cloud-GC Documentation

Release 0.1

Jiawei Zhuang

Dec 16, 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).

Note: For any questions, bug reports, or functionality requests, please raise a new issue on the GitHub issue tracker. It is a much better approach than sending emails, because all discussions are public and can be easily found by anyone with similar problems.

How to use this documentation

For GEOS-Chem users, this website contains everything you need in order to use GEOS-Chem on the cloud. You will be able to finish a complete research workflow, from model simulations to output data analysis and management. **If it is your first time trying GEOS-Chem, this project is perhaps your best starting point**, because *you don't need to do any initial setup* and the model is guaranteed to work correctly (see *quick start guide*). Note that this website is not a user guide on the GEOS-Chem model itself. Please refer to our comprehensive user guide and wiki for all details about GEOS-Chem. To run on the cloud, we only support versions newer than v11-02a for GNU-Fortran compatibility.

For non-GEOS-Chem-users, this documentation can be used as an introduction to AWS for scientific computing, especially for **Earth science model simulations**. Since all Earth science models are highly similar from a software perspective, it should be quite easy to adapt this guide for you specific use case. More than 90% of this website is about general AWS concepts and tutorials, which doesn't require GEOS-Chem-specific knowledge. Please get a feeling of cloud computing workflow by exploring *beginner tutorials* and then refer to the *developer guide* to build your own model. Although cloud computing has a lot of potential in Earth science, it is still significantly under-utilized due to *the lack of accessible tutorials* for Earth science researchers. This project tries to fill this gap.

For general reference, GEOS-Chem is a Chemical Transport Model for simulating atmospheric chemical compositions. It has been developed over 20 years and is used by more than 100 research groups worldwide. The program is mainly written in Fortran 90. All model source code is distributed freely under the MIT license. Input and output data formats are mostly NetCDF, which can be analyzed easily by most languages such as Python, R and MATLAB. IDL (Interactive Data Language) has historically been the major data analysis tool but now we embrace open-source tools especially Python, Jupyter and xarray. The classic version of GEOS-Chem uses OpenMP parallelization (shared-memory, multi-threading). The MPI version of GEOS-Chem has also been developed and we have an *an experimental version that runs on the cloud*.

Table of Contents

2.1 Overview

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

2.1.1 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 writing 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. 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. This is *introduced in advanced tutorials*.

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.

2.1.2 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.

Relevant 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!)

2.1.3 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.

2.2 Beginner tutorials

This chapter provides hands-on tutorials for absolute beginners with no cloud computing experience. **Those beginner tutorials are aimed to be read in-order!**

2.2.1 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.
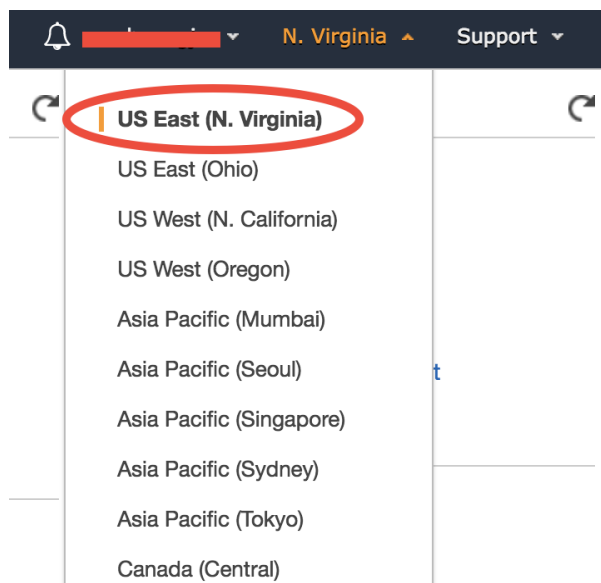

In the EC2 console, make sure you are in the **US East (N. Virginia)** region as shown in the upper-right corner of your console. Choosing a region closer to your physical location will give you better network. To keep this tutorial minimal, I built the system in only one region. But working across regions 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-08c83a8b3ebd20b63** or **GEOSChem_tutorial_20180926** – that's the system with GEOS-Chem installed. Select it and click on "Launch".

An AMI full specifies the "software" side of your virtual server, including the operating system, software libraries, and default data files. Then it's time to specify the "hardware" side, mostly about CPUs.

You can select from a large number of CPU types at "Step 2: Choose an Instance Type". In this toy example, choose "Memory optimized"-"r5.large" which meets the minimal hardware requirement for GEOS-Chem:

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 will be asked to create and download a file called "Key Pair". It is equivalent to the password you enter to `ssh` to your local server, but much more secure.

Give your "Key Pair" 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 (shown in the above figure) near the blue "Launch Instance" button, then you should see this instruction 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`)

EC2 Dashboard

Events

Tags

Reports

Limits

- INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

Scheduled Instances

- IMAGES

AMIs

Bundle Tasks

Launch

Actions

Public images ⌄

☑ | **Name**

☑
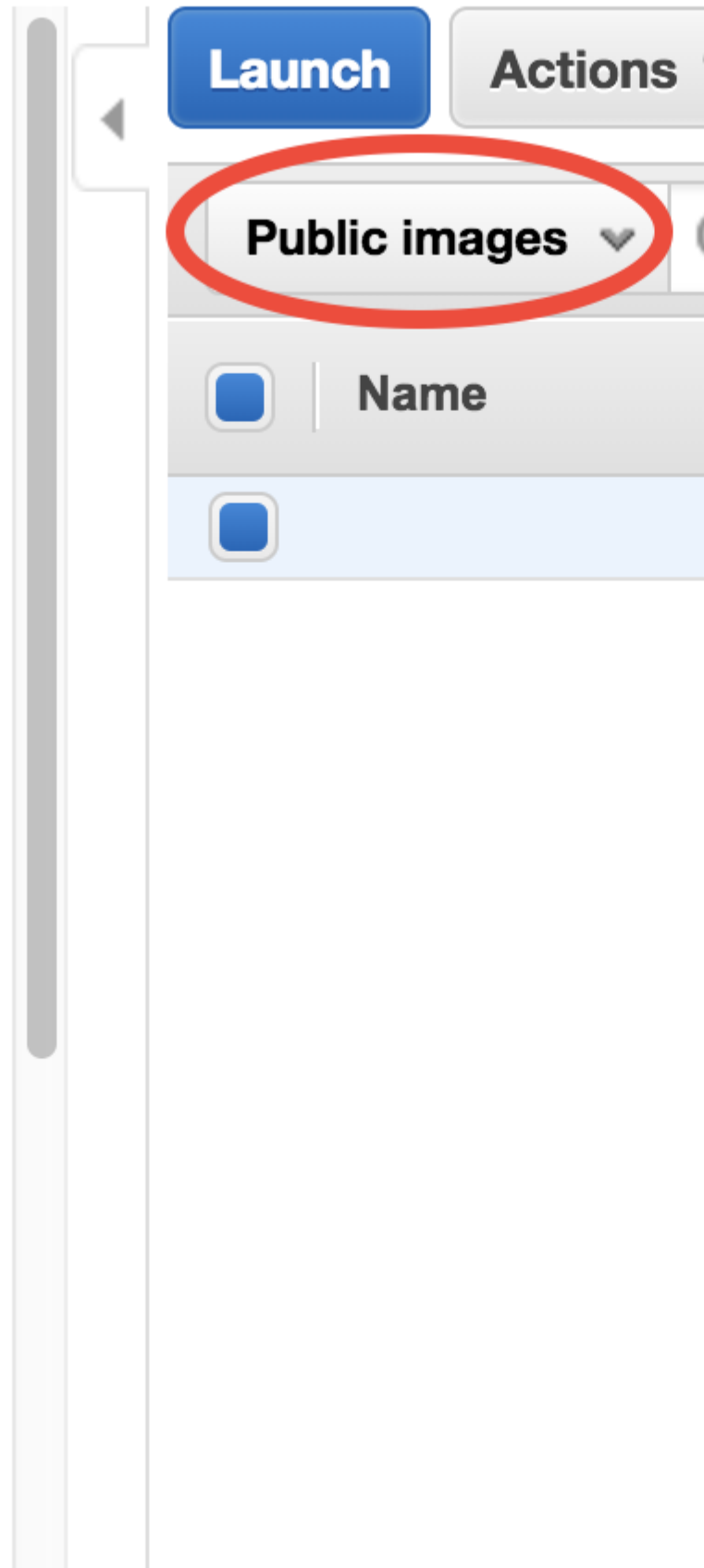
Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types

capacity, and give you the flexibility to choose the appro

Filter by: **Memory optimized** ▼   **Current ge**

Currently selected: r5.large (10 ECUs, 2 vCPUs, 2.5 G

	Family	Type
☐	Memory optimized	r5d.large
☐	Memory optimized	r5d.xlarg
☐	Memory optimized	r5d.2xlarg
☐	Memory optimized	r5d.4xlarg
☐	Memory optimized	r5d.12xlar
☐	Memory optimized	r5d.24xlar

(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 (MobaXterm should be a lot easier). Alternatively, you can use the Windows Subsystem for Linux and then simply follow the above steps for Mac/Linux.

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 ~/tutorial/geosfp_4x5_standard
```

Just run the pre-compiled the model by:
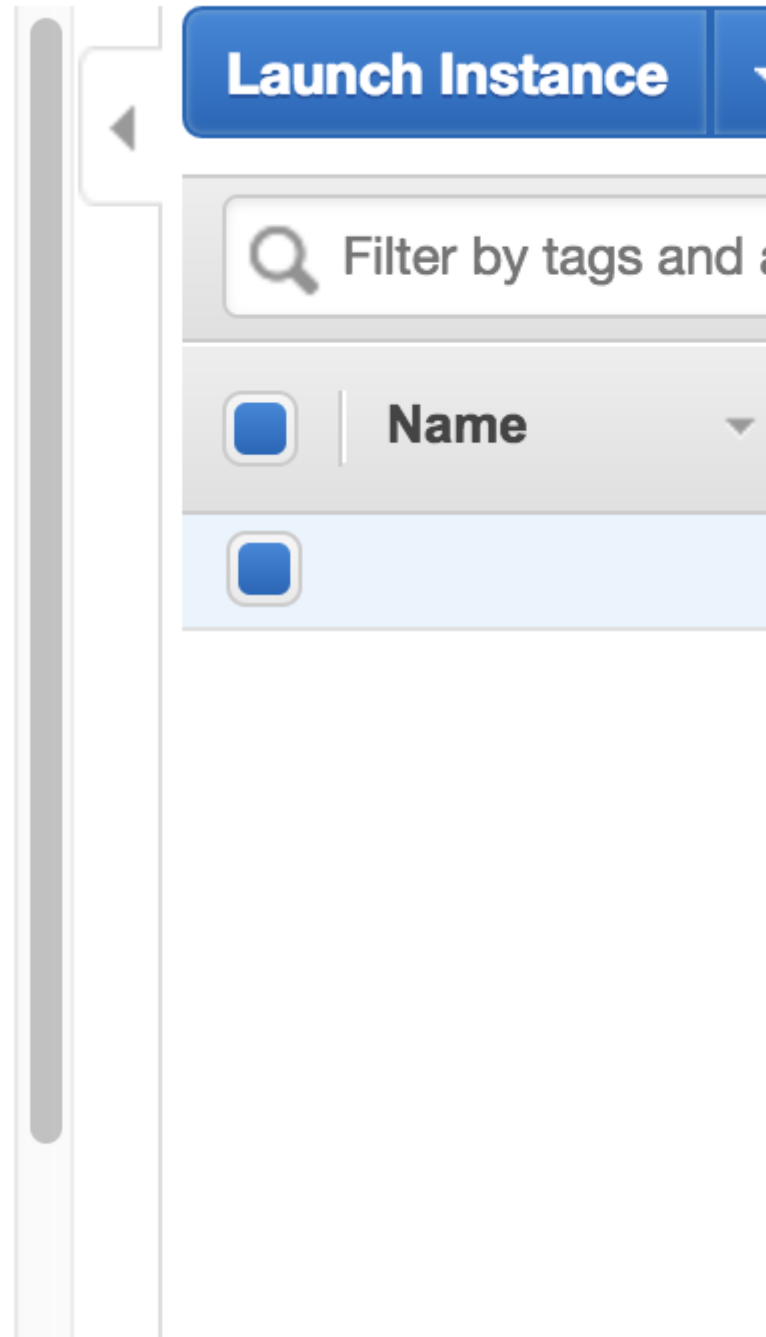
```
$ ./geos.mp
```

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

```
$ make realclean
$ 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!

EC2 Dashboard

Events

Tags

Reports

Limits

INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

Scheduled Instances

Launch Instance

Filter by tags and a

Name

Connect To Your Instance ✕

I would like to connect with ⦿ A standalone SSH client
◯ A Java SSH Client directly from my browser (Java required)

To access your instance:

1. Open an SSH client. (find out how to connect using PuTTY)

2. Locate your private key file (my-aws-key.pem). The wizard automatically detects the key you used to launch the instance.

3. Your key must not be publicly viewable for SSH to work. Use this command if needed:

   `chmod 400 my-aws-key.pem`

4. Connect to your instance using its Public DNS:

   `ec2-54-236-245-121.compute-1.amazonaws.com`

Example:

`ssh -i "my-aws-key.pem" root@ec2-54-236-245-121.compute-1.amazonaws.com`

Please note that in most cases the username above will be correct, however please ensure that you read your AMI usage instructions to ensure that the AMI owner has not changed the default AMI username.

The default simulation length is only 20 minutes, for demonstration purpose. The "r5.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.

Note: This system is a **general environment** for GEOS-Chem, **not just a specific version of the model**. This pre-configured run directory in the "tutorial" folder is only for demonstration purpose. *Later tutorials* will show you how to set up custom versions and configurations.

Step 4: Analyze output data with Python (Optional)

If you wait for the simulation to finish (takes 5~10 min), it will produce NetCDF diagnostics called `GEOSChem.SpeciesConc.20160701.nc4` inside `OutputDir/` of the run directory. To save time, you can also cancel the simulation and use the pre-generated file with the same name:

```
$ cd ~/tutorial/geosfp_4x5_standard/OutputDir/
$ ncdump -h GEOSChem.SpeciesConc.20160701_0000z.nc4
netcdf GEOSChem.SpeciesConc.20160701_0000z {
dimensions:
     time = UNLIMITED ; // (1 currently)
     lev = 72 ;
     ilev = 73 ;
```

(continues on next page)

```
.ssh $chmod 400 my-aws-key.pem
.ssh $ssh -i "my-aws-key.pem" ubuntu
Welcome to Ubuntu 16.04.5 LTS (GNU/L

   * Documentation:  https://help.ubun
   * Management:     https://landscape
   * Support:        https://ubuntu.co

    Get cloud support with Ubuntu Adv
        http://www.ubuntu.com/business/s

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

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade

Last login: Thu Sep 27 16:44:39 2018
ubuntu@ip-172-31-27-55:~$ ls
ExtData  miniconda  tutorial
ubuntu@ip-172-31-27-55:~$ cd tutoria
```

```
      lat = 46 ;
      lon = 72 ;
variables:
      double time(time) ;
              time:long_name = "Time" ;
    time:units = "minutes since 2016-07-01 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 [Python/xarray tutorial for GEOS-Chem users](#).

Activate the pre-installed [geoscientific Python environment](#) by `source activate geo` (it is generally a bad idea to directly install things into the root Python environment), and then start `ipython` from the command line:

```
$ source activate geo  # I also set a `act geo` alias
$ ipython
Python 3.6.6 |Anaconda, Inc.| (default, Jun 28 2018, 17:14:51)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import xarray as xr

In [2]: ds = xr.open_dataset('GEOSChem.SpeciesConc.20160701_0000z.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`:

```
$ source activate geo
$ 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* that you can just execute. It is located at:

```
~/tutorial/python_example/sample-python-code.ipynb
```

Besides being a data analysis environment, Jupyter can also be used as a graphical text editor on remote servers so you don't have to use `vim`/`emacs`/`nano`. The Jupyter console also allows you to download/upload data without using `scp`. The next generation of notebooks, namely Jupyter Lab, is also installed. Just change the launching command from `jupyter notebook ...` to `jupyter lab ...` if you want to have a try.

Note: There are many ways to connect to 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.

We encourage users to try the new NetCDF diagnostics, but you can still use the old BPCH diagnostics if you want to. Just compile with `NC_DIAG=n BPCH_DIAG=y` instead. The Python package xbpch can read BPCH data into xarray format, so you can use very similar code for NetCDF and BPCH output. xbpch is pre-installed in the `geo` environment. My xESMF package is also pre-installed, which can fulfill almost all horizontal regridding needs for GEOS-Chem data (and most of Earth science data).

Also, you could indeed download the output data and use old tools like IDL & MATLAB to analyze them, but we highly recommend the open-source Python/Jupyter/xarray ecosystem. It will vastly improve user experience and working efficiency, and also help open science and reproducible research.

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

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

EC2 Dashboard

Events

Tags

Reports

Limits

⊟ INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

Scheduled Instances

⊟ IMAGES

AMIs

Bundle Tasks

Launch Instance

Q Filter by tags and a

Name

Co
Ge
Lau

Ins
Ins
Ima
Net
Clo
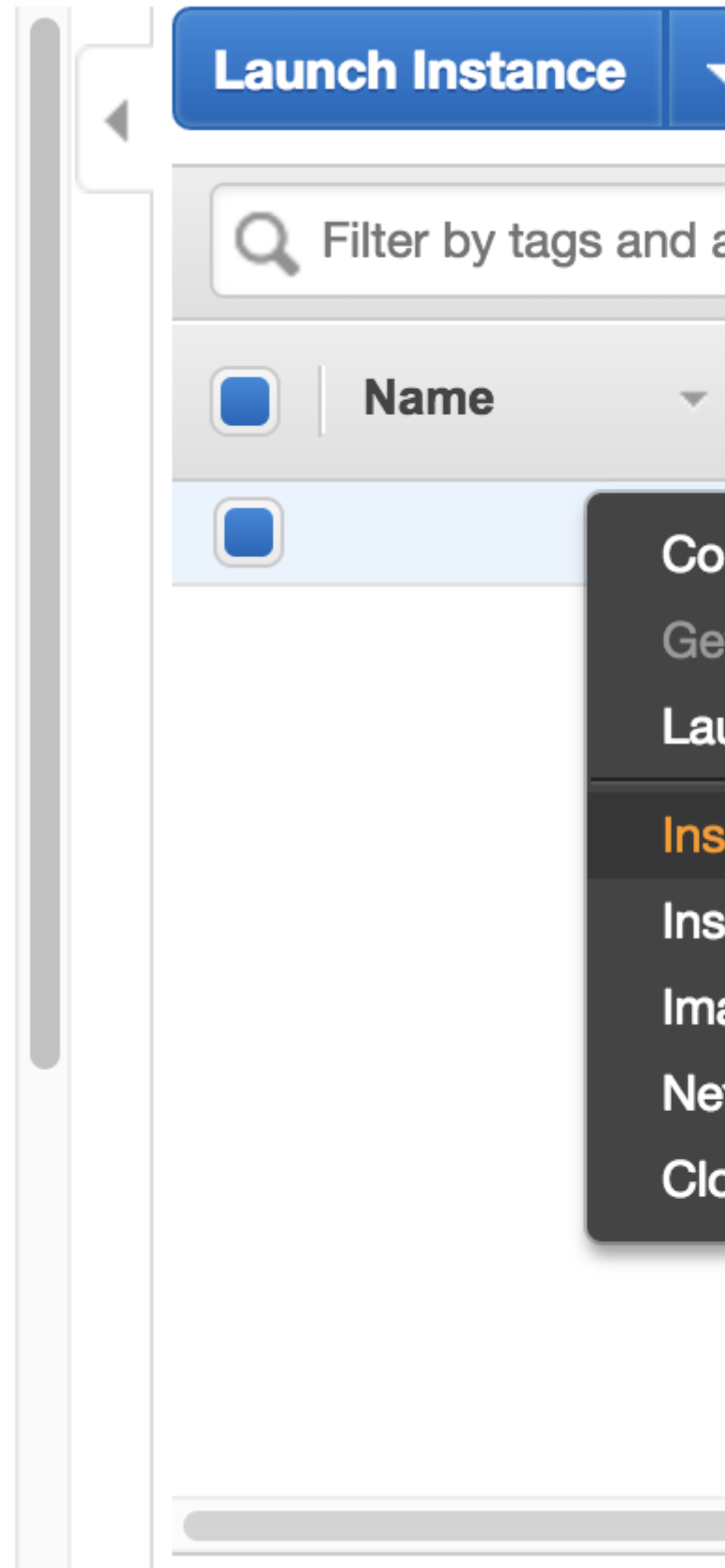
There are two different ways to stop being charged:

- "Stop" will make the system inactive, so that you'll not be charged by the CPU time, but 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. You might also want to *simplify your ssh login command*.

2.2.2 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 targeted at IT/Business applications that scientists can safely ignore:

Core AWS concepts for scientific computing

In this section you will familiarize 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, persistent 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 **transferring 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.

That's all core concepts you need to remember. For a more complete review, see *AWS concepts and services in detail*. For first-time readers, simply go to the next hands-on tutorial. The best way to learn cloud computing is trying it yourself.

2.2.3 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, so I defer "IAM users" to advanced tutorials. For now, simply choose "Continue to Security Credentials" to get the science done as quickly as possible. (If you want to follow the IAM user approach right now, see the official tutorial.)
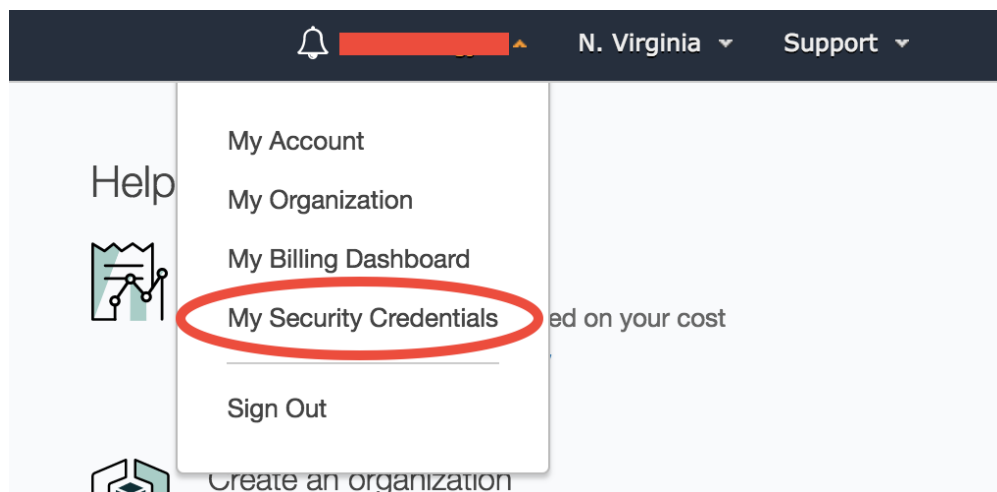
In the Security Credential console, choose Access keys – Create New Access Key:

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.

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]: xxxxxxxxxxxxxxxxxxxx
AWS Secret Access Key [None]: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
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 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, or 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/
```

(continues on next page)

Your Security Credentials

Use this page to manage the credentials for your AWS

To learn more about the types of AWS credentials and

+	Password
+	Multi-factor authentication (MFA)
+	Acc
+	Clou
+	X.50
+	Acc

You are accessing the security cre

access to your AWS resources.

To help secure your account, follo

Management (IAM) users with limit

Continue t

Your Security Credentials

Use this page to manage the credentials for your AWS a

To learn more about the types of AWS credentials and h

+	Password
+	Multi-factor authentication (MFA)
–	Access keys (access key ID and secr

You use access keys to sign programmatic requests t
your access keys securely and do not share them. In

Note: You can have a maximum of two access keys (a

Created	**Deleted**	**Acc**

Create New Access Key

⚠ Important Change: Managing You

As described in a previous announcement,

Create Access Key

☑ **Your access key (access key ID and secret acces**

**Download your key file now, which contains your ne
download the key file now, you will not be able to r**

To help protect your security, store your secret access

▶ Show Access Key

Download

Created	Deleted	Acces
Mar 9th 2018		

Create New Access Key

```
                         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? After all, 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` every time 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 credentials – it is pretty quick!

2.2.4 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 recursively 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 example 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 preserve 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 ;
      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.

Access GEOS-Chem input data repository in S3

List our bucket by:

```
$ 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/
```

(continues on next page)

(continued from previous page)

```
                             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. Transferring data from S3 to EC2 (in the same region) has no cost. But you do need to pay for the *egress fee* if you download data to local machines.

The tutorial AMI only has 4x5 GEOS-FP metfield for 1-month (2016/07). You can get other metfields from that S3 bucket, to support simulations with any configurations.

For example, download the 4x5 GEOS-FP data over the next month (2016/08)

```
$ aws s3 cp --request-payer=requester --recursive \
  s3://gcgrid/GEOS_4x5/GEOS_FP/2016/08/ \
  ~/ExtData/GEOS_4x5/GEOS_FP/2016/08/

download: s3://gcgrid/GEOS_4x5/GEOS_FP/2016/08/GEOSFP.20160801.A3mstC.4x5.nc to␣
↪ExtData/GEOS_4x5/GEOS_FP/2016/08/GEOSFP.20160801.A3mstC.4x5.nc
...
```

Downloading this ~2.5 GB data should just take 10~20s.

To download more months (but not the entire year), consider simple bash "for" loop:

```
for month in 09 10
do
aws s3 cp --request-payer=requester --recursive \
  s3://gcgrid/GEOS_4x5/GEOS_FP/2016/$month \
  ~/ExtData/GEOS_4x5/GEOS_FP/2016/$month
done
```

Wildcards are also supported, but it feels pretty different from common Linux wildcards. I often find writing bash scripts a lot quicker.

Then you may want to change the simulation date in `input.geo` to test the new data. For example, change to next month:

```
Start YYYYMMDD, hhmmss  : 20160801 000000
End   YYYYMMDD, hhmmss  : 20160901 000000
Run directory           : ./
Input restart file      : GEOSChem_restart.201607010000.nc
```

(Note that the restart file is still at 2016/07 in this case.)

The EC2 instance launched from the tutorial AMI only has limited disk by default, so the disk will be full very soon. You will learn how to increase the disk size, right in the next tutorial.

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. However, it doesn't work well with requester-pay buckets yet (issue#635). If that issue is resolved we will add more instructions.

2.2.5 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 a limited 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 70~100 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:

INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

Scheduled Instances

IMAGES

AMIs

Bundle Tasks

ELASTIC BLOCK STORE

Volumes

Snapshots

Create Volume

Filter by tags and a

Name

1. Choose AMI   2. Choose Instance Type   3. Configure

Step 4: Add Storage

Your instance will be launched with the following storage
edit the settings of the root volume. You can also attach
storage options in Amazon EC2.

Volume Type ⓘ	Device ⓘ	Snapshot
Root	/dev/sda1	snap-04da595

Add New Volume

Volumes > Create Volume

Create Volume

Volume Type	General Pu
Size (GiB)	200
IOPS	600 / 3000
Availability Zone*	us-east-1a

Q Filter b

us-east-1

us-east-1

Throughput (MB/s)

Snapshot ID

us-east-1

us-east-1

Instance Type	▼	Availability Zone	▼	Instance State	▼
r4.large		us-east-1c		🟢 running	

Size	▼	Volume Type	▼	IOPS	▼	Snaps
200 GiB		gp2		600 / 3000		
70 GiB		gp2		210 / 3000		snap-0

The "in-use" one is the disk for the running 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":

Size	▼	Volume Type	▼	IOPS	▼	Snapsh
200 GiB		g	Modify Volume			
70 GiB		g	Create Snapshot			ap-04
			Delete Volume			
			Attach Volume			
			Detach Volume			
			Force Detach Volume			
			Change Auto-Enable IO Setting			
			Add/Edit Tags			

You should be prompted with the ID of the running EC2 instance. If nothing gets prompted, double check if you have chosen 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

Attach Volume ✕

Volume ⓘ	vol-0563f0a163da7ca62 in us-east-1c	
Instance ⓘ	Search instance ID or Name tag	in us-east-1c
Device ⓘ	i-0aef9f4f469a2a37d (running)	

Cancel **Attach**

"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! The 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 console, 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/month and S3 price is $23/TB/month. There is something in between, called "snapshot EBS volumes to S3", which causes $50/TB/month. 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.

2.2.6 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 network 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 to be stable for a long time, but it is often an overkill for scientific computing workflows 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.

EC2 Dashboard

Events

Tags

Reports

Limits

INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

Scheduled Instances

IMAGES

AMIs

Bundle Tasks

Request Spot I

Request typ

Reques

Select a Spot

cloud-GC Documentation, Release 0.1

2.2. Beginner tutorials 41

Each "*Availability Zone*" has its own spot price. You will get the cheapest one by default.

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.

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. It is also OK to leave the "Maximum price" option blank; the on-demand price will be used as default.

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).

Spot Instance Pricing History

Product: Linux/UNIX ⬍   Instance type: p2.xlarge

1. Choose AMI   2. Choose Instance Type   3. Configure

Step 3: Configure Instance Detail

Configure the instance to suit your requirements. You ca

more.

Number of instances ⓘ   1

Purchasing option ⓘ   ☑ Re

Current price ⓘ   Avai

us-ea

us-ea

us-ea

us-ea

us-ea

us-ea

Maximum price ⓘ $ 0.7

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 transfered 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.

2.2.7 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:

---

1. Choose AMI   2. Choose Instance Type   3. Configure

Step 6: Configure Security Group

A security group is a set of firewall rules that control the
Internet traffic to reach your instance, add rules that allo
groups.

Assign a security group: ● Create

○ Select

Security group name: laun

Description: laun

Type ⓘ	Protocol ⓘ
SSH ⇕	TCP

Add Rule

⚠ Warning

Rules with source of 0.0.0.0/0 allow all IP add

Assign a security group: ◯Create

◉Select

Security Group ID

☑ sg-0c67bd7b

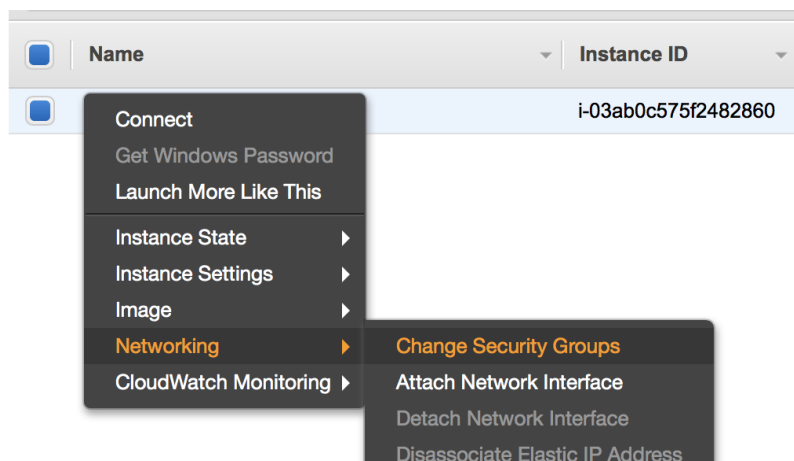**Inbound rules for sg-0c67bd7b (Selected security gr**

Type ⓘ   **Protocol**

All traffic   All

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: 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.

2.2.8 Put everything together: a complete workflow

Congrats! If you've been reading the tutorials in order, now you should know enough AWS stuff to perform most of simulation, data analysis, and data management 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.

General comments on cloud vs local computer

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 forever (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 utilize the power and flexibility of cloud platforms, directly use native, basic services like EC2 and S3.

A reference workflow

Here's the outline of a typical research workflow:

IMAGES

AMIs

Bundle Tasks

ELASTIC BLOCK STORE

Volumes

Snapshots

NETWORK & SECURITY

Security Groups

Elastic IPs

Placement Groups

Key Pairs

Network Interfaces

Create Security Gro

Q Filter by tags and

Name

Select a security gro

1. Launch EC2 instances from pre-configured AMI. Consider spot instances for big computing. *Consider AWSCLI to simply the above process to one shell 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. Re-login at anytime to check progress.

5. Use Python/Jupyter to analyze output data.

6. When the EC2 instance is not needed anymore, transfer output data and customized model configuration (mostly run directories) to S3. Or download them to local machines if necessary (Recall that *egress charge* is $90/TB; for several GBs the cost is negligible).

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).

9. Whenever need to continue computing, launch new EC2 instances, pull previous data from S3.

Talk is cheap. Let's actually walk through them.

Below are reproducible steps (copy & paste-able commands) to set up a custom model run. We use a half-day, 2x2.5 simulation as an example, but the same idea applies to other types of runs. **Most laborious steps only need to be done once**. Subsequent workflow will be much simpler.

I assume you've read all previous sections. Don't worry if you can't remember everything – there will be links to previous sections whenever necessary.

Launch EC2 instance with custom configuration

A complete EC2 configuration has 7 steps, with tons of options throughout the steps:

You typically only touch very few options, as listed in order below.

- Choose our tutorial AMI *just as in the quick start guide*. This effectively did "Step 1: Choose an Amazon Machine Image (AMI)". You will be brought to "Step 2: Choose an Instance Type".

- At Step 2, choose the "Compute optimized" family, select `c5.4xlarge`, which is suitable for medium-sized simulations. For longer-term, higher-resolution runs, consider even bigger ones like `c5.9xlarge` and `c5.18xlarge`.

- At "Step 3, Configure Instance Details", select "Request Spot instances". *See here to review spot instances configuration*. (At this step you also have the chance to select an "IAM role" to simplify AWSCLI configuration for S3, as *explained in advanced tutorial*.)

- At "Step 4: Add Storage", **increase the size to 400~500 GB** to host more input/output data. *See here to review EBS volume configuration*.

- Nothing to do for "Step 5: Add Tags". Just go to the next step. You can always add resource tags (just convenient labels) anytime later.

- At "Step 6: Configure Security Group", select a proper security group. *See here to review security group configuration*. If you don't bother with security group configurations, simply choose "Create a new security group" (it works but not optimal).

- Nothing to do for "Step 7: Review Instance Launch". Just click on "Launch".

Occasionally you might hit EC2 instance limit, especially when you try to launch a very large instance on a new account. Just request for limit increase. if that happens.

Advanced tutorial will show you how to *use AWSCLI to simply the above process to one shell command*.

Set up your own model configuration

Log into the instance *as in the quick start guide*. Here you will set up you own model configuration, instead of using the pre-configured tutorial run directory. You can also change model version – any versions newer than v11-02a should work smoothly. The system will still work with future releases of GEOS-Chem, unless there are big structural changes that break the compile process.

Existing GEOS-Chem users should feel quite familiar about the steps presented here. New users might need to refer to our user guide for more complete explanation.

Get source code and checkout model version

You can obtain the latest copy of the code from GEOS-Chem's GitHub repo:

```
$ mkdir ~/GC  # make you own folder instead using the "tutorial" folder.
$ cd ~/GC
$ git clone https://github.com/geoschem/geos-chem Code.12.0.1  # might use different
→names for future versions
$ git clone https://github.com/geoschem/geos-chem-unittest.git UT
```

You may list all versions (they are just git tags) in chronological order:

```
$ cd Code.12.0.1
$ git log --tags --simplify-by-decoration --pretty="format:%ci %d"
2018-08-22 16:57:08 -0400  (HEAD -> master, tag: 12.0.1, origin/master, origin/HEAD)
2018-08-09 16:59:22 -0400  (tag: 12.0.0)
2018-07-30 10:31:40 -0400  (tag: 12.0.0-1yr-bm, tag: 12.0.0-1mo-bm)
2018-06-21 10:04:24 -0400  (tag: v11-02-release-candidate, tag: v11-02-rc)
2018-05-11 16:31:42 -0400  (tag: v11-02f-1yr-Run1)
...
```

New users had better just use the default, latest version to minimize confusion. Experienced users might want to checkout to a different version, say `12.0.0`:

```
$ git checkout 12.0.0  # just the name of the tag
$ git branch
* (HEAD detached at 12.0.0)
$ git checkout master  # restore the latest version if you want
```

You need to do version checkout for both source code and unit tester.

Configure unit tester and generate run directory

Then you need to generate run directories from unit tester:

In `UT/perl/CopyRunDirs.input`, change the default paths:

```
GCGRID_ROOT     : /n/holylfs/EXTERNAL_REPOS/GEOS-CHEM/gcgrid
DATA_ROOT       : {GCGRIDROOT}/data/ExtData
...
UNIT_TEST_ROOT : {HOME}/UT
...
COPY_PATH       : {HOME}/GC/rundirs
```

to:

```
GCGRID_ROOT     : /home/ubuntu
DATA_ROOT       : {GCGRIDROOT}/ExtData
...
UNIT_TEST_ROOT : {HOME}/GC/UT
...
COPY_PATH       : {HOME}/GC
```

Then uncomment the run directory you want:

```
geosfp   2x25        -       standard    2016070100   2016080100      -
```

In `UT/perl/Makefile`, make sure the source code path is correct:

```
CODE_DIR    :=$(HOME)/GC/Code.$(VERSION)
```

Finally, generate the run directory:

```
$ ./gcCopyRunDirs
```

Go to the run directory and compile:

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

Note that you almost have to execute `make` command **in the run directory**. This will ensure the correct combination of compile flags for this specific run configuration. GEOS-Chem's compile flags have become so complicated that you will almost never get the right compile settings by compiling in the source code directory. See our wiki for more information.

Get more input data from S3

If you just run the executable `./geos.mp`, it will complain about missing input data. Remember that the default `~/ExtData` folder only contains sample data for a demo 4x5 simulation; other data need to be retrieved from S3 using AWSCLI commands (*see here to review S3 usage*). In order to use AWSCLI on EC2, you need to either *configure credentials (beginner approach)* or *configure IAM role (advanced approach)*.

Try `aws s3 ls` to make sure AWSCLI is woking. Then retrieve data by:

```
# GEOSFP 2x2.5 CN metfield
aws s3 cp --request-payer=requester --recursive \
s3://gcgrid/GEOS_2x2.5/GEOS_FP/2011/01/ ~/ExtData/GEOS_2x2.5/GEOS_FP/2011/01/
```

(continues on next page)

```
# GEOSFP 2x2.5 1-month metfield
aws s3 cp --request-payer=requester --recursive \
s3://gcgrid/GEOS_2x2.5/GEOS_FP/2016/07/ ~/ExtData/GEOS_2x2.5/GEOS_FP/2016/07/

# 2x2.5 restart file
aws s3 cp --request-payer=requester \
s3://gcgrid/SPC_RESTARTS/initial_GEOSChem_rst.2x25_standard.nc ~/ExtData/SPC_RESTARTS

# fix the softlink in run directory
ln -s ~/ExtData/SPC_RESTARTS/initial_GEOSChem_rst.2x25_standard.nc ~/GC/geosfp_2x25_
↪standard/GEOSChem_restart.201607010000.nc
```

Tweak run-time configurations

Here shows very common customizations in the run directory. You might further tweak any settings as needed.

In `input.geos`, change the simulation length to 12 hours instead of 1 month.

```
End   YYYYMMDD, hhmmss  : 20160701 120000
```

Note: If you do need to run the simulation over months, remember to pull more metfields in the previous step. For example, metfields for the entire year can be retrieved by `aws s3 cp --request-payer=requester --recursive s3://gcgrid/GEOS_2x2.5/GEOS_FP/2016/ ~/ExtData/GEOS_2x2.5/GEOS_FP/2016/`

In `HEMCO_Config.rc`, tweak emission configurations as needed. Here I disable CEDS due to its data size issue, which will be fixed in 12.1.0.

```
--> CEDS                    :          false
```

In `HISTORY.rc`, change the output path:

```
EXPID:  ./OutputDir/GEOSChem
```

Remember to `mkdir OutputDir` so the path you specified actually exists.

Say I am only interested in Species Concentrations diagnostics. Comment out others:

```
COLLECTIONS: 'SpeciesConc',
#            'AerosolMass',
#            'Aerosols',
#            'CloudConvFlux',
#            'ConcAfterChem',
#            'DryDep',
#            'JValues',
#            'JValuesLocalNoon',
#            'LevelEdgeDiags',
#            'ProdLoss',
#            'StateChm',
#            'StateMet',
#            'WetLossConv',
#            'WetLossLS',
```

Output hourly instantaneous field, instead of the original monthly mean:

```
SpeciesConc.frequency:       00000000 010000
SpeciesConc.duration:        00000000 010000
SpeciesConc.mode:            'instantaneous'
```

Note: To massively change the date for all collections, in `vim` you can perform a subsitition by `:%s/00000100 000000/00000000 010000/g`

Now you should be able to execute the model without problems.

Perform long-term simulation

With tmux, you can keep the program running after logging out.

```
$ tmux
$ ./geos.mp | tee run.log
Type `Ctrl + b`, and then type `d`, to detach from the tmux session

$ tail -f run.log  # display the output message dynamically
Type `Ctrl + c` to quit the message display. Won't affect model simulation.
```

Log out of the server (`Ctrl + d` or just close the terminal). The model will be safely running in the background. You can re-login anytime and check the progress by looking at `run.log`. If you need to cancel the simulation, type `tmux a` to resume the interactive session and then `Ctrl + c` to kill the program.

This half-day simulation will take about half an hour. In the meantime, do whatever you like such as having a cup coffee... Just come back and re-login after half an hour. The same strategy applies to simulations that run over many days. You don't have to keep the terminal open.

Note: What if the model finishes at mid-night? Any way to automatically terminate the instance to stop paying for charge? I tried multiple auto-checking methods but they often bring more troubles than benefits. For example, *the HPC cluster solution* will handle server termination for you, but that often makes the workflow more complicated, especially if you are not a heavy user. Manually examining the simulation on next day is usually the easiest way. The cost of EC2 piles up for simulations that last for many days, but for just one night it is negligible.

Analyze output data

Output data will be inside `OutputDir/` as specified in `HISTORY.rc`. You can *use Jupyter notebooks* to analyze them, or simply `ipython` for a quick check. One tip is that multi-file time-series can be opened as a single object by `xarray.open_mfdataset()`:

```
$ source activate geo
$ ipython
Python 3.6.6 |Anaconda, Inc.| (default, Jun 28 2018, 17:14:51)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import xarray as xr

In [2]: ds = xr.open_mfdataset("GEOSChem.SpeciesConc.20160701_*00z.nc4")  # multiple␣
↪many files at once
```

(continues on next page)

```
In [3]: ds  # 12 time frames in the same object
Out[3]:
<xarray.Dataset>
Dimensions:                 (ilev: 73, lat: 91, lev: 72, lon: 144, time: 12)
...
```

Save your files to S3

Before terminate the EC2 instance, always make sure that input files are transferred to persistent storage (S3 or local). Here we push our custom files to S3 (*see here to review S3+AWSCLI usage*).

```
aws s3 mb s3://my-custom-gc-files  # use a different name for the bucket, with all␣
↪lower cases
aws s3 cp --recursive ~/GC/ s3://my-custom-gc-files  # transfer data
aws s3 ls s3://my-custom-gc-files/  # show the bucket content
```

Only the `~/GC/` folder contains custom configurations. Input data can be easily retrieved from the `s3://gcgrid` bucket. However, if you made you own changes to the input data, remember to also transfer them to S3.

Terminate server, start over whenever needed

Now you can safely *terminate the server*. The next time you want to continue working on this project, **you only need to do two simple things**:

1. Launch EC2 instance. It takes one second if you *use AWSCLI*.

2. Retrieve data files. In this example, the commands are:

```
# Assume that AWSCLI is already configured by either credentials or IAM roles

# customized code, config files, and output data
aws s3 cp --recursive s3://my-custom-gc-files ~/GC/
chmod u+x ~/GC/geos.mp  # restore execution permission

# standard input data from public bucket
aws s3 cp --request-payer=requester --recursive \
s3://gcgrid/GEOS_2x2.5/GEOS_FP/2011/01/ ~/ExtData/GEOS_2x2.5/GEOS_FP/2011/01/
aws s3 cp --request-payer=requester --recursive \
s3://gcgrid/GEOS_2x2.5/GEOS_FP/2016/07/ ~/ExtData/GEOS_2x2.5/GEOS_FP/2016/07/
```

The files on this new EC2 instance will look exactly the same as on the original instance that you terminated last time. In this way, you can get a sustainable workflow on the cloud.

2.3 Advanced tutorials

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

2.3.1 Enable S3 access from EC2 by IAM role

I've promised you *in the beginner tutorial* that you can skip `aws configure` before using AWSCLI on EC2. Here is how. The initial configuration takes a few steps, but once it's done your overall workflow will be simplified quite a bit.

What is Identity and Access Management (IAM)

Identity and Access Management (IAM) is a very powerful utility to **control the permissions** of your AWS resources. More specifically, a "permission" is:

> **X is allowed to use Y**

"Y" is generally some AWS resources, like EC2 or S3. In the most common case, "X" is a specific user. You can create multiple **users** under a single AWS **account** (an "account" is tied to a single credit card); each user can have its unique ID, password, and permissions. This is useful for managing a research group, but not quite useful if you are the only user.

The *Researcher's Handbook* has very detailed instructions on how to set up multiple users (called "IAM users"), so I will not repeat it here. You, the account owner, are also encouraged to create an IAM user for yourself, instead of using the root AWS account to log in (as you've been doing till now). This is again for security reasons. An IAM user will never have access to the billing information and your credit card number, even if that user has the most powerful "AdministratorAccess" which is almost equivalent to root access.

Note: Yes, there are just so many "security best practices" on AWS (Key Pairs, security groups, IAM users. . .), and their benefits are not intuitive to researchers who really just want to get the computing done and publish papers. But please do check out those security stuff when you have time.

To further complicate things, **"X" doesn't have to be a human user, but can also be a AWS resource**. This is what we want to do here – grant S3 access to our EC2 instances, i.e.

> **"X" = our EC2 instances**
> **"Y" = S3 buckets**

"Y" can also be as detailed as "read-only access to S3" (so, no write access) or even "read-only access to a specific S3 bucket" (so, no access to other buckets). All those possible combinations make the IAM console kind of daunting for beginners. Fortunately, here we only need to enable a simple permission rule, which is very easy to do.

Grant S3 permission to EC2

Create a new IAM role

Choose "IAM" in the AWS main console:

It can also be searched from the top search bar, so you don't have to look through hundreds of AWS services:
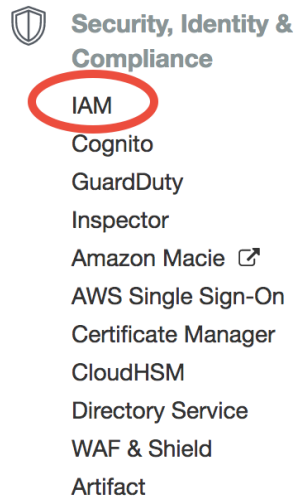
Then choose "Roles" in the IAM console and click on "Create role":

The first step is to choose "X" (which will be allowed to access "Y"). AWS calls it "trusted entity". Select EC2, of course.

The second step is to choose "Y". Search for "S3" and then select "AmazonS3FullAccess":

Finally, give this role a descriptive name. Here I use "full_S3_access_from_EC2". (For the "Role description", enter whatever you like or just keep default.)

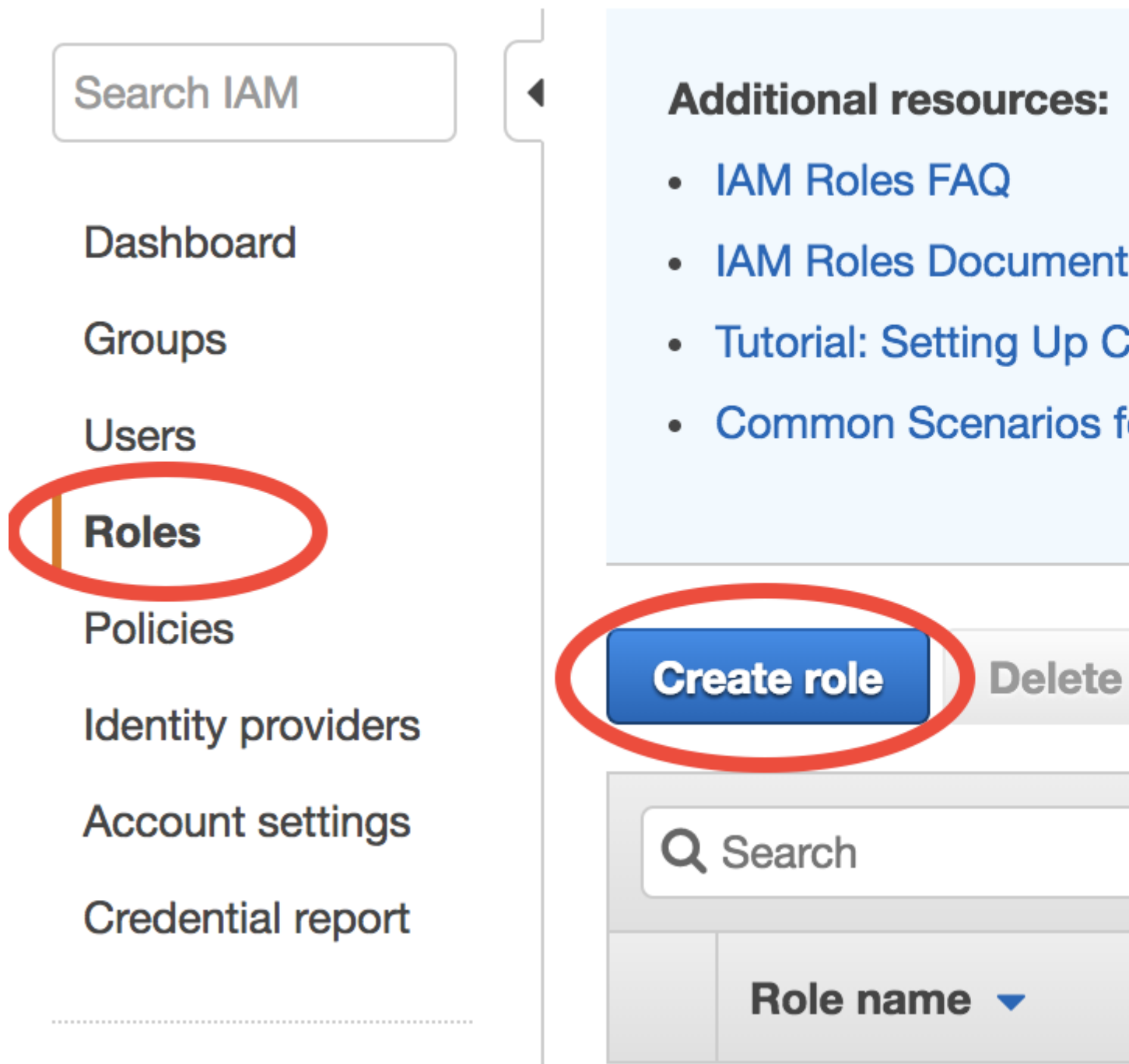Now a new IAM role is created. This only needs to be done once.

Security, Identity &
Compliance

IAM

Cognito

GuardDuty

Inspector

Amazon Macie ⬀

AWS Single Sign-On

Certificate Manager

CloudHSM

Directory Service

WAF & Shield

Artifact

AWS services

IAM

IAM

Manage User Access and Encryption Keys

Create role

Select type of trusted entity

AWS service EC2, Lambda and others	**Anothe** Belonging

Allows AWS services to perform actions on your behalf.

Choose the service that will use this

EC2

Allows EC2 instances to call AWS services on your beh

Lambda

Allows Lambda functions to call AWS services on your

Create role

Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy   **⟳ Refresh**

Filter: **Policy type** ∨   🔍 S3

		Policy name ▼
☐	▶	📦 AmazonDMSRedshiftS3Role
☑	▶	📦 AmazonS3FullAccess
☐	▶	📦 AmazonS3ReadOnlyAccess
☐	▶	📦 QuickSightAccessForS3StorageManage

Create role

Review

Provide the required information below and review this rol

Role name*   full_

Use al

Role description   Allo

Maxim

Trusted entities   AWS

Policies

Assign that role to EC2

Whenever you launch a new EC2 instance, in "Step 3: Configure Instance Details", select the IAM role you created previously for the "IAM role" option.

No need to touch other options on this page and just launch as usual. On this EC2 instance, you don't need to run `aws configure`, and commands like `aws s3 ls` will just work (as long as AWSCLI is installed). This is actually a better practice since you never type your security credentials on this server (which might be stolen if your server gets hacked).

This IAM role configuration can be further *automated by AWSCLI scripts*.

2.3.2 More advanced usages of AWSCLI

Besides accessing S3, AWSCLI can control any kinds of AWS resources you can imagine. A very useful one is `aws ec2 run-instances` (official doc) , as it saves a lot of time clicking through the console.

While `aws s3 xxx` are often used on EC2 instances to interact with S3, `aws ec2 xxx` are mostly used on local computers to control remote servers.

Launch on-demand instances

Use this bash script to launch an on-demand instance:

```bash
#!/bin/bash

# == often change ==
TYPE=r4.large   # EC2 instance type

# ==  set it once and seldom change ==
AMI=ami-xxxxxxx # AMI to launch from
SG=sg-xxxxxxxx  # security group ID
KEY=xxx-key     # EC2 key pair name
COUNT=1         # how many instances to launch
IAM=xxxxx       # EC2 IAM role name

# == almost never change; just leave it as-is ==
aws ec2 run-instances --image-id $AMI \
    --security-group-ids $SG --count $COUNT \
    --instance-type $TYPE --key-name $KEY \
    --iam-instance-profile Name=$IAM
```

- **TYPE**: EC2 instance type.

- **AMI**: The AMI you want to launch from, such as *our tutorial AMI*.

- **SG**: The *security group* you want to assign to your EC2 instance.

- **KEY**: The EC2 Key Pair for `ssh`. For example, in the quick start demo I used *my-aws-key*.

- **COUNT**: You can launch multiple instances with exactly the same configurations

- **IAM**: The *IAM role* for your EC2 instance. Primarily for granting S3 access.

1. Choose AMI   2. Choose Instance Type   3. Configure

Step 3: Configure Instance Detail

Configure the instance to suit your requirements. You car

more.

Number of instances	(i)	1
Purchasing option	(i)	☐ Re
Network	(i)	vpc-
Subnet	(i)	No
Auto-assign Public IP	(i)	Use
IAM role	(i)	full_

Request spot instances

For spot instances, simply add `--instance-market-options '{"MarketType":"Spot"}` option to `aws ec2 run-instances`. Other options are exactly the same as the on-demand case:

```bash
#!/bin/bash

# == often change ==
TYPE=c5.4xlarge  # EC2 instance type

# ==  set it once and seldom change ==
AMI=ami-xxxxxxx # AMI to launch from
SG=sg-xxxxxxxx  # security group ID
KEY=xxx-key     # EC2 key pair name
COUNT=1         # how many instances to launch
IAM=xxxxx       # EC2 IAM role name

# == almost never change; just leave it as-is ==
aws ec2 run-instances --image-id $AMI \
    --security-group-ids $SG --count $COUNT \
    --instance-type $TYPE --key-name $KEY \
    --iam-instance-profile Name=$IAM \
    --instance-market-options '{"MarketType":"Spot"}'
```

By default, the on-demand pricing will be used as the spot price limit. You can further set a custom limit (generally not necessary):

```
--instance-market-options '{"MarketType":"Spot", "SpotOptions": {"SpotPrice": "0.68"}}
↪'
```

The command `aws ec2 request-spot-instances` (doc) can also launch spot instances, but its syntax is slightly more complicated.

Other use cases

You may also describe EC2 instances or stop EC2 instances using AWSCLI. But it is very quick to do so from the console, too.

2.3.3 Use containers to enhance research reproducibility

The reproducibility crisis in science

A large number of research papers cannot be reproduced. Reproducing Earth science model simulations is particularly hard, due to the difficulty of getting adequate computing power, downloading model input data, and configuring software environment. The cloud *helps a lot with all those problems*, but is still not ideal because you are essentially locked-in by AWS (or other cloud vendors). We want to go one step further to make research projects reproducible everywhere.

The container technology can ensure exactly the same software environment everywhere, including your own machines, shared HPC clusters, and different cloud platforms. Thus you can easily combine cloud platforms with your already-invested local computing resources.

What are containers and how can they help

A brute-force way to get the same software environment on different computers is to use a virtual machine (VM) to deliver the entire system. But VMs incur a significant performance penalty since a new operating system needs to run inside the existing system. A container behaves pretty much the same as a VM, while having almost no performance penalty. (Their underlying implementations are vastly different but that's out of scope here.)

Docker is the most widely used container in the software world. The major use case is web applications but it is also used for scientific computing & data science (e.g. Anaconda Docker image) and deep learning (e.g. NVIDIA Docker). NCAR has a Docker-WRF image which might be the easiest way to start WRF.

However, Docker has many limitations for HPC workloads. Using Docker for cross-node MPI programs is possible but very tricky. Further, due to security reasons, there is almost no chance to get Docker installed on shared HPC clusters. Also, domain scientists might find Docker's workflow quite unintuitive because it is mainly designed for web apps, not for numerical computing.

The Singularity container, developed by Lawrence Berkeley National Lab, overcomes the above limitations of Docker. It is already installed on many shared HPC clusters including Harvard's. Because Singularity is designed for scientific use, its workflow feels much more intuitive for domain scientists. Thus we choose Singularity as the container environment to run GEOS-Chem.

For more information about containers, read this Singularity article.

Install Singularity container

Follow the official docs for installation on different platforms. On our tutorial AMI (based on Ubuntu), the full commands are:

```
sudo apt-get update && \
 sudo apt-get install \
   python \
   dh-autoreconf \
   build-essential \
   libarchive-dev

git clone https://github.com/sylabs/singularity.git
cd singularity
git fetch --all
git checkout 2.6.0
./autogen.sh
./configure --prefix=/usr/local
make
sudo make install
```

We deliberately do not include pre-installed Singularity in the tutorial AMI, to let you go through the installation process on your own – you will see that installation is very easy and quick. On your own machine, the installation process is the same, and you will be able to use exactly the same environment on your own machine as on the cloud. On shared HPC clusters, you need to ask the cluster administrator to install Singularity for you if it is not there already.

Do some basic tests like in the official start guide:

```
$ which singularity
/usr/local/bin/singularity
$ singularity pull shub://vsoch/hello-world
Progress |===================================| 100.0%
Done. Container is at: /home/ubuntu/vsoch-hello-world-master-latest.simg
$ ./vsoch-hello-world-master-latest.simg
RaawwWWWWWRRRR!!
```

Run GEOS-Chem inside Singularity container

Pull the GEOS-Chem Singularity image and name it `GEOS-Chem-env.simg` (the default name is quite lengthy):

```
$ singularity pull --name GEOS-Chem-env.simg shub://JiaweiZhuang/Singularity_GC
```

Then just "go into the container" with interactive mode:

```
$ singularity shell GEOS-Chem-env.simg
```

The effect is kind of similar to `module load xxx` on HPC clusters – you load all the libraries needed for GEOS-Chem. Then, just compile and run the model as usual.

If you use *screen/tmux*, remember to start a new session **before** starting the container.

2.3.4 Overview of HPC cluster options on AWS

For heavy-duty work, it might be useful to create HPC-cluster-like environment.

Why do you need an HPC cluster

Basic EC2 instances can already fulfill most of computing needs, as you've learned in the beginner tutorials. An "HPC cluster" just provides those additional functionalities:

- **Jobs scheduler**. On the cloud, the job scheduler is generally not for sharing resources between multiple users, since the entire server belongs to you. Instead, the scheduler is for **requesting new resources automatically**. Whenever you submit a new job by `qsub` or `sbatch`, a new instance is automatically launched (with spot pricing if needed) to run that job, and gets automatically terminated after the job is finished.

- **Shared disk storage**. By default, an EBS data volume can only be attached to one EC2 instance at a time. It is also possible to share an EBS volume between instances using the Network File System (NFS), but doing that by hand is kind of complicated and tedious. Instead, HPC management tools (see the next section) can set up shared disks for you.

- **Cross-node MPI connection**. To run large-scale MPI jobs with hundreds and thousands of cores, you need to connect multiple EC2 instances (just like connecting multiple "nodes" on traditional clusters). This can be done by hand, but using existing HPC tools is much easier.

For most of daily computing workload, it doesn't actually worth the effort to set up clusters (also consider that all HPC management tools have some initial learning curves). Using the basic EC2 is often much easier and controllable. For a single simulation, the "c5.18xlarge" type with 36 CPU cores is already pretty decent. For a few simultaneous runs, just launch separate EC2 instances (pretty quick with AWSCLI) and duplicate input data (pretty quick with S3).

So, when do you actually need an HPC environment?

- You need to use hundreds of cores to run a single MPI job.

- You have tens or hundreds of jobs to submit, which is too inconvenient to do with basic EC2 instances.

- When you decide that AWS should be your major working environment (i.e. "all-in on cloud"). Then it worths the effort to really get familiar with one HPC management tool. (For occasional use, the basic EC2 is more convenient.)

In any case, understand basic AWS services and concepts (finish all the beginner tutorials) before trying HPC clusters.

> **Warning:**   **The major caveat of HPC clusters on cloud:** Those "clusters" are built upon tons of different AWS services to behave like a traditional cluster. The underlying AWS services are abstracted away and not directly exposed to users. However, all abstractions are leaky. If the cluster just works as you expect, then everything is fine. However, if you encounter a bug, the solution would generally require a fix to the underlying AWS services which you might have no idea about. That's another reason why I recommend really getting used to basic AWS services before trying clusters.

HPC cluster management tools

Many cloud-HPC tools have been developed over the past 10+ years, but only a few have relatively large communities. They are introduced below. I personally recommend AlcesFlight, according to the current development status of these tools.

CfnCluster

CfnCluster is an open-source tool developed directly by AWS. The full name "cloud formation cluster" comes from the CloudFormation service, which is for combining different AWS resources together. Understanding CloudFormation is far beyond the IT knowledge of domain scientists. Fortunately, you can just use CfnCluster with terminal commands `cfncluster xxx` and don't have to understand how CloudFormation works under the hood. Its official doc provides a simple tutorial on running a 2-node MPI job.

However, CfnCluster is not an official AWS service, as you cannot find it anywhere in the AWS console. Its documentaion is not as user-friendly as AlcesFlight (see below) and there is a lack of comprehensive instructions for new users.

AlcesFlight

Alces Flight provides free and open-source tools for HPC cluster management on AWS as well as paid services for advanced tasks. The free one ("Community Version") is generally enough for common research workloads. Alces Flight has a pretty decent, comprehensive documentation which keeps in mind the fact that domain scientists have very limited IT knowledge. Their quick start guide teaches you how to create an HPC cluster.

Their cluster environment contains many pre-built software modules, but they are not necessarily up-to-date. Fortunately, they support Singularity container so you can *run the model inside containers*.

StarCluster

StarCluster is one of the earliest cloud-HPC tool. It dominated the majority of cloud-HPC applications including research papers in the early time. Unfortunately, its development has stalled after 2014, so I don't recommend using it now.

2.3.5 GEOS-Chem High-Performance version (GCHP) (experimental)

We've successfully made GCHP running on the cloud. **It is functioning correctly** but there are several issues to be resolved:

- GCHP compiles with gfortran, but the overall performance is ~20% slower than with ifort. The major slow down comes from the new advection module (GFDL-FV3).
- The initial I/O takes a long time (does not affect long-term simulations).

- The data analysis pipeline is not fully documented. We do have some preliminary scripts to process and regrid cubed-sphere data, though.

- It is only set up to run in a single node, with at most 36 CPUs (c5.18xlarge).

Right now it is pretty good for learning and for small experiments. We will make a major update after the formal release of v11-02.

GCHP inside Singularity container

We will be using *containers* to run GCHP. It allows you to set up GCHP quickly on almost **any machines**, not just on Amazon cloud. You can adapt this guide for your own server.

The Singularity image for GCHP can be obtained from Singularity Hub, with the command:

```
$ singularity pull --name GCHP.simg shub://JiaweiZhuang/Singularity_GCHP
```

Launch server

Launch from the AMI ID `ami-21f37a5e`. This AMI is just to provide sample input data and pre-configured run directory. The software libraries will be provided by Singularity container.

The minimum hardware requirement is `r4.2xlarge` with 8 CPUs and 60 GB memory. The minimum number of MPI processes for GCHP is 6 (one for each cubed-sphere panel). You can still start a GCHP simulation on an instance with <6 CPUs, but the program is likely to die somewhere.

Test run

After launching the instance and logging-in (username is `ubuntu`), you should see:

```
$ ls
gcdata  GCHP  GCHP.simg  miniconda  singularity
```

Run the container interactively by:

```
$ singularity shell GCHP.simg
```

If you just execute the container by `./GCHP.simg`, it will print some instructions.

Go to the run directory and execute the pre-compiled executable:

```
$ cd ~/GCHP/gchp_standard
$ mpirun -np 6 ./geos
```

Test compile

To re-compile the model, for now you need to specify the code directory when starting the container:

```
$ SINGULARITYENV_GC_CODE_DIR=~/GCHP/Code.v11-02_gchp singularity shell GCHP.simg
```

Then re-compile the model in the run directory:

```
$ cd ~/GCHP/gchp_standard
$ make compile_clean
```

For more information please see the official tutorial on GCHP wiki.

2.4 Developer guide

This chapter shows how to build models on the cloud from scratch. Ironically, this is perhaps the easiest chapter. Installing software on the cloud is just exactly the same as doing so on local computers, so this chapter contains mostly old Linux stuff with very few cloud-specific concepts.

2.4.1 Install compilers and NetCDF libraries

Start with a brand new system

Please follow AWS official tutorial to launch an EC2 instance with a branch new Linux system. The only difference from our *quick start guide* is in "Step 1: Choose an Amazon Machine Image (AMI)". Instead of starting with our tutorial AMI, here you will start with a basic AMI with almost nothing installed.

AWS promotes their Amazon Linux AMI but there are many other options:

So which to choose? Recall that Linux distributions fall into two big categories:

1. **The Debian family**, such as Debian GNU/Linux and Ubuntu. They use `apt` as the high-level package manager and `dpkg` as the low-level one.

2. **The Red Hat family**, such as Red Hat Enterprise Linux (RHEL) and CentOS. Amazon Linux also belongs to this family. They use `yum` as the high-level package manager and `rpm` as the low-level one.

Ubuntu tends to have the largest user base; CentOS seems to be widely used on HPC clusters; Amazon Linux has the most native AWS support. I find the model performance almost the same on different systems; but this could depend on specific programs.

In this guide, we use with **Ubuntu 16.04 LTS (ami-66506c1c)** as the major example. Commands for **Amazon Linux 2017.09.1 (ami-1853ac65)** are also provided for reference.

To test software installation, the smallest instance "t2.micro" is good enough.

Install C and Fortran compilers

Ubuntu

On Ubuntu, first update package metadata:

```
$ sudo apt-get update
```

We use the GNU compiler family which is free and open source:

```
$ sudo apt-get install gcc gfortran
```

(Alternatively, you can install Intel compilers if you have the license, or PGI compilers for CUDA Fortran and OpenACC support.)

Executables will be installed to `/usr/bin/`:

```
$ which gcc gfortran
/usr/bin/gcc
/usr/bin/gfortran
```

Amazon Linux AMI 2017.09.1 (HVM

Amazon Linux

Free tier eligible

The Amazon Linux AMI is an EBS-backed

The repositories include Docker, PHP, My

Root device type: ebs   Virtualization type: hv

Amazon Linux 2 LTS Candidate A

Amazon Linux

Free tier eligible

Amazon Linux 2 is the next generation of

support, newer versions of glibc, gcc and

Root device type: ebs   Virtualization type: hv

Red Hat Enterprise Linux 7.4 (HVM

Red Hat

Free tier eligible

Red Hat Enterprise Linux version 7.4 (HV

Root device type: ebs   Virtualization type: hv

SUSE Linux Enterprise Server 12

SUSE Linux

Free tier eligible

SUSE Linux Enterprise Server 12 Service

and Scripting, and Legacy modules enab

Root device type: ebs   Virtualization type: hv

By default the package manager gets 5.4.0:

```
$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609
...
$ gfortran --version
GNU Fortran (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609
...
```

You can also install higher versions from the Toolchain test builds, for example:

```
$ sudo add-apt-repository ppa:ubuntu-toolchain-r/test
Press [ENTER]
$ sudo apt-get update
$ sudo apt-get install gfortran-7
```

Amazon Linux

For Amazon Linux, the equivalent command is:

```
$ sudo yum install gcc gcc-gfortran
```

It gets 4.8.5 by default:

```
$ gcc --version
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-11)
$ gfortran --version
GNU Fortran (GCC) 4.8.5 20150623 (Red Hat 4.8.5-11)
```

Install NetCDF library with package manager

The NetCDF library is ubiquitously used in Earth science models. Getting it from the package manager is extremely easy and quick.

Ubuntu

On Ubuntu, simply:

```
$ sudo apt-get install libnetcdf-dev libnetcdff-dev
```

Note that "dev" stands for "development tool" since you are going to use it to compile models. (it is not "developing version" – the package repository is quite mature and stable!) Also note that after version 4.2, the NetCDF-C and NetCDF-Fortran libraries are installed separately.

Check NetCDF-C configuration:

```
$ nc-config --all

This netCDF 4.4.0 has been built with the following features:
...
```

Check NetCDF-Fortran configuration:

```
$ nf-config --all

This netCDF-Fortran 4.4.3 has been built with the following features:
...
--prefix    -> /usr
--includedir-> /usr/include
--version   -> netCDF-Fortran 4.4.3
```

`--includedir` will be used to include this NetCDF library when compiling Fortran code.

Amazon Linux

For Amazon Linux, NetCDF libraries are not in the default repository. You need to first enable the Extra Packages for Enterprise Linux (EPEL):

```
$ sudo yum-config-manager --enable epel
```

(See AWS guide for enabling EPEL on CentOS and Ret Hat).

Then get NetCDF from the EPEL repo:

```
$ sudo yum install netcdf-devel
```

This currently gives you 4.1.1 with C and Fortran bundled together:

```
$ nc-config --all

This netCDF 4.1.1 has been built with the following features:
...
```

(CentOS's package registry has a newer version of NetCDF which separates Fortran and C libraries, so you would need `sudo yum install -y netcdf-devel netcdf-fortran-devel`)

However, this NetCDF distribution seems to lack `/usr/include/netcdf.mod` for `use netcdf` statement in Fortran 90 code. The `include 'netcdf.inc'` statement would work fine.

CentOS will put one at `/usr/lib64/gfortran/modules/netcdf.mod`. The include statement needs to be changed to `-I/usr/lib64/gfortran/modules/`.

Test sample NetCDF code

Get some sample code, such as simple_xy_wr.f90.

```
$ wget https://www.unidata.ucar.edu/software/netcdf/examples/programs/simple_xy_wr.f90
$ gfortran simple_xy_wr.f90 -o test_nc.exe -I/usr/include -lnetcdff
$ ./test_nc.exe
*** SUCCESS writing example file simple_xy.nc!
```

Install `ncdump` to check data content:

```
$ sudo apt install netcdf-bin
$ ncdump -h simple_xy.nc
netcdf simple_xy {
dimensions:
    x = 6 ;
```

(continues on next page)

```
      y = 12 ;
variables:
      int data(x, y) ;
}
```

(Optional) Install NetCDF from source code

You might want to build NetCDF from source if:

1. To install into a different directory. Package managers can only install libraries into `/usr`.

2. To ensure the latest version. Package managers are not necessarily up-to-date.

Doing so is quite tedious so we will not cover it here. Please refer to the official guide.

For NetCDF library, you generally won't get better performance by compiling it from source with better optimized compiler settings. That's because NetCDF is just an I/O library, not for numerical computation. However, for other compute-oriented libraries, compiling from source can sometimes make a big difference in performance.

2.4.2 Save your system and share with others

After you've installed compilers and libraries, you would like to save those changes. Libraries are system-dependent so cannot be simply uploaded to S3. You need to save the entire system instead. (The system image will *physically live on S3* anyway, but the backup process is handled by AWS to ensure system integrity)

Right-click on your EC2 instance, and select "Image" - "Create Image":

> **Warning:** If you are going to make an AMI for public use, make sure you did not leave any private information on the EC2 instance, such as your AWS security credentials or your GitHub credentials. Also clean your command history by `shred -u ~/.*history`. Please refer to AWS official doc for the full guidelines.

Give your AMI a meaningful name, edit "Image description" as you like. and then simply click on "Create Image". Then, your EC2 instance will reboot and you will lose connection for several seconds.

AMI creation can take a while. For this small one it will take several minutes; bigger ones might take an hour. You will final see the new AMI in EC2 console:

It is private by default. You can change the permission to public if you wish. The change might take several minutes to take effect. Note that the AMI entry's first column, "Name", is just a convenient label ("Name Tag") to help you remember what this resource is for. It is only visible to you and can be modified at any time. Other users can only see the second column, "AMI Name", which cannot be changed after AMI creation.
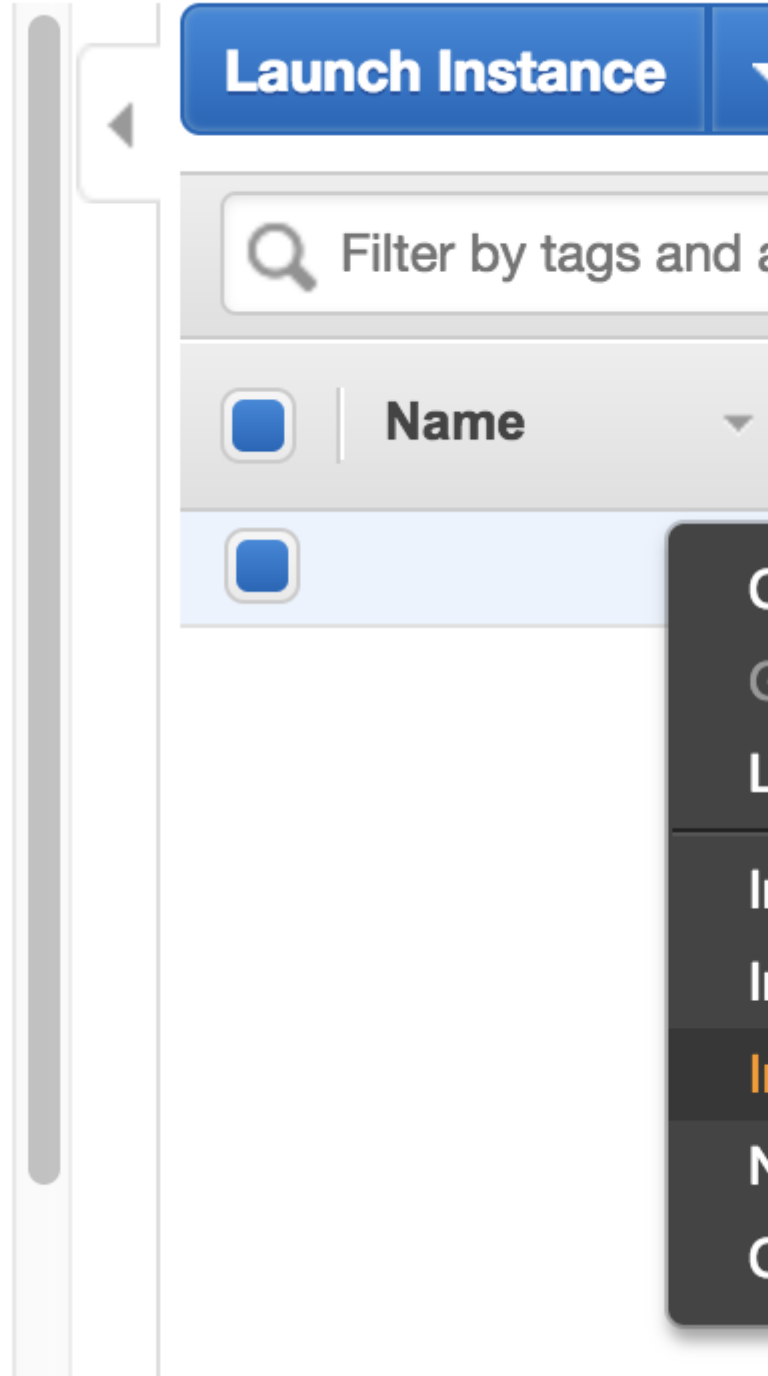
An *EBS snapshot* is also created automatically to host the actual data for that AMI. It looks pretty indistinguishable from EBS snapshots created directly from EBS volumes (except that the "Description" column shows "Created by CreateImage. . . "). Edit its name tag as you like to make it look less confusing.

That's all you need to know about AMIs. AMIs are convenient for infrequent build. If you need to update software constantly, consider containers instead.

2.4.3 Set up GEOS-Chem environment

GEOS-Chem-classic only uses OpenMP parallelization so there is no need for MPI libraries. Having Fortran compiler and NetCDF library installed is sufficient for GC-classic to compile and run.

EC2 Dashboard

Events

Tags

Reports

Limits

INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

Scheduled Instances

Launch Instance

Filter by tags and a

Name

Create Image

Instance ID ⓘ	i-04584db00b0da
Image name ⓘ	my-test-image
Image description ⓘ	
No reboot ⓘ	☐

Instance Volumes

Volume Type ⓘ	Device ⓘ	Snapshot ⓘ	Size (GiB) ⓘ
Root	/dev/sda1	snap-03c91645beefa0b0d	8

Add New Volume

Total size of EBS Volumes: 8 GiB

When you create an EBS image, an EBS snapshot will
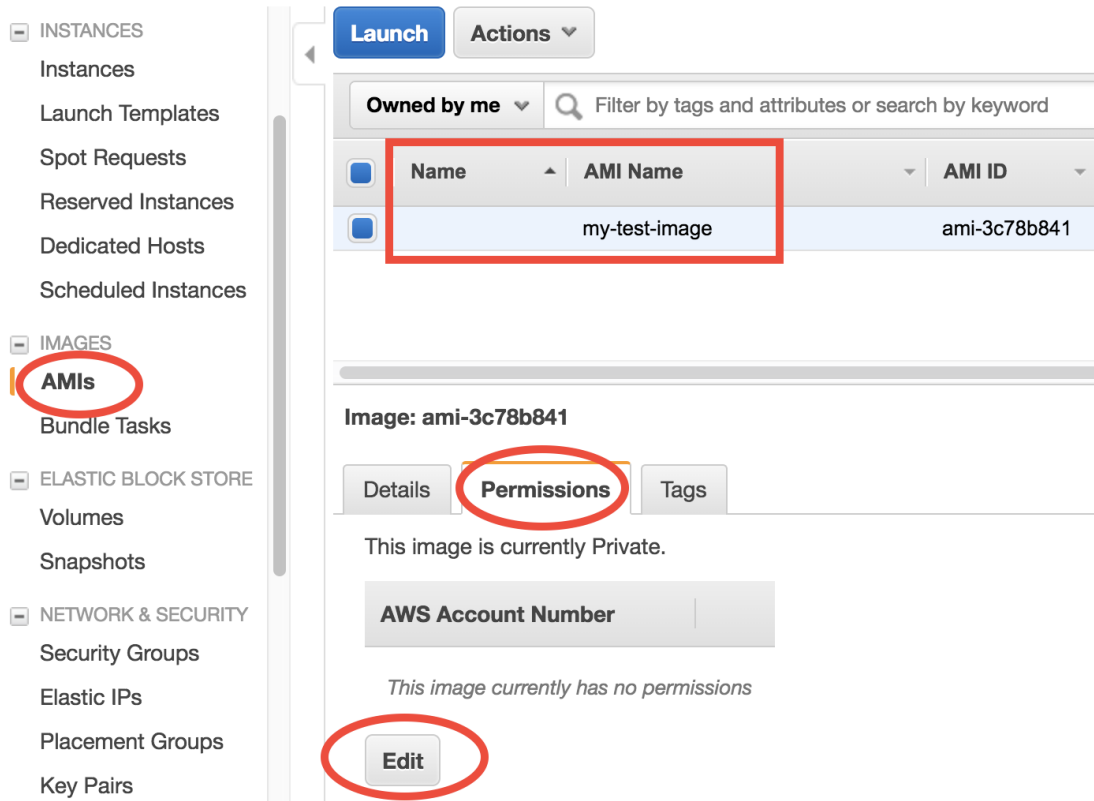
Environment variables

Just follow our wiki. The bashrc used for the tutorial AMI is available for reference.

Source code, run directory and input data

Again, just follow the wiki.

Since all data are now *available on S3*, no need to pull them from other places.

2.4.4 Install scientific Python environment

Just use Anaconda. It doesn't require root access and can be easily installed into almost any environment (including shared HPC clusters).

Please refer to our GCPy page for detailed steps to set up geoscientific Python Environment.

Scripts used for the tutorial AMI are available for reference.

2.4.5 Install MPI libraries

GCHP would need MPI library. Most of other models would need MPI, too.

Enabling in-node MPI parallelization is pretty trivial since no network configurations are required. For cross-node MPI run, using the pre-built MPI library provided by *HPC cluster management tools* would be much easier.

INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

Scheduled Instances

IMAGES

AMIs

Bundle Tasks

ELASTIC BLOCK STORE

Volumes

Snapshots

Create Snapshot

Owned By Me

Name

Back up my test

More details coming soon... The Dockerfile for WRF is a good reference for installing MPI libraries using the package manager. They use CentOS as the base OS.

2.4.6 Build your own containers

Docker is ubiquitous in the software world but Singularity feels much more intuitive for people from the scientific computing world. To engage scientists more easily, consider Singularity. Its documentation is pretty easy to follow. The recipe for the container used in the tutorial is available for reference.

Note that Singularity interoperates pretty well with Docker. Because Docker has a lot more base images, Singularity images are often started with Docker images (shown by `BootStrap: docker` on the top of the recipe).

2.5 AWS concepts and services in detail

This chapter dives deeper into AWS services.

2.5.1 Common AWS concepts (but often confuse non-IT people)

While this entire documentation is about using AWS for **scientific computing**, one should always keep in mind that AWS is mainly designed for **IT/web applications**. Although I've been trying to avoid any IT concepts throughout the tutorials, all kinds of IT jargons will pop up here and there, when you are navigating through the AWS console or looking at their documentations.

Here I try to give a gentle introduction to the most basic IT stuff, to make you feel more comfortable with AWS jargons.

Regions and Availability Zones

Regions and Avail Zones describe the geographical distribution of AWS services and data centers (i.e. actual buildings). **Region** is a top-level concept that refers to a pretty big area such as us-east-1 (N. Virginia) and us-west-2 (Oregon). All AWS resources (EC2, S3...) in one region are fairly isolated from other regions. The EC2 console only shows one single region (selectable from the upper-right corner). The S3 console displays your buckets in all regions on the same page, but each S3 bucket can live in a different region, as shown by the "Region" column. Transferring data within a region is always free (no matter EC2 to S3, S3 to EC2, or EC2 to EC2); transferring data across regions costs $10~20/TB. Note that cross-region data transfer is significantly slower than in-region transfer.

Inside a region, there are ~5 **Avail Zones** (called us-east-1a, us-east-1b...). An Avail Zone corresponds to a **physical** place where AWS data centers are actually located. Compared to "Region", an Avail Zone is a much smaller area, maybe like a town. Having machines distributed over multiple places ("towns") makes the entire system much more resilient. If a town suffers from power outage, only one Avail Zone will die but others will work as usual. EC2 instances and EBS volumes only run in a specific Avail Zone. S3 buckets have no concept of Avail Zone and operate over the entire region.

The concept of Avail Zone is extremely useful for deploying web servers, but much less useful for numerical computing workloads. To make a website highly robust (especially for business applications), the underlying servers can be deployed across multiple Avail Zones, hosting exactly the same content for the website. Even if one Avail Zone dies (a severe accident that rarely happens), the website is still alive, so your business will not be affected. But if you use EC2 only for number crunching, this functionality would make little sense. Thus you only need to know:

1. An EBS volume can only be *attached to an EC2 instance in the same Avail Zone* .

2. Different Avail Zones *have different spot prices*, and you will get the cheapest on by default.

Virtual Private Clouds (VPCs) and Subnets

VPCs and Subnets define the **private IP range** (detailed later) of your EC2 instances. In a newly created AWS account, there will be a default **VPC** within each **Region**, and a default **Subnet** within each **Avail Zone**. You can see them in the VPC console, or during EC2 launching:

So what is a private IP? For any running EC2 instance, you will see a bunch of IPs and addresses:

The Public DNS, `ec2-34-230-38-159.compute-1.amazonaws.com` in this case, is what you've been using in the `ssh` command. Equivalently, you can also use the **Public IP** `34.230.38.159` in the `ssh` command. Then it comes to the **private IP** `172.31.76.121`. By default, the private IPs of EC2 instances are always `172-31-xxx-xxx`. This IP is only for internal connection (say, connect with other EC2 instances), thus you cannot use it in the `ssh` command.

Why are there both public and private IPs? You should already know that **a public IP address must be unique across the entire Internet**. Computers talk to each other through the Internet using IP addresses. However, we don't always want to use the **Internet** (i.e. the biggest, public network) to connect computers because:

1. We often need secret connections between computers, without the use of any public IP addresses. One example is *managing HPC clusters on the cloud*. An HPC cluster typically has a "login node" (or "home node"), and multiple "compute nodes". You won't want to give the compute nodes public IP addresses, because this will increase security risks. A public IP means it is possible to directly `ssh` to compute nodes from the external world. Instead, compute nodes should only be accessible from the home node, through **private IP connection**. Only the home node should get a public IP.

2. There aren't that many unique IP addresses available for public use. IPv4 only allows 2^32 addresses in total. That's why AWS is moving towards IPv6 to get more digits.

Thus, we often use smaller, private networks (**i.e. VPCs!**) to connect computers. IP address like `172-31-xxx-xxx` or `192-168-xxx-xxx` are designed to work in private networks. I bet you've seen those numbers many times in your daily life, when working with any digital products. Those addresses don't need to be unique across the Internet so can be reused on every digital device. A **subnet** refers to a subset of (private) IP addresses, such as `172-31-46-xx`

So what do you need to do with VPCs? For simple computing workloads, you don't need to change anything. Just keep the default VPC and subnets during EC2 launching. But you will see those concepts constantly when using HPC clusters on the cloud. As an HPC cluster **user**, you mostly don't need to tweak VPCs manually, but it's good to understand what's going on.

IAM users and IAM roles

They control who are allowed to use your AWS resources. See *Enable S3 access from EC2 by IAM role* for details.

Security groups

They control who are allowed to `ssh` your EC2 instances. See *Notes on security groups (EC2 firewall)* for details.

2.5.2 Monitoring AWS cost

Time to talk about money. The *Researcher's Handbook* has a very comprehensive introduction to all kinds of budget management tools. I find that the AWS Cost Explorer is the most straightforward and intuitive one.

Go to your billing console:

Select "Cost Explorer". For the first time, you need to enable it and wait for 24 hours.

AWS provides several pre-configured cost reports in "Saved Reports". Let's choose "Monthly costs by service":

1. Choose AMI 2. Choose Instance Type 3. Configure

Step 3: Configure Instance Detai

Configure the instance to suit your requirements. You ca

more.

Number of instances	(i)	1
Purchasing option	(i)	☐ R
Network	(i)	vpc
Subnet	(i)	✓ No
		sub
Auto-assign Public IP	(i)	sub
		sub
IAM role	(i)	sub
		sub
		sub

Dashboard

Bills

Cost Explorer

Budgets

Reports

Cost Allocation Tags

Payment Methods

Welcome to Cost Explorer

Cost Explorer provides reporting, analytics and visualization capabilities that you ca

You will be able to see your spend data within 24 hours of enabling Cost Explorer.

⚠ **Spend data for your account is being populated**

It can take up to 24 hours to populate spend data for your AWS account.

Cost Explorer › **Saved Reports**

⊕New report   📋 Create Copy   **Delete**

		Report name		Chart style	
☐	›	AWS Marketplace	🔒	�£᥇	
☐	›	Daily costs	🔒	ᥣ᥇	
☐	›	Monthly costs by linked account	🔒	ᥣ᥇	
☐	›	Monthly costs by service	🔒	ᥣ᥇	
☐	›	Monthly EC2 running hours costs ...	🔒	ᥣ᥇	
☐	›	RI Coverage	🔒	ᥥᥥ	
☐	›	RI Utilization	🔒	ᥥᥥ	

The default, monthly frequency won't give you too much information if you just start to use AWS. Let's change it to "Daily" over the "Last 7 Days":

You can see the day-to-day cost very clearly, broken into EC2, S3, and other charges. "EC2-Other" generally refers to EBS volumes. You can also click on "Save as" to create a new "Daily costs by service" report for future use.

Note: For some reason, you might be directed to the Simple Monthly Calculator, say if you just google "estimate aws cost". This tool is not going to help you because it is for estimating the cost of web servers. Web servers run stably for a long time, so their costs are very easy to pre-calculate. On the other hand, scientific computing workloads are intermittent and hard to predict. So, the most helpful thing you can do is looking at the history.

2.6 Appendix

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

2.6.1 List of public AWS resources for GEOS-Chem

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

Latest resources:

Resource	ID/name	Size	Content
Tutorial AMI	ami-08c83a8b3ebd20b63	100 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
GCHP experimental AMI	ami-21f37a5e	100 GB	1. Pre-configured GCHP rundir 2. Sample GCHP input data 3. GCHP container environment
S3 bucket for all GC data	s3://gcgrid (requester-pay)	~30 TB	All current GEOS-Chem input data

Old resources (for record):

Resource	ID/name	Size	Content
Older tutorial AMI	ami-ab925cd6	70 GB	Same as tutorial AMI but uses v11-02d

2.6.2 Simplify SSH login with config file

In the *quick start guide* you used a lengthy command `ssh -i "xx.pem" ubuntu@xxx.com` to login. To copy files by `scp`, the command would be even longer:

```
scp -i "xx.pem" local_file ubuntu@xxx.com:~/
```

Fortunately, those commands can be simplified to `ssh ec2` and `scp local_file ec2:~/`, skipping the Key Pair argument `-i "xx.pem"` and even the EC2 address.

Set up SSH config file

For example, say the full ssh command is:

```
ssh -i "~/.ssh/my-aws-key.pem" ubuntu@ec2-35-169-93-188.compute-1.amazonaws.com
```

Just open the file ~/.ssh/config (create it if doesn't exist), and enter the following content. Then you will be able to use the shortcuts.

```
Host ec2
  Hostname ec2-35-169-93-188.compute-1.amazonaws.com
  user ubuntu
  IdentityFile ~/.ssh/my-aws-key.pem
  Port 22
```

"Host" is the shortcut you want to use in ssh and scp commands. "user", "IdentityFile" (EC2 Key Pair), and "Port" (always 22 for ssh) are almost never changed for different EC2 instances. So you only need to change "Hostname" every time you have a new instance.

Enable port forwarding

Port forwarding for Jupyter can be done as usual:

```
ssh ec2 -L 8999:localhost:8999
```

Or can be also included in ~/.ssh/config so you simply need ssh ec2:

```
Host ec2
  Hostname ec2-35-169-93-188.compute-1.amazonaws.com
  user ubuntu
  IdentityFile ~/.ssh/my-aws-key.pem
  Port 22
  LocalForward 8999 localhost:8999
```

For multiple servers

You can add multiple entries for multiple EC2 instances. Say you've launched a bigger instance in addition to the existing one:

```
Host ec2
  Hostname ec2-35-169-93-188.compute-1.amazonaws.com
  user ubuntu
  IdentityFile ~/.ssh/my-aws-key.pem
  Port 22
Host ec2-big
  Hostname xxxxx.com
  user ubuntu
  IdentityFile ~/.ssh/my-aws-key.pem
  Port 22
```

Additional notes

1. Note that if you stop and then re-start an EC2 instance, its address will change. The address can be fixed by AWS Elastic IP, but I find that modifying "Hostname" in ~/.ssh/config is generally quicker.

---

2. Our tutorial AMI is based on Ubuntu, thus the user name. Other operating systems would have different user names.

3. There are similar tutorials for SSH Config online.

2.6.3 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 allows 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 `Ctrl + 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 `Ctrl + 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.

2.6.4 Sample Python code to analyze GEOS-Chem data

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

np.seterr(invalid='ignore'); # disable a warning from matplotlib + cartopy
```

GEOS-Chem NetCDF diagnostics

```
[2]: ls /home/ubuntu/tutorial/geosfp_4x5_standard/OutputDir/
```

```
GEOSChem.SpeciesConc.20160701_0000z.nc4
```

```
[3]: ds = xr.open_dataset("/home/ubuntu/tutorial/geosfp_4x5_standard/OutputDir/"
                          "GEOSChem.SpeciesConc.20160701_0000z.nc4")

     ds
```

```
[3]: <xarray.Dataset>
     Dimensions:         (ilev: 73, lat: 46, lev: 72, lon: 72, time: 1)
     Coordinates:
       * time            (time) datetime64[ns] 2016-07-01T00:10:00
       * 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: Species...
         history:
         format:                         CFIO
         conventions:                    COARDS
         ProdDateTime:
         reference:                      www.geos-chem.org; wiki.geos-chem.org
         contact:                        GEOS-Chem Support Team (geos-chem-suppor...
         simulation_start_date_and_time: 2016-07-01 00:00:00z
         simulation_end_date_and_time:   2016-07-01 00:20:00z
```

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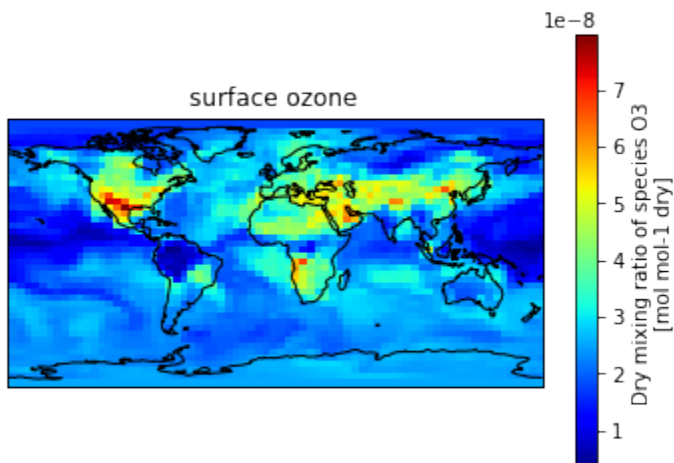
GEOS-FP metfield

```
[5]: ds_met = xr.open_dataset("~/ExtData/GEOS_4x5/GEOS_FP/2016/07/GEOSFP.20160701.I3.4x5.nc
     ↪")
     ds_met
```

```
[5]: <xarray.Dataset>
     Dimensions: (lat: 46, lev: 72, lon: 72, time: 8)
     Coordinates:
       * time     (time) datetime64[ns] 2016-07-01 2016-07-01T03:00:00 ...
       * 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.20160701.I3.4x5.nc
         History:                File generated on: 2016/08/04 09:56:57 GMT-0300
         ProductionDateTime:     File generated on: 2016/08/04 09:56:57 GMT-0300
         ModificationDateTime:   File generated on: 2016/08/04 09:56:57 GMT-0300
         Format:                 NetCDF-4
         SpatialCoverage:        global
         Conventions:            COARDS
         Version:                GEOS-FP
         Model:                  GEOS-5
         Nlayers:                72
         Start_Date:             20160701
         Start_Time:             00:00:00.0
         End_Date:               20160701
         End_Time:               23:59:59.99999
         Delta_Time:             030000
         Delta_Lon:              5
         Delta_Lat:              4
```

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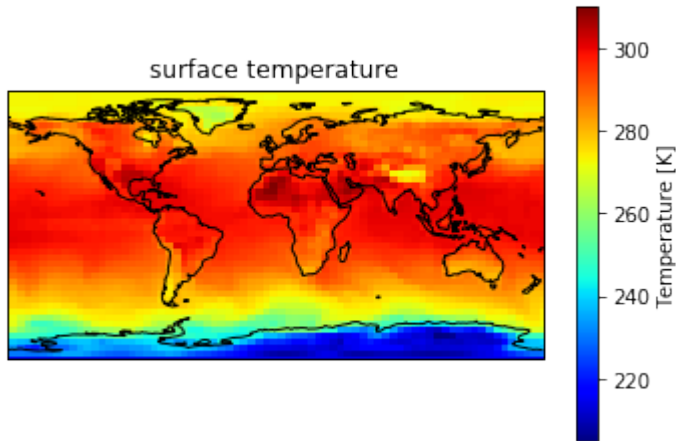

2.6.5 Sample Python code to analyze NASA-NEX data

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

```
[2]: # Data already downloaded by
     # aws s3 cp s3://nasanex/NEX-GDDP/BCSD/rcp85/day/atmos/tasmax/r1i1p1/v1.0/tasmax_day_
     ↪BCSD_rcp85_r1i1p1_inmcm4_2100.nc ./
     ds = xr.open_dataset("./tasmax_day_BCSD_rcp85_r1i1p1_inmcm4_2100.nc")
     ds
```

```
[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
```

(continues on next page)

```
        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
```

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

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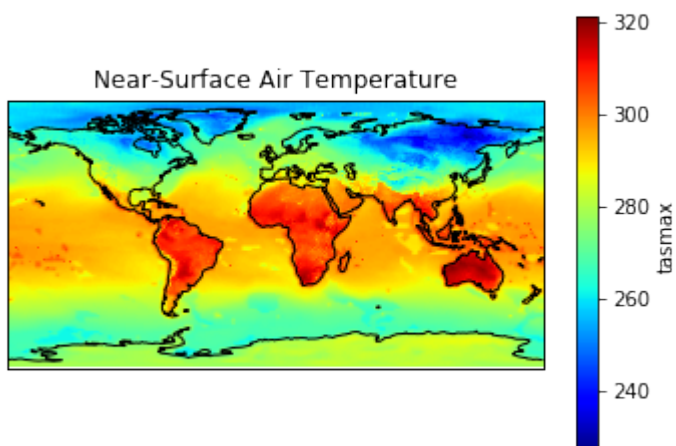
```
[4]: ax = plt.axes(projection=ccrs.PlateCarree())
     ds['tasmax'][0].plot(cmap='jet')
     ax.coastlines()
     plt.title('Near-Surface Air Temperature');
```

```
[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');
```