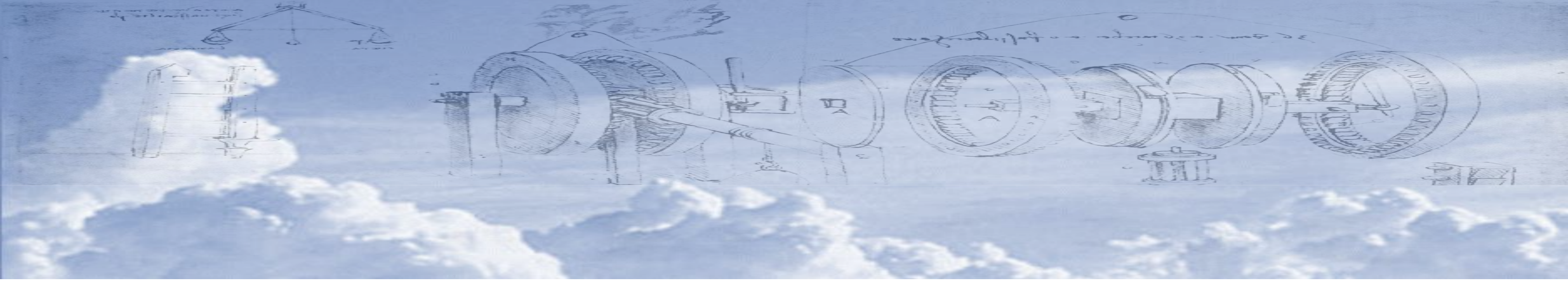


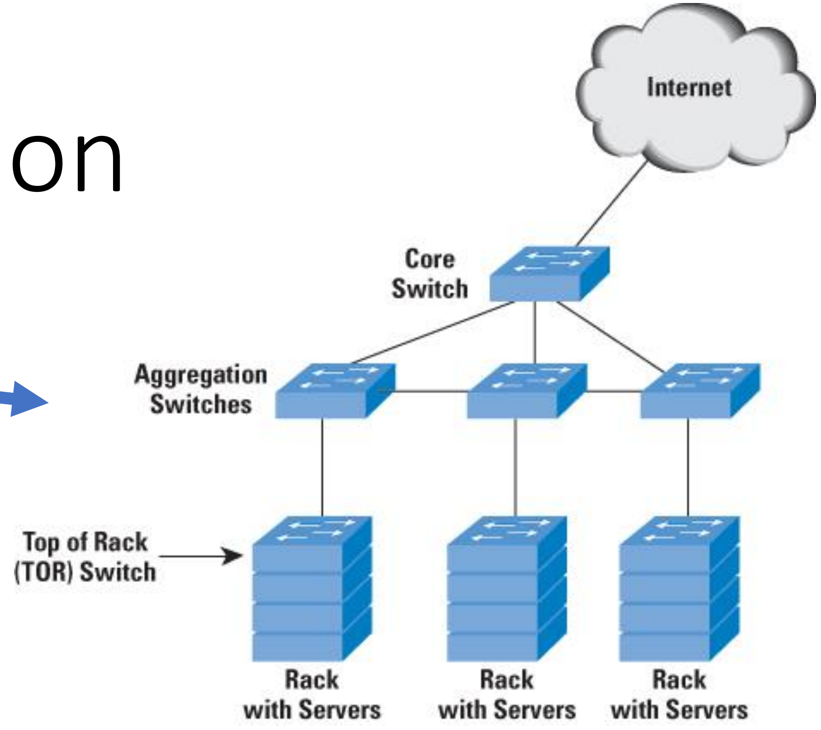
Cloud Computing for Science

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



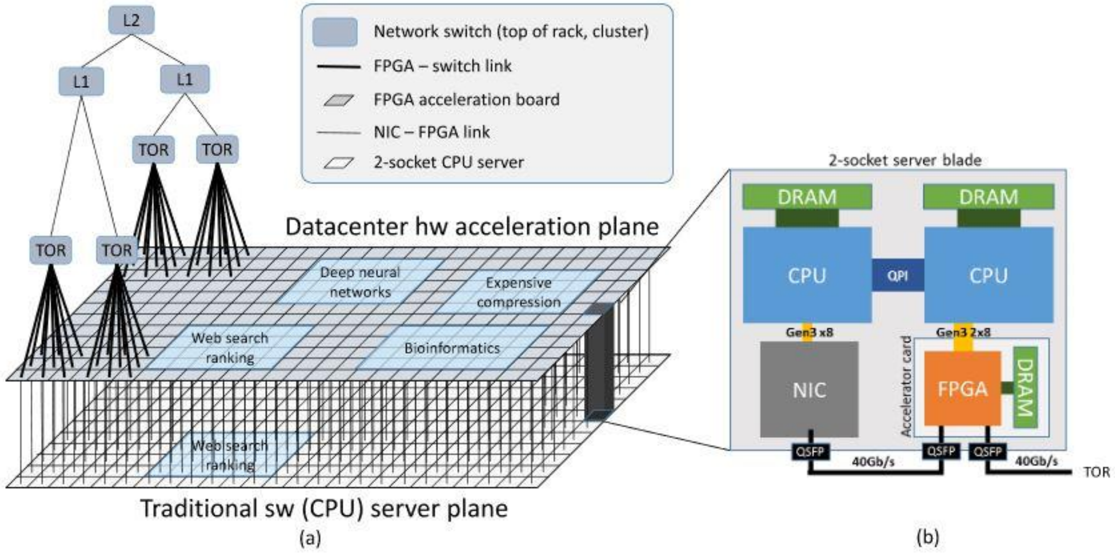


Fig. 1. (a) Decoupled Programmable Hardware Plane, (b) Server + FPGA schematic.

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

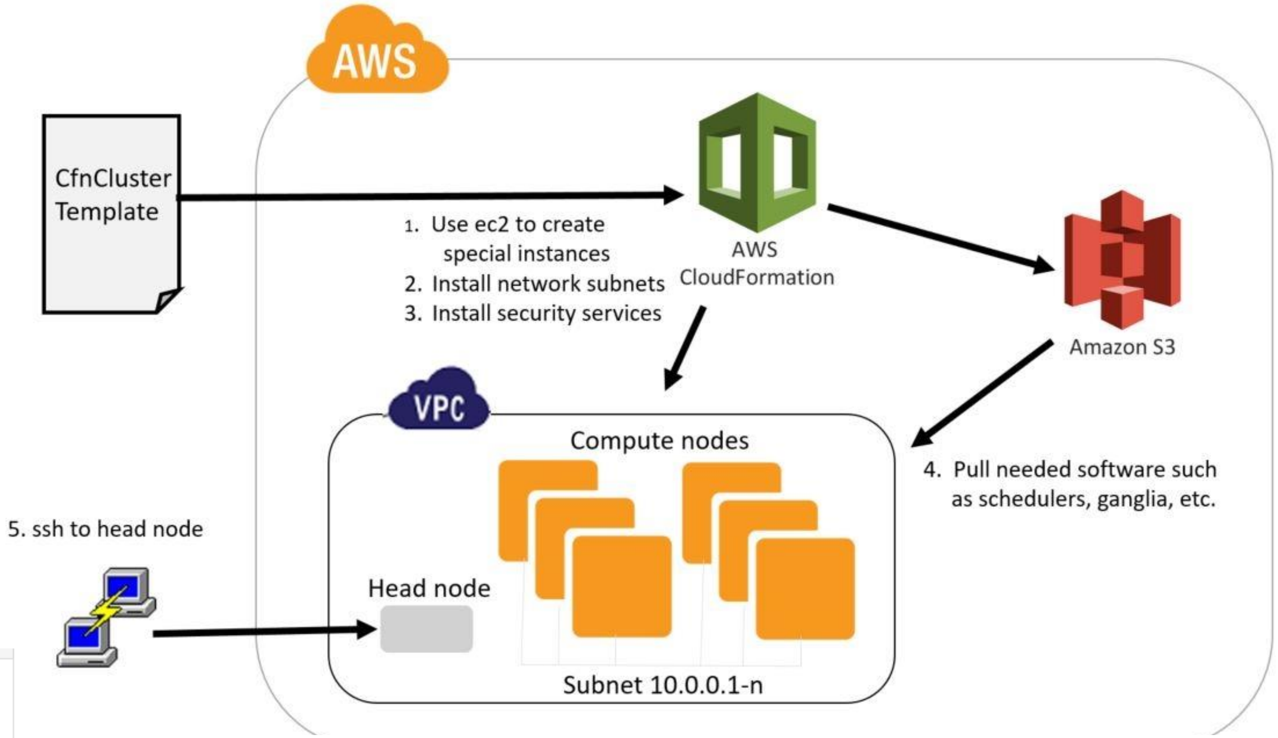
Deploy to Azure | Visualize

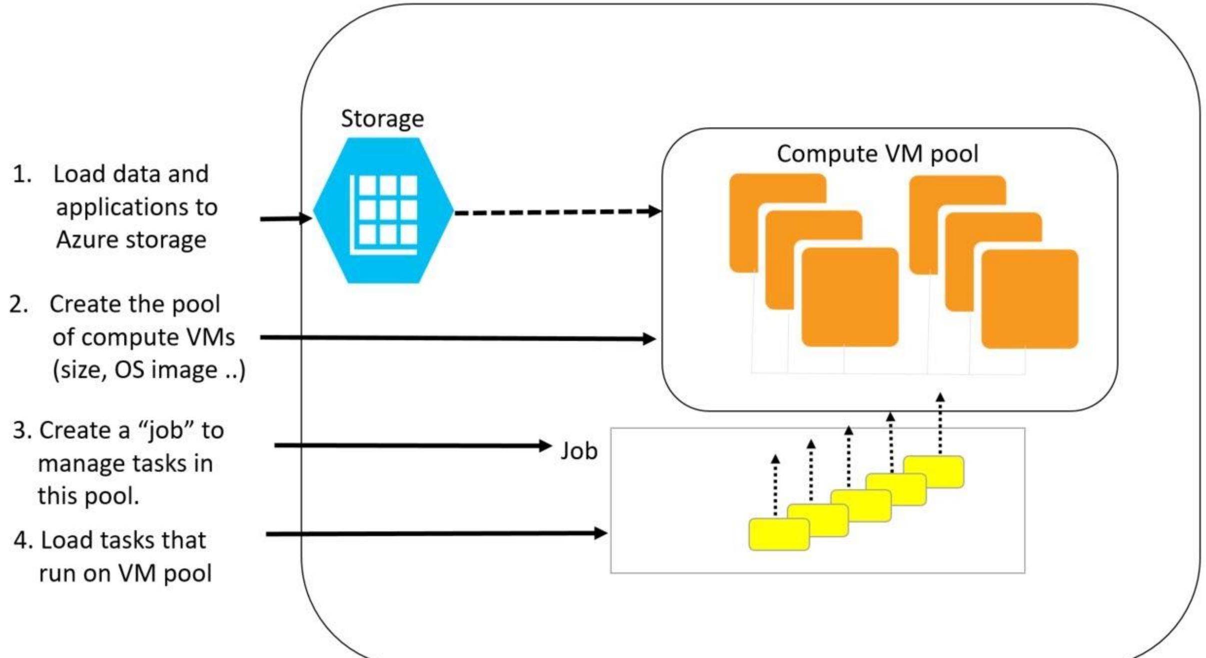
1. Fill in the 3 mandatory parameters - public DNS name, a storage account to hold VM image, and admin user password.
2. Fill in other info and click "OK".

Using the cluster

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

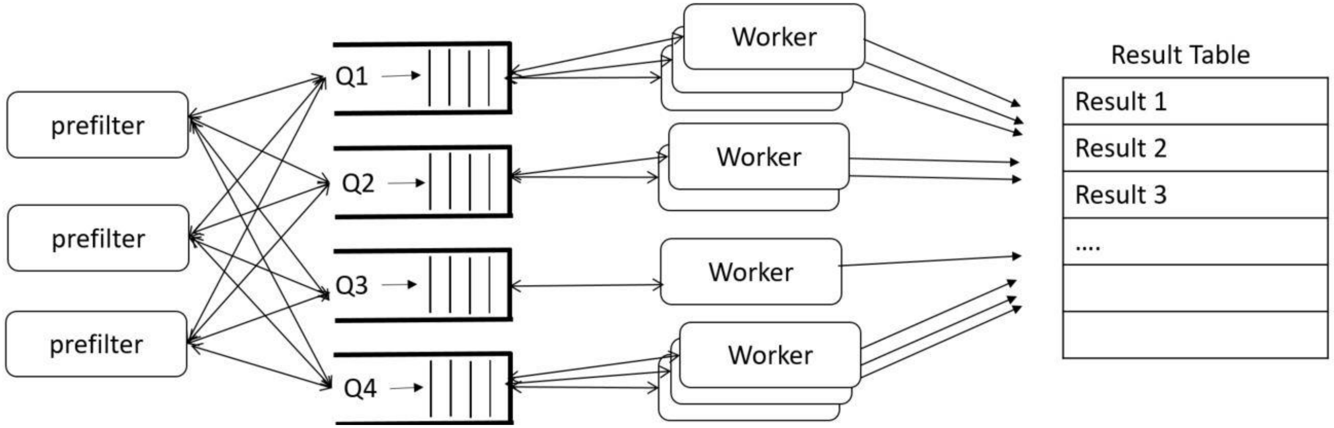
- Or use Azure Batch
  - Similar to AWS batch

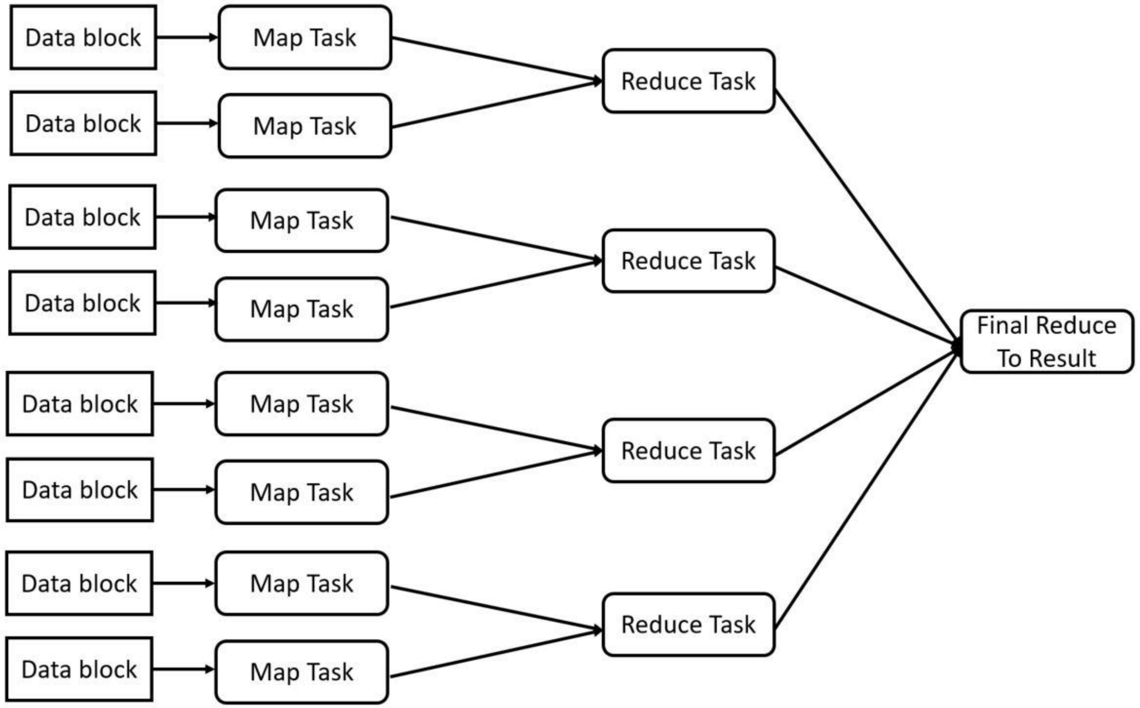




Task Parallel and Map Reduce

- 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



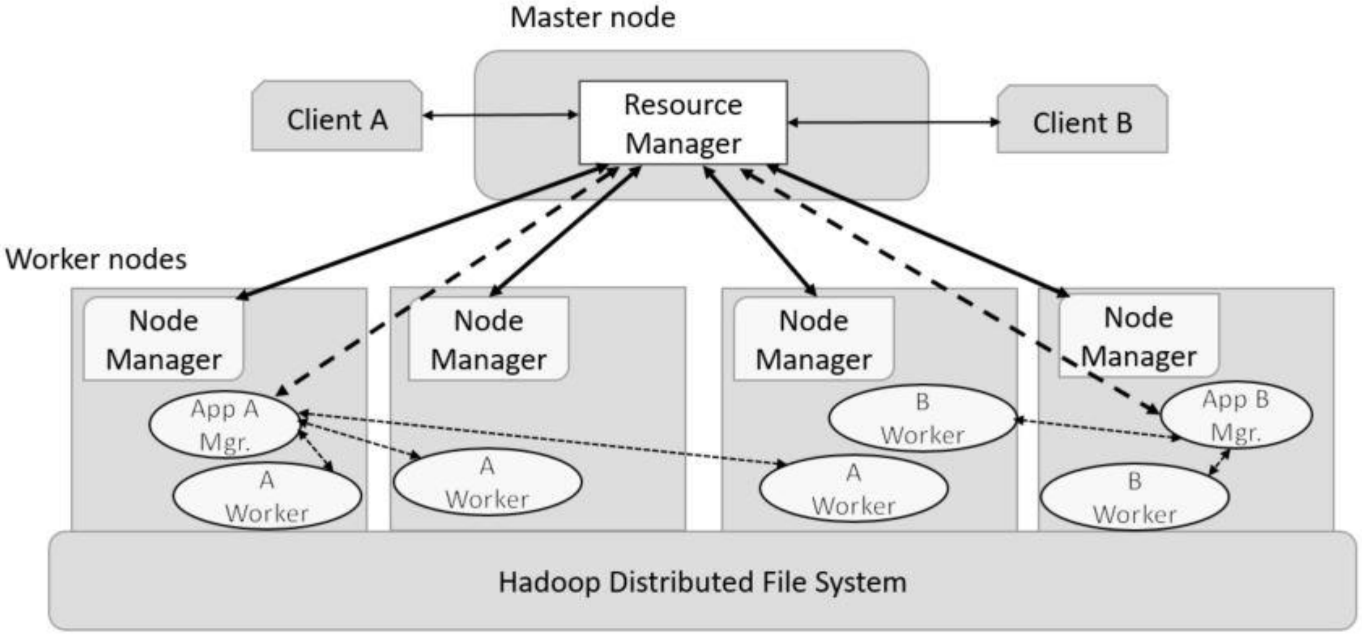


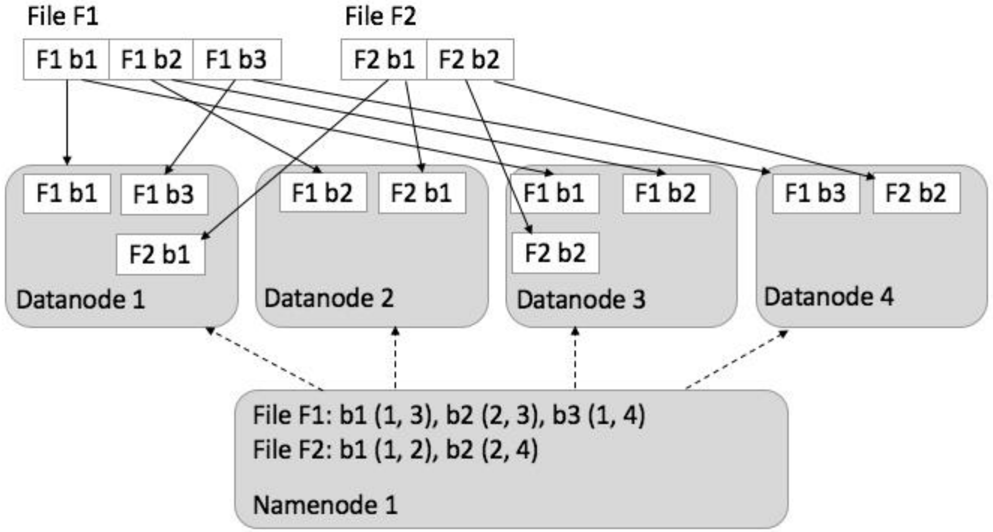
The Hadoop- Yarn ecosystem

- Yarn is the name of a project containing many elements

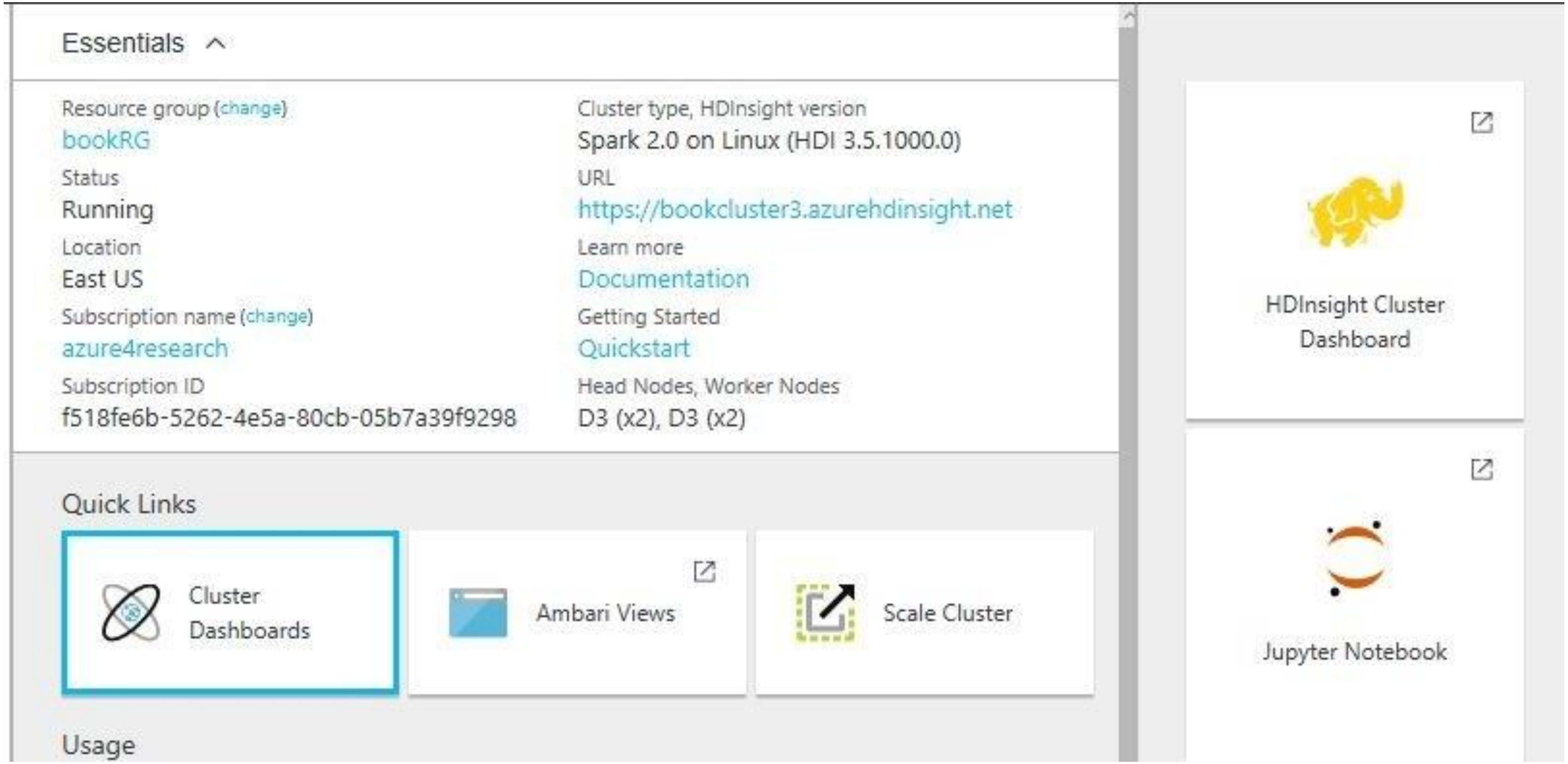
- The runtime system is distributed
- Hadoop, Spark run in distributed mode
- Multiple clients can access the resource manager
- Jupyter and Zeppelin are interactive clients

- HDFS is the Hadoop File system
- Distributed over data node servers
- Files are blocked, distributed and replicated
- Files are write-once.



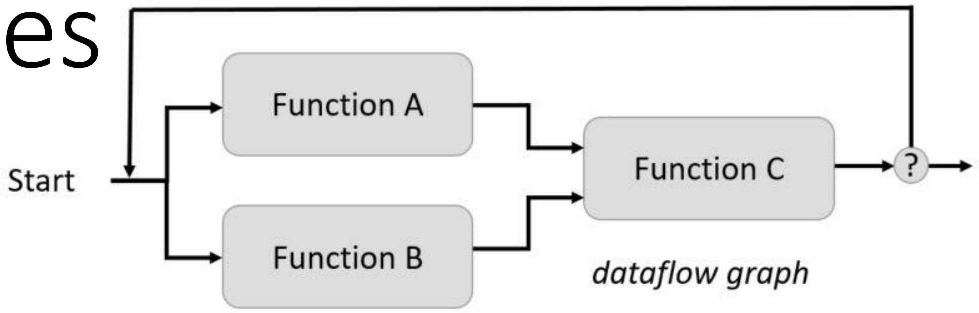


Azure HDInsight is a Yarm Environment

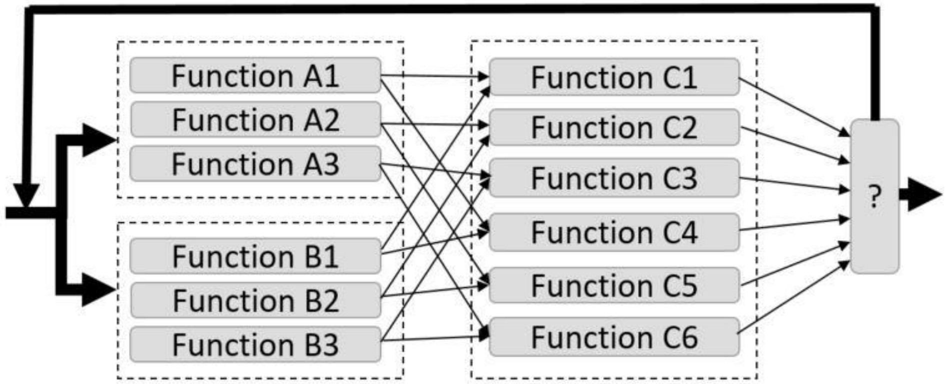


Graph Parallel and Microservices

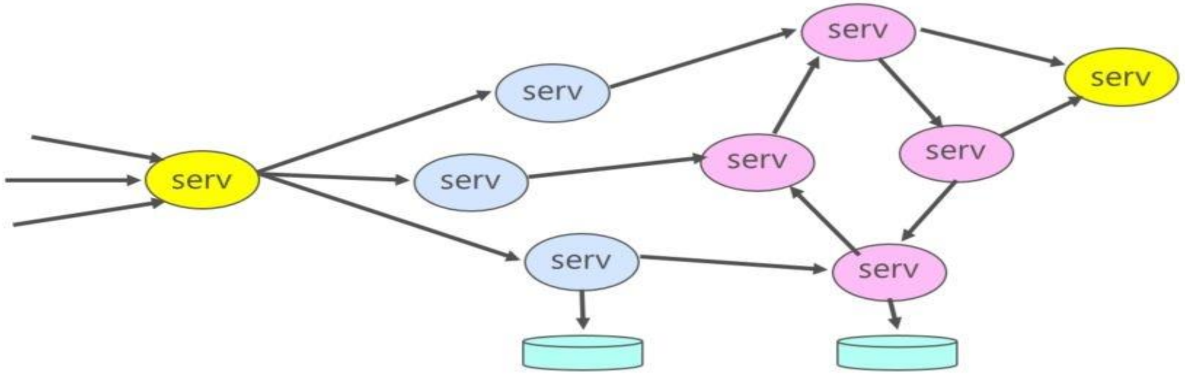
- Graph Parallel
  - The data is in distributed arrays or streams.
  - build a data flow graph of the algorithms functions.
  - The graph is compiled into parallel operators that are applied to the distributed data structures.
- Microservices
  - Divide a computation into small, mostly stateless components that can be
    - Easily replicated for scale
    - Communicate with simple protocols
  - Computation is as a swarm of communicating workers.



dataflow graph



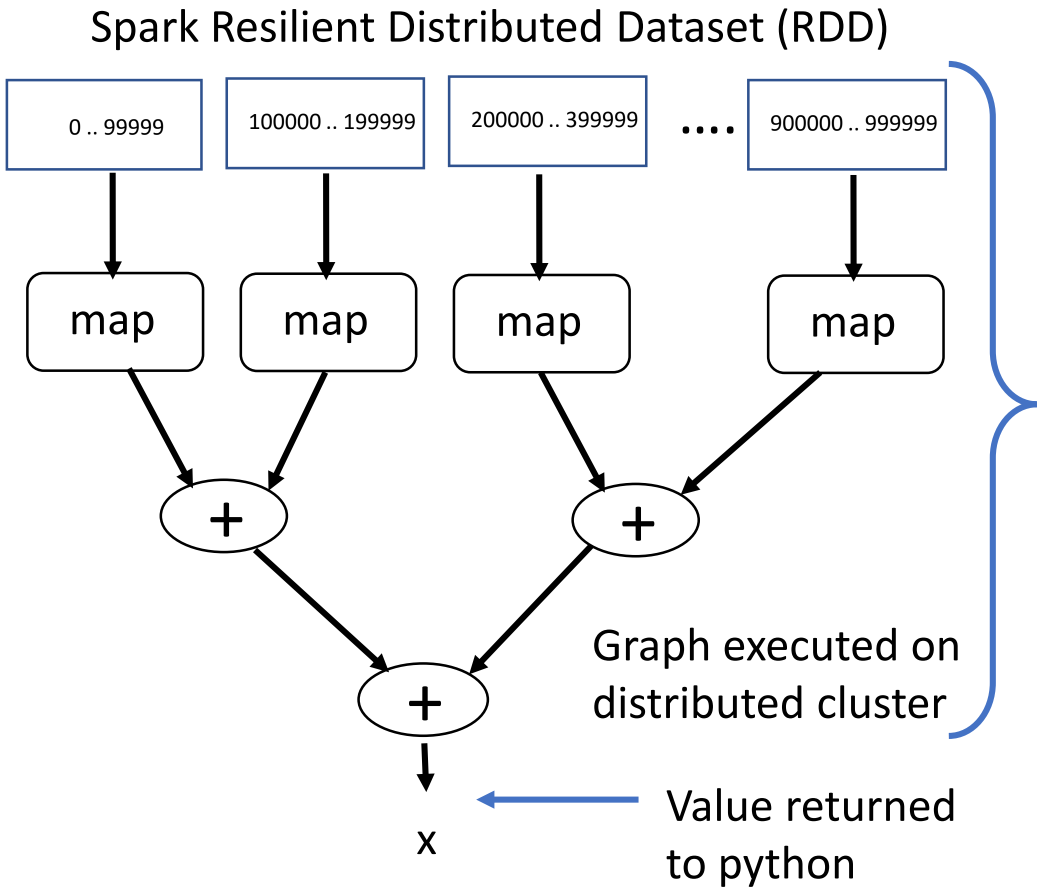
parallel execution graph



Graph computation example: Spark

- A simple map reduce: Compute $\lim_{n->\infty} \sum_{i=1}^{n} \frac{1}{i^2} = \frac{\pi^2}{6}$
- For n = 10,000,000
- In Spark on Python is:

```
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, (sum(p)/sum(1)))
    - Set q[j] to be the new centroid sum(p)/sum(1)

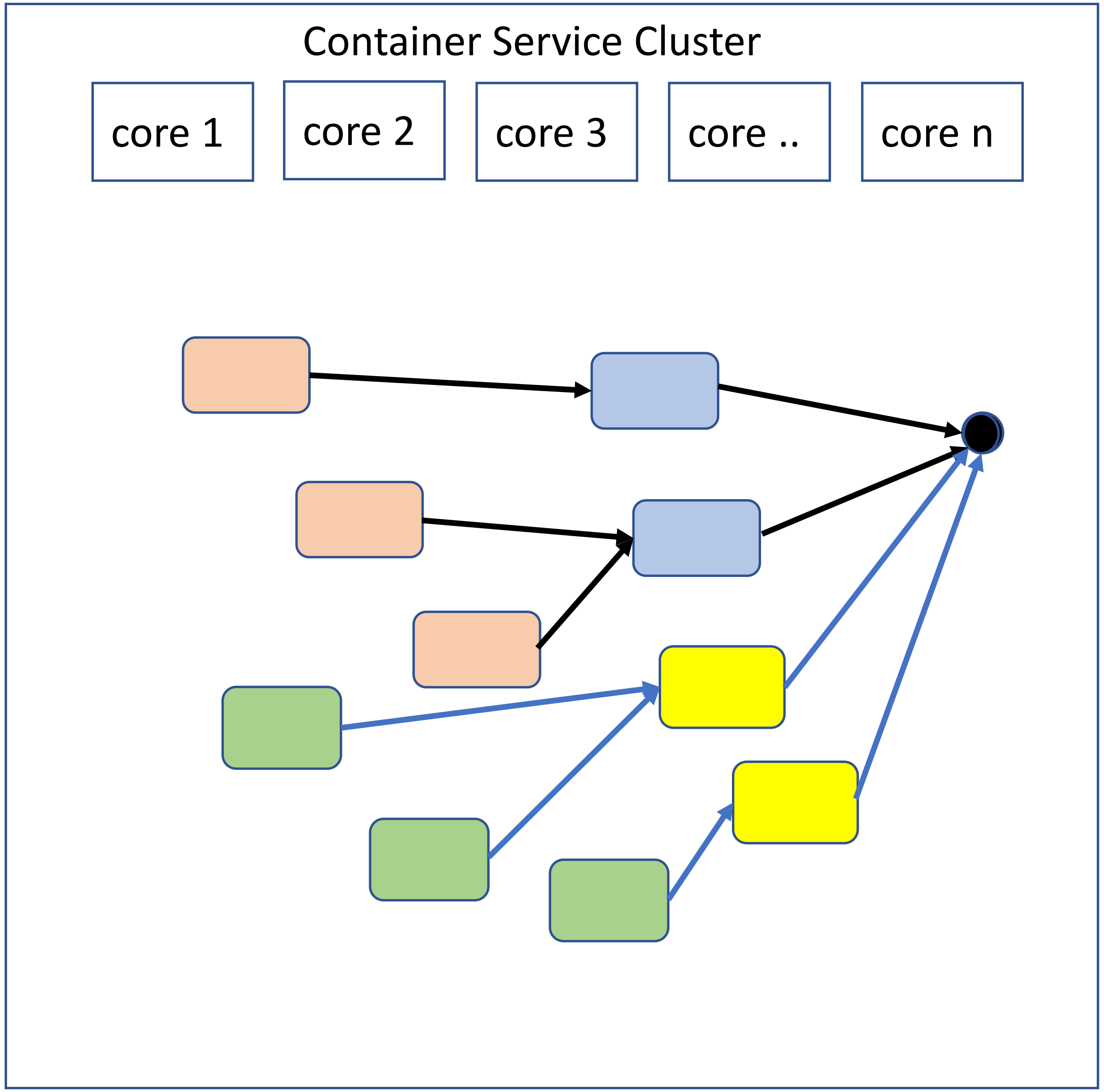
```
tempDist = 1.0
while tempDist > convergeDist:
    newPoints = data \
            .map( lambda p: (closestPoint(p, kPoints), (p, 1))) \
            .reduceByKey(lambda x, y : (x[0] + y[0], x[1] + y[1])) \
            .map(lambda x : (x[0], x[1][0]/ x[1][1])) \
            .collect()

    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
  - run dbgannon/tutorial
    run -it --rm -p 8888:8888 dbgannon/tutorial
  - You should see the spark.ipynb in the notbooks.   Fire it up.   Make sure it is running with kernel python 2 and shutdown other big apps.   This needs memory!
- If you don't have docker, but do have the data science vm:
  - Go to https://your-vm-ip:8000 and login.
  - Upload spark.ipynb from the tutorial tarball folder
- For something different: Signup for https://notebooks.azure.com
  - Do the twitter analysis demo

Microservices

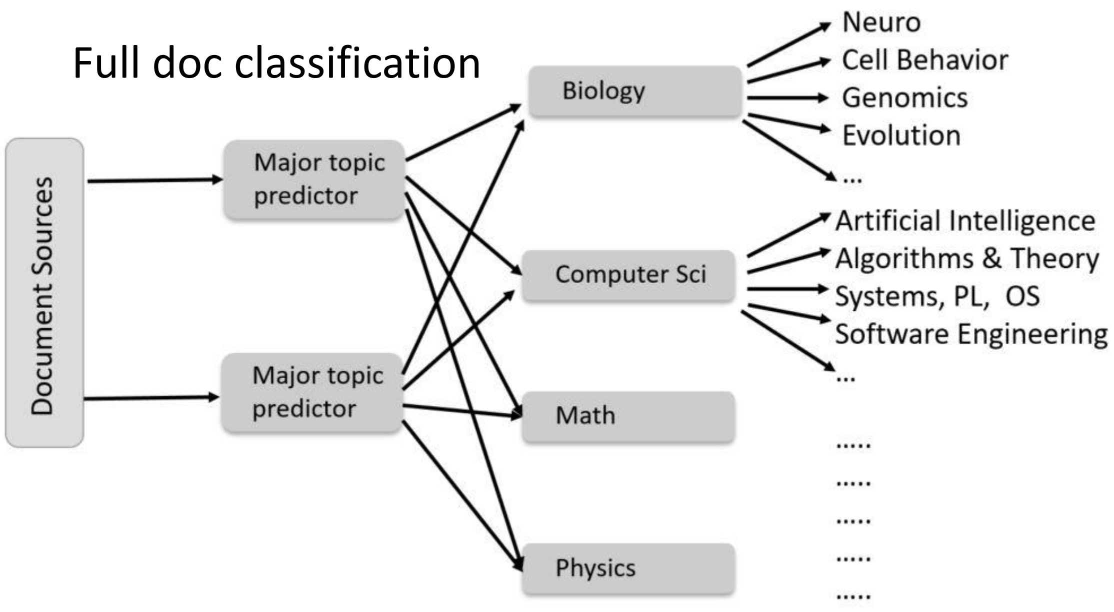
- Divide a computation into small, mostly stateless components that can be
  - Easily replicated for scale
  - Communicate with simple protocols
  - Computation is as a swarm of communicating workers.
- Typically run as containers using a service deployment and management service
  - Amazon EC2 Container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



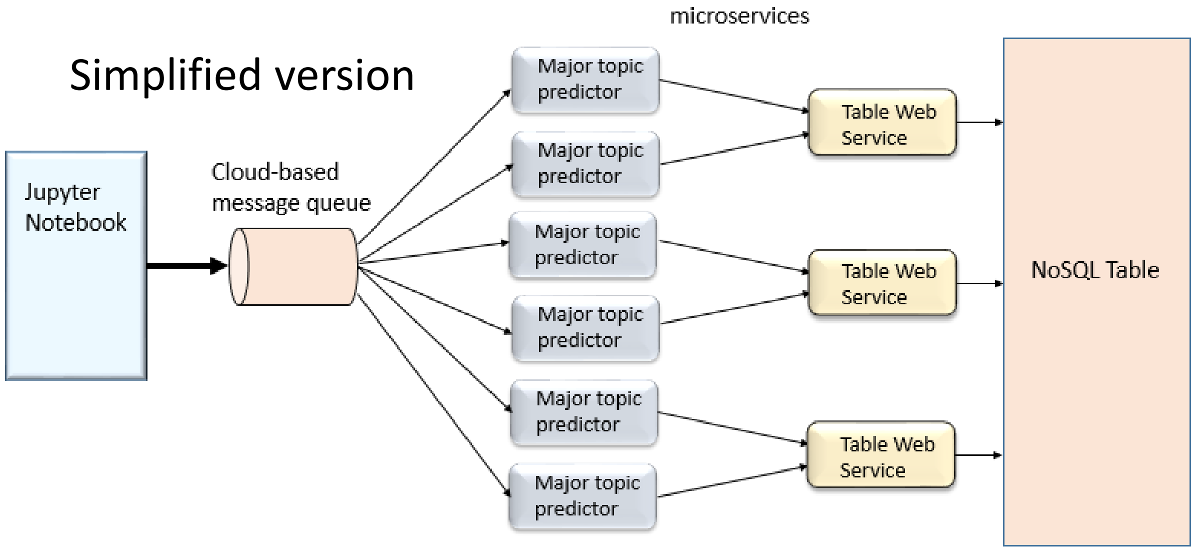
Examples

- 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.

Full doc classification

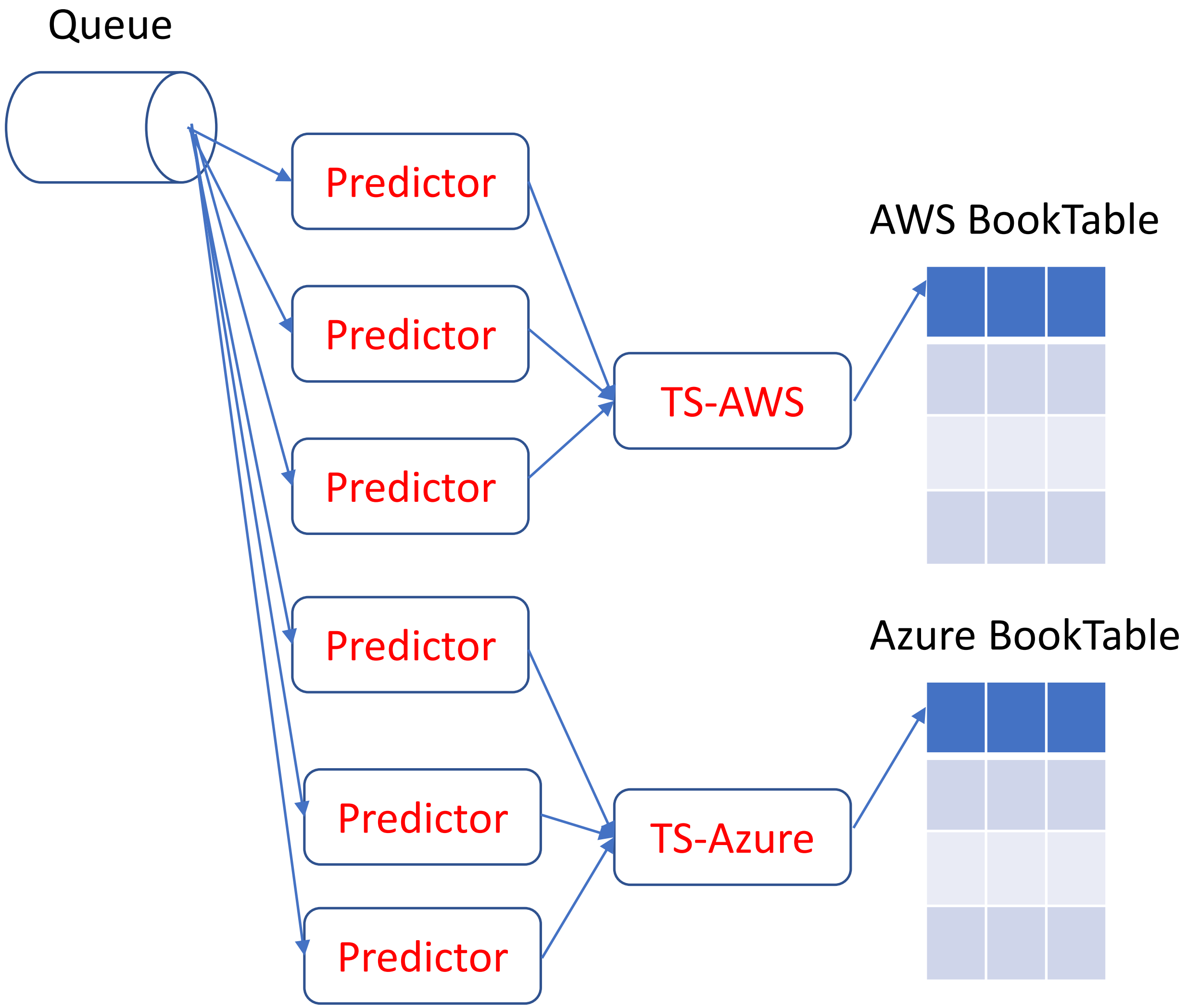


Simplified version

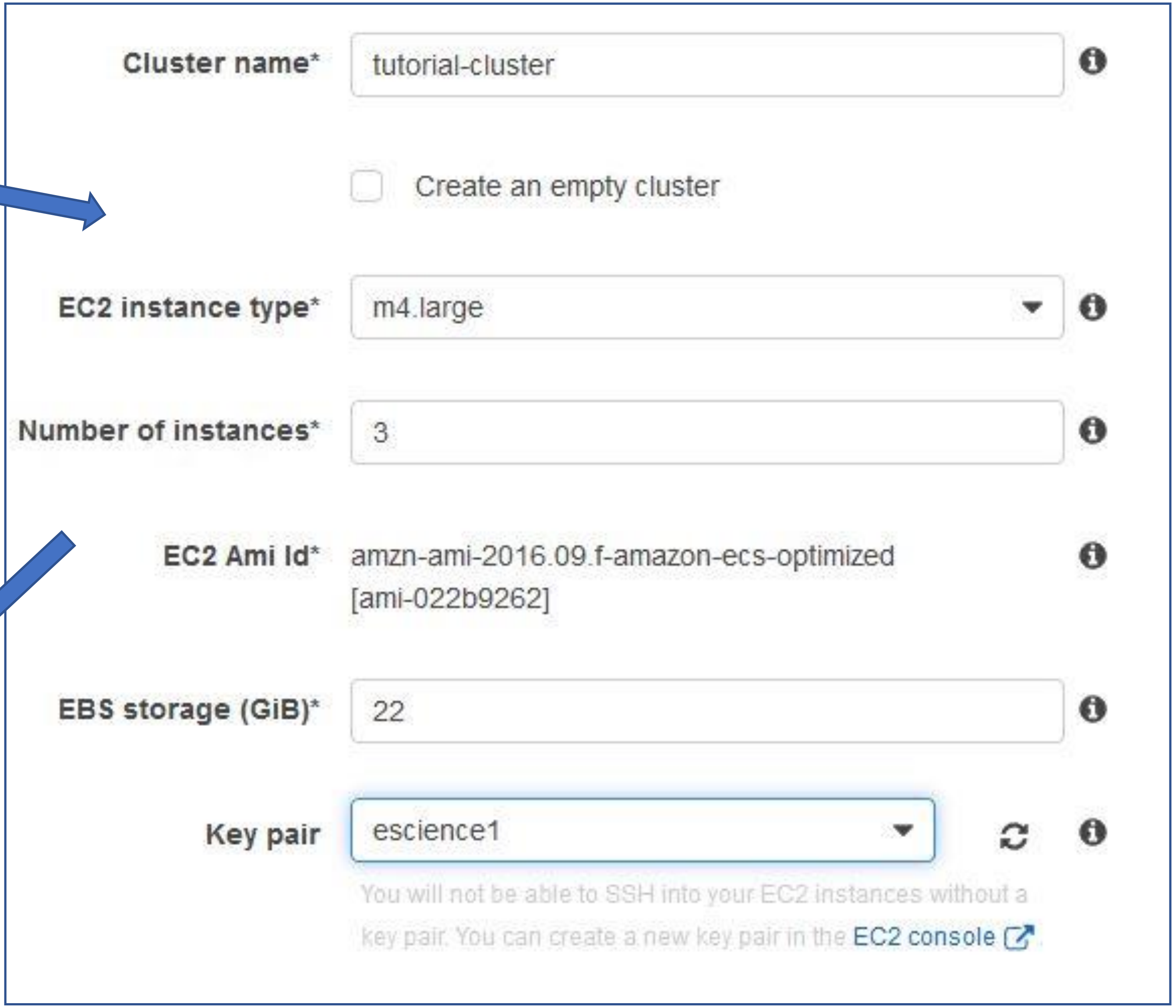


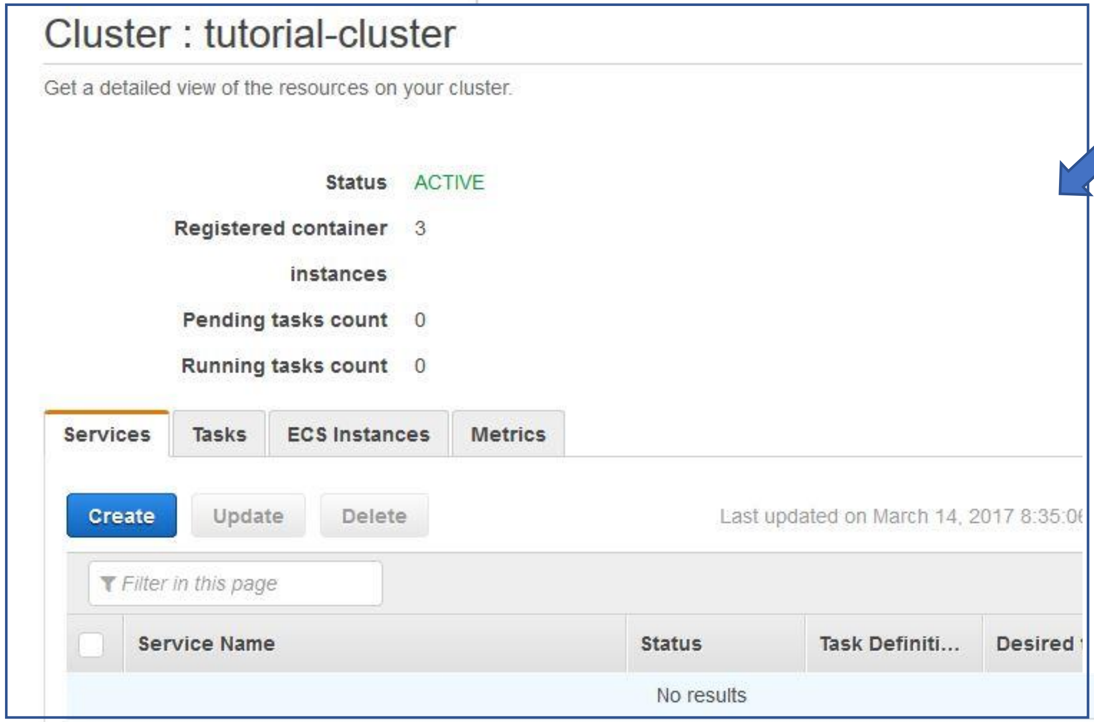
Demo - Using Amazon AWS and Azure Together

- 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
- 1st step: create a AWS elastic container service cluster



Create a cluster





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.