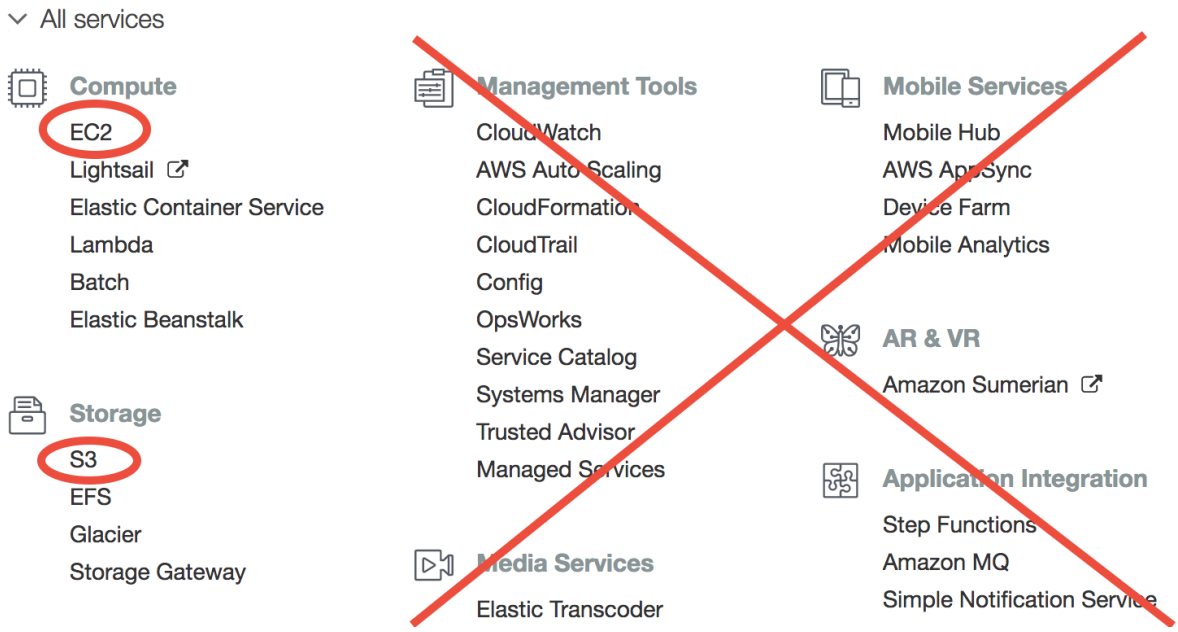
- "Stop" will make the system inactive, so that you'll not be charged by the CPU time, and only be charged by the negligible disk storage fee. You can re-start the server at any time and all files will be preserved.
- "Terminate" will completely remove that virtual server so you won't be charged at all after that. Unless you save your system as an AMI or transfer the data to other storage services, you will lose all your data and software.

You will learn how to save your data and configurations persistently in the next tutorials.

1.2.2 Overview of basic AWS compute and storage services

In the *quick start guide*, you have used **EC2 (Elastic Compute Cloud)** to perform simulations and data analysis. Unlike your local server, cloud platforms are designed to be ephemeral – you can launch or shut down servers at any time. No up-front investment, no hardware maintenance. This flexibility is a great advantage of cloud platforms, but it also means that you need to take special care of your data. On local servers, one can simply log out after the work is done. But if you simply terminate your cloud server, all data will disappear. Instead of keeping your server running (which incurs hourly cost), a much cheaper and cleverer way is to move data to other storage services.

AWS has hundreds of services (shown in the main console; see the figure below), and EC2 is just one of them. Fortunately, a tiny subset of services is enough for scientific computing. The most important services are **EC2** for compute and **S3** for storage. Tons of other services are targetted at IT/Business applications that scientists can safely ignore:



Core AWS concepts for scientific computing

In this section you will familarize yourself with the following concepts and their costs: EC2, Spot Instances, AMI, EBS, S3, and Data egress charge.

- EC2 (Elastic Compute Cloud) is the major computing service. You can create any number of servers (called "EC2 instances" in AWS context). They are just like normal servers that you can `ssh` to, to perform various computing tasks. Unlike local servers that have fixed hardware capacity and static software environment, EC2 instances are highly-customizable. For hardware, there are tons of EC2 instances types with different capacities in CPUs, memory, and disk storage. For software, you can start with a brand new operating system, or use other people's system images as you did in the quick start guide. The price of EC2 is roughly $0.01 /CPU/hour.

Note: EC2 used to charge by hours but it now uses per-second billing. That's a great saving for short simulations – a 10-min simulation is only charged as 1/6 hour.

- Spot instances are a special pricing model for EC2 that is particularly suitable for scientific computing. It can reduce EC2 cost by ~70%. You will learn how to use it later in this tutorial.
- AMI (Amazon Machine Image) is a frozen system image of an EC2 instance. It contains the operating system, the software libraries, and all the files on a server. An AMI can be used to create any number of EC2 instances. Once a model is configured and saved as an AMI, any researchers can replicate the same environment and start using the model instantly. Some good examples are the Deep Learning AMI and the OpenFOAM AMI.
- EBS (Elastic Block Storage) is a disk storage service to increase the disk capacity of existing EC2 instances. You create an "EBS volume" and "attach" it to an EC2 instance, just like attaching a USB drive to a computer. You will learn how to do this later. The default disk storage of the EC2 instance itself is also an "EBS volume". EBS is suitable for temporarily hosting files that you are directly working with. For long-term, persisent storage, S3 (see below) is a much better option. The price of EBS volumes is $0.1/GB/month.
- S3 (Simple Storage Service) is the major storage service on AWS. Unlike traditional hard disk storage, S3 uses object storage model which is much more scalable. Traditional disks have limited storage capacity – once you hit the limit you need to either delete some data or buy new disks; EBS volumes are just like physical disks so also have limits, although you can create new volumes easily; **S3, on the other hand, has almost no capacity**

limit – you can dump as many data into it as you like. The price of S3 is $0.023/GB/month, only 23% of EBS price.

S3 is thus the default storage mechanism for most of *Earth Science Big Data*. Later in this tutorial you will learn how to access all 30 TB of GEOS-Chem input data on S3, as well as other public Earth Science data on AWS. You will also upload your own data (e.g. model simulation results) to S3.

- **Data egress charge** is an additional charge besides compute (EC2) and storage (EBS, S3). While **transfering data into the cloud is free**, almost all commercial cloud providers **charge for data transferring out of their cloud** – that's how they make money and encourage people to keep stuff within their cloud. The data egress fee on AWS is $0.09/GB. AWS does also offer Data egress discount to researchers, but the discount cannot exceed 15% of total AWS cost.

  The data egress fee is actually not a big worry because:

  1. For small data (~GBs), the cost is quite low.
  2. For large data (~TBs), downloading takes a long time so we would like to avoid downloading them anyway. After all, the key idea of cloud computing is "bringing compute to data". With Python and Jupyter, analyzing simulation results on the cloud is just as convenient as having data locally available.

See "AWS services in human language" for a more complete review of AWS services.

1.2.3 Set up AWS Command Line Interface (AWS-CLI)

Before using the *S3 storage*, you need to set up AWSCLI first. AWSCLI is a command line tool that can replicate everything you can do with the graphical console. It can control hundreds of AWS services, but the major use case is S3. The initial setup takes some time, but this is a one-off effort, so be patient!

Install AWSCLI

If you are on the EC2 instance launched from my tutorial AMI, AWSCLI is already installed:

```
ubuntu@ip-172-31-46-2:~$ which aws
/home/ubuntu/miniconda/bin/aws
```

You can also use AWSCLI on your own computer (which is a very common practice):

```
pip install awscli
```

Note: AWSCLI is a like "Python package" that you can install into your Anaconda environment. However, it is directly used in the shell (or bash shell scripts), not an "importable package" in Python code. The actual Python tool to control AWS resources is boto3. For most cases, AWSCLI is enough. For very complicated administrative tasks (say, launching hundreds of EC2 instances on a dedicated schedule) that are too complicated for shell scripts, consider writing Python scripts with boto3. We will not cover boto3 in this tutorial as it is generally an overkill.

Then try `aws help` to get general help information, or `aws s3 help` to get help specifically on S3.

Try to configure AWSCLI

If you try any actual commands like `aws s3 ls` (listing your S3 resources), it will fail:

```
ubuntu@ip-172-31-46-2:~$ aws s3 ls
Unable to locate credentials. You can configure credentials by running "aws configure
↪".
```

That's because you haven't configured your AWS account. Think about this: if you are using AWSCLI on your local computer, you can control your AWS resources **without** logging into the graphical console. Then, how can AWS know that it is you accessing your account? Thus you need to save your account information somewhere on your computer.

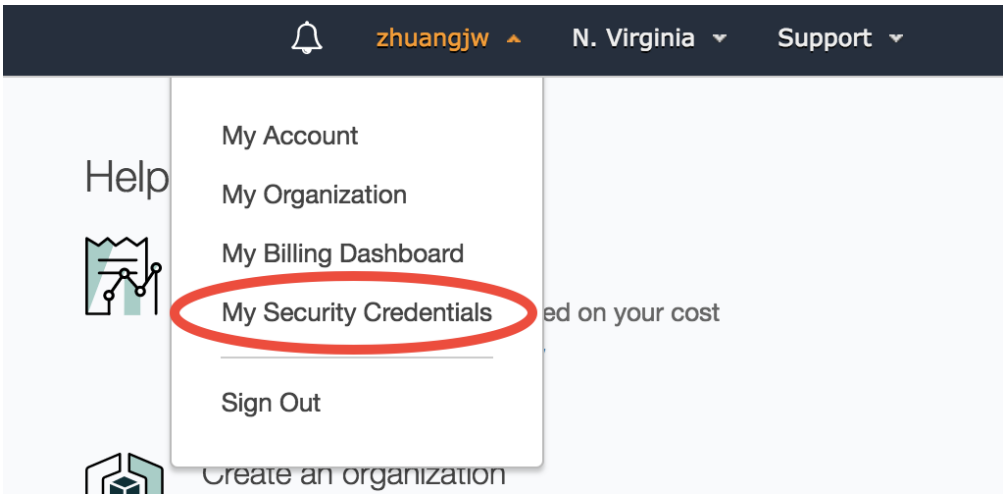
Running `aws configure`, you will be asked 4 questions:

```
ubuntu@ip-172-31-46-2:~$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

"Access Key ID" and "Secret Access Key" are just like your AWS account name and password. For security reasons they are not the one you use to log into the console. You can obtain them in the "My Security Credentials" console.

Obtaining security credentials

Click on your account name on the upper right corner of AWS console to get this memu:



Then you will likely be prompted with this warning:

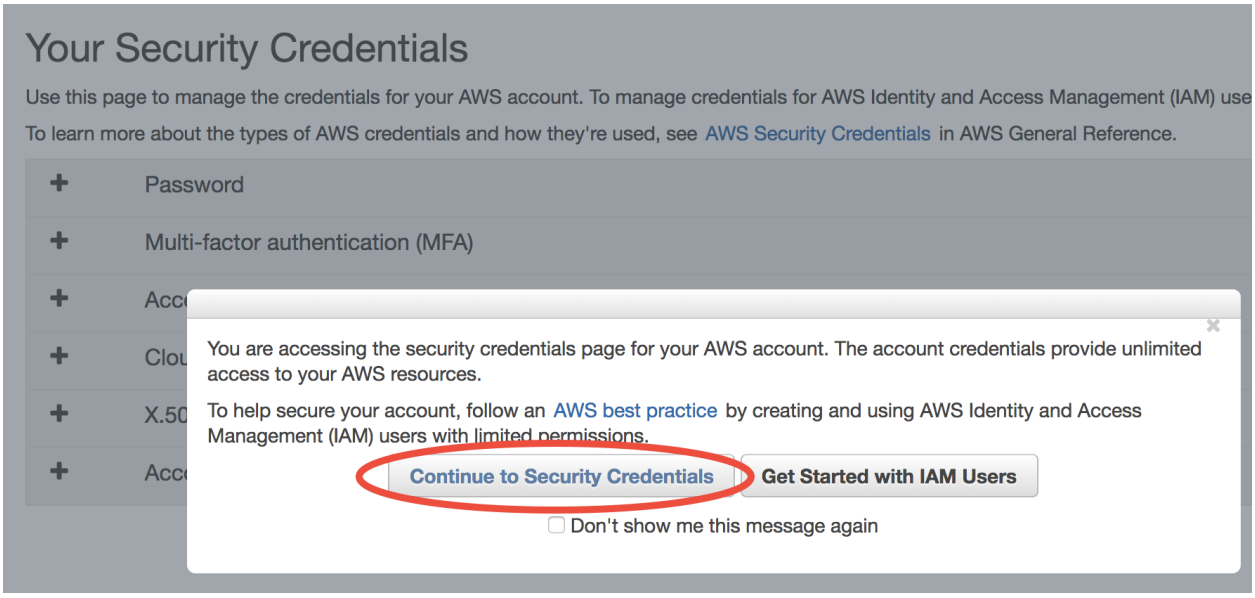
AWS encourages you to create "IAM users" instead of simply getting your Security Credentials. However, I find the concept of "IAM users" a bit overwhelming for who just starts to use the cloud. As scientists we want to get the science done as quickly as possible, so I defer the concept of "IAM users" to advanced tutorials. For now, simply choose "Continue to Security Credentials".

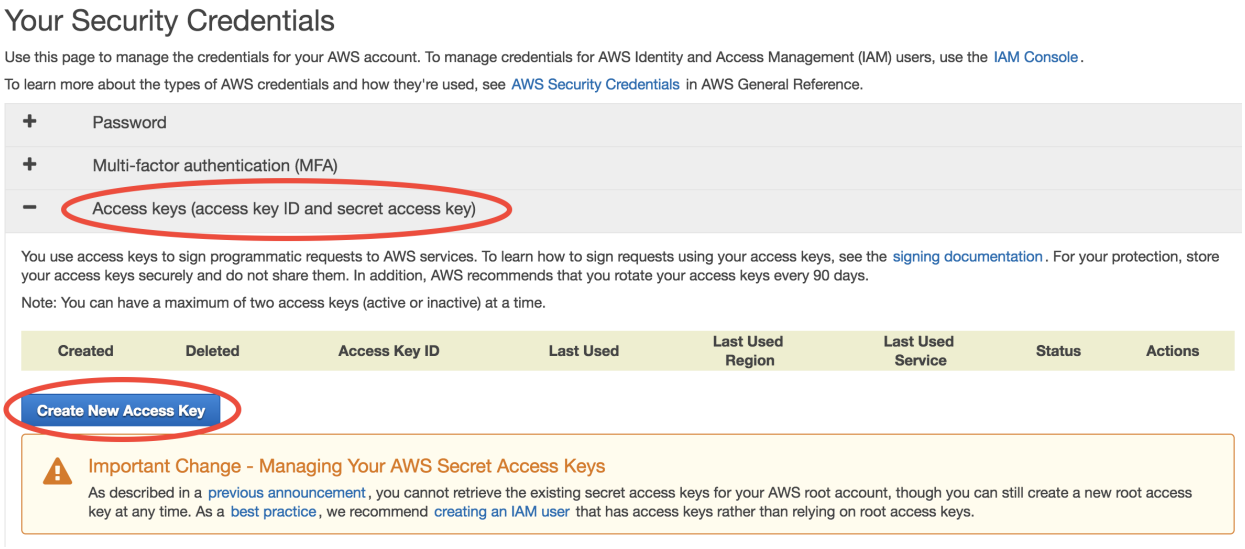
In the Security Credential console, choose Access keys – Create New Access Key:

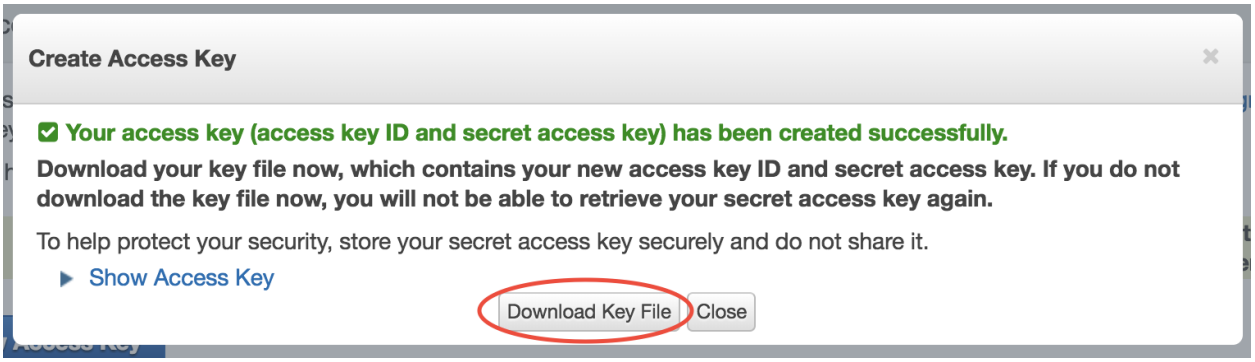
In the prompted window, click on "Download Key File". That's the only chance you can download the key file. If you forget to download it before closing the window, simply delete your key and create a new one.

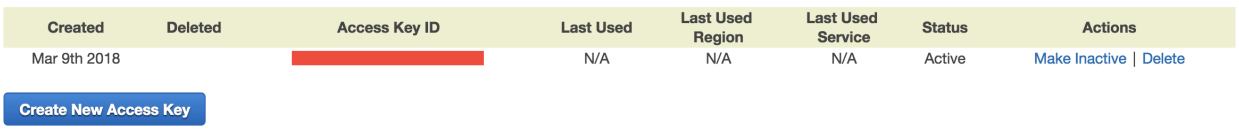
Then your new key will appear in the console. Here you can only see the "Access Key ID" (like your AWS account), but not the "Secret Access Key" (like password, has much more digits than ID, and is only visible in the downloaded Key File).

Keep your Key File in a safe place. Now we can answer `aws configure` questions.


Your Security Credentials
Use this page to manage the credentials for your AWS account. To manage credentials for AWS Identity and Access Management (IAM) use
To learn more about the types of AWS credentials and how they're used, see AWS Security Credentials in AWS General Reference.
Password
Multi-factor authentication (MFA)
You are accessing the security credentials page for your AWS account. The account credentials provide unlimited access to your AWS resources.
To help secure your account, follow an AWS best practice by creating and using AWS Identity and Access Management (IAM) users with limited permissions.
Continue to Security Credentials
Get Started with IAM Users
Don't show me this message again


Your Security Credentials
Use this page to manage the credentials for your AWS account. To manage credentials for AWS Identity and Access Management (IAM) users, use the IAM Console.
To learn more about the types of AWS credentials and how they're used, see AWS Security Credentials in AWS General Reference.
Password
Multi-factor authentication (MFA)
Access keys (access key ID and secret access key)
You use access keys to sign programmatic requests to AWS services. To learn how to sign requests using your access keys, see the signing documentation. For your protection, store your access keys securely and do not share them. In addition, AWS recommends that you rotate your access keys every 90 days.
Note: You can have a maximum of two access keys (active or inactive) at a time.
Created
Deleted
Access Key ID
Last Used
Last Used Region
Last Used Service
Status
Actions
Create New Access Key
Important Change - Managing Your AWS Secret Access Keys
As described in a previous announcement, you cannot retrieve the existing secret access keys for your AWS root account, though you can still create a new root access key at any time. As a best practice, we recommend creating an IAM user that has access keys rather than relying on root access keys.


Create Access Key
Your access key (access key ID and secret access key) has been created successfully.
Download your key file now, which contains your new access key ID and secret access key. If you do not download the key file now, you will not be able to retrieve your secret access key again.
To help protect your security, store your secret access key securely and do not share it.
Show Access Key
Download Key File
Close


Created
Deleted
Access Key ID
Last Used
Last Used Region
Last Used Service
Status
Actions
Mar 9th 2018
N/A
N/A
N/A
Active
Make Inactive | Delete
Create New Access Key

Finish AWSCLI configuration

Copy and paste your Key ID and Secret Key from the Key File:

```
ubuntu@ip-172-31-46-2:~$ aws configure
AWS Access Key ID [None]: xxxxxxxxxxxxxxxxxxx
AWS Secret Access Key [None]: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Default region name [None]: us-east-1
Default output format [None]: json
```

- For the default region, enter `us-east-1` . It is just an alias to the "US East (N. Virginia)" region that you chose in the quick start guide. Currently all GEOS-Chem resources are within in this region, so use it as default.
- For output format, enter `json`. (JSON is the most widely used format in web services. You don't need to worry about it right now. It looks almost the same as Python dictionaries and lists.)

The answers you typed are saved in `~/.aws/credentials` and `~/.aws/config`. You can rerun `aws configure` to overwrite them, just edit the files directly.

Now `aws s3 ls` should run smoothly. Since you don't have your own data on S3 yet, that command is likely to show nothing. However, you can already access tons of AWS Public Datasets. For example, let's view the NASA-NEX data by `aws s3 ls s3://nasanex/`:

```
ubuntu@ip-172-31-46-2:~$ aws s3 ls s3://nasanex/
                           PRE AVHRR/
                           PRE CMIP5/
                           PRE LOCA/
                           PRE Landsat/
                           PRE MAIAC/
                           PRE MODIS/
                           PRE NAIP/
                           PRE NEX-DCP30/
                           PRE NEX-GDDP/
```

You will learn how to retrieve and analyze those data in the next tutorial.

Another major use case of AWSCLI is to launch EC2 servers. You must already get tired of clicking through the EC2 console to launch a new server. You can actually launch a server with one single AWSCLI command, which is far more convenient than clicking tons of buttons. We defer this to advanced tutorials, as there are more important things to learn (S3, spot..) right now.

Additional notes

1. About various "keys"

Note: Secret Access Key? EC2 Key Pair? Why are there are so many keys? Do not be confused: the **AWS Secret Access Key** is tied to your AWS account itself, while the **EC2 Key Pair** is only for accessing a specific server. In general, the Access Keys are stored in `~/.aws/` as they are general AWS configurations; while EC2 Key Pairs are stored in `~/.ssh/`, as they are only for `ssh`.

It is totally fine to give your EC2 Key Pair to your friend to allow them to log into a your EC2 instances. You can easily create a new EC2 Key Pair to launch another EC2 instance that your friend have no access to. On the other hand, **NEVER** give you Secret Access Key to others. This will allow them to purchase AWS resources on your behalf!

---

2. Simplifying AWSCLI configuration on EC2

Note: If you are using AWSCLI on EC2 instances, not on your local computer, you might wonder why you still need to configure those credentials? Afterall, it's on AWS's server, and AWS should know that you are using AWSCLI on your own EC2 instances. Yes, you can avoid running `aws configure` everytime you launch a new EC2 instance. It is just not enabled by default because of security reasons. For example, you might want to allow your friend to log into your EC2 servers, but you don't want to let them control your other AWS resources using AWSCLI.

Enabling AWSCLI by default requires some understandings of IAM (Identity and Access Management), so we defer it to advanced tutorials. For now, simply copy and paste your credientials – it is pretty quick!

1.2.4 Use S3 as major storage

S3 is the most important storage service on AWS. Knowing how to use it crucial for almost any projects on the cloud.

Use S3 from the console

Please follow the official S3 tutorial to learn how to use S3 in the graphical console. It feels pretty much like Dropbox or Google Cloud Drive, which allows you to upload and download files by clicking on buttons. Before continuing, you should know how to:

1. Create a S3 bucket
2. Upload a file (also called "object" in S3 context) into that bucket
3. Download that file
4. Delete that file and Bucket

The S3 console is convenient for viewing files, but most of time you will use AWSCLI to work with S3 because:

- It is much easier to recurisively upload/download directories with AWSCLI.
- To transfer data between S3 and EC2, you have to use AWSCLI since there is no graphical console on EC2 instances.
- To work with public data set, AWSCLI is almost the only way you can use. Recall that in the previous chapter you use `aws s3 ls s3://nasanex/` to list the NASA-NEX data. But you cannot see the "s3://nasanex/" bucket in S3 console, since it doesn't belong to you.

Working with S3 using AWSCLI

On an EC2 instance launched from the GEOSChem tutorial AMI, configure AWSCLI by `aws configure` as in the *previous chapter* and make sure `aws s3 ls` can run without error.

Now, say you've made some changes to the `geosfp_4x5_standard/` run directory, such as tweaking model configurations in `input.geos` or running simulations to produce new diagnostics files. You want to keep those changes after you terminate the server, so you can retrieve them when you continue the work next time.

Create a new bucket by `aws s3 mb s3://your-bucket-name`. Note that S3 bucket names must be unique across all accounts, as this facilitates sharing data between different people (If others' buckets are public, you can access them just like how the owners access them). Getting a unique name is easy – it the name already exists, just add your name initials or some prefix. If you just use the name in the exmaple below, you are likely to get an `make_bucket failed` error since that bucket already exists in my account:

```
$ aws s3 mb s3://geoschem-run-directory
make_bucket: geoschem-run-directory
```

Then you can see your bucket by `aws s3 ls` (you can also see it in the S3 console)

```
$ aws s3 ls
2018-03-09 18:54:18 geoschem-run-directory
```

Now use `aws s3 cp local_file s3://your-bucket-name` to transfer the directory to S3 (add the `--recursive` option is to recursively copy a directory, just like the normal Linux command `cp -r`)

```
$ aws s3 cp --recursive geosfp_4x5_standard s3://geoschem-run-directory/
upload: geosfp_4x5_standard/FJX_spec.dat to s3://geoschem-run-directory/FJX_spec.dat
upload: geosfp_4x5_standard/HEMCO.log to s3://geoschem-run-directory/HEMCO.log
upload: geosfp_4x5_standard/HISTORY.rc to s3://geoschem-run-directory/HISTORY.rc
...
```

The default bandwidth between EC2 and S3 is ~100 MB/s so copying that run directory would just take seconds.

Note: To make incremental changes to existing S3 buckets, `aws s3 sync` is more efficient then `aws s3 cp`. Instead of overwriting the entire bucket, `sync` only write the files that have actually changed.

Now list your S3 bucket content by `aws s3 ls s3://your-bucket-name`:

```
$ aws s3 ls s3://geoschem-run-directory
2018-03-09 19:13:45        364 .gitignore
2018-03-09 19:13:45       9712 FJX_j2j.dat
2018-03-09 19:13:45      50125 FJX_spec.dat
...
```

You can also see all the files in the S3 console, which is a quite convenient way to view your data without launching any servers.

Then, try to get data back from S3 by swapping the arguments to `aws s3 cp`:

```
ubuntu@ip-172-31-46-2:~$ aws s3 cp --recursive s3://geoschem-run-directory/ rundir_
↪copy
download: s3://geoschem-run-directory/.gitignore to rundir_copy/.gitignore
download: s3://geoschem-run-directory/FJX_j2j.dat to rundir_copy/FJX_j2j.dat
download: s3://geoschem-run-directory/FJX_spec.dat to rundir_copy/FJX_spec.dat
...
```

Since your run directory is now safely living in the S3 bucket that is independent to any servers, terminating your EC2 instance won't cause data loss. You can use `aws s3 cp` to get data back from S3, on any number of newly-launched EC2 instances.

Warning: S3 is not a standard Linux file system and thus cannot preserve Linux file permissions. After retrieving your run directory back from S3, the executable `geos.mp` and `getRunInfo` will not have execute-permission by default. Simply type `chmod u+x geos.mp getRunInfo` to grant permission again.

Another approach to preserve permissions is to use `tar -zcvf` to compress your directory before loading to S3, and then use `tar -zxvf` to decompress it after retrieving from S3. Only consider this approach if you absolutely want to preverse all the permission information.

S3 also has no concept of symbolic links created by `ln -s`. By default, it will turn all links into real files by making real copies. You can use `aws s3 cp --no-follow-symlinks ...` to ignore links.

Those simplifications make S3 much more scalable (and cheaper) than normal file systems. You just need to be aware of those caveats.

Access NASA-NEX data in S3 (Optional but recommended)

Before accessing GEOS-Chem input data repository, let's play with some NASA-NEX data first. "NASA-NEX on AWS" is one of the earliest "Earth Data on cloud" project that was launch around 2013. Unlike other newer projects that are still evolving (and might change constantly), the NASA-NEX repository is very stable, so it is a good starting example.

Let's download the NEX-GDDP dataset produced by CMIP5:

```
$ aws s3 ls s3://nasanex/NEX-GDDP/
                           PRE BCSD/
2015-06-04 17:15:58          0
2015-06-04 17:18:35         35 doi.txt
2015-06-12 21:08:34    4346867 nex-gddp-s3-files.json
```

You can explore sub-directories by, for example:

```
$ aws s3 ls s3://nasanex/NEX-GDDP/BCSD/
$ aws s3 ls s3://nasanex/NEX-GDDP/BCSD/rcp85/
$ aws s3 ls s3://nasanex/NEX-GDDP/BCSD/rcp85/day/
$ aws s3 ls s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/
$ aws s3 ls s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/tasmax/
$ aws s3 ls s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/tasmax/r1i1p1/
$ aws s3 ls s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/tasmax/r1i1p1/v1.0/
```

Or just get down to one of the the lowest level folders

```
$ aws s3 ls --summarize --human-readable s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/
↪tasmax/r1i1p1/v1.0/
...
2015-09-25 21:07:06    8.3 KiB tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2099.json
2015-06-10 22:48:30  759.0 MiB tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2099.nc
2015-09-25 21:21:34    8.3 KiB tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2100.json
2015-06-10 22:48:52  757.9 MiB tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2100.nc

Total Objects: 3986
   Total Size: 1.4 TiB
```

The `--summarize --human-readable` options print the total size in human-readable formats (like the normal Linux command `du -sh`) As you see, that subfolder has 1.4 TB of data. Just get one file to play with:

```
$ aws s3 cp s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/tasmax/r1i1p1/v1.0/tasmax_day_
↪BCSD_rcp85_r1i1p1_inmcm4_2100.nc ./
download: s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/tasmax/r1i1p1/v1.0/tasmax_day_
↪BCSD_rcp85_r1i1p1_inmcm4_2100.nc to ./tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2100.nc
```

With ~100 MB/s bandwidth, downloading this 750 MB file to EC2 should just take seconds. If you download it to a local machine with 1 MB/s bandwidth, it would take 10 minutes. Not to mention downloading the entire 200 TB NEX dataset to your machine!

It is the daily maximum surface air temperature under the RCP 8.5 scenario:

```
$ ncdump -h
ubuntu@ip-172-31-46-2:~$ ncdump -h tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2100.nc
netcdf tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2100 {
dimensions:
      time = UNLIMITED ; // (365 currently)
      lat = 720 ;
```

(continues on next page)

(continued from previous page)

```
      lon = 1440 ;
...
      float tasmax(time, lat, lon) ;
              tasmax:time = 32850.5 ;
              tasmax:standard_name = "air_temperature" ;
              tasmax:long_name = "Daily Maximum Near-Surface Air Temperature" ;
...
```

Here's an *example notebook* to plot the data with Jupyter and xarray. See the previous *quick start guide* for starting Jupyter.

Congrats! You know how to access and analyze public datasets on AWS. Accessing GEOS-Chem's input data repository will be exactly the same.

Note: Get tired of lengthy S3 commands? The s3fs-fuse tool can make S3 buckets and objects behave just like normal directories and files on disk. We will mention it in advanced tutorials.

Access GEOS-Chem input data repository in S3

```
$ aws s3 ls --request-payer=requester s3://gcgrid/
                           PRE BPCH_RESTARTS/
                           PRE CHEM_INPUTS/
                           PRE GCHP/
                           PRE GEOS_0.25x0.3125/
                           PRE GEOS_0.25x0.3125_CH/
                           PRE GEOS_0.25x0.3125_NA/
                           PRE GEOS_0.5x0.625_AS/
                           PRE GEOS_0.5x0.625_NA/
                           PRE GEOS_2x2.5/
                           PRE GEOS_4x5/
                           PRE GEOS_MEAN/
                           PRE GEOS_NATIVE/
                           PRE GEOS_c360/
                           PRE HEMCO/
                           PRE SPC_RESTARTS/
2018-03-08 00:18:41       3908 README
```

GEOS-Chem input data bucket uses requester-pay mode. Transfering data from S3 to EC2 (in the same region) has no cost. But you do need to pay for the egress fee it you download data to local machines.

The tutorial AMI only has 1-month GEOS-FP metfield. You can get other metfields from that S3 bucket, to support simulations wtih any configurations.

1.2.5 Use EBS volumes as temporary disk storage

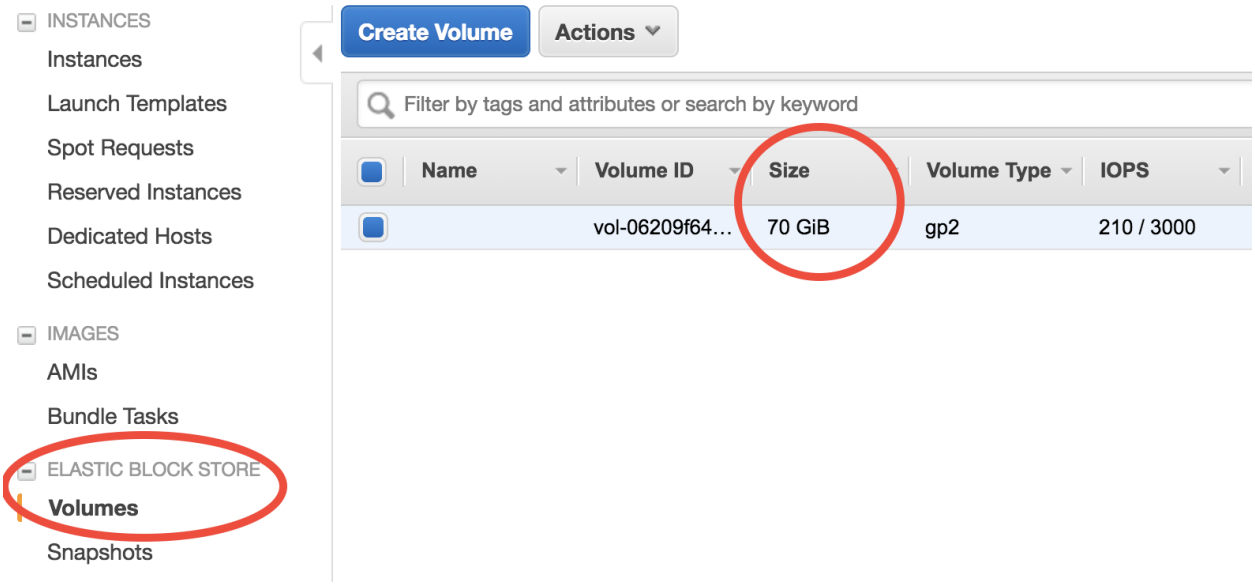
In the previous tutorial you've learned S3, which is independent of any EC2 instances. *EBS volumes*, on the other hand, are traditional disks that directly used by EC2. Whenever you are using EC2 you are also implicitly using EBS (it's just the disk!). Here we just briefly show how to view&control them in the console.

Viewing existing EBS volumes

The EC2 instance launched from the tutorial AMI has 70 GB of disk storage by default:

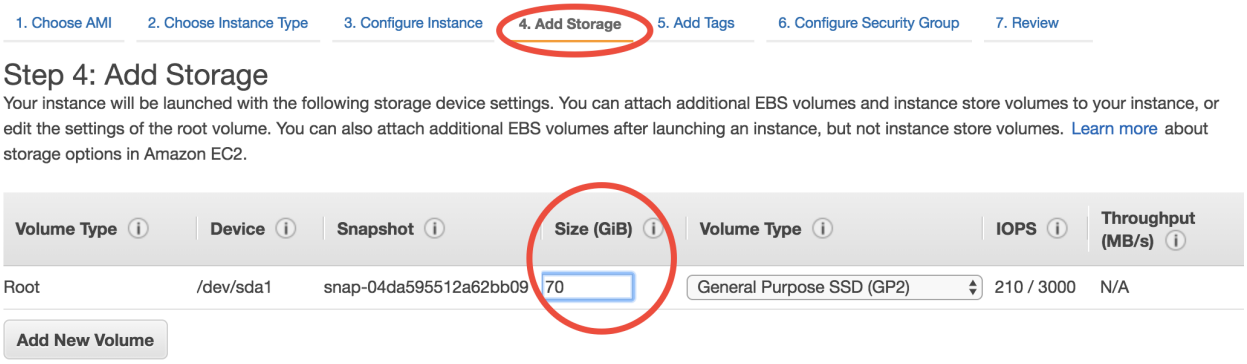
```
ubuntu@ip-172-31-46-2:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7.5G     0  7.5G   0% /dev
tmpfs           1.5G  8.7M  1.5G   1% /run
/dev/xvda1       68G   58G   11G  85% /
```

When the instance is running, you can see the underlying EBS volume in the EC2 console:



Choose volume size at launch time

The easiest way to increase your EC2 instance's disk size is during launching. In the *quick start guide* you've skipped all EC2 configuration details. Step 4 of the configuration specifies disk size:



(we will gradually cover other configuration details throughout the rest of tutorials)

The default number is the minimum storage requirement of the AMI. For a fresh operating system, the requirement is 8 GB. My tutorial AMI contains input data so I require it to be at least 70 GB. If you need a larger disk to host more output data, just enter a larger number. The maximum size of a single volume is 16 TB. You don't need to change Volume Type in most cases.

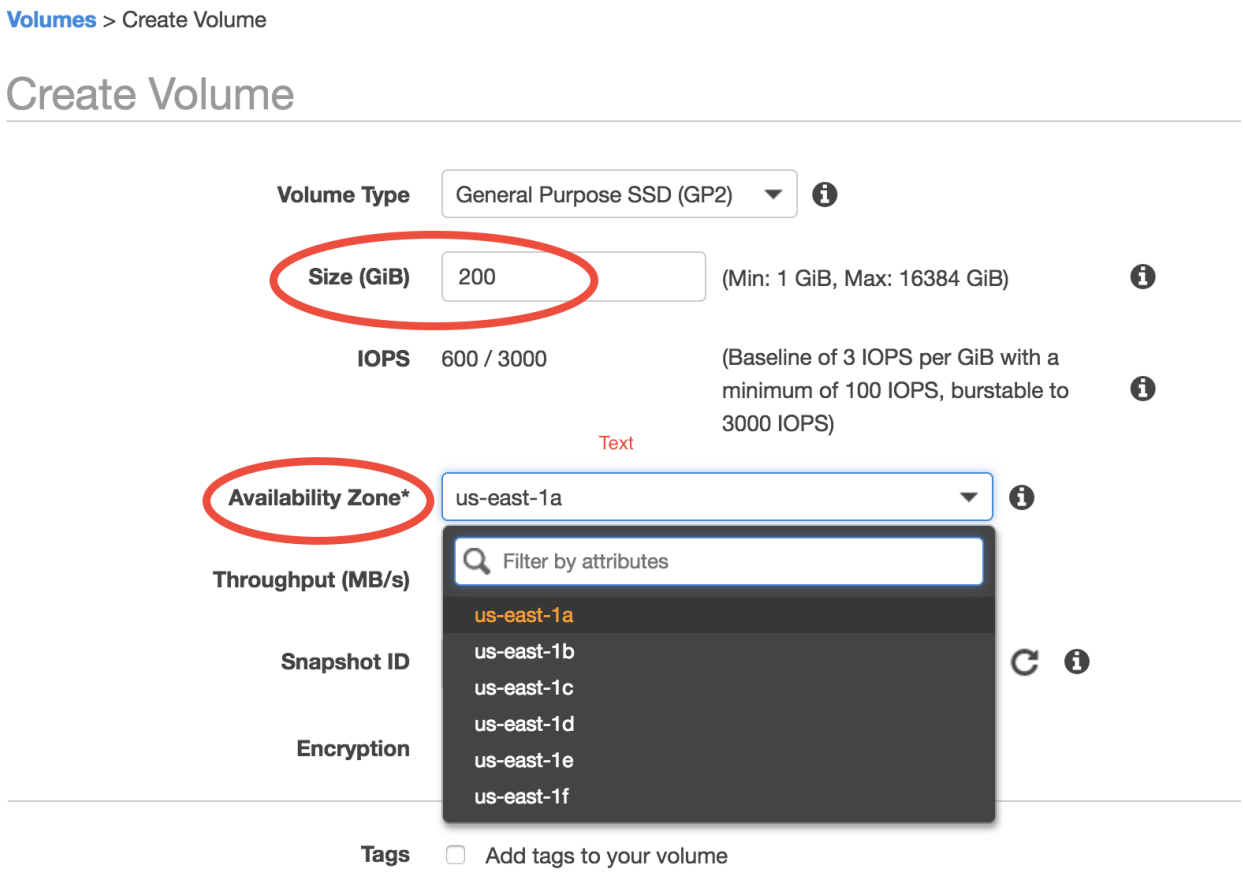
But you might already have an EC2 instance running and don't want to start over. Another way to add disk storage is creating additional volumes, as detailed below.

Attach new volumes after EC2 launch (Optional)

Note: This part is not absolutely necessary for a typical research workflow so free feel to jump to the *next tutorial on Spot Instances* which is more important for scientific computing.

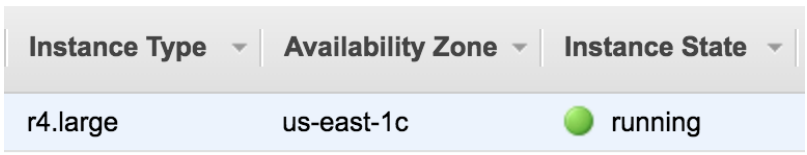
Launch and attach a volume

Click on the "Create Volume" button in the "Volumes" page. You should see:



Key parameters are "Size" (say you need a 200 GB disk in this case) and "Availability Zone" (a new concept). Keep other options as default. Currently there are 6 Avail Zones in the us-east-1 region (i.e. N. Virginia). EBS volumes are only attachable to EC2 instances in the same Avail Zone, because different Avail Zones are physically located at different locations (how can you attach a USB drive to a computer in another building?).

You can see the Avail Zone of your running instances in the EC2 console, "Instance" page:



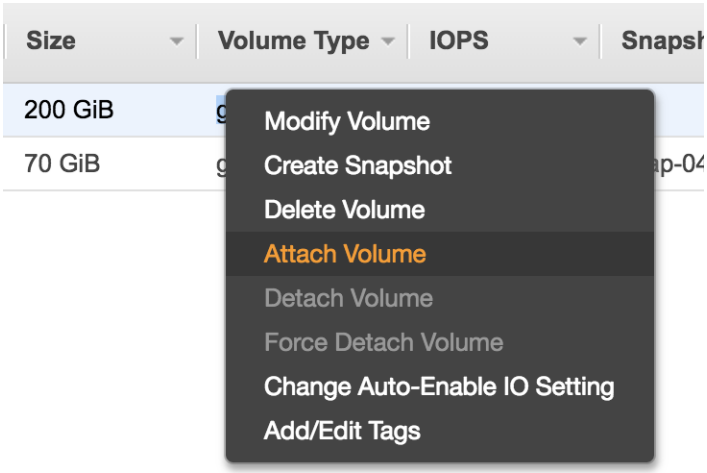
In this case my EC2 instance is running in us-east-1c, thus I need to also launch the EBS volume into us-east-1c.

Then you should see two active volumes in the EC2 console, "Volumes" page:

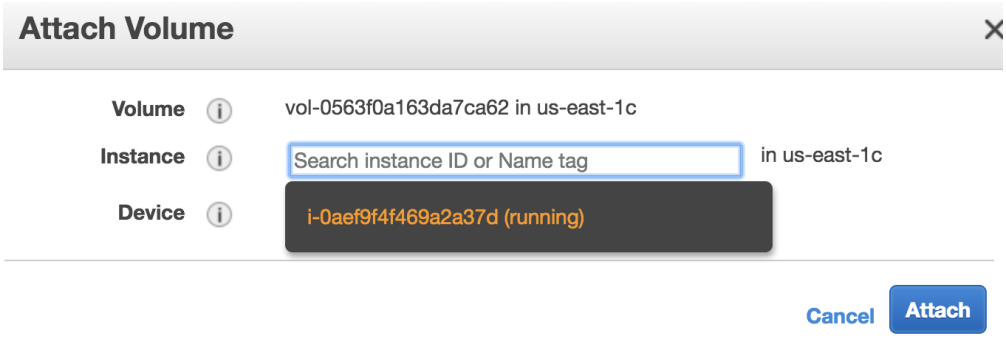
Size	Volume Type	IOPS	Snapshot	Created	Availability Zone	State
200 GiB	gp2	600 / 3000		March 9, 2018 at 8:...	us-east-1c	available
70 GiB	gp2	210 / 3000	snap-04da595...	March 9, 2018 at 10...	us-east-1c	in-use

The "in-use" one is the disk for the runnning EC2 instance. The "available" one is the newly-created volume that isn't attached to an EC2 instance yet.

Right click on the new volume and choose "Attach Volume":



You should be prompted with ID of the running EC2 instance. If nothing gets prompted, double check if you choose the same Avail Zone for your EC2 instance and EBS volume.



After attaching, the `lsblk` command will show the new 200 GB volume.

```
$ lsblk
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
xvda    202:0    0   70G  0 disk
└─xvda1 202:1    0   70G  0 part /
xvdf    202:80   0  200G  0 disk
```

(An equivalent way to replicate the above steps is during launching the EC2 instance, "Step 4: add storage", click on "Add New Volume". But you still need to do the below steps to make that volume usable)

Make that volume usable

Before actually using this additional disk, you need to type a few commands. If you have no idea about file system management, simpliy copy and paste the following commands without thinking too much (adapted from AWS official guide).

Create a file system (only needed for newly-created volumes):

```
$ sudo mkfs -t ext4 /dev/xvdf
mke2fs 1.42.13 (17-May-2015)
...
Writing superblocks and filesystem accounting information: done
```

Mount it to a new directory (use any directory name you like):

```
$ mkdir new_disk
$ sudo mount /dev/xvdf new_disk
```

Then you should see the `/dev/xvdf` file system is mounted on the `/home/ubuntu/new_disk` directory:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7.5G     0  7.5G   0% /dev
tmpfs           1.5G  8.6M  1.5G   1% /run
/dev/xvda1       68G   58G   11G  85% /
tmpfs           7.5G     0  7.5G   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           7.5G     0  7.5G   0% /sys/fs/cgroup
tmpfs           1.5G  4.0K  1.5G   1% /run/user/1000
/dev/xvdf       197G   60M  187G   1% /home/ubuntu/new_disk
```

By default, the new directory belongs to the root user. Change the ownership so you don't need root permission to access it:

```
$ sudo chown ubuntu new_disk
```

Test if you can write files into that new disk:

```
$ touch new_disk/test_file
[no error occurs]
```

Done! This disk size of your server is now much bigger. EBS volumes are useful for hosting input/output data temporarily. For long-term, persistently storage, alway upload your stuff to S3. S3 is much more "transparent" than EBS. To know what's in an EBS volume, you have to attach it to an EC2 instance and view the files through EC2. On the other hand, you can view all your files on S3 directly in the graphical conole, without having any EC2 instances running.

You can also detach the volume and re-attach it to another EC2 instance, as a way to share data between two EC2 instances. However, using S3 as the medium of data transfer is generally more convenient, and it doesn't require two EC2 instances to be in the same Avail Zone.

> **Warning:** Terminating your EC2 instance will not remove attached EBS volumes. You need to delete them manually.

Save volumes into snapshots (Optional)

Recall that EBS price is $100/TB and S3 price is $23/TB. There is something in between, called "snapshot EBS volumes to S3", which causes $50/TB. You seldom need to use this functionality (since simply using S3 itself is more convenient), but the concept is quite important – AMIs are actually backed by "EBS snapshots", which physically live on S3.

Note: Remember the "warm-up" time I mentioned in the quick start guide? It is not any physical "warm-up" at all – it is because the data are being pulled from S3 to the EBS volume under the hood. For a newly-created EC2 instance, although it looks like all files are already on that server, the actual data content actually live on S3. The data will be pulled from S3 on-the-fly whenever your try to access it. Thus the first simulation has quite slow I/O. After the data actually live on EBS, the subsequent I/O will be much faster.

1.2.6 Use Spot Instances to reduce EC2 cost

In the *quick start guide*, you've chosen the "r4.large" instance type to conduct proof-of-concept simulations. For real, serious simulations, definitely use bigger instances with more CPU cores in the Compute Optimized families (e.g. c5.4xlarge, c4.4xlarge). Larger instances also have higher bandwidth, allowing faster data transfer between EC2 and S3.

However, bigger instances are also more expensive. While "r4.large" only costs $0.1/hour, the most powerful instance "c5.18xlarge" with 72 cores costs $3/hour. If the simulation runs for days or weeks, the total cost will be not a trivial number. The good news is, the Spot Instance pricing model can generally reduce the cost by ~70%.

What are spot instances and why they exist

The default EC2 pricing is called "on-demand" pricing. This service model is extremely flexible (whenever you request a server, you get it almost instantly; you can stop and restart it at any time) and very stable (the server is guaranteed to run for no matter how long, as long as you don't stop it on purpose). This kind of high quality is needed by web servers which have be stable for a long time, but it is often an overkill for scientific computing workflow, which are generally intermittent. By allowing the server to be a little bit sloppy, the cost can be drastically reduced. That's the role of spot instances.

In the EC2 console, go to "Spot Requests" page, and then click on "Pricing History".

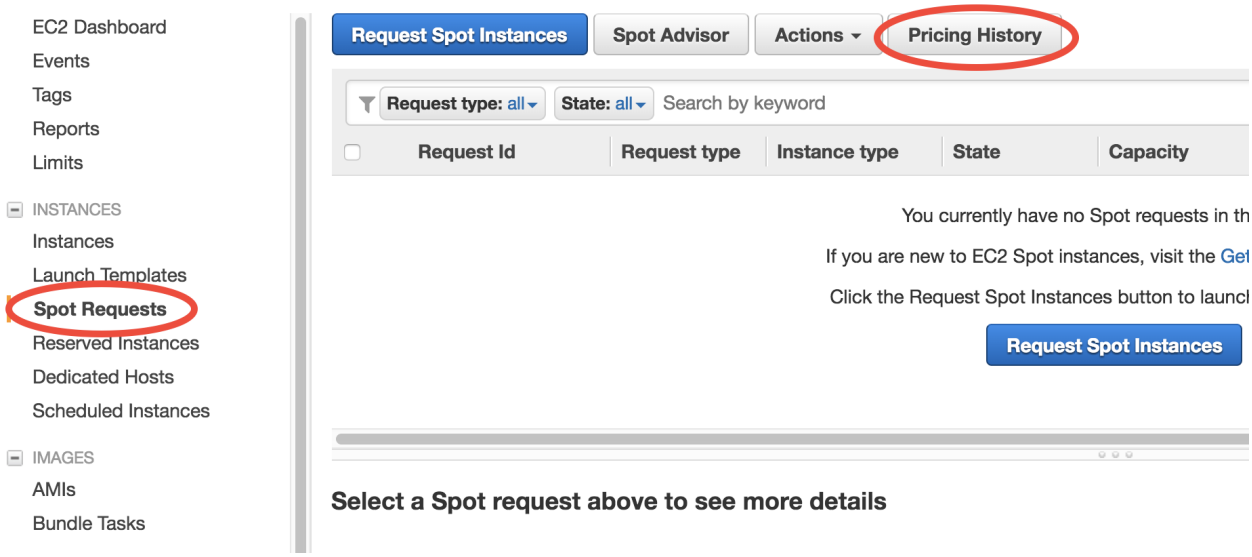
Choose an instance type, for example "c4.4xlarge". You should see that spot prices are only 1/3~1/4 of the on-demand price. Only $0.2/hour for 16 modern CPU cores? Almost insane.

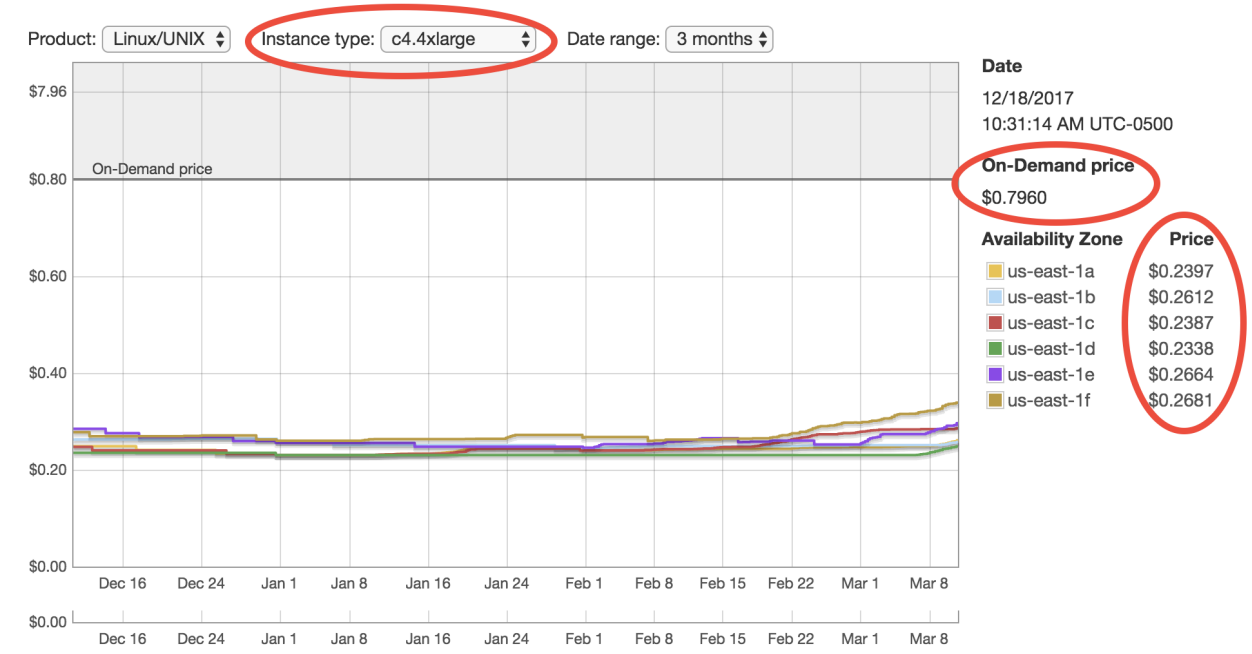
Each "Availability Zone" has its on spot price. You will get the cheapest one by default. (Recall that different Avail Zones are physically located at different locations. This was mentioned in the *EBS tutorial*.)

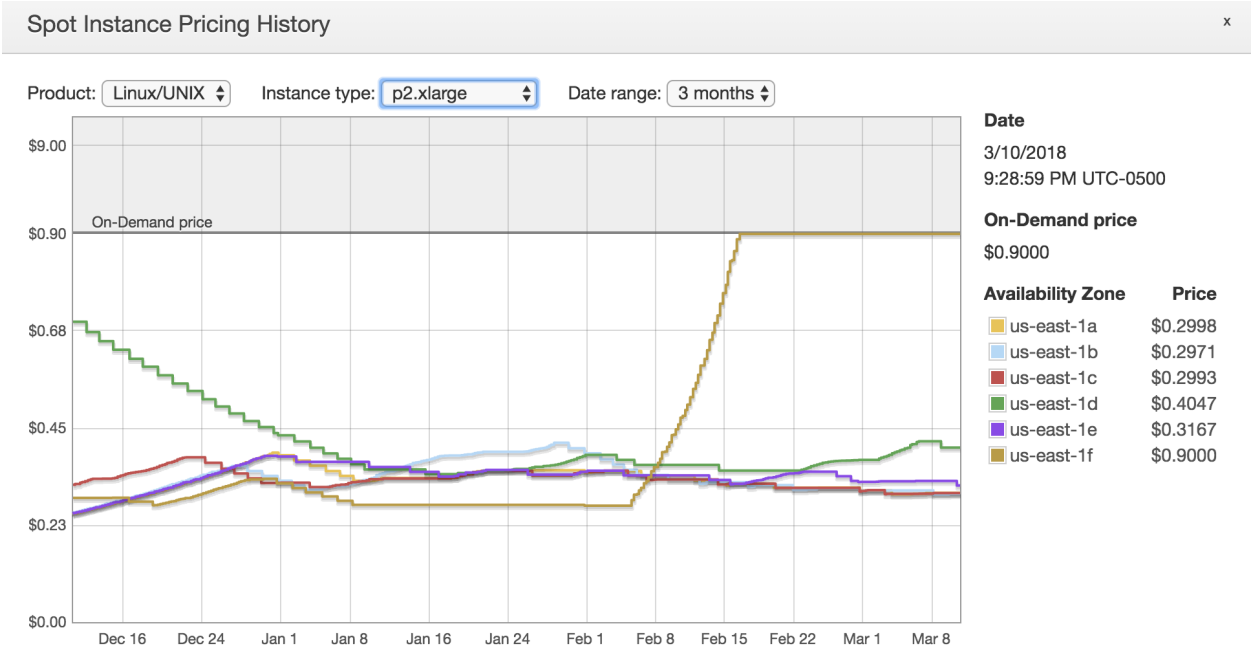
Spot prices are fluctuating according to current user demand. **Once the price exceeds the on-demand price, your spot instance will be reclaimed by AWS to serve on-demand users who pay much more.** That's why they are "sloppy" and thus much cheaper than standard, on-demand instances.

But from the above figure it seems like the spot prices are consistently much lower than the on-demand price, for the entire 3 months? Does the price ever exceeds on-demand price? Yes, it does. For example this GPU instance (perhaps too many people are *training neural nets*):

However, in general, the chance of spot instance shut-down is pretty low, especially during a model simulation (at most takes several days, sometimes just hours). The cost saving is quite big so I recommend using spot instances for serious, compute-intensive simulations.


EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Launch Templates
Spot Requests
Reserved Instances
Dedicated Hosts
Scheduled Instances
IMAGES
AMIs
Bundle Tasks
Request Spot Instances
Spot Advisor
Actions
Pricing History
Request type: all
State: all
Search by keyword
Request Id
Request type
Instance type
State
Capacity
You currently have no Spot requests in th
If you are new to EC2 Spot instances, visit the Get
Click the Request Spot Instances button to launcl
Request Spot Instances
Select a Spot request above to see more details


Product: Linux/UNIX
Instance type: c4.4xlarge
Date range: 3 months
$7.96
$0.80
$0.60
$0.40
$0.20
$0.00
On-Demand price
Dec 16
Dec 24
Jan 1
Jan 8
Jan 16
Jan 24
Feb 1
Feb 8
Feb 15
Feb 22
Mar 1
Mar 8
Date
12/18/2017
10:31:14 AM UTC-0500
On-Demand price
$0.7960
Availability Zone
Price
us-east-1a $0.2397
us-east-1b $0.2612
us-east-1c $0.2387
us-east-1d $0.2338
us-east-1e $0.2664
us-east-1f $0.2681



Note: Still feel uncomfortable about the possibility of losing your server? The chance is really low and there are even people wondering why on-demand instances are still being used since spot instances look so sweet. Think about this: "100% stability" (on-demand) can be much more expensive than "99.5% stability" (spot). There are people who are willing to pay much more for the last 0.5%. Those people are generally not scientific researchers.

Use spot instances for big computing

You could click on "Request Spot Instances" in the "Spot Requests" page, but the user interface looks kind of unfamiliar for new AWS users and contains some advanced settings that I'd like to skip for now. Instead, we can launch spot instances using the old way in the *quick start guide*.

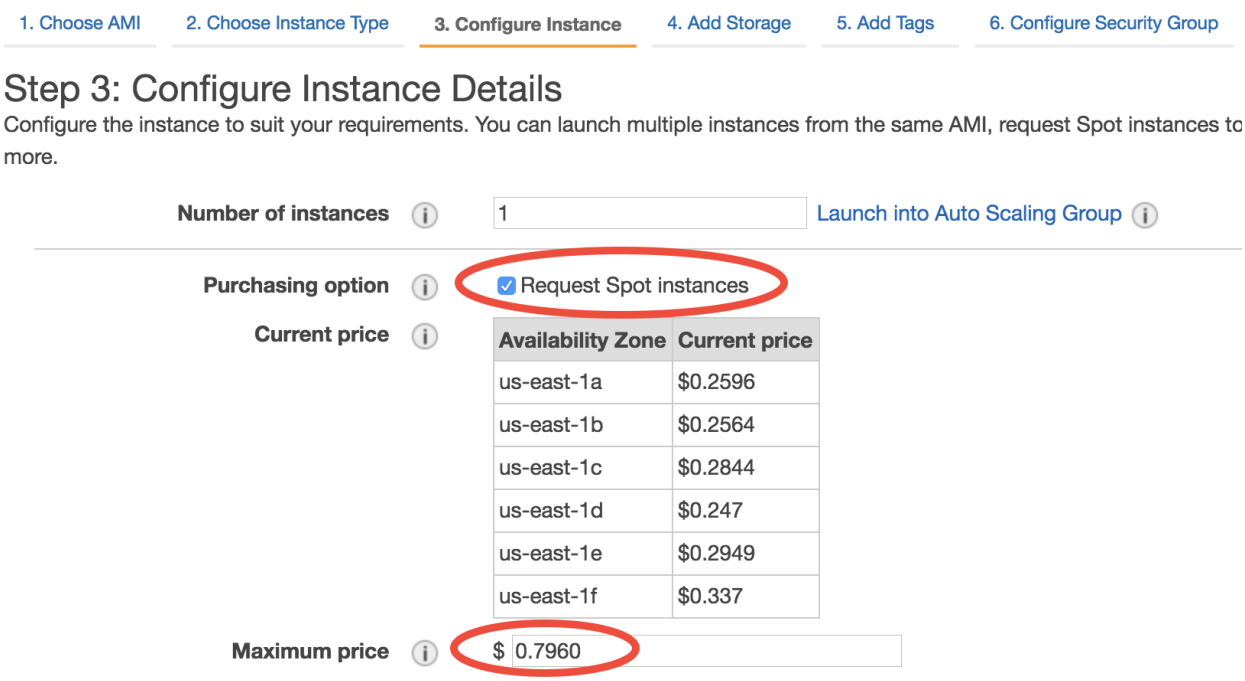
Launch a new EC2 instance just like in the quick start guide, but in "Step 2: Choose an Instance Type" select a bigger instance such as "c4.4xlarge", and go to "Step 3: Configure Instance Details":

Select "Request Spot instances", enter the on-demand price for the "Maximum price" option. You can also use a different price limit other than the on-demand price. Once the spot price goes beyong that limit, your server will be reclaimed. In other words, you will never pay a price higher than the limit you set. You can even set the price higher than on-demand but this is generally not recommended.

No need to touch other options. Then just launch as usual. This spot request should be fulfilled pretty quickly, and your server will be running as usual. Just `ssh` to your server. You can make sure you do get 16 cores by `lscpu`:

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
...
```

Then set OpenMP thread number and run the model as usual:




```
$ export OMP_NUM_THREADS=16
$ ./geos.mp
```

(It is also OK to have `OMP_NUM_THREADS` unset, as the program will use the number of cores by default, which is exactly 16 here.)

The model will run ~10x faster on this advanced instance than on the previous "r4.large". Because you're using spot, you don't pay a lot more (likely just double, from $0.1/hour to $0.2/hour).

Note: c4.4xlarge or c5.4xlarge? C5 is a newer generation, and is ~10% faster than C4. Further, the on-demand price of C5 is ~10% cheaper than C4. So seems like C5 is clearly more cost-effective. But this might not be true for spot prices which depend on the current market. In general, both families are pretty good for HPC workloads.

If you like, try performing a 1-month simulation on this fast machine. This would take several hours so you might want to *keep the program running even after logging off the server*.

Spot instances cannot be stopped and can only be terminated. Make sure you've transfering important data to S3 before terminating the server.

Deal with spot instance interruptions

Well, most of time I simply ignore the fact that they might be interrupted. After using AWS for a year, I haven't experienced a true spot shut-down, unless I set the price limit to a very low value intentionally.

If you are super cautious, put your run directory and output data *in an additional EBS volume*. When the spot instance dies, additional volumes will not be affected, and you can attach them to another EC2 instances. No need to worry about input data unless you've made your own modifications to them, since all input data can be retrieved from *our public S3 bucket*.

It is also possible to retrieve data in the root EBS volume of the spot instance, but that is a bit cumbersome since

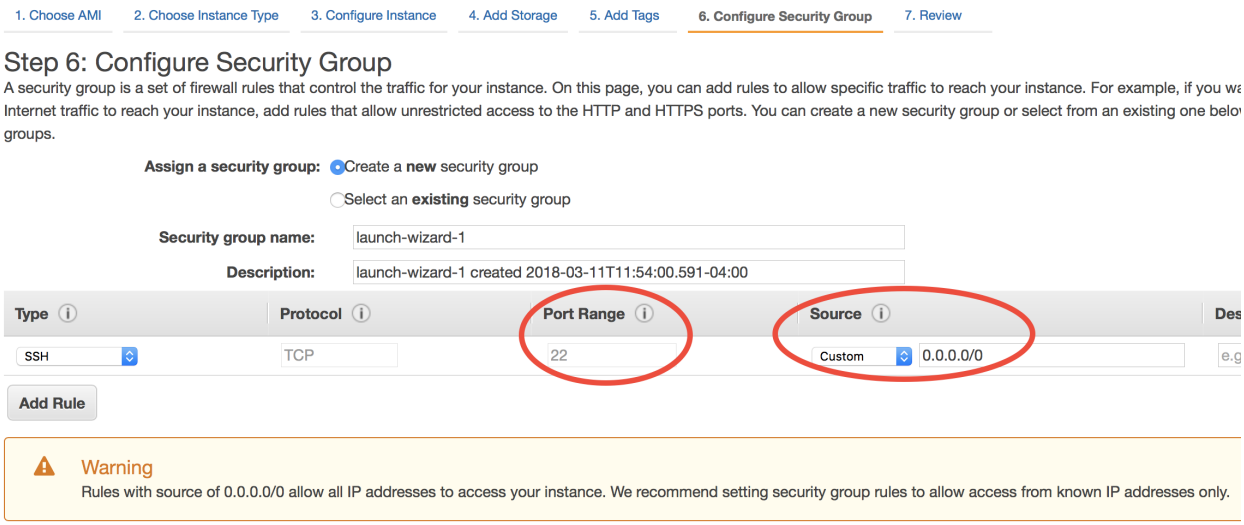
the root volume also contains system files (which feels kind of messy if you are unfamiliar with Linux system file structure). On the other hand, additional volumes have nothing but your own data.

Note: Besides "On-demand" and "Spot", there is also a "Reserved Instance" pricing model. Unless you are running models 24 hours a day, 7 days a week, this type won't help too much.

1.2.7 Notes on security groups (EC2 firewall)

In the *quick start guide* I asked you to *skip EC2 configuration details*. If you didn't follow my word and messed up "Step 6: Configure Security Group", you might have trouble `ssh` to your EC2 instance. (Mis-configured security group should be the most common reason why you cannot `ssh` to your server. For other possible situations, see troubleshooting EC2 connection.)

In Step 6, a new "security group" will be created by default, with the name "launch-wizard-x":



"Security group" controls what IPs are allowed to access your server. As you already see in the warning message above, the current setting is to allow any IP to connect (the IP number "0.0.0.0/0" stands for all IPs, equivalent to selecting "Anywhere" in the "Source" option above), but only for SSH type of connection (recall that "22" is the port number for SSH). This is generally fine, as you also need to have the *EC2 Key Pair* in order to access that server. You can further set "Source" to "My IP", to add one more layer of security (which means your friend won't be able to access your server even if they have your EC2 key pair).

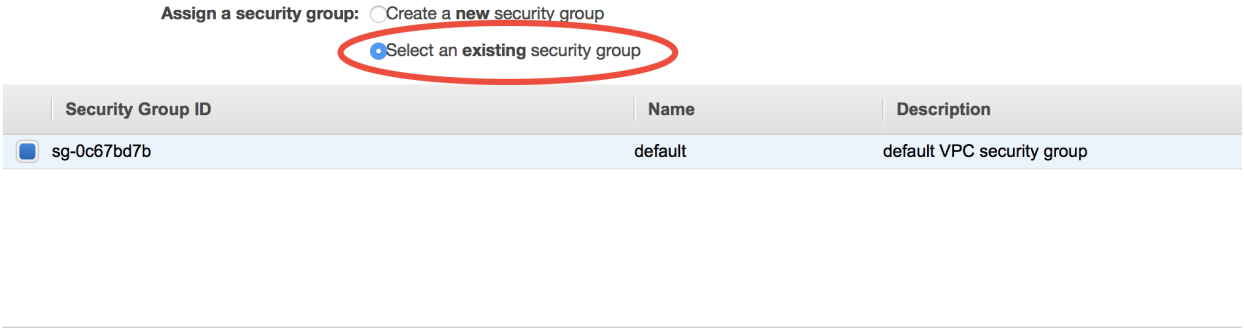
However, if you messed it up, say selected the "default" security group:

In this case, what's under the "Source" option is the ID of the default security group itself. This means NO external IPs are allowed to connect to that server, so you won't be able to `ssh` to it from your own computer.

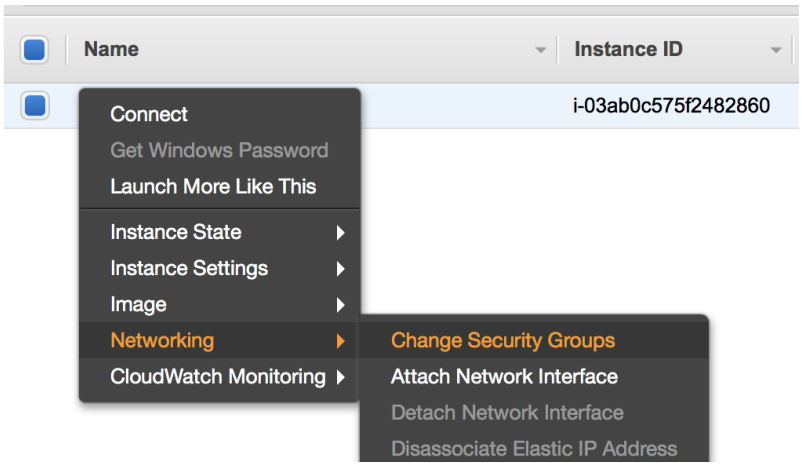
If you've already messed it up and launched the EC2 instance, right-click on your EC2 instance in the console, choose "Networking" - "Change Security Groups" to assign a more permissive security group.

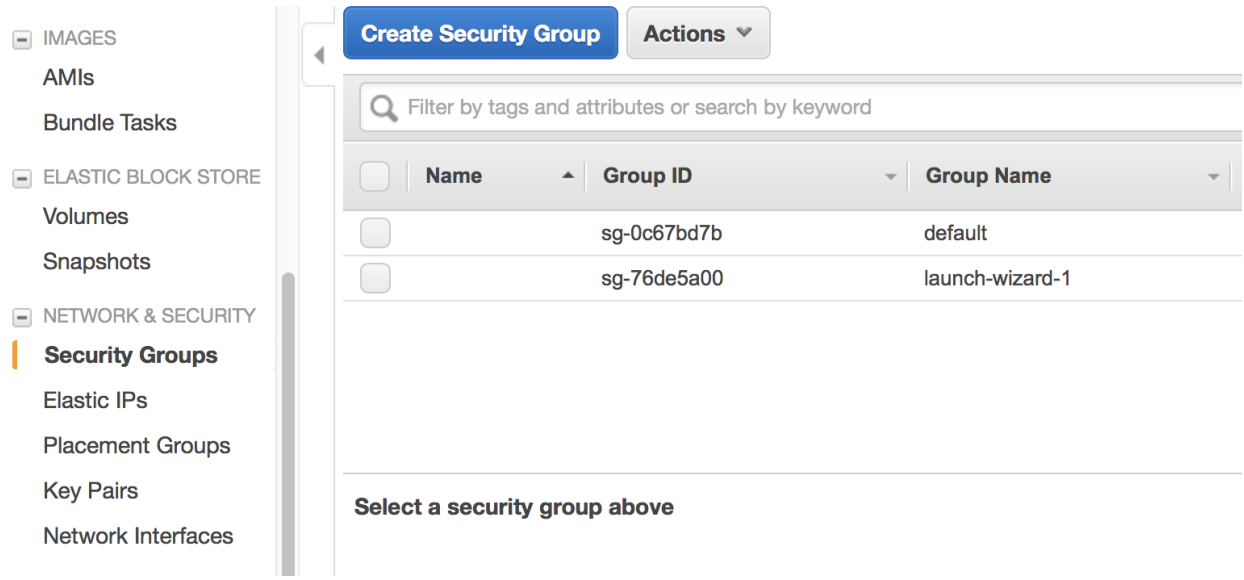
You can view existing security groups in the "Security Groups" page in the EC2 console:

If you've launched multiple EC2 instances following the exact steps in the *quick start guide* and always skipped "Step 6: Configure Security Group", you would see multiple security groups named "launch-wizard-1", "launch-wizard-2"... They are created automatically each time you launch new EC2 instances. They have exactly the same settings (allow SSH connection from all IPs), so you only need to keep one (delete others) and just choose that one in "Step 6:


Assign a security group: Create a new security group
Select an existing security group
Security Group ID
Name
Description
sg-0c67bd7b
default
default VPC security group


Inbound rules for sg-0c67bd7b (Selected security groups: sg-0c67bd7b)
Type
Protocol
Port Range
Source
All traffic
All
All
sg-0c67bd7b (default)


Name
Instance ID
i-03ab0c575f2482860
Connect
Get Windows Password
Launch More Like This
Instance State
Instance Settings
Image
Networking
CloudWatch Monitoring
Change Security Groups
Attach Network Interface
Detach Network Interface
Disassociate Elastic IP Address


IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Create Security Group
Actions
Filter by tags and attributes or search by keyword
Name
Group ID
Group Name
sg-0c67bd7b
default
sg-76de5a00
launch-wizard-1
Select a security group above

Configure Security Group" during EC2 instance launching. You can also modify an existing security group by right clicking on it and choose "Edit inbound rules".

That's all you need to know about security groups. Unlike local HPC clusters that can force strict security control, cloud platforms are exposed to the entire world and thus need complicated security settings to deal with different situations. Say, do you plan to access the server only from you own computer, or want to open the access to your group members, or even open to a broader public? This complexity can be a bit confusing for beginners.

1.2.8 Final word on research workflow on cloud

Congrats! You've finished all beginner tutorials. Now you've learned enough AWS stuff to perform most of simulation, data analysis, and data managment tasks. These tutorials could feel pretty intense if you are new to cloud computing (although I really tried to make them as user-friendly as possible). Don't worry, repeat these practices several times and you will get familiar with the research workflow on the cloud very quickly. There are also advanced tutorials, but they are just add-ons and are really not necessary for just getting science done.

The major difference (in terms of research workflow) between local HPC clusters and cloud platforms is **data management**, and that's what new users might feel uncomfortable with. To get used to the cloud, the key is to use and love S3! On traditional local disks, any files you create will stay there overever (so I often end up leaving tons of random legacy files in my home directory). On the other hand, the pricing model of cloud storage (charge you by the exact amount of data) will force you to really think about what files should kept by transferring to S3, and what should be simply discarded (e.g. TBs of legacy data that are not used anymore).

There are also ways to make cloud platforms *behave like traditional HPC clusters*, but they can often bring more restrictions than benefits. To fully ultilize the power and flexibility of cloud platforms, directly use native, basic services like EC2 and S3.

Here's my typical research workflow for reference:

1. Launch EC2 instances from pre-configured AMI. Consider spot instances for big computing. (Use AWSCLI to launch them with one command.)
2. Prepare input data by pulling them from S3 to EC2. Put commonly used `aws s3 cp` commands into bash scripts.
3. Tweak model configurations as needed.
4. Run simulations *with tmux*. Log out and go to sleep if the model runs for a long time.
5. Use Python/Jupyter to analyze output data.
6. After simulation and data analysis tasks are done, upload output data and customized model configuration (mostly run directories) to S3. Or download them to local machines if necessary (Recall that data egress charge is $90/TB).
7. Once important data safely live on S3 or on your local machine, shut down EC2 instances to stop paying for CPU charges.
8. Go to write papers, attend meetings, do anything other than computing. During this time, no machines are running on the cloud, and the only cost is data storage on S3 ($23/TB/month). Consider S3 - Infrequent Access which costs half if the data will not be used for several months.
9. Whenever need to continue computing, launch EC2 instances, pull stuff from S3, start coding again.

I often use big instances (e.g c5.8xlarge) with spot pricing for computationally-expensive model simulations. But I also keep a less-powerful, on-demand instance (e.g. c5.large) for frequent data analysis workloads. Whenever I need to make a quick plot of model output data, I just start this on-demand instance, write Python code in Jupyter, and stop this instance when I am done. Since this on-demand instance just switches between "running" and "stopped" states but never "terminates", all files are preserved on disk, so I don't have to backup temporary files to S3 all the time.

When I need large computing power to process data in parallel, this on-demand instance can be easily resized to a bigger type like c5.8xlarge. Since data analysis tasks typically just take 1~2 hours (unlike model simulations that often take days), it doesn't worth the effort to set up spot instances to save one dollar.

1.3 Advanced tutorials

This chapter provides advanced tutorials to improve your research workflow. Make sure you've gone through all beginner tutorials first.

1.3.1 Use containers to enhance research reproducibility

What are containers?

Run GEOS-Chem inside Singularity container

1.3.2 Overview of HPC cluster options on AWS

Why do you need an HPC cluster

HPC cluster management tools

CfnCluster

AlcesFlight

StarCluster

AWS HPC partners

1.3.3 Use CfnCluster to manage HPC clusters

1.3.4 Use AlcesFlight to manage HPC clusters

1.4 Developer guide

This chapter shows how to build models on the cloud from stratch.

1.4.1 Install compilers and NetCDF libraries

GCC and gfortran compilers

Install NetCDF library with package manager

Test NetCDF sample code

(Optional) Install NetCDF from source code

1.4.2 Share your AMI with others

1.4.3 Set up GEOS-Chem environment

Environment variables

Source code and run directory

Test compile

Prepare input data

Test run

1.4.4 Install scientific Python environment

1.4.5 Build your own containers

1.4.6 Install MPI libraries

1.5 AWS services in detail

This chapter dives deep into the nitty-gritty of AWS services.

1.5.1 AWS services in human language

1.5.2 Monitoring AWS cost

1.5.3 EC2 configuration details

1.6 Appendix

This chapter provides additional resources that don't fit into the main tutorials.

1.6.1 List of public AWS resources for GEOS-Chem

Currently all resources are in us-east-1 (N. Virginia).

Resource	ID/name	Size	Content
Tutorial AMI	ami-c135ccbc	70 GB	1. gfortran 5.4.0, netCDF-Fortran 4.4.3 2. GC environment variables 3. GC source code and Unit Tester 4. 4x5 geosfp run directory 5. Pre-compiled executable in rundir 6. Geoscientific Python environment 7. Sample gcdata directory
S3 bucket for all GC data	s3://gcgrid (requester-pay)	~30 TB	All current GEOS-Chem input data

1.6.2 Keep a program running after logoff

Shared clusters often have job schedulers to handle multiple users' job submissions. On the cloud, however, the entire server belongs to you so there's generally no need for a scheduler.

Note: Have multiple jobs? Why schedule them? Just launch multiple instances to run all of them at the same time. Running 5 instances for 1 hour costs exactly the same as running 1 instance for 5 hours, but the former approach saves you 80% of time without incurring any additional charges.

Thus, instead of using `qsub` (with PBS) or `sbatch` (with Slurm), you would simply run the executable `./geos.mp` from the terminal. To keep the program running after logoff or internet interruption, use simple tools such as the nohup command, GNU screen or tmux. I personally like tmux as it is very easy to use and also allow highly advanced terminal management if needed. It is also quite useful for managing other time-consuming computations such as big data processing or training machine learning models, so worth learning.

Use nohup command (not recommended)

`nohup` is a built-in Linux command to prevent a program from being interrupted. This works but is **not recommended** since monitoring nohup jobs is kind of a mess. Instead, use `screen` or `tmux` as detailed in the next section. I just put basic `nohup` commands here for record.

Start the simulation with nohup mode:

```
$ nohup ./geos.mp > run.log &
$ nohup: ignoring input and redirecting stderr to stdout
```

Type `Crtl + c` to go back to normal terminal. Use `tail -f run.log` to monitor the log file if necessary. Log off and re-login the server if you like.

List nohup jobs by `ps x`:

```
$ ps x
...
13067 pts/0    Rl     4:56 ./geos.mp
...
```

If necessary, kill the job by its ID. In this case, it is:

```
kill 13067
```

Use GNU Screen

The `screen` commmand creates terminal sessions that can persist after logoff. Here's a nice tutorial offered by Harvard Research Computing.

Start a screen session with any name you like

```
$ screen -S run-geoschem
```

Inside the screen session, run the model as usual:

```
$ ./geos.mp | tee run.log
```

(Here I use `tee` to print model log to both the terminal screen and a file.)

Type `Crtl + a`, and then type `d`, to **detach** from the current session. You will be back to the normal terminal but the model is still running inside that detached session. You can log off the server and re-login if you like.

List existing sessions by:

```
$ screen -ls
There is a screen on:
      13279.run-geoschem      (03/12/2018 12:25:39 AM)        (Detached)
1 Socket in /var/run/screen/S-ubuntu.
```

Resume that session by

```
screen -x run-geoschem
```

Use tmux (recommended)

The `tmux` command behaves almost the same as `screen` for single-panel sessions. But it is also useful for splitting one terminal window into multiple panels (tons of quick tutorials online, say this, and this). `screen` also does terminal splitting but is not as convenient as tmux.

Start a new session by

```
$ tmux
```

Inside the session, run the model as usual, just like in the `screen` session:

```
$ ./geos.mp | tee run.log
```

Type `Crtl + b`, and then type `d`, to **detach** from the current session. Use `tmux ls` to list existing sessions and `tmux a` (shortcut for `tmux attach`) to resume the session.

To handle multiple sessions, use `tmux new -s session_name` to create a session with a name and `tmux a -t session_name` to resume that specific session.

1.6.3 Sample Python code to analyze GEOS-Chem data

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import xarray as xr
        import cartopy.crs as ccrs
```

GEOS-Chem NetCDF diagnostics

```
In [2]: ds = xr.open_dataset("/home/ubuntu/geosfp_4x5_standard/"
                             "GEOSChem.inst.20130701_backup.nc4")
        ds
```

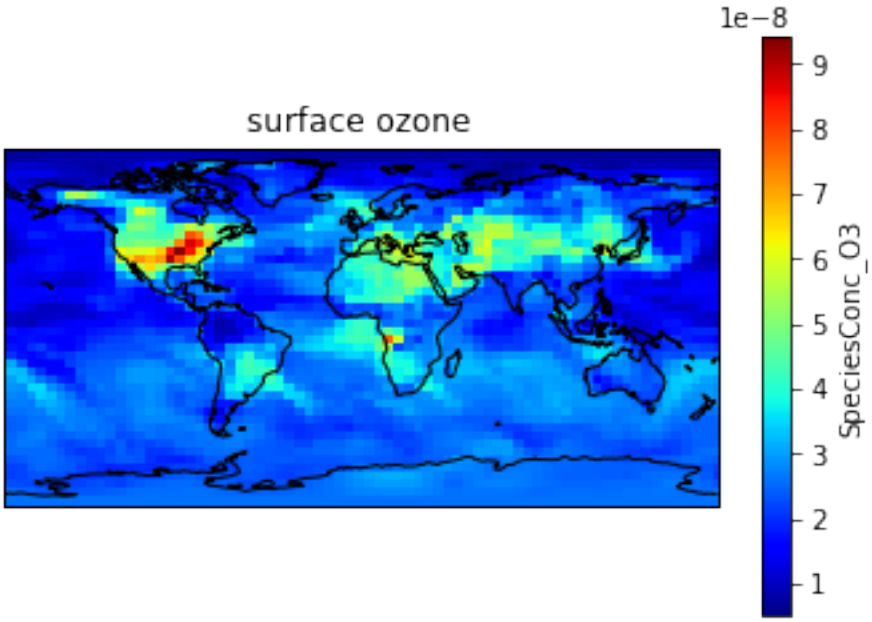
```
Out[2]: <xarray.Dataset>
        Dimensions:          (ilev: 73, lat: 46, lev: 72, lon: 72, time: 1)
        Coordinates:
          * time             (time) datetime64[ns] 2013-07-01T00:20:00
          * lev              (lev) float64 0.9925 0.9775 0.9625 0.9475 0.9325 0.9175 ...
          * ilev             (ilev) float64 1.0 0.985 0.97 0.955 0.94 0.925 0.91 ...
          * lat              (lat) float64 -89.0 -86.0 -82.0 -78.0 -74.0 -70.0 -66.0 ...
          * lon              (lon) float64 -180.0 -175.0 -170.0 -165.0 -160.0 -155.0 ...
        Data variables:
            hyam             (lev) float64 ...
            hybm             (lev) float64 ...
            hyai             (ilev) float64 ...
            hybi             (ilev) float64 ...
            P0               float64 ...
            AREA             (lat, lon) float32 ...
            SpeciesConc_CO   (time, lev, lat, lon) float32 ...
            SpeciesConc_O3   (time, lev, lat, lon) float32 ...
            SpeciesConc_NO   (time, lev, lat, lon) float32 ...
        Attributes:
            title:         GEOS-Chem diagnostic collection: inst
            history:
            format:        not found
            conventions:   COARDS
            ProdDateTime:
            reference:     www.geos-chem.org; wiki.geos-chem.org
            contact:       GEOS-Chem Support Team (geos-chem-support@as.harvard.edu)
```

```
In [3]: ds['SpeciesConc_O3']
```

```
Out[3]: <xarray.DataArray 'SpeciesConc_O3' (time: 1, lev: 72, lat: 46, lon: 72)>
        [238464 values with dtype=float32]
        Coordinates:
          * time     (time) datetime64[ns] 2013-07-01T00:20:00
          * lev      (lev) float64 0.9925 0.9775 0.9625 0.9475 0.9325 0.9175 0.9025 ...
          * lat      (lat) float64 -89.0 -86.0 -82.0 -78.0 -74.0 -70.0 -66.0 -62.0 ...
```

```
  * lon      (lon) float64 -180.0 -175.0 -170.0 -165.0 -160.0 -155.0 -150.0 ...
Attributes:
    long_name:         Dry mixing ratio of species O3
    units:             mol mol-1 dry
    averaging_method:  instantaneous
```

```
In [4]: ax = plt.axes(projection=ccrs.PlateCarree())
        ds['SpeciesConc_O3'][0,0].plot(cmap='jet', ax=ax)
        ax.coastlines()
        plt.title('surface ozone');
```



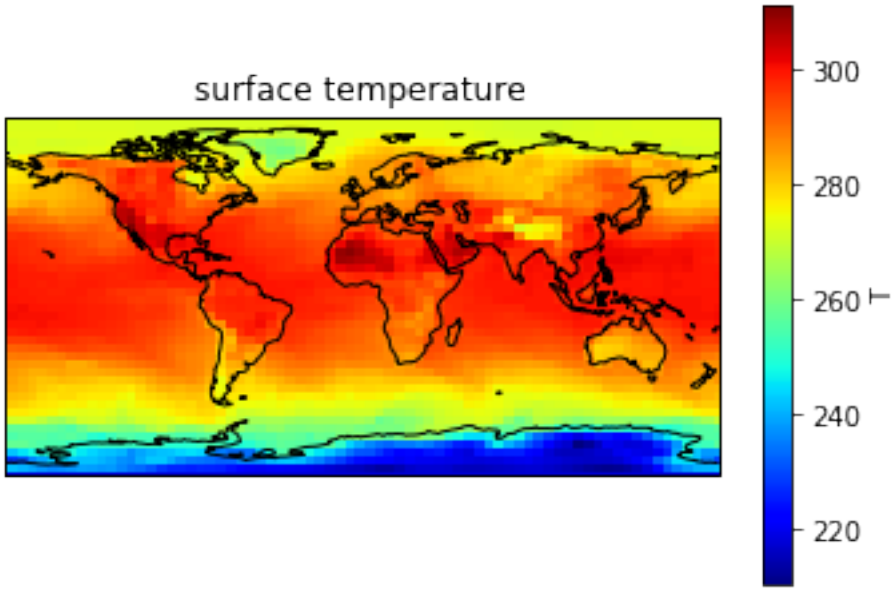
GEOS-FP metfield

```
In [5]: ds_met = xr.open_dataset("/home/ubuntu/gcdata/ExtData/GEOS_4x5/GEOS_FP/"
                                 "2013/07/GEOSFP.20130701.I3.4x5.nc")
        ds_met
```

```
Out[5]: <xarray.Dataset>
        Dimensions:  (lat: 46, lev: 72, lon: 72, time: 8)
        Coordinates:
          * time     (time) datetime64[ns] 2013-07-01 2013-07-01T03:00:00 ...
          * lev      (lev) float32 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
          * lat      (lat) float32 -90.0 -86.0 -82.0 -78.0 -74.0 -70.0 -66.0 -62.0 ...
          * lon      (lon) float32 -180.0 -175.0 -170.0 -165.0 -160.0 -155.0 -150.0 ...
        Data variables:
            PS       (time, lat, lon) float32 ...
            PV       (time, lev, lat, lon) float32 ...
            QV       (time, lev, lat, lon) float32 ...
            T        (time, lev, lat, lon) float32 ...
        Attributes:
            Title:                GEOS-FP instantaneous 3-hour parameters (I3), proc...
            Contact:              GEOS-Chem Support Team (geos-chem-support@as.harva...
            References:           www.geos-chem.org; wiki.geos-chem.org
            Filename:             GEOSFP.20130701.I3.4x5.nc
            History:              File generated on: 2013/10/24 12:26:39 GMT-0300
```

```
ProductionDateTime:    File generated on: 2013/10/24 12:26:39 GMT-0300
ModificationDateTime:  File generated on: 2013/10/24 12:26:39 GMT-0300
Format:                NetCDF-3
SpatialCoverage:       global
Conventions:           COARDS
Version:               GEOS-FP
Model:                 GEOS-5
Nlayers:               72
Start_Date:            20130701
Start_Time:            00:00:00.0
End_Date:              20130701
End_Time:              23:59:59.99999
Delta_Time:            030000
Delta_Lon:             5
Delta_Lat:             4
```

```
In [6]: ax = plt.axes(projection=ccrs.PlateCarree())
        ds_met['T'][0,0].plot(cmap='jet', ax=ax)
        ax.coastlines()
        plt.title('surface temperature');
```



1.6.4 Sample Python code to analyze NASA-NEX data

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import xarray as xr
        import cartopy.crs as ccrs
```

```
In [2]: # Data already downloaded by
        # aws s3 cp s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/tasmax/r1i1p1/v1.0/tasmax_day_BCSD_rcp
        ds = xr.open_dataset("./tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2100.nc")
        ds
```

```
Out[2]: <xarray.Dataset>
        Dimensions:  (lat: 720, lon: 1440, time: 365)
        Coordinates:
```

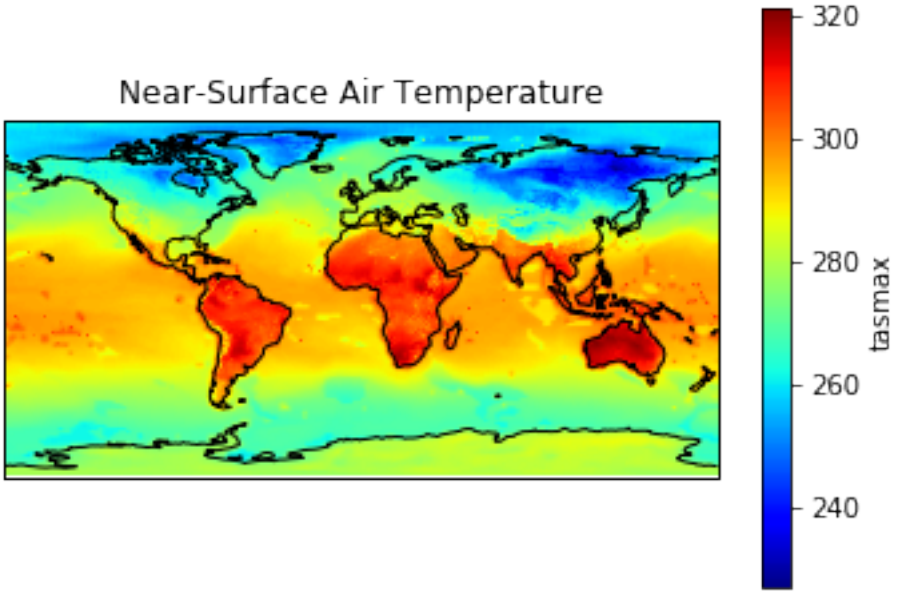
```
  * time     (time) datetime64[ns] 2100-01-01T12:00:00 2100-01-02T12:00:00 ...
  * lat      (lat) float32 -89.875 -89.625 -89.375 -89.125 -88.875 -88.625 ...
  * lon      (lon) float32 0.125 0.375 0.625 0.875 1.125 1.375 1.625 1.875 ...
Data variables:
    tasmax   (time, lat, lon) float32 ...
Attributes:
    parent_experiment:              historical
    parent_experiment_id:           historical
    parent_experiment_rip:          r1i1p1
    Conventions:                    CF-1.4
    institution:                    NASA Earth Exchange, NASA Ames Research C...
    institute_id:                   NASA-Ames
    realm:                          atmos
    modeling_realm:                 atmos
    version:                        1.0
    downscalingModel:               BCSD
    experiment_id:                  rcp85
    frequency:                      day
    realization:                    1
    initialization_method:          1
    physics_version:                1
    tracking_id:                    f97f5681-30cf-4b49-9380-e6dae253fb6c
    driving_data_tracking_ids:      N/A
    driving_model_ensemble_member:  r1i1p1
    driving_experiment_name:        historical
    driving_experiment:             historical
    model_id:                       BCSD
    references:                     BCSD method: Thrasher et al., 2012, Hydro...
    DOI:                            http://dx.doi.org/10.7292/W0MW2F2G
    experiment:                     RCP8.5
    title:                          INMCM4 global downscaled NEX CMIP5 Climat...
    contact:                        Dr. Rama Nemani: rama.nemani@nasa.gov, Dr...
    disclaimer:                     This data is considered provisional and s...
    resolution_id:                  0.25 degree
    project_id:                     NEXGDDP
    table_id:                       Table day (12 November 2010)
    source:                         BCSD 2014
    creation_date:                  2015-01-07T20:33:31Z
    forcing:                        N/A
    product:                        output
```

In [3]:
```
ds['tasmax']
```

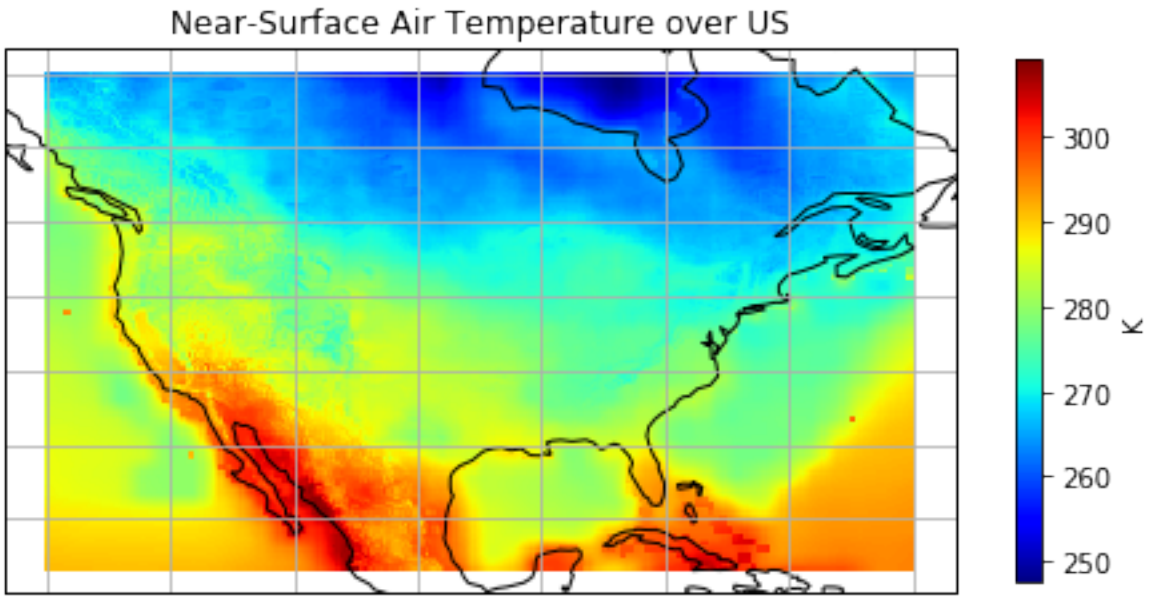
Out[3]:
```
<xarray.DataArray 'tasmax' (time: 365, lat: 720, lon: 1440)>
[378432000 values with dtype=float32]
Coordinates:
  * time     (time) datetime64[ns] 2100-01-01T12:00:00 2100-01-02T12:00:00 ...
  * lat      (lat) float32 -89.875 -89.625 -89.375 -89.125 -88.875 -88.625 ...
  * lon      (lon) float32 0.125 0.375 0.625 0.875 1.125 1.375 1.625 1.875 ...
Attributes:
    time:             32850.5
    standard_name:    air_temperature
    long_name:        Daily Maximum Near-Surface Air Temperature
    comment:          daily-maximum near-surface (usually, 2 meter) air temp...
    units:            K
    original_name:    tasmax
    cell_methods:     time: maximum (interval: 1 day)
    cell_measures:    area: areacella
    history:          2010-10-25T09:20:20Z altered by CMOR: Treated scalar d...
    coordinates:      height
```

```
            associated_files:  baseURL: http://cmip-pcmdi.llnl.gov/CMIP5/dataLocation...
```

```
In [4]: ax = plt.axes(projection=ccrs.PlateCarree())
        ds['tasmax'][0].plot(cmap='jet')
        ax.coastlines()
        plt.title('Near-Surface Air Temperature');
```




```
In [5]: plt.figure(figsize=[8, 6])
        ax = plt.axes(projection=ccrs.PlateCarree())
        (ds['tasmax'][0].
         sel(lon=slice(230, 300), lat=slice(20, 60)).
         plot(cmap='jet', cbar_kwargs={'shrink': 0.6, 'label': 'K'})
        )
        ax.coastlines()
        ax.gridlines()
        plt.title('Near-Surface Air Temperature over US');
```


Near-Surface Air Temperature over US
300
290
280
270
260
250
K