

# Cloud Computing for Science

## Part 3. Scaling Computation

Dennis Gannon

# The Cloud Data Center Evolution

- Early days: 2005
  - Very simple servers
  - Network outward facing poor interconnect
- 2008-2016
  - Software defined networks
  - Special InfiniBand sub networks
  - Many different server types
    - 2 cores to 32 cores to GPU accelerations
  - Efficiency experiments
    - Geothermal, wind, wave
    - Containerized server
- 2017
  - Azure FPGA accelerated mesh
  - Google Tensor Processing Unit
  - Facebook – Open Compute Project
  - ARM based servers

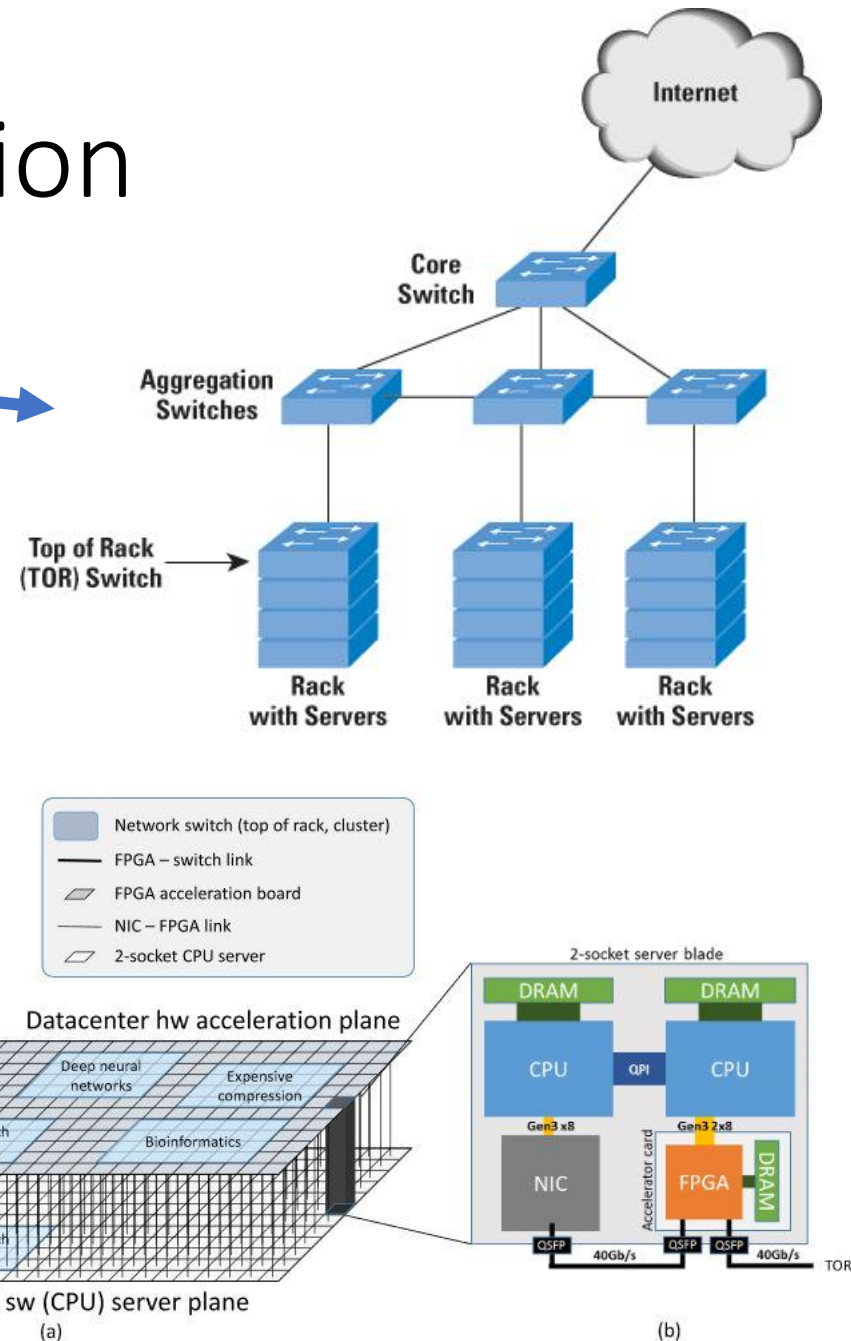


Fig. 1. (a) Decoupled Programmable Hardware Plane, (b) Server + FPGA schematic.

# Azure and AWS Global Data Center Network



# How to scale: Models of Parallelism

- Classic HPC
  - SPMD MPI programming
- Task Parallel
  - Also called “embarrassingly parallel”
- MapReduce
  - Hadoop style
- Graph Execution
  - Spark and streaming systems
- Microservices
  - Similar to actor model

# Classic HPC

- AWS CloudFormation Cluster
  - Fill out CfnCluster template
  - Use aws command line to submit
  - Log into head node
- Azure create a slurm cluster
  - See Azure slurm tutorial

## Deploy a slurm cluster

 Deploy to Azure

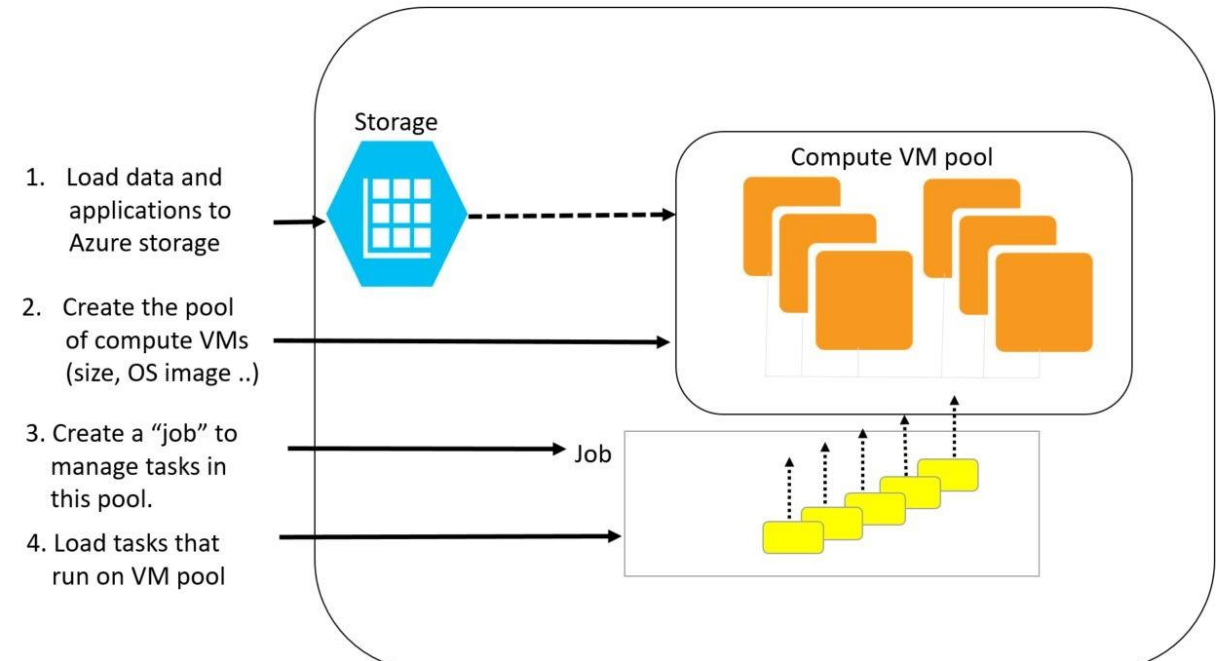
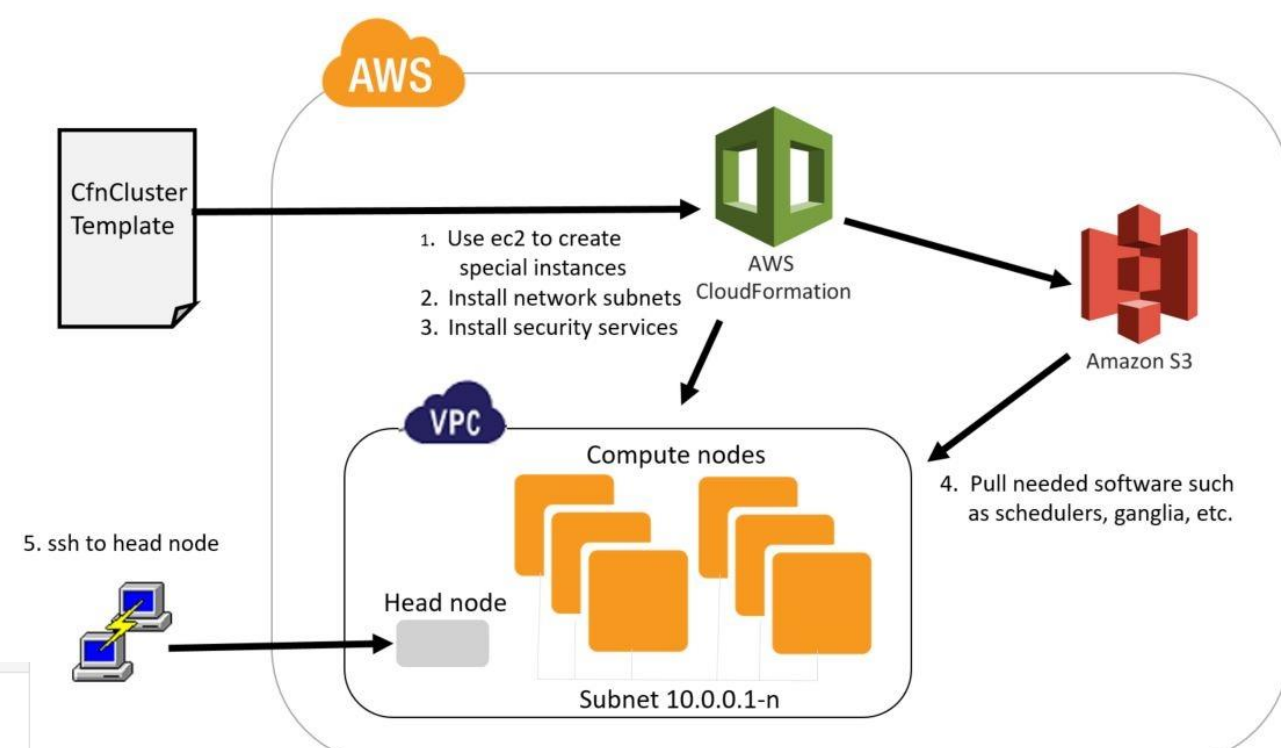
 Visualize

1. Fill in the 3 mandatory parameters - public DNS name, a storage account to hold VM image, and admin user password.
2. Fill in other info and click "OK".

## Using the cluster

Simply SSH to the master node and do a `slurm`! The DNS name is `dnsName.location.cloudapp.azure.com`, for example, `yidingslurm.westus.cloudapp.azure.com`.

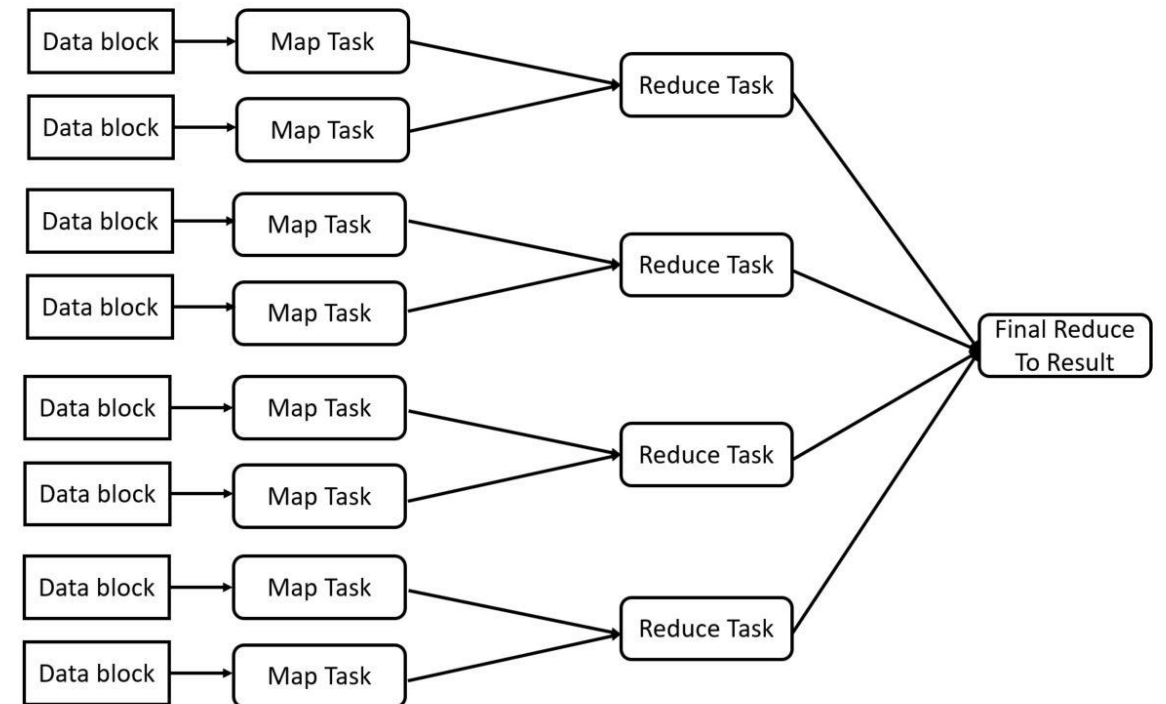
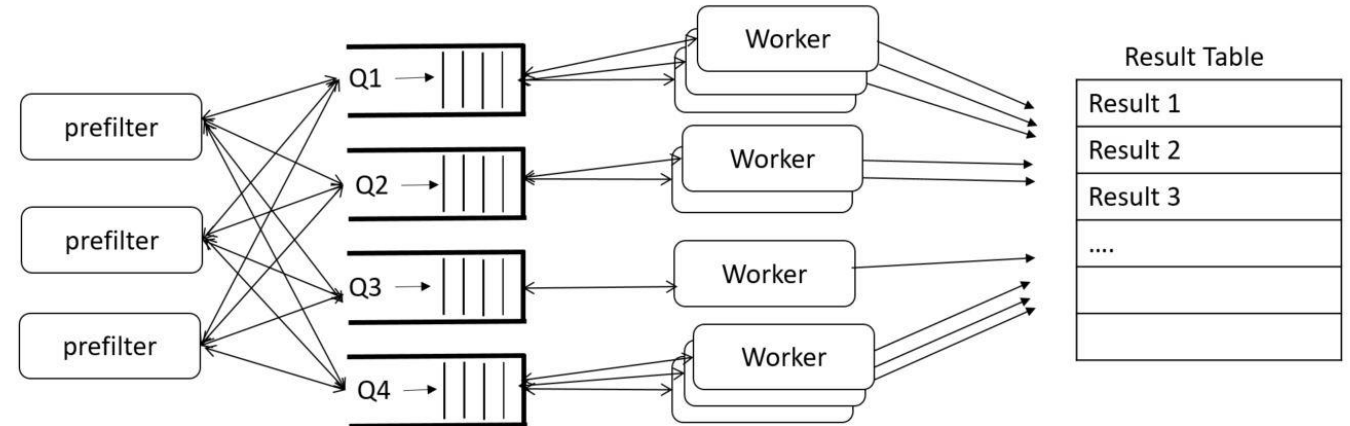
- Or use Azure Batch
  - Similar to AWS batch





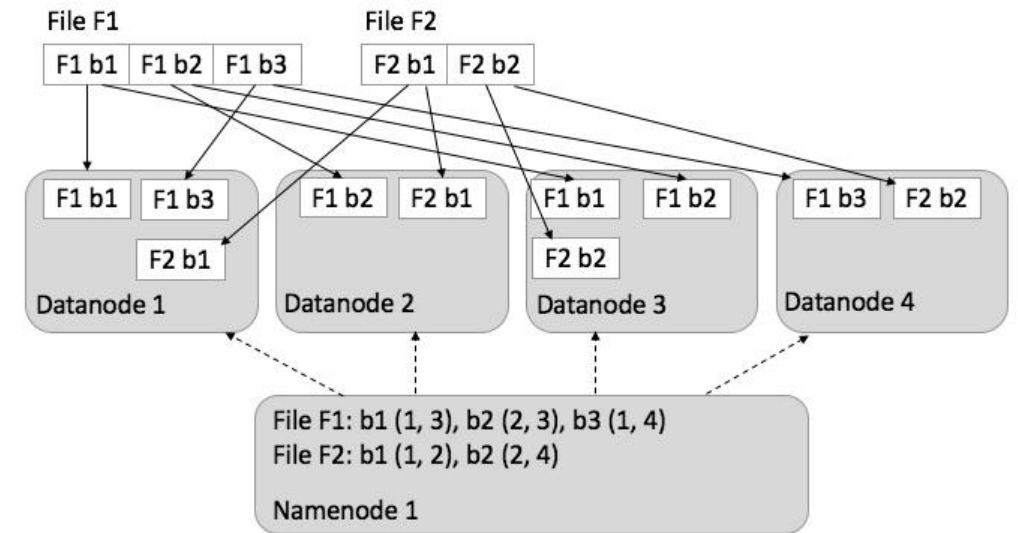
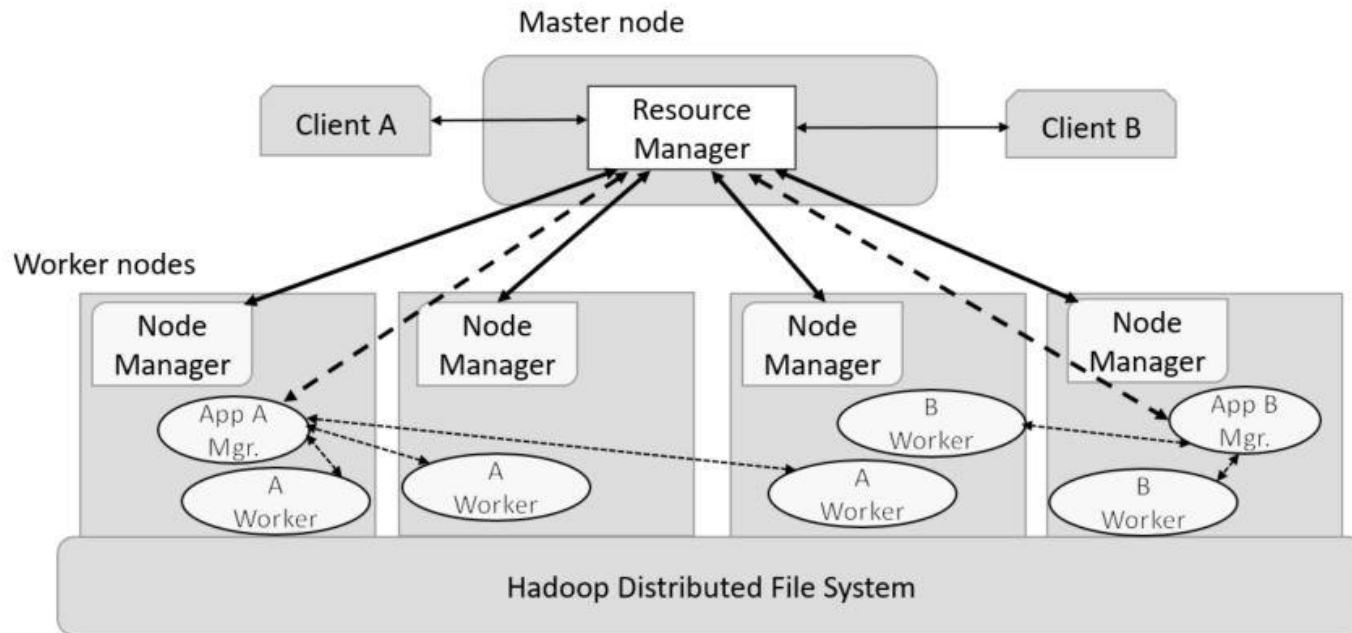
# Task Parallel and Map Reduce

- Task parallel model is great for solving problems that involve doing many independent computations.
- Map Reduce
  - Bulk Synchronous Parallel (BSP)
  - Map Task = an operation applied to blocks of data in parallel
  - Reduce Task- when maps are “done” reduce the results to a single result
- Examples of both later



# The Hadoop- Yarn ecosystem

- Yarn is the name of a project containing many elements
  - The runtime system is distributed
  - Hadoop, Spark run in distributed mode
  - Multiple clients can access the resource manager
  - Jupyter and Zeppelin are interactive clients
- HDFS is the Hadoop File system
  - Distributed over data node servers
  - Files are blocked, distributed and replicated
  - Files are write-once.



# Azure HDInsight is a Yarm Environment

Essentials ^

Resource group [\(change\)](#)  
[bookRG](#)

Status  
Running

Location  
East US

Subscription name [\(change\)](#)  
[azure4research](#)

Subscription ID  
f518fe6b-5262-4e5a-80cb-05b7a39f9298


Cluster type, HDInsight version  
Spark 2.0 on Linux (HDI 3.5.1000.0)


URL  
<https://bookcluster3.azurehdinsight.net>


Learn more  
[Documentation](#)  
[Getting Started](#)  
[Quickstart](#)

Head Nodes, Worker Nodes  
D3 (x2), D3 (x2)


Quick Links


 Cluster Dashboards

 Ambari Views

 Scale Cluster

Usage

  
HDInsight Cluster Dashboard

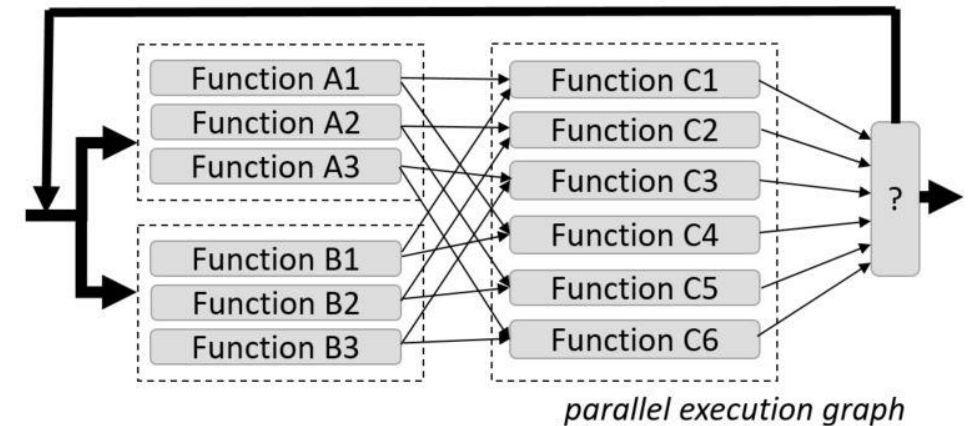
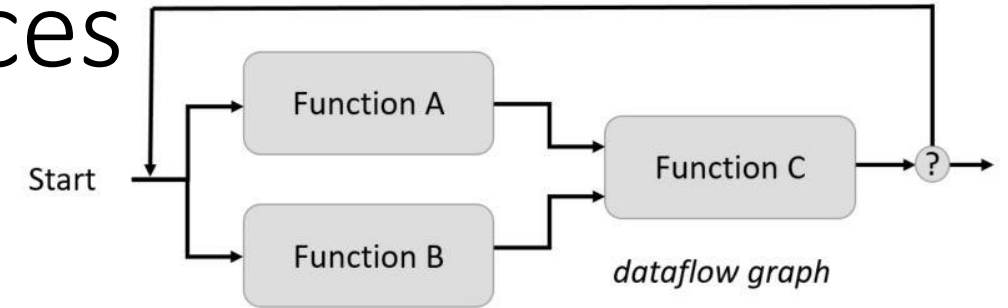
  
Jupyter Notebook



# Graph Parallel and Microservices

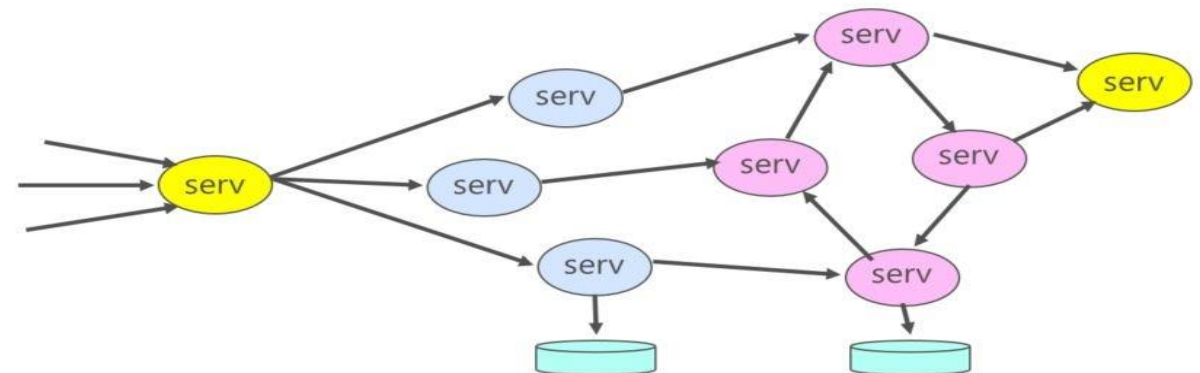
- Graph Parallel

- The data is in distributed arrays or streams.
- build a data flow graph of the algorithms functions.
- The graph is compiled into parallel operators that are applied to the distributed data structures.



- Microservices

- Divide a computation into small, mostly stateless components that can be
  - Easily replicated for scale
  - Communicate with simple protocols
- Computation is as a swarm of communicating workers.



# Graph computation example: Spark

- A simple map reduce: Compute
- For  $n = 10,000,000$
- In Spark on Python is:

$$\lim_{n \rightarrow \infty} \sum_{i=1}^n \frac{1}{i^2} = \frac{\pi^2}{6}$$

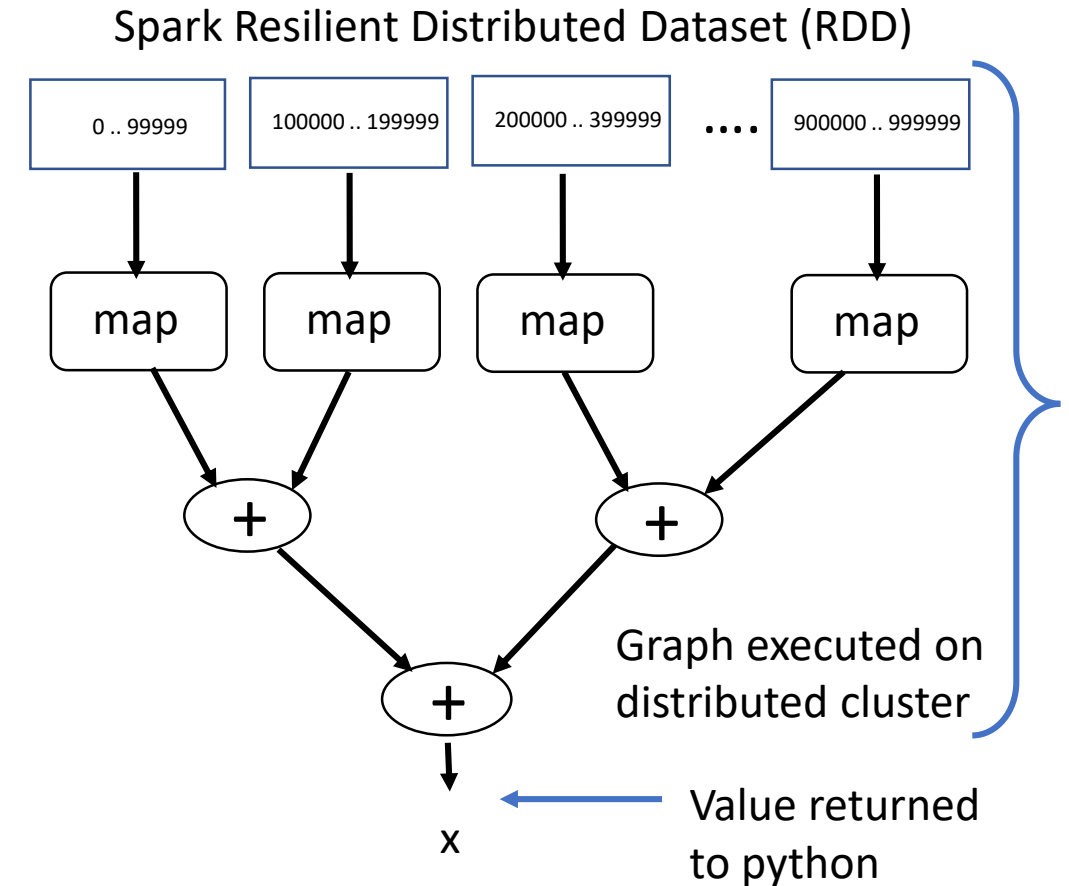
```
import numpy as np
ar = np.arange(n) # an array from 0 to 9999999
nump = 100

rdd = sc.parallelize(ar, nump)

x = rdd.map(lambda i: 1.0/(i+1)**2)
      .reduce(lambda a,b: a+b)

print("x=%f"%x)
print("pi**2/6=%f"%(np.pi**2/6))

1.644934
1.644934
```



# More interesting example: k-means clustering

- The algorithm basics
  - $n=1000000$
  - Start with a vector P of  $n$  2-d points and a vector kPoints of  $k$  random cluster centroids.
  - Iterate until kPoints don't move:
    - For each  $j$  in  $[0, k-1]$  pick  $q[j]$  from kPoints. Then find all the points  $p$  in P near  $q[j]$  and create the tuples  $(j, (p, 1))$  for  $p$  nearest to  $q[j]$
    - For each  $j$  compute the centroid of all points “near”  $q[j]$  in kPoints”  $(j, (\text{sum}(p)/\text{sum}(1)))$
    - Set  $q[j]$  to be the new centroid  $\text{sum}(p)/\text{sum}(1)$

```
tempDist = 1.0
while tempDist > convergeDist:
    newPoints = data \
        .map( lambda p: (closestPoint(p, kPoints), (p, 1))) \
        .reduceByKey(lambda x, y : (x[0] + y[0], x[1] + y[1])) \
        .map(lambda x : (x[0], x[1][0]/ x[1][1])) \
        .collect()

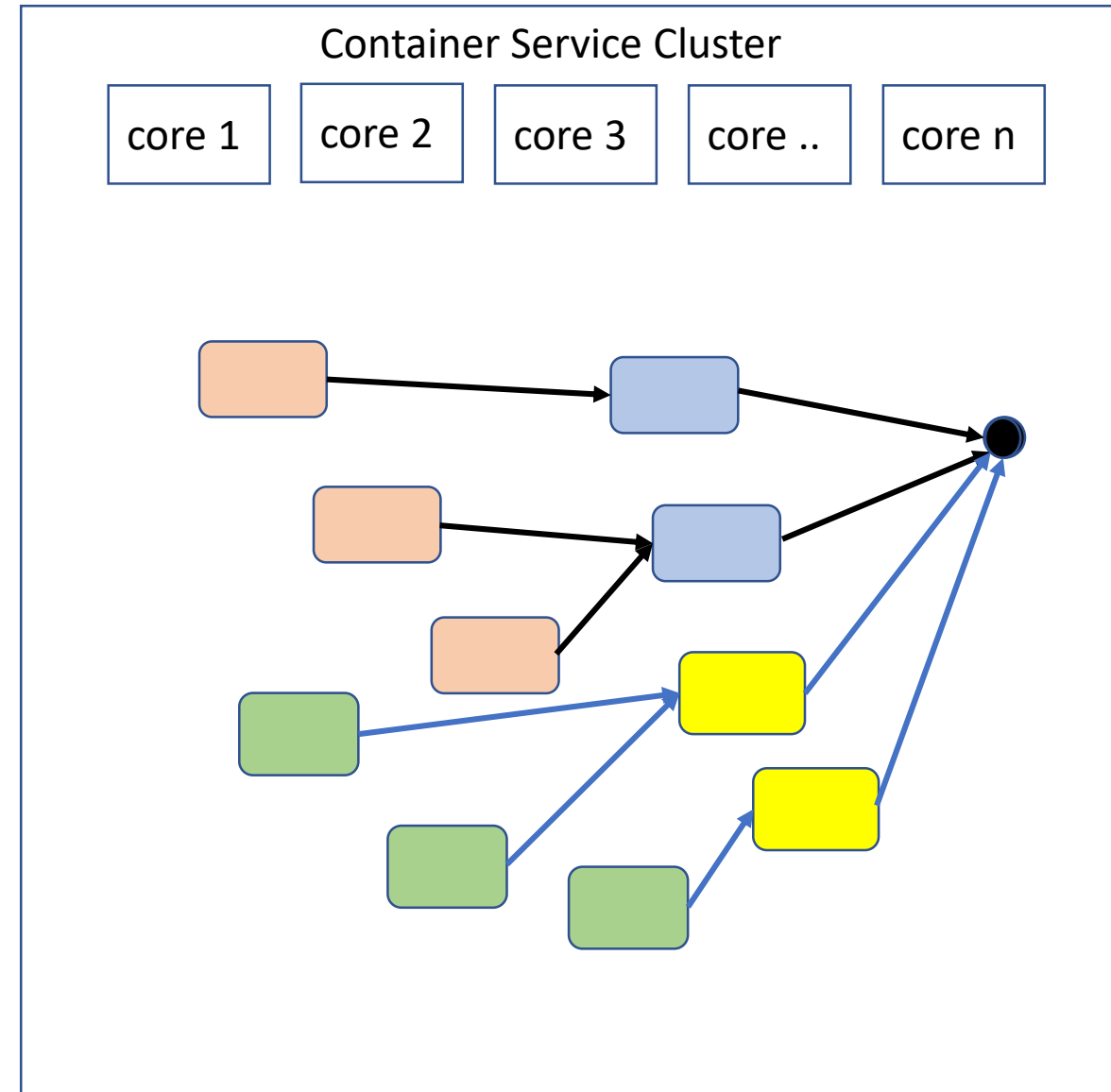
    tempDist = sum(np.sum((kPoints[i] - y) ** 2) \
                      for (i, y) in newPoints)
    for (i, y) in newPoints:
        kPoints[i] = y
```

# Exercises

- If you have Docker installed
  - download this the k-means demo  
<https://sciengcloud.github.io/spark.ipynb>  
to a local directory called notebooks
  - run dbgannon/tutorial  
run -it --rm -p 8888:8888 -v /Users/you/notebooks:/home/joyvan/work \  
dbgannon/tutorial
  - You should see the spark.ipynb. Fire it up. Make sure it is running with kernel python 2 and shutdown other big apps. This needs memory!
- Signup for <https://notebooks.azure.com>
  - Do the twitter analysis demo

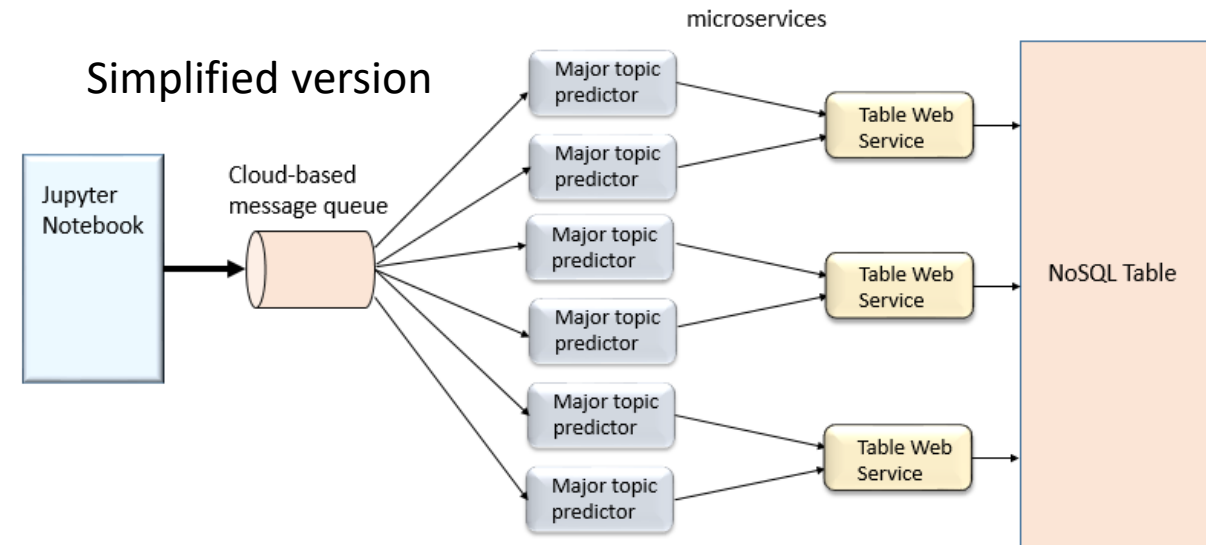
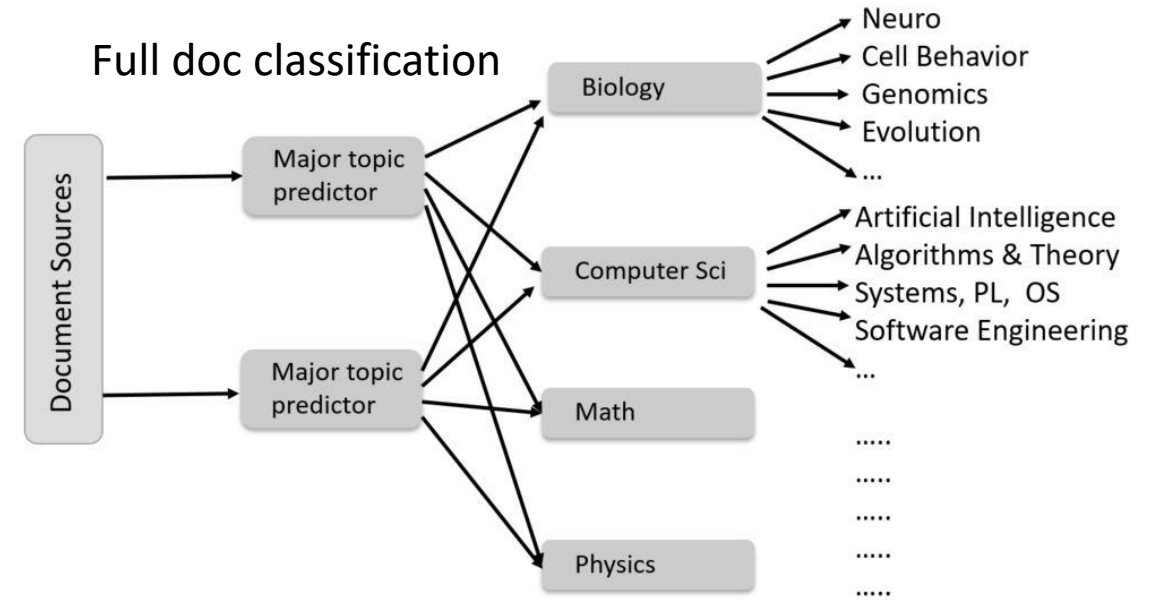
# Microservices

- Divide a computation into small, mostly stateless components that can be
  - Easily replicated for scale
  - Communicate with simple protocols
  - Computation is as a swarm of communicating workers.
- Typically run as containers using a service deployment and management service
  - Amazon EC2 Container Service
  - Google Kubernetes
  - DCOS from Berkeley/Mesosphere
  - Docker Swarm



# Examples

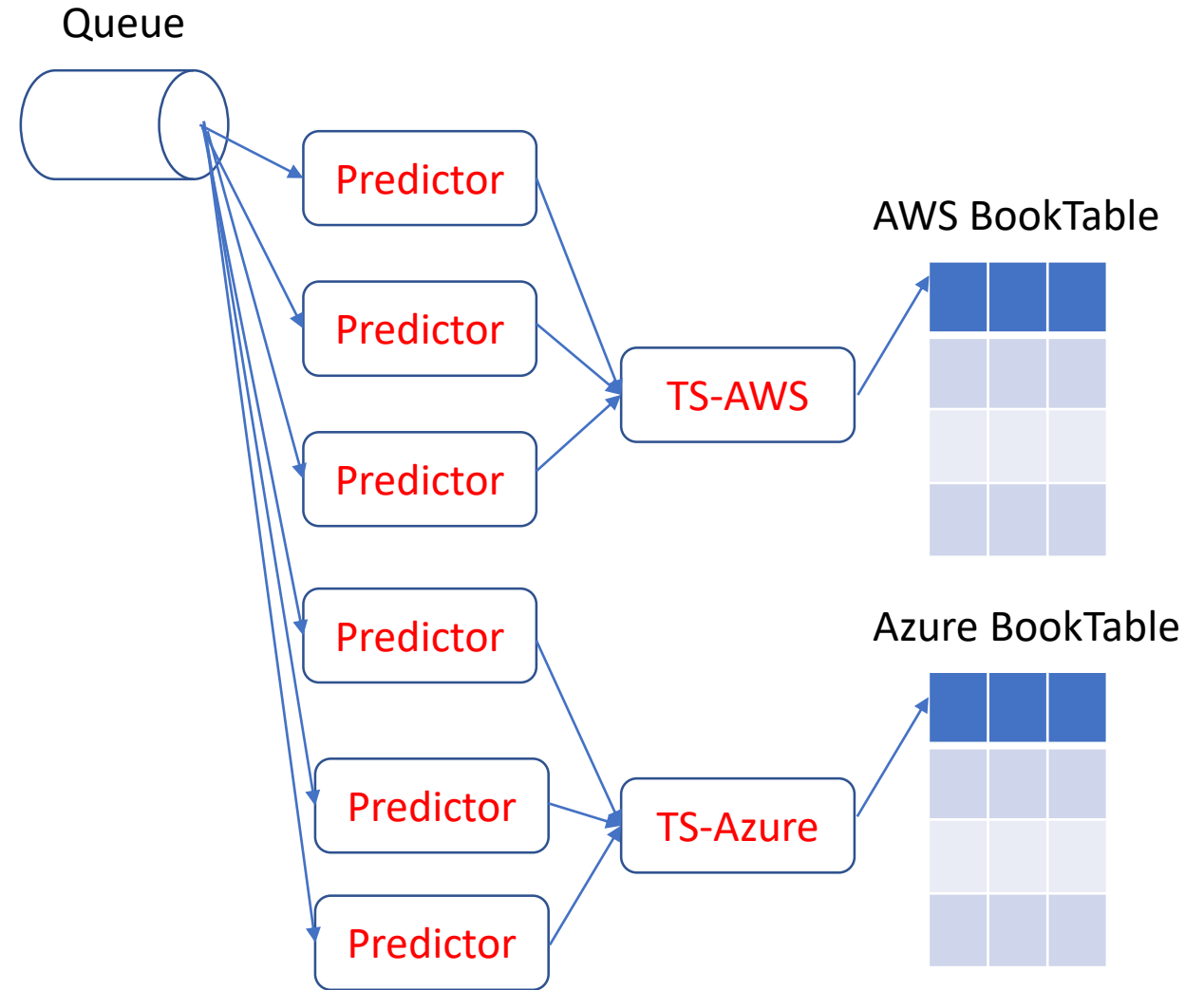
- Netflix, Google Docs, Azure services, eBay, Amazon, the UK Government Digital Service, Twitter, PayPal, Gilt, Bluemix, Soundcloud, The Guardian
- JetStream Genomics Docker swarm to spinup container instance of Galaxy for users on demand
- Processing Document streams
  - Lots of RSS feeds describing recent scientific documents
  - Let's classify them by topic
    - Physics, Math, CS, Biology, Finance, ...
  - By reading the abstracts and using a little machine learning.





# Demo - Using Amazon AWS and Azure Together

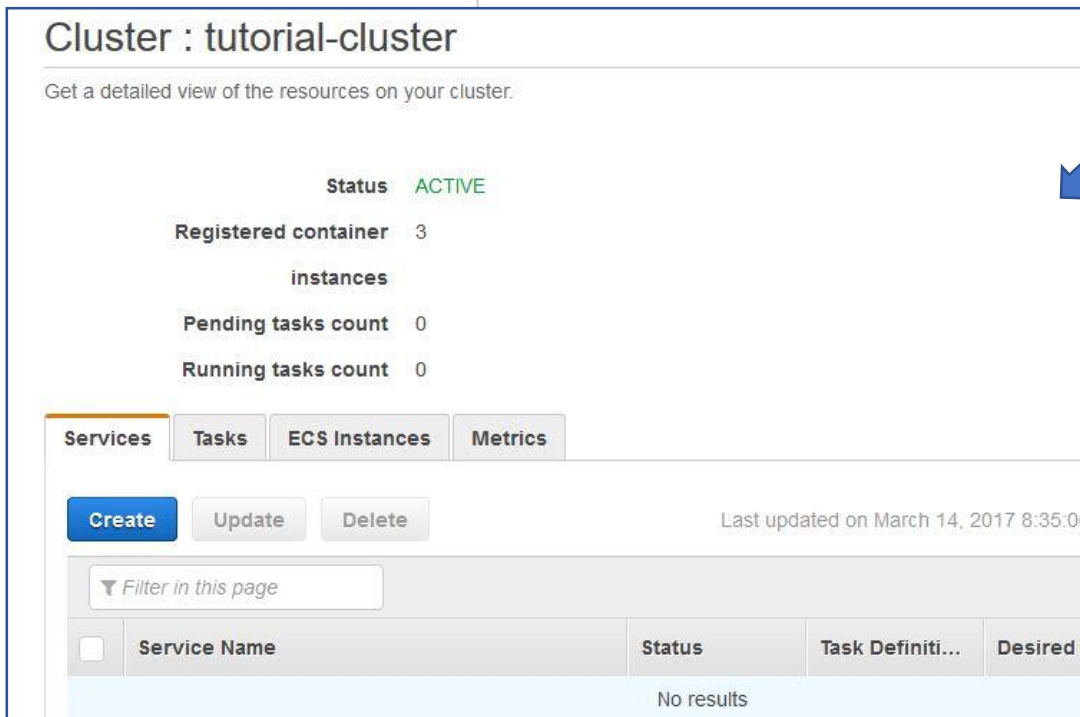
- Create
  - An instance of a message Queue based on AWS SQS
  - An dynamoDB table BookTable
  - An Azure table called BookTable
- Create 3 services
  - Predictor – one parameter (port)
  - TableServiceAWS
  - TableServiceAzure
- 1<sup>st</sup> step: create a AWS elastic container service cluster



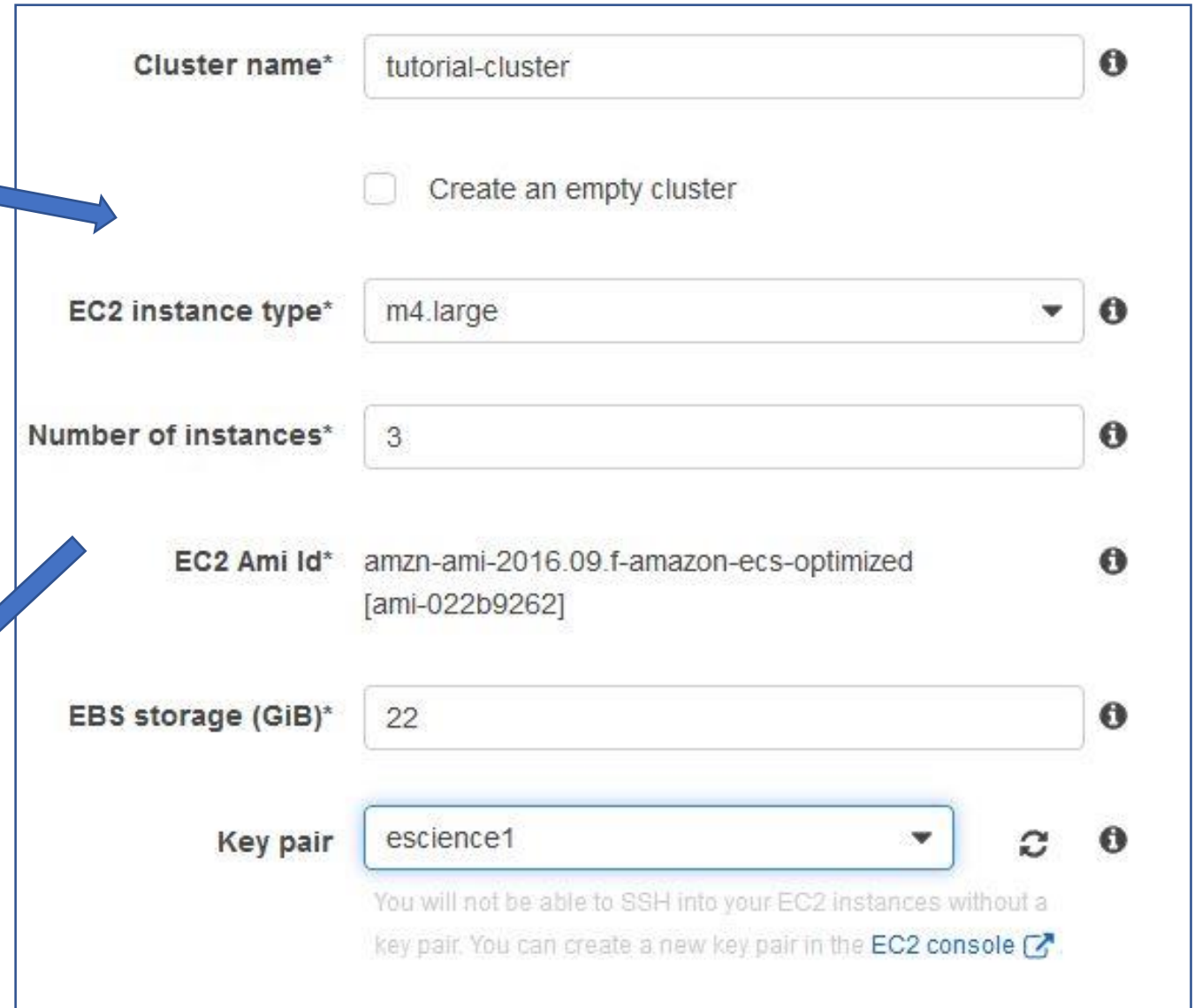
# Create a cluster



The screenshot shows the Amazon ECS Clusters page. The left sidebar contains links for Amazon ECS, Clusters (selected), Task Definitions, and Repositories. The main content area is titled 'Clusters' and includes a description: 'An Amazon ECS cluster is a regional grouping you use the Amazon ECS service. Clusters in'. Below the description is a link to 'ECS documents' and a blue 'Create Cluster' button. A blue arrow points from the 'Create Cluster' button to the 'Create a cluster' form on the right.



The screenshot shows the details page for the 'tutorial-cluster'. The status is 'ACTIVE'. It displays metrics: 'Registered container instances' (3), 'Pending tasks count' (0), and 'Running tasks count' (0). Below the metrics are tabs for 'Services', 'Tasks', 'ECS Instances', and 'Metrics'. The 'Services' tab is active, showing a 'Create' button, 'Update', and 'Delete' buttons. A filter bar is present with the text 'Filter in this page'. Below the filter bar is a table with columns: 'Service Name', 'Status', 'Task Definiti...', and 'Desired'. The table is currently empty, showing 'No results'. A blue arrow points from the 'Create Cluster' button in the previous screenshot to the 'Create' button in this screenshot.



The screenshot shows the 'Create a cluster' form. The form fields are as follows:

- Cluster name\***: tutorial-cluster
- ☐ Create an empty cluster
- EC2 instance type\***: m4.large
- Number of instances\***: 3
- EC2 Ami Id\***: amzn-ami-2016.09.f-amazon-ecs-optimized [ami-022b9262]
- EBS storage (GiB)\***: 22
- Key pair**: escience1

Below the key pair field, there is a note: 'You will not be able to SSH into your EC2 instances without a key pair. You can create a new key pair in the [EC2 console](#)'.

# Code to create a service

```
response = client.register_task_definition(  
    family='predictorAzure',  
    networkMode='bridge',  
    taskRoleArn= 'arn:aws:iam::066301190734:role/mymicroservices',  
  
    containerDefinitions=[  
        {  
            'name': 'predictorAzure',  
            'image': 'dbgannon/predictor-new',  
            'cpu': 20,  
            'memoryReservation': 400,  
            'essential': True,  
            'command': [ '8055' ]  
        },  
    ],  
)  
  
response = client.create_service( cluster='tutorial-cluster',  
    serviceName='predictorAzure',  
    taskDefinition='predictorAzure:1',  
    desiredCount=1, deploymentConfiguration={  
        'maximumPercent': 100,  
        'minimumHealthyPercent': 50 }  
)
```

[Go to Demo](#)

# Section Summary

- The cloud data centers are designed to scale
  - Traditional HPC MPI programming is possible, but a Cray is better.
- The cloud is best at distributed scale, interactive computation
  - Spark in Yarn with Jupyter is a good example
- MapReduce and Graph models are well supported
- Microservices provide a means to support very large scale parallelism in continuously running applications.