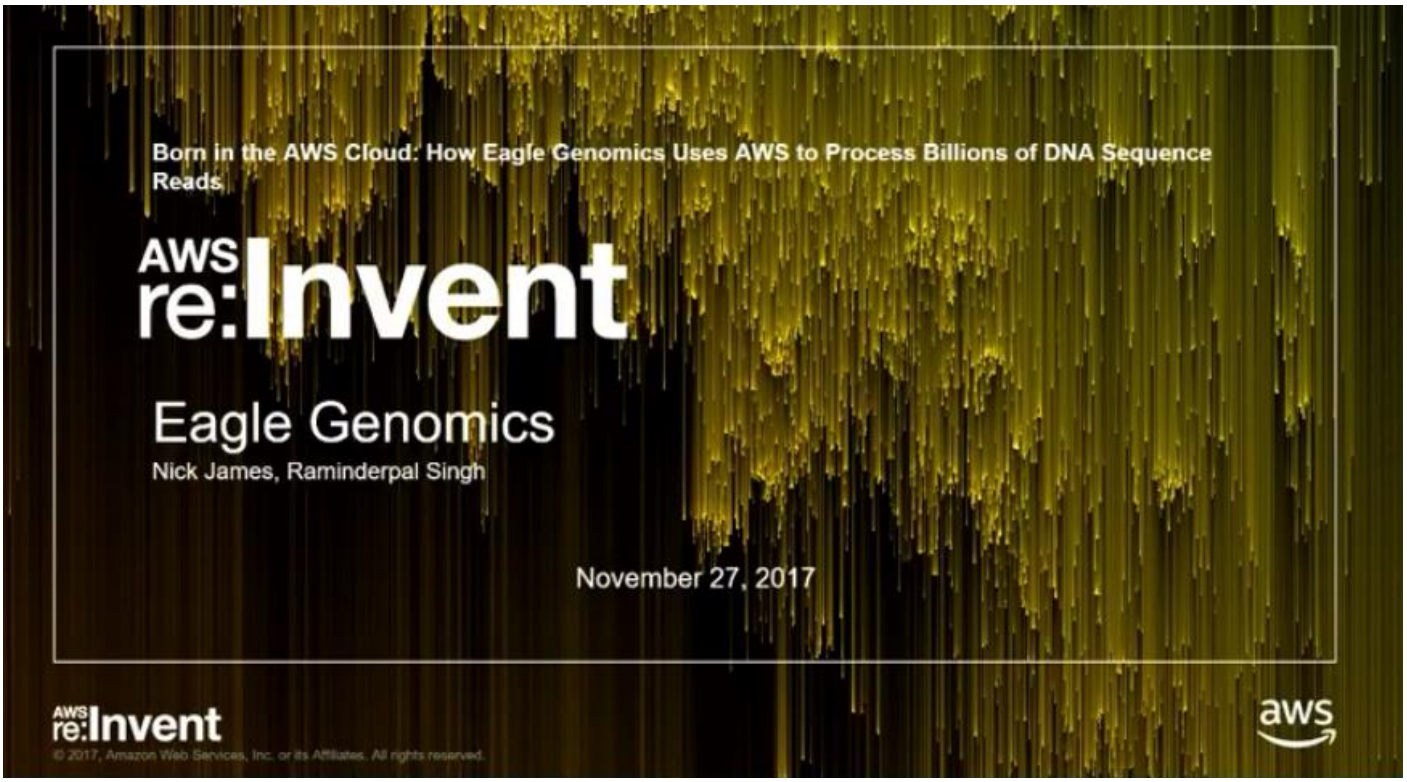


With the increasing use of genomic sequencing for scientific discovery, the rate-limiting step for researchers is not in obtaining genetic code, but in having the capacity for storage and computing power to analyze it. In this session, you learn how Eagle Genomics built a cloud platform that uses an open-source workflow engine (eHive), Docker containers to process jobs, and a REST service to manage pipeline runs, all to help customers process genetic sequences up to 20 times faster without additional costs. You also learn how Eagle used Amazon EC2 Spot instances to provide low-cost compute power, and services such as Amazon S3 to power their cost-effective and scalable genetic processing platform.

About Eagle Genomics

Founded in 2008
Based in Cambridge, UK, on the Wellcome Genome Campus

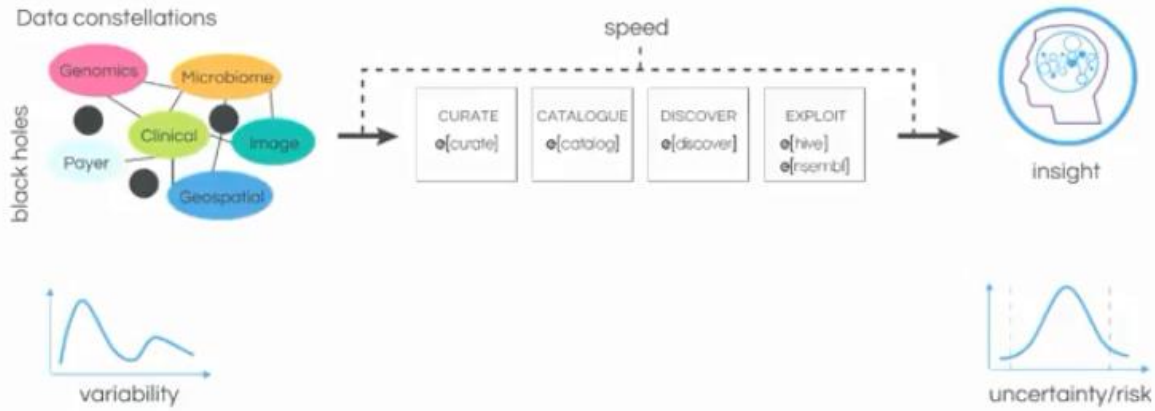
Smart data management for Life Sciences—software and services

- Human and animal health
- Personal care and cosmeceuticals
- Food and nutriceuticals

Blue-chip customers across US and Europe

One of the first companies ever to run bioinformatics pipelines on AWS cloud, in 2008

The challenge



Key requirements:

- Trustworthy and validated evidence
- Cater for future and unforeseen scientific questions
- Manage data as assets, quantify their quality and value
- No single data source can answer complex scientific questions

Evolution of data processing

Past: Stand alone

Weeks / Months

Present: Server/Cluster

Days / Weeks

Emerging: Secure Scalable Cloud

e[catalog] e[hive]

MRC Cloud Infrastructure for Microbial Bioinformatics

amazon web services

Hours

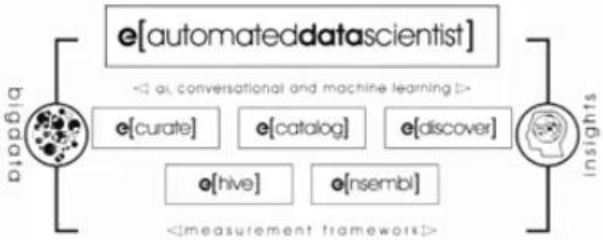
100K reads → **5 Billion reads**

Empowering data biologists into the future

Today: Simple to deploy and scalable on cloud

- "Always On", access from anywhere
- Effective collaboration across global teams
- Empowering biologists to become data scientists
- Streamlined workflows—"Click and Go"
- Scalable storage and analysis power

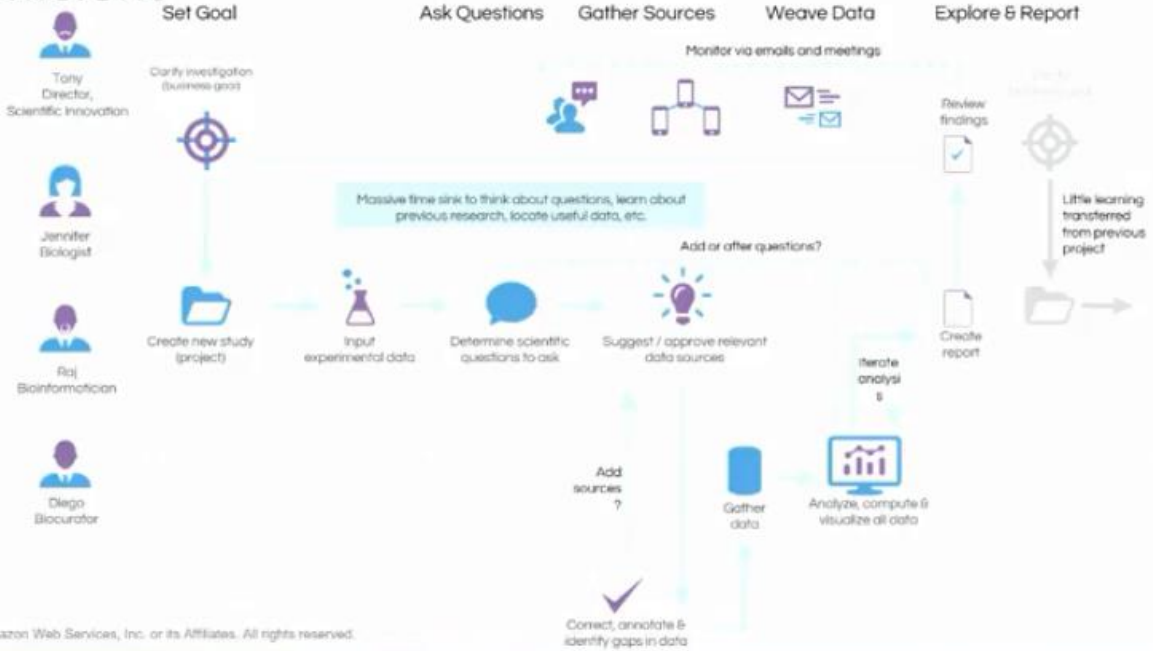
Tomorrow: True data-driven innovation



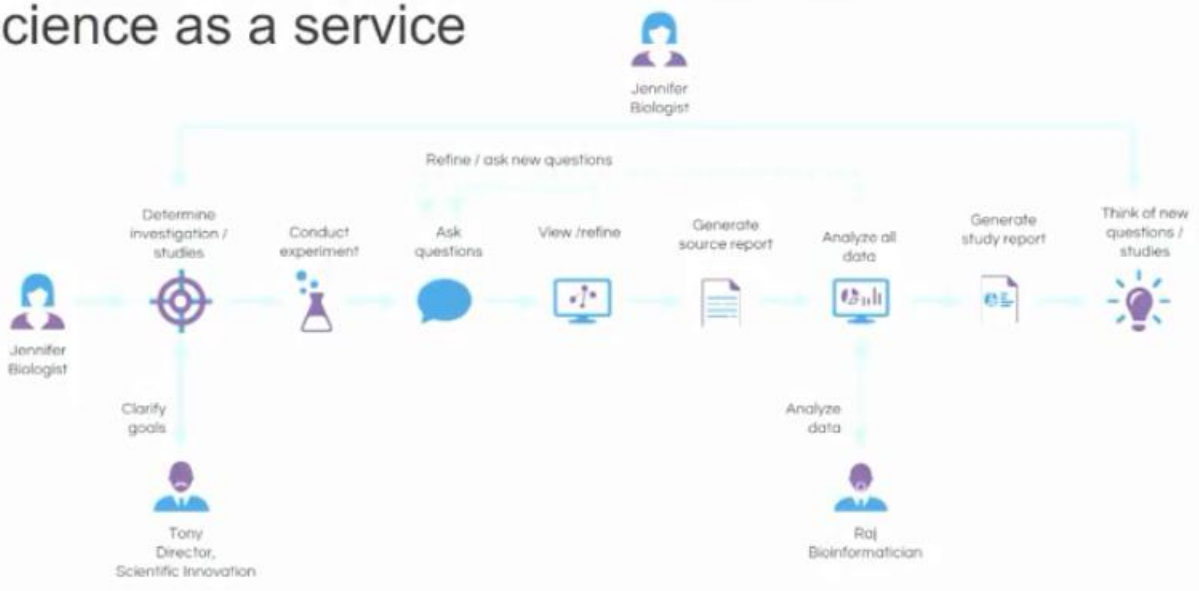
- Questions-driven, AI-enabled
- Radically improved productivity of your scientist
- Transformational interaction between executive and scientist
- Data governance by design
- Data-driven innovation

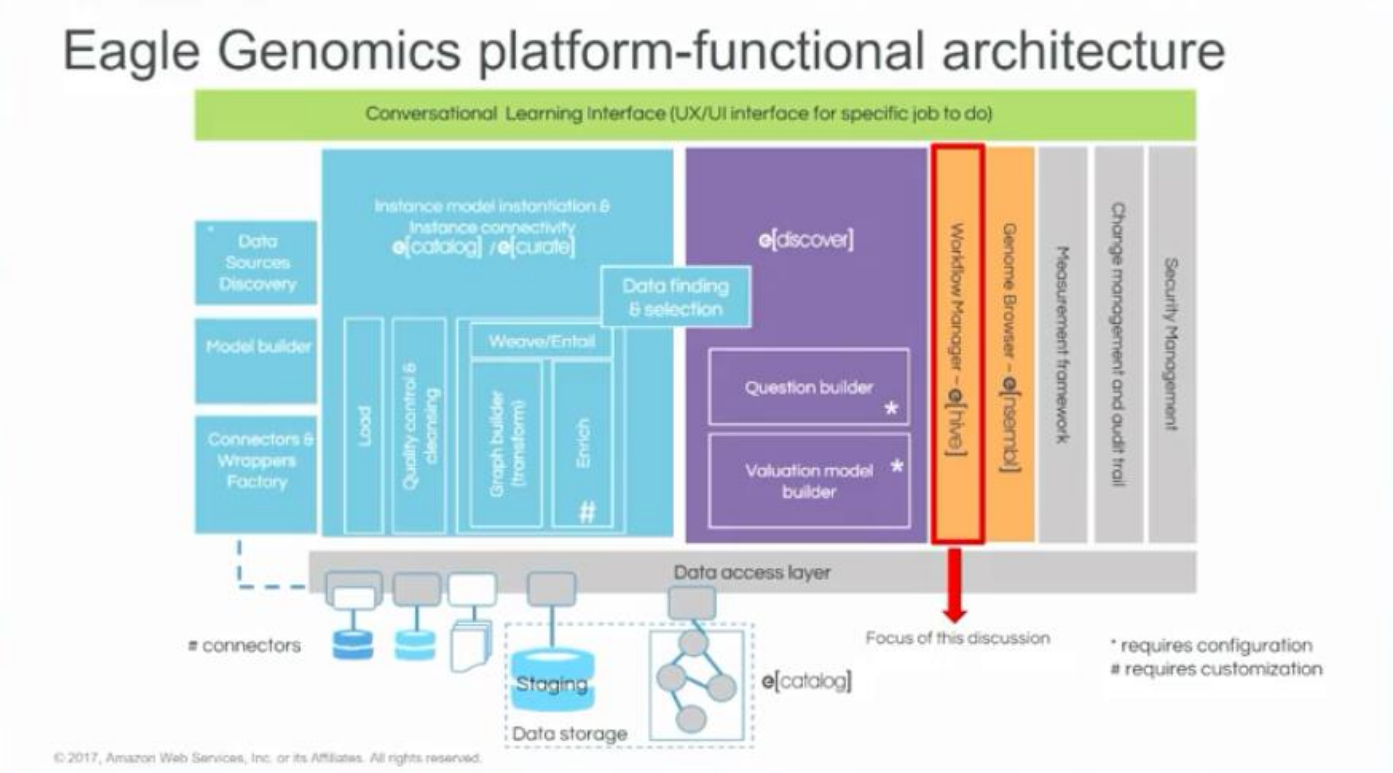
Presented by Unilever at July 2017 webinar:
https://goo.gl/SYrz5

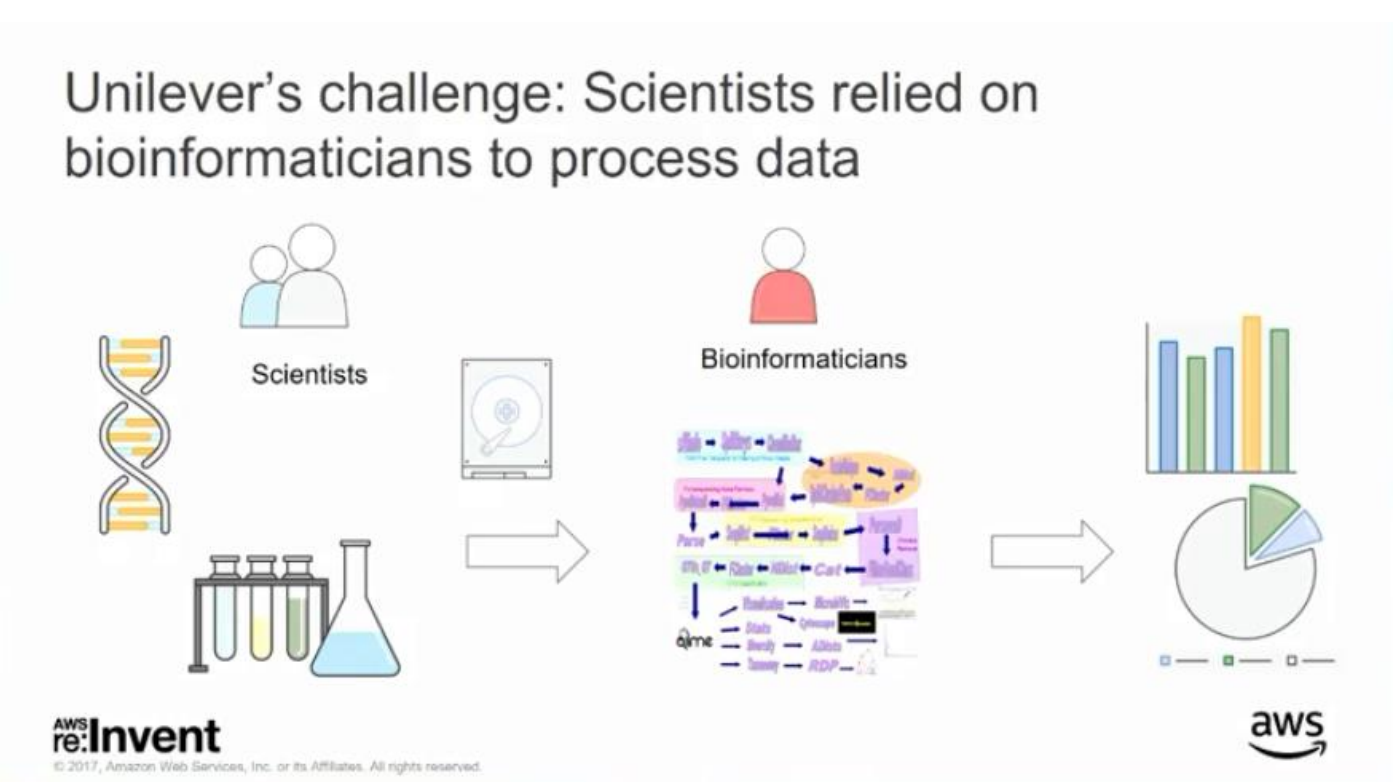
Current process entangled, human intensive, and inefficient



The future—the automated data scientist—data science as a service







We need to industrialized the middle part and reduce the number of bioinformaticians needed and make it self service for the scientists to use on their own with little help

Unilever's requirements

Background

New biology and informatics program promotes access to public data

Underlying architecture must keep pace with expanding scientific discoveries

Simple but robust solution combines Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3) with the open-source workflow system, eHive

Results

Ten times as many scientists can process studies simultaneously, compared to non-cloud architecture

Genetic sequence processing is twenty times faster, without increasing compute costs

Confidence that the AWS-based deployment helps Unilever's scientists create market-leading innovations

Successful outcome: Reference scientific project

First-of-a-kind project providing product competitiveness using microbiome data;
Eagle's Microbiome Platform accelerated the project timeline across global teams


SCIENTIFIC REPORTS

OPEN

A randomised clinical study to determine the effect of a toothpaste containing enzymes and proteins on plaque oral microbiome ecology

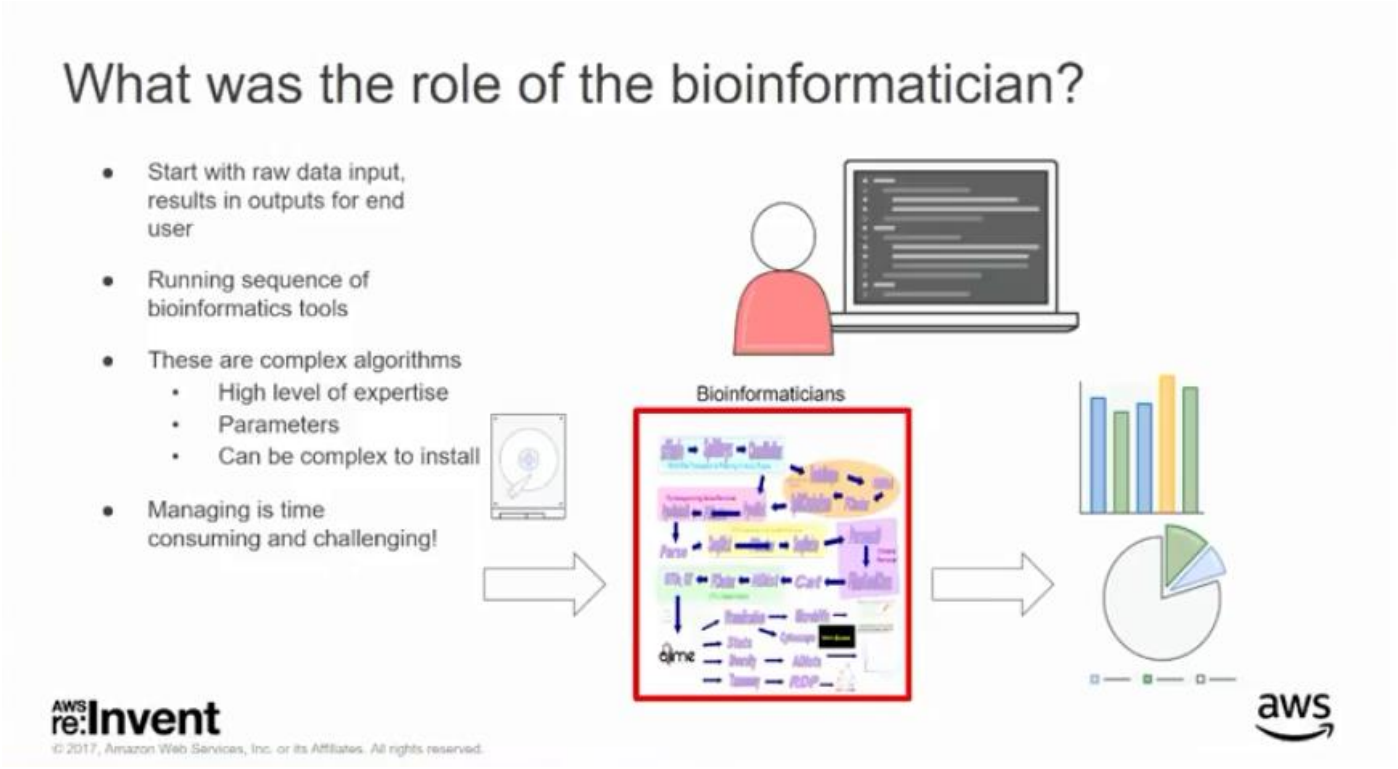
"Unilever's digital data program now processes genetic sequences twenty times faster - without incurring higher compute costs. In addition, its robust architecture supports ten times as many scientists, all working simultaneously."

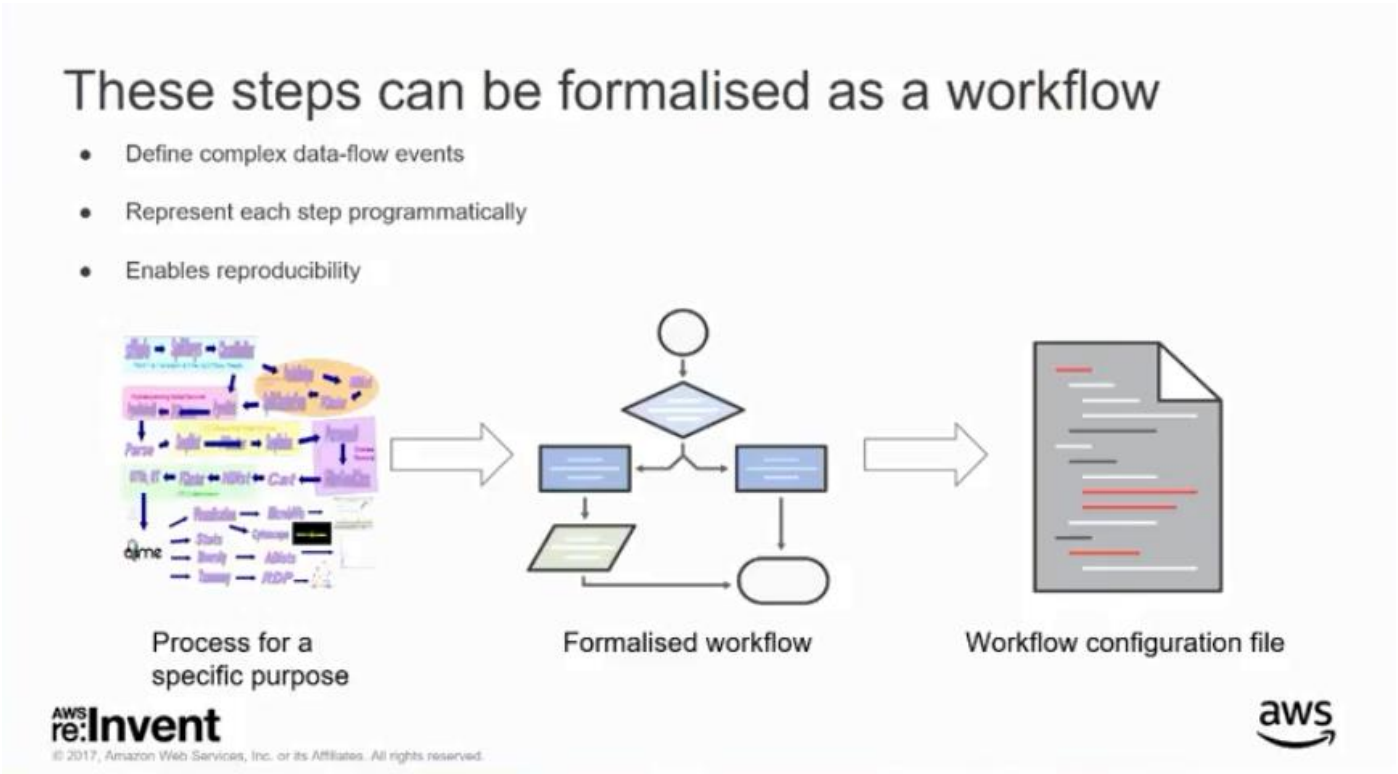
–Pete Keeley, eScience Technical Lead, R&D IT at Unilever

Background concepts

Workflows and workflow engines

Introducing the need for workflows and workflow engines in biology



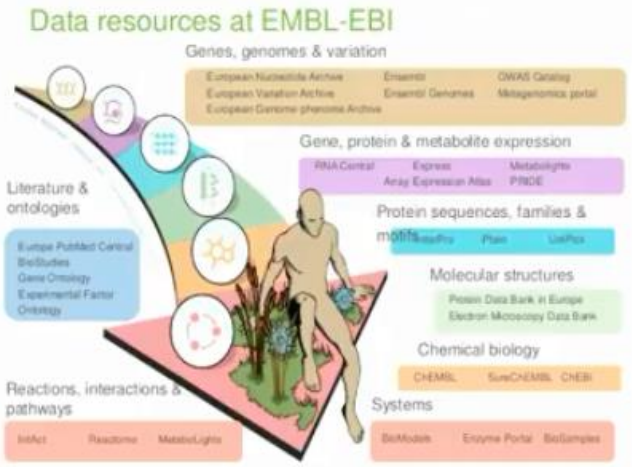


The Workflow configuration file allows us to represent the formalized workflow in a programmatic and repeatable way using automation via workflow engines.

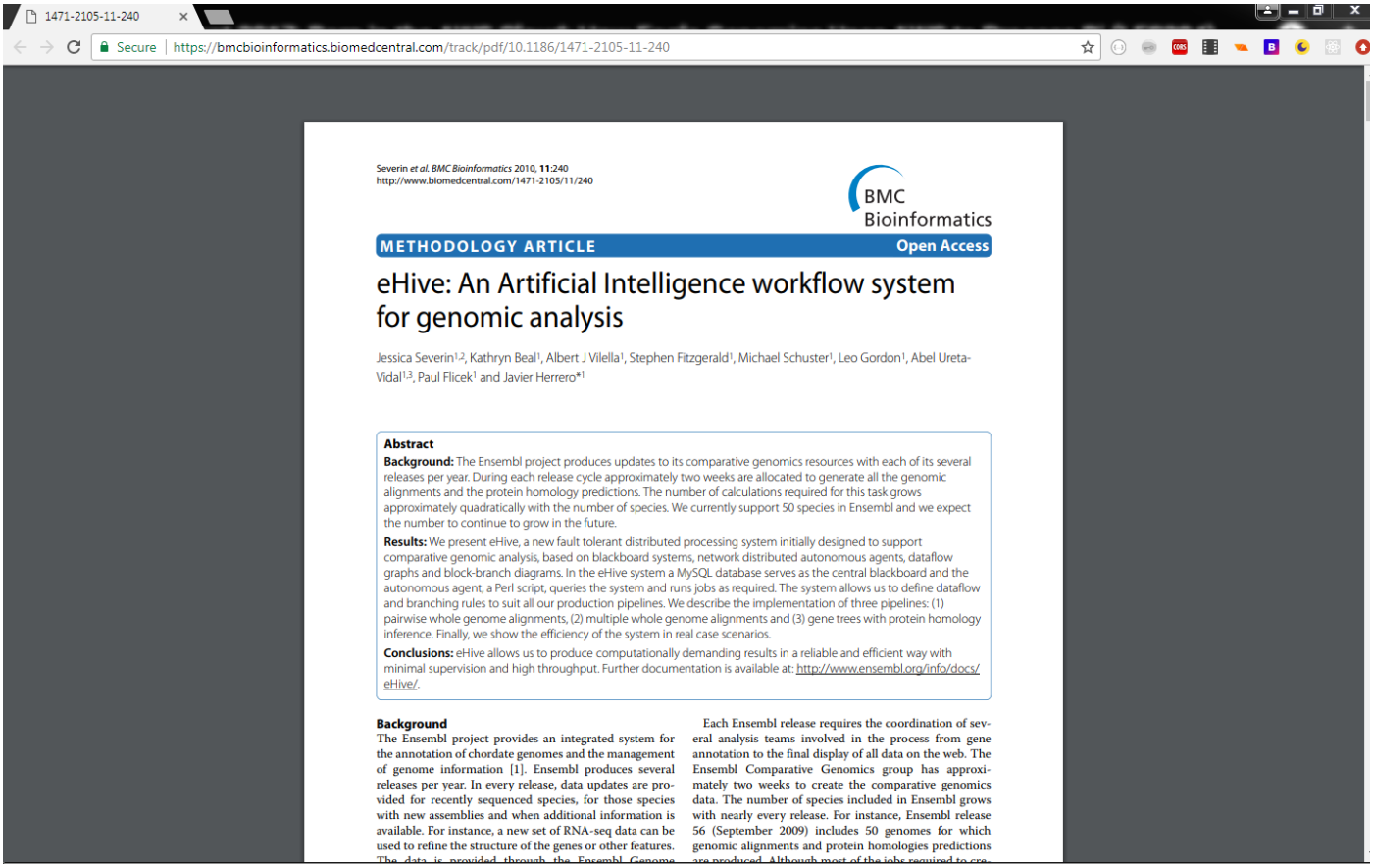
Workflow engines

- Need a way of running the workflow configuration file
- Lots of workflow engines Azkaban CloudSlang digdag OOZIE RUNDECK WORKFLOWENGINE AirFlow
- Key requirements:
  - Scalable
  - Reproducible
  - Traceable
  - Robust (e.g., handle failures)
  - Need to handle very complicated workflows

eHive has a track record in genomics workflows



- Since 2004 at the European Bioinformatics Institute (EBI)
- Proven scalability on big data sets:
  - Approximately 450 CPU years of compute every year
  - Some pipelines reach > 6,000 of concurrent workers and manage millions of jobs
- Tried and tested with complex data sets and analysis



1471-2105-11-240

Secure | https://bmcbioinformatics.biomedcentral.com/track/pdf/10.1186/1471-2105-11-240

Severin *et al. BMC Bioinformatics* 2010, **11**:240
http://www.biomedcentral.com/1471-2105/11/240

BMC Bioinformatics

METHODOLOGY ARTICLE Open Access

eHive: An Artificial Intelligence workflow system for genomic analysis

Jessica Severin[1,2], Kathryn Beal[1], Albert J Vilella[1], Stephen Fitzgerald[1], Michael Schuster[1], Leo Gordon[1], Abel Ureta-Vidal[1,3], Paul Flicek[1] and Javier Herrero*[1]

Abstract

Background: The Ensembl project produces updates to its comparative genomics resources with each of its several releases per year. During each release cycle approximately two weeks are allocated to generate all the genomic alignments and the protein homology predictions. The number of calculations required for this task grows approximately quadratically with the number of species. We currently support 50 species in Ensembl and we expect the number to continue to grow in the future.

Results: We present eHive, a new fault tolerant distributed processing system initially designed to support comparative genomic analysis, based on blackboard systems, network distributed autonomous agents, dataflow graphs and block-branch diagrams. In the eHive system a MySQL database serves as the central blackboard and the autonomous agent, a Perl script, queries the system and runs jobs as required. The system allows us to define dataflow and branching rules to suit all our production pipelines. We describe the implementation of three pipelines: (1) pairwise whole genome alignments, (2) multiple whole genome alignments and (3) gene trees with protein homology inference. Finally, we show the efficiency of the system in real case scenarios.

Conclusions: eHive allows us to produce computationally demanding results in a reliable and efficient way with minimal supervision and high throughput. Further documentation is available at: http://www.ensembl.org/info/docs/eHive/.

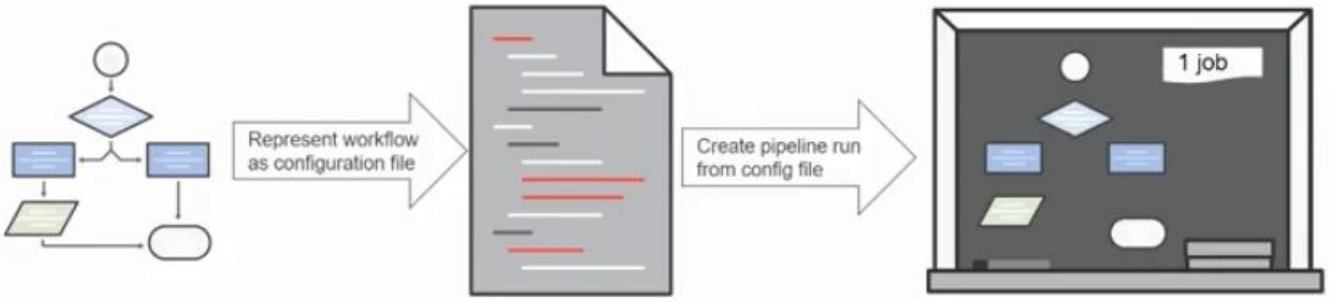
Background

The Ensembl project provides an integrated system for the annotation of chordate genomes and the management of genome information [1]. Ensembl produces several releases per year. In every release, data updates are provided for recently sequenced species, for those species with new assemblies and when additional information is available. For instance, a new set of RNA-seq data can be used to refine the structure of the genes or other features. The data is provided through the Ensembl Genome

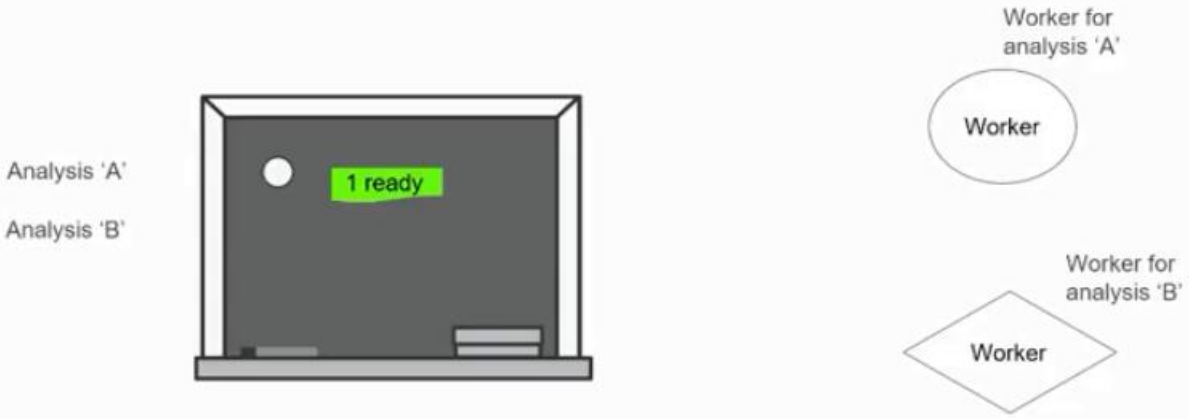
Each Ensembl release requires the coordination of several analysis teams involved in the process from gene annotation to the final display of all data on the web. The Ensembl Comparative Genomics group has approximately two weeks to create the comparative genomics data. The number of species included in Ensembl grows with nearly every release. For instance, Ensembl release 56 (September 2009) includes 50 genomes for which genomic alignments and protein homologies predictions are produced. Although most of the jobs required to cre-

eHive uses a blackboard system

- Workflow analysis steps described by config file
- During workflow run jobs are updated in a central blackboard database
- Represents flow structure and job list

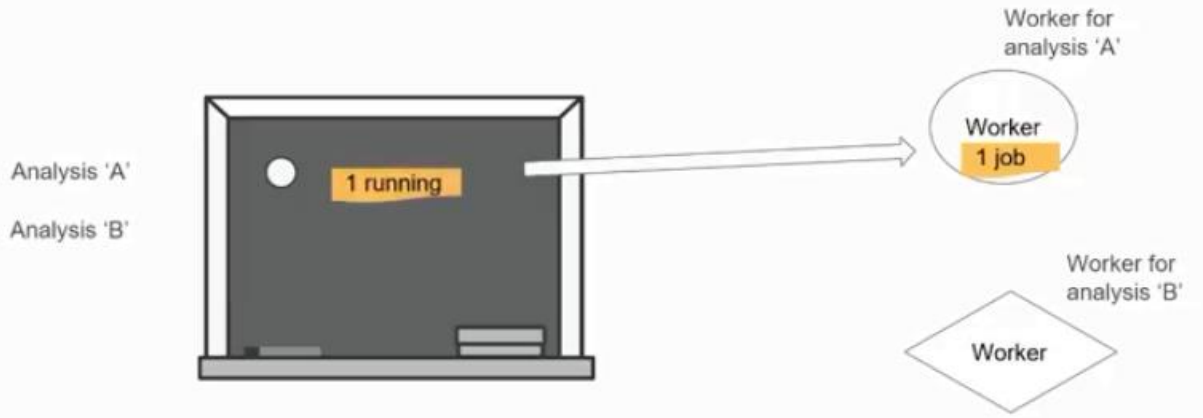


Jobs are processed by autonomous workers



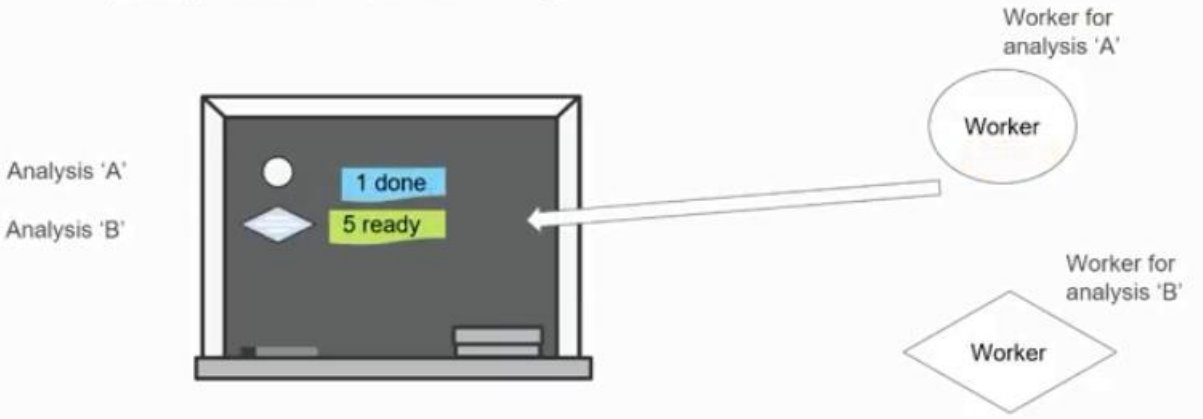
Jobs are processed by autonomous workers

- The worker queries the blackboard for jobs of a type that it can run
- Workers process jobs that match the analysis type



Jobs are processed by autonomous workers

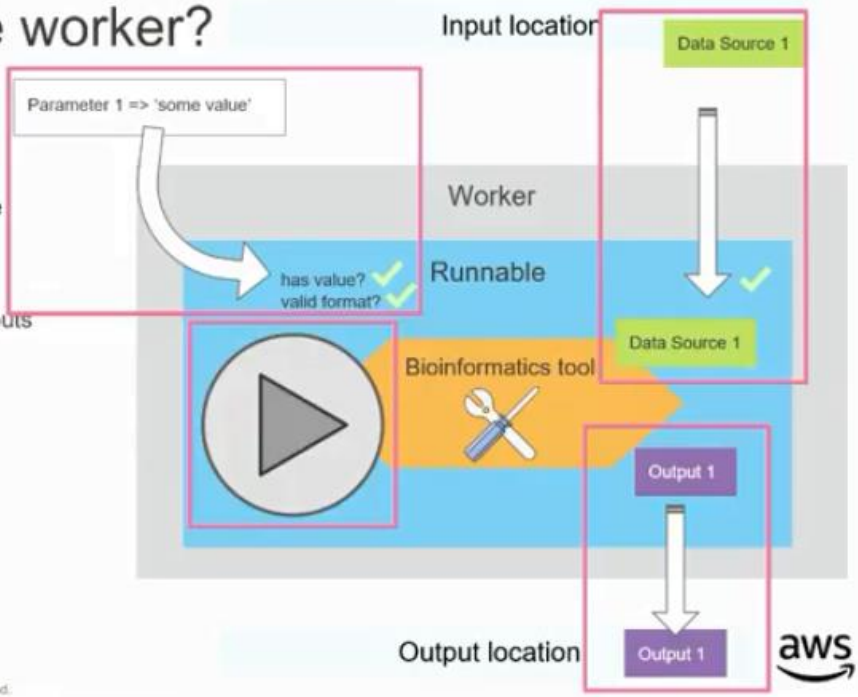
- The worker queries the blackboard for jobs of a type that it can run
- Workers process jobs that match the analysis type
- When a job completes it can create 1 or more new jobs



The job creation is a dynamic process in the Hive and this makes the process scalable

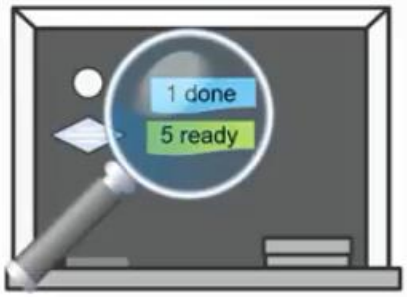
What is an eHive worker?

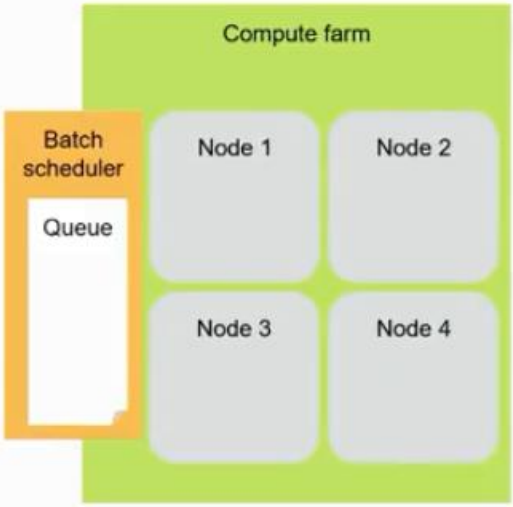
- Bioinformatics tool
- 'Runnable'—wrapper around the bioinformatics tool
  - Handling parameters
  - Fetching and validating inputs
  - Running the tool
  - Handling outputs
  - Logging



eHive runs workers using a batch scheduler

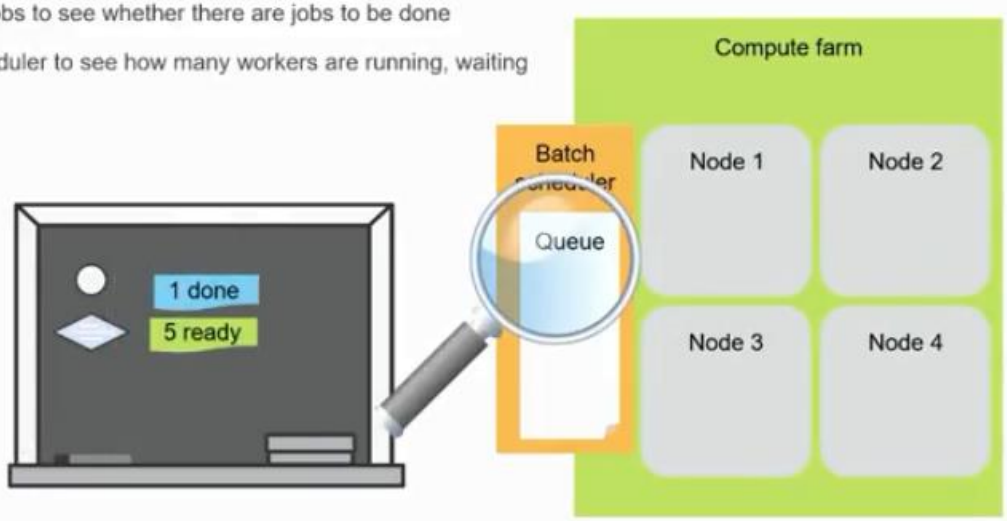
- eHive monitors jobs to see whether there are jobs to be done





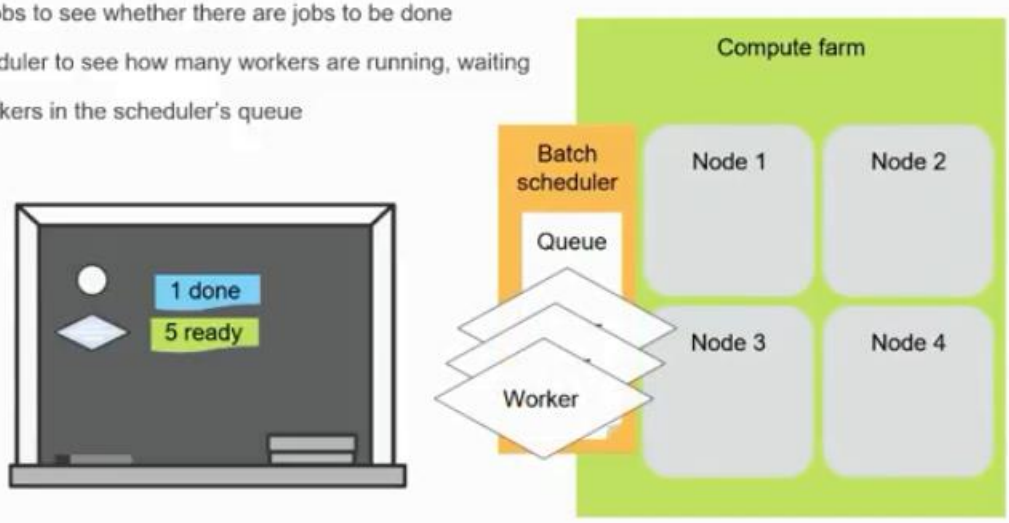
eHive runs workers using a batch scheduler

- eHive monitors jobs to see whether there are jobs to be done
- Queries job scheduler to see how many workers are running, waiting



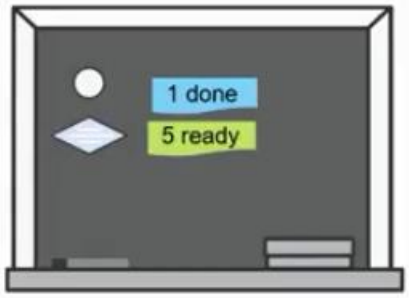
eHive runs workers using a batch scheduler

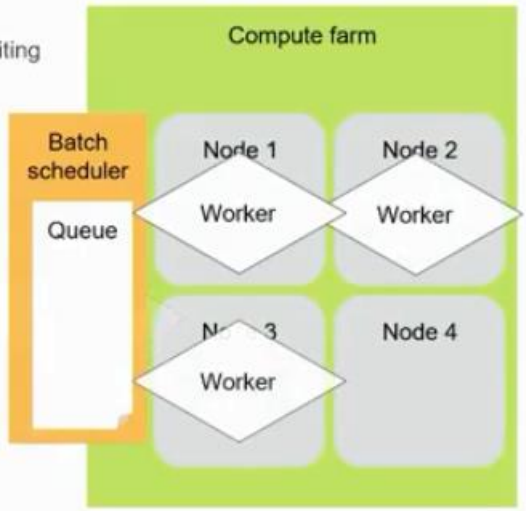
- eHive monitors jobs to see whether there are jobs to be done
- Queries job scheduler to see how many workers are running, waiting
- eHive places workers in the scheduler's queue



eHive runs workers using a batch scheduler

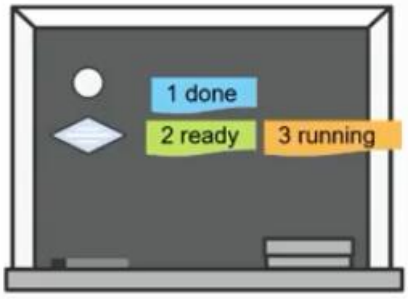
- eHive monitors jobs to see whether there are jobs to be done
- Queries job scheduler to see how many workers are running, waiting
- eHive places workers in the scheduler's queue
- The scheduler finds available compute to run the worker

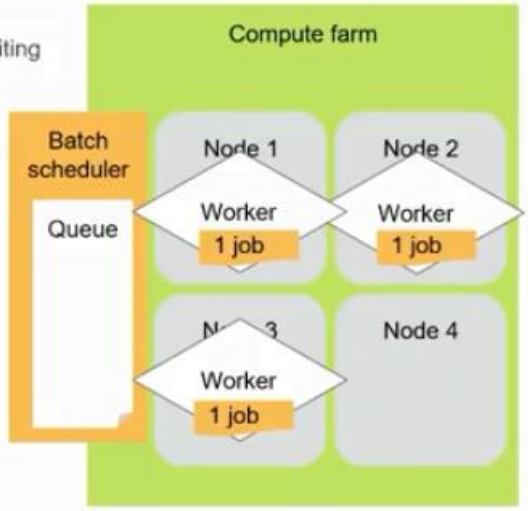




eHive runs workers using a batch scheduler

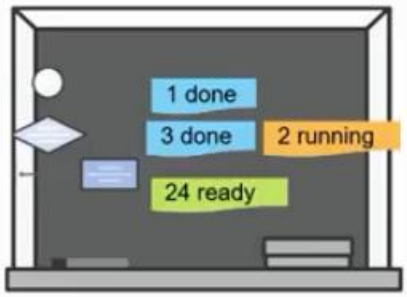
- eHive monitors jobs to see whether there are jobs to be done
- Queries job scheduler to see how many workers are running, waiting
- eHive places workers in the scheduler's queue
- The scheduler finds available compute to run the worker

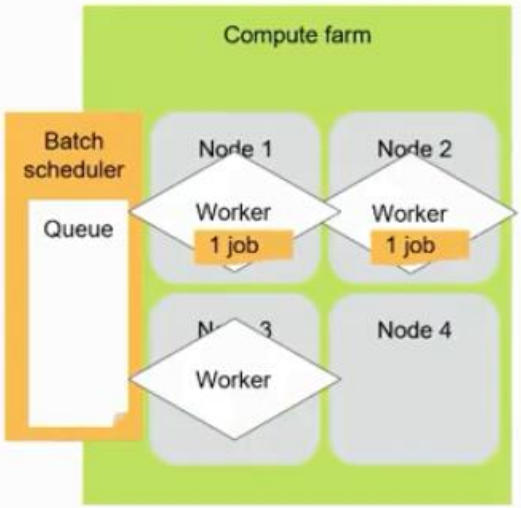




AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

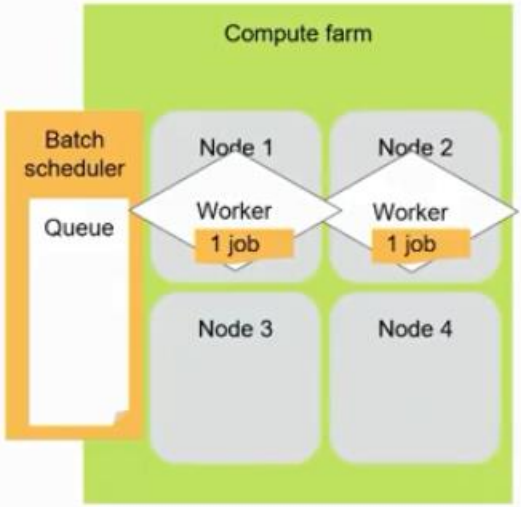
Any number and type of workers can run together





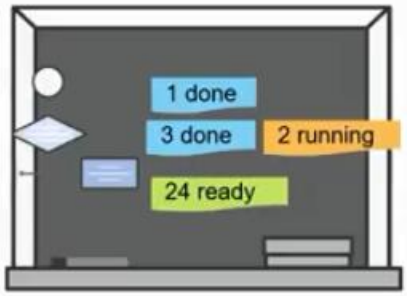
Any number and type of workers can run together

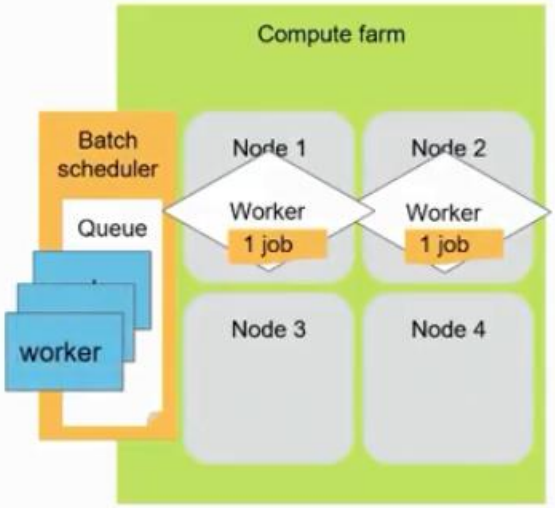
- Workers exit if they have no more jobs



Any number and type of workers can run together

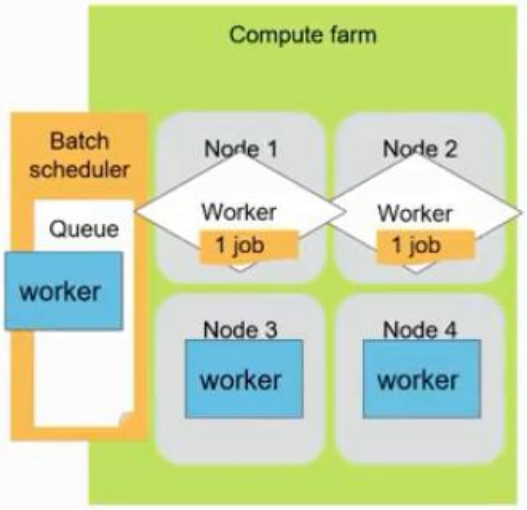
- Workers exit if they have no more jobs
- More workers of the correct type are requested





Any number and type of workers can run together

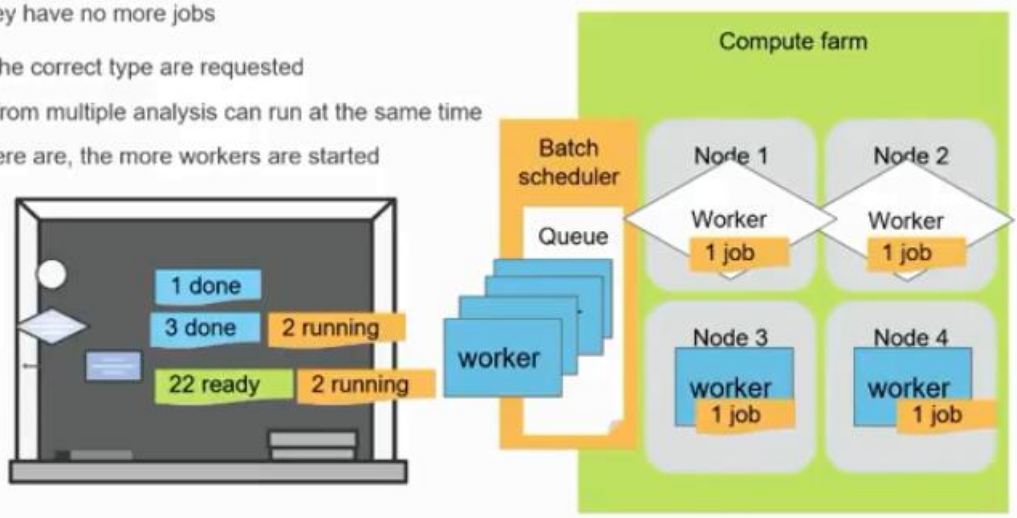
- Workers exit if they have no more jobs
- More workers of the correct type are requested
- Multiple workers from multiple analysis can run at the same time



aws re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Any number and type of workers can run together

- Workers exit if they have no more jobs
- More workers of the correct type are requested
- Multiple workers from multiple analysis can run at the same time
- The more jobs there are, the more workers are started



Implementation approach

From eHive to **e[**hive**]**

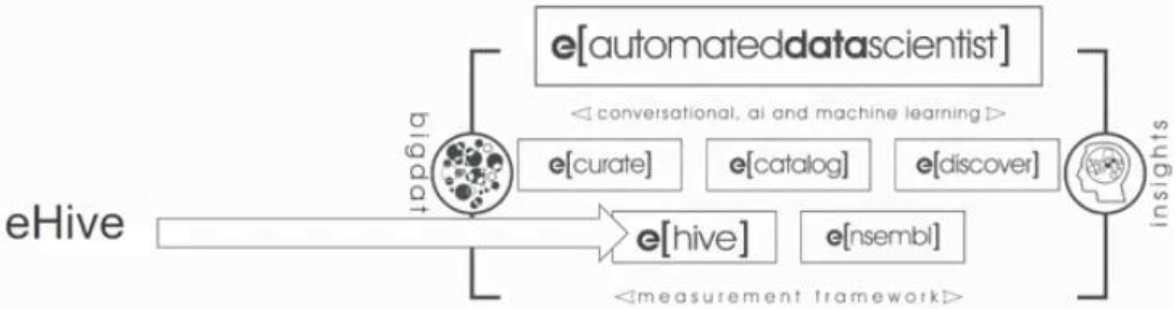
Taking eHive to the AWS cloud...and back again

Making **e[**hive**]** a part of our platform

eHive → **e[**hive**]**

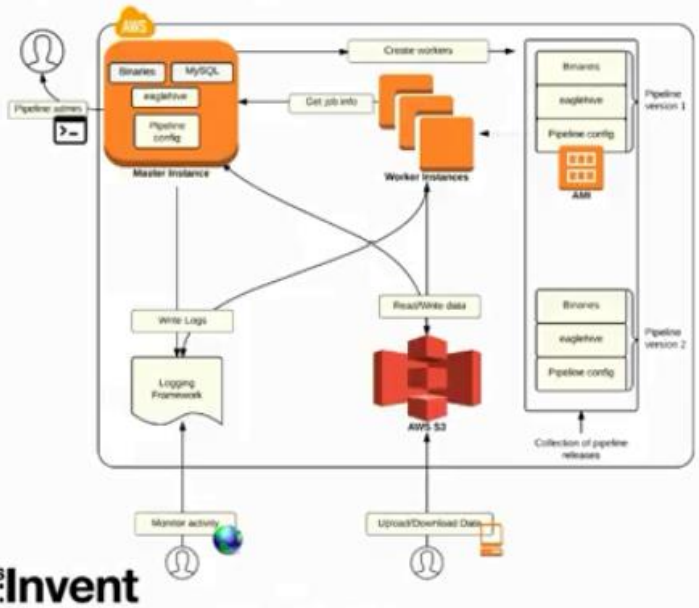
- Designed to work on site at EBI (LSF)
- Requires technical user
- Want to run in cloud environments
- Enable scientists to run pipeline without knowledge of **e[**hive**]**

Making e[hive] a part of our platform



- Designed to work on site at EBI (LSF)
- Requires technical user
- Want to run in cloud environments
- Enable scientists to run pipeline without knowledge of e[hive]
- Integrate with platform providing analytics capability

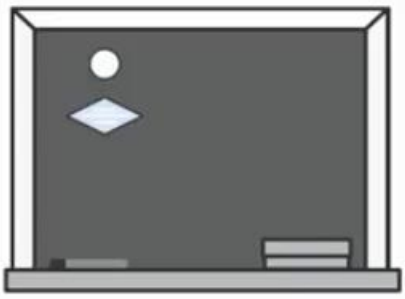
e[hive] v1 (2008)—Workers as AWS instances

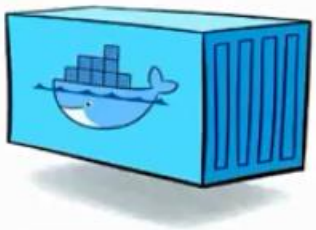


- Each worker is an AWS instance
- Many production workflows implemented in AWS!
- Poor resource usage
  - Workflow steps specify instance type
  - Resource bottleneck
  - Same instance for multiple analysis
- Whole workflow on AMI
  - Slows development
  - Difficult to test analysis
  - Poor reusability
- Automated dependency installation using Chef

e[hive] v2 (2015) - Workers as Docker containers?

- Could e[hive] workers be made into a docker image?
- How can e[hive] manage scheduling workers across a cluster?





Worker as Docker Container

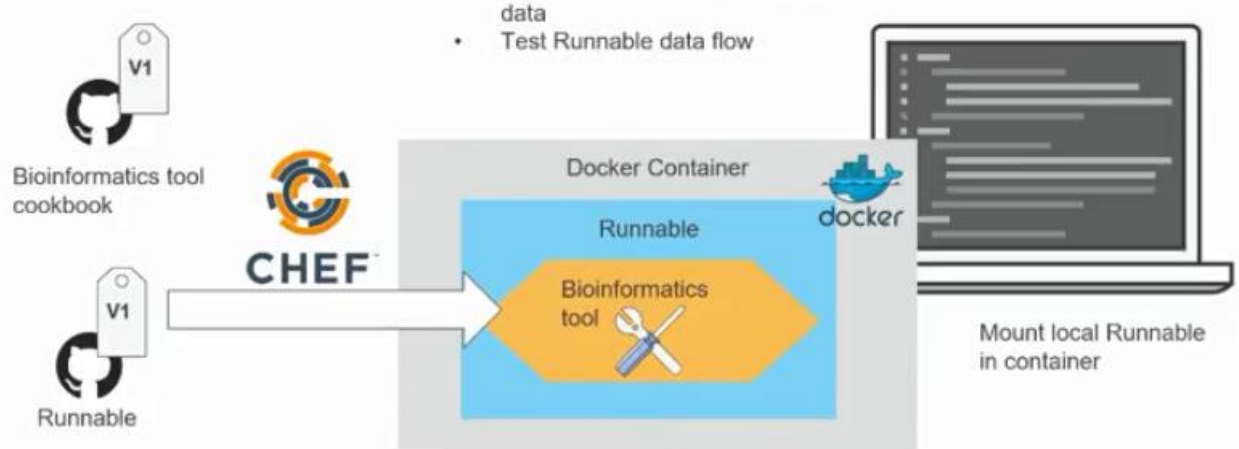
From eHive to e[hive]: Stage 1

Dockerising the e[hive] worker

Building tested, versioned, and reusable analysis components with Docker

Building the e[hive] worker with Docker

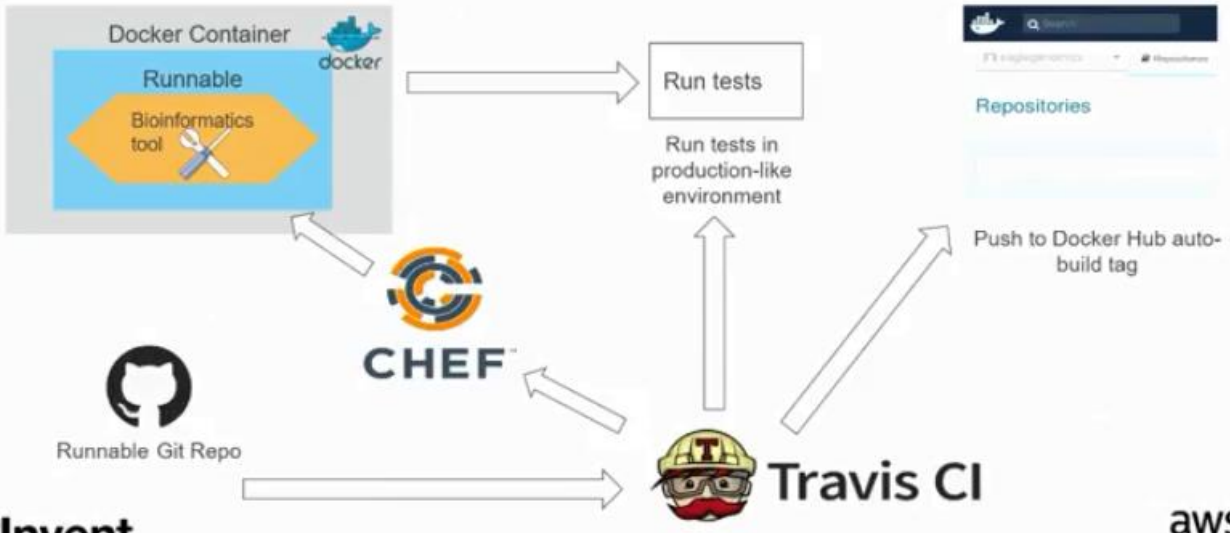
- Each Runnable has tests:
  - Run bioinformatics tool with test data
  - Test Runnable data flow



e[hive] worker Docker container

e[hive] worker development cycle

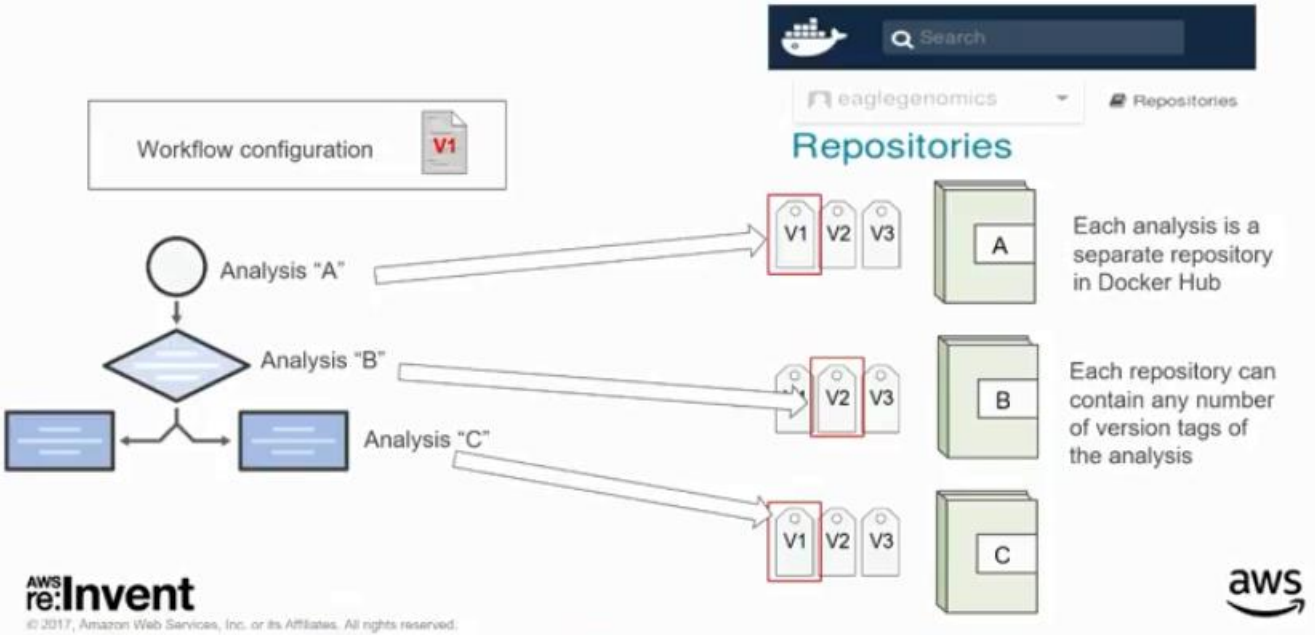
- Runnable tests are run from CI tool on each commit and weekly



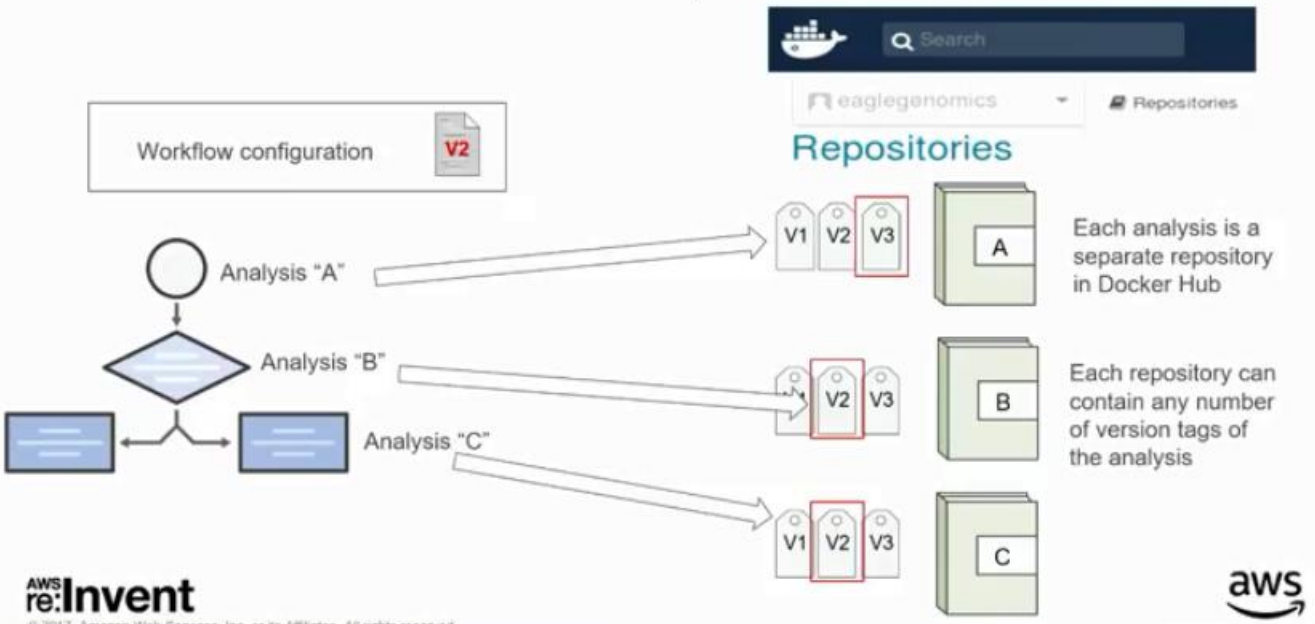
Some key questions about building Docker images

Question	Reason	Solution(s)
How was the image built? What is on the image?	We shouldn't trust something if we don't know how it is built	Dockerfile, configuration management
Does the build still work now?	Or are we stuck with a 'golden image' we can't reproduce anymore?	Build regularly, e.g., via continuous integration tool
Does the tool work? How do we test the image?	This is essential! Build may 'succeed' but not be a working install!	Continuous integration tool to run tests
Can we redeploy elsewhere? Or are we tied to Docker? Is this a problem?	In some cases we can't use Docker (e.g., on an HPC)	Need a suitable build method that takes this into account

The workflow and each step are versioned



The workflow and each step are versioned



Versioning is critical

- Reproducibility in scientific workflows is essential
  - Replicate other people's work
  - Necessary for quality control, compliance, etc.
  - Requires clear instructions
  - Versioning at many levels is key for this

- Pipeline run report for end user
  - Workflow description
  - Each tool and version

Workers log to a centralised logging server

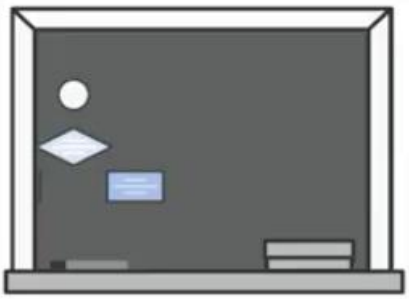
- Workers log to Amazon Elasticsearch Service (Amazon ES) via FluentD—jobs execute on different hosts, but we see all the logs together
- Visualise and explore in Kibana
- Can quickly see any errors even with 100 1000s jobs

From eHive to e[hive]: Stage 2

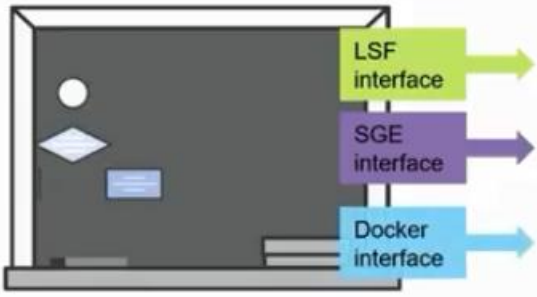
Building the Docker Swarm Orchestrator

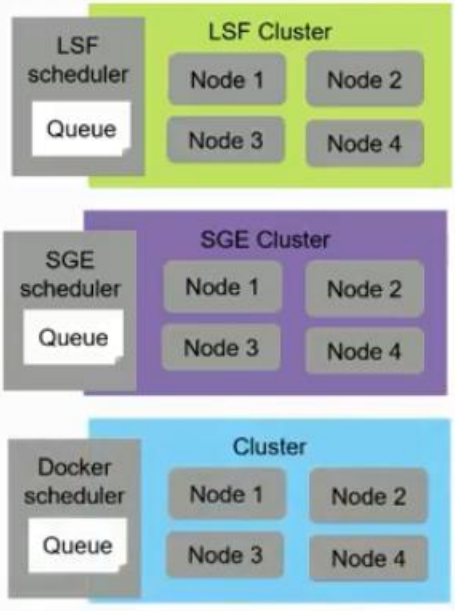
Creating an auto scaling container scheduler for **e[hive]**

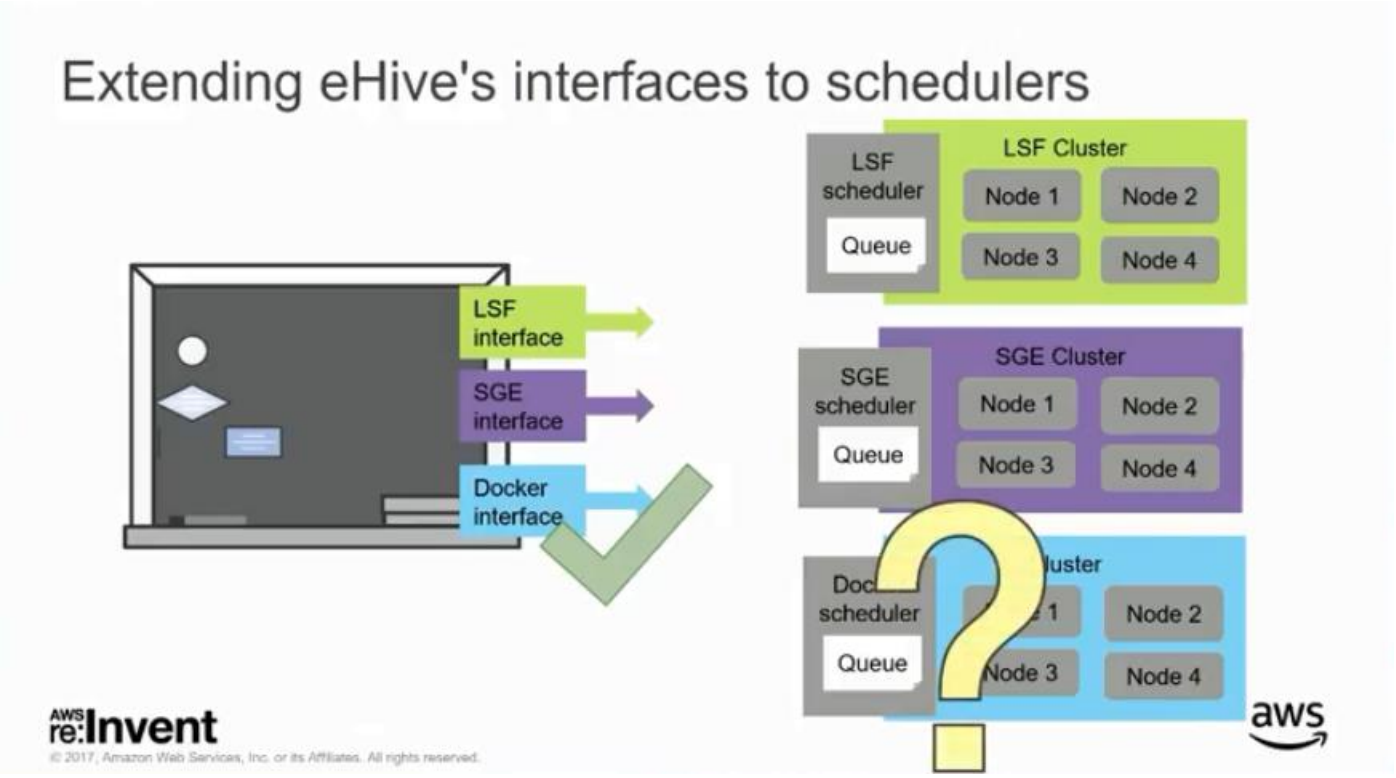
Extending eHive's interfaces to schedulers

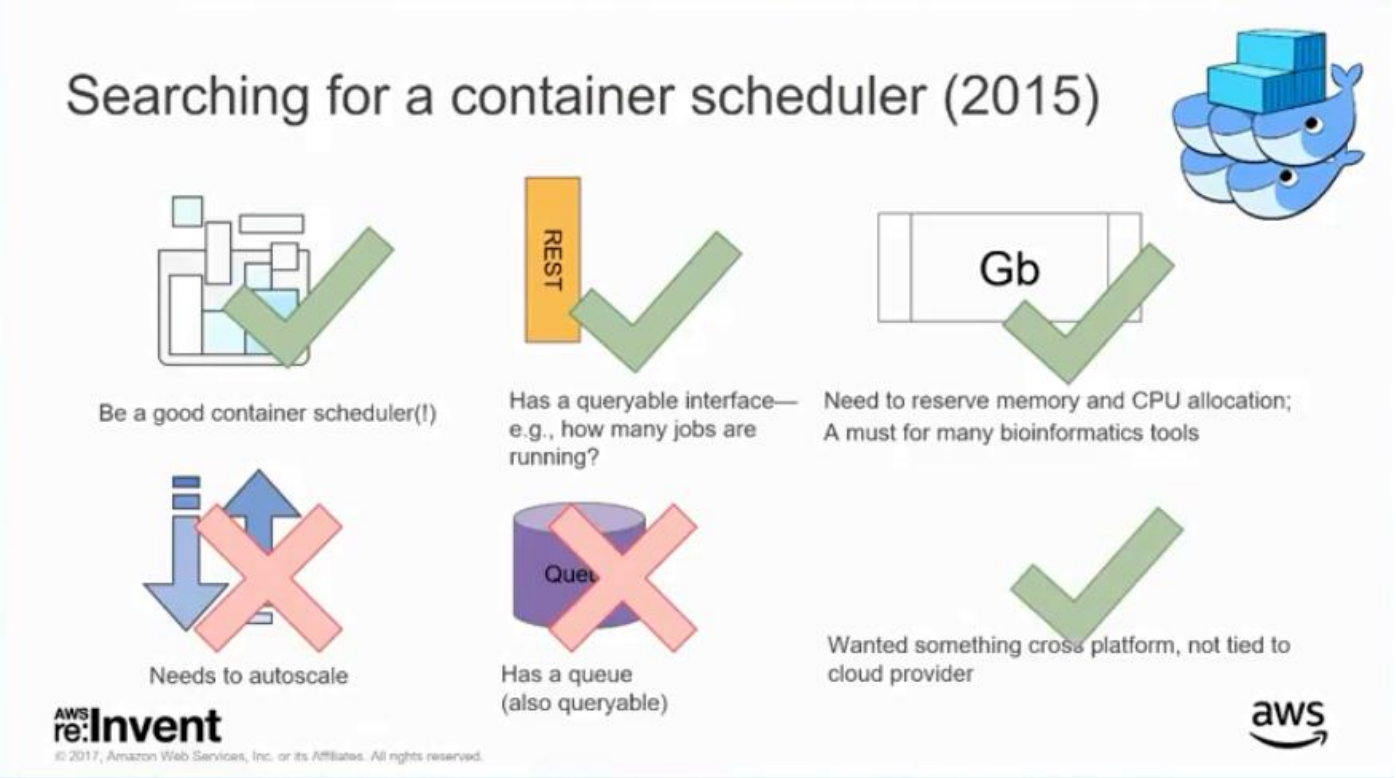


Extending eHive's interfaces to schedulers

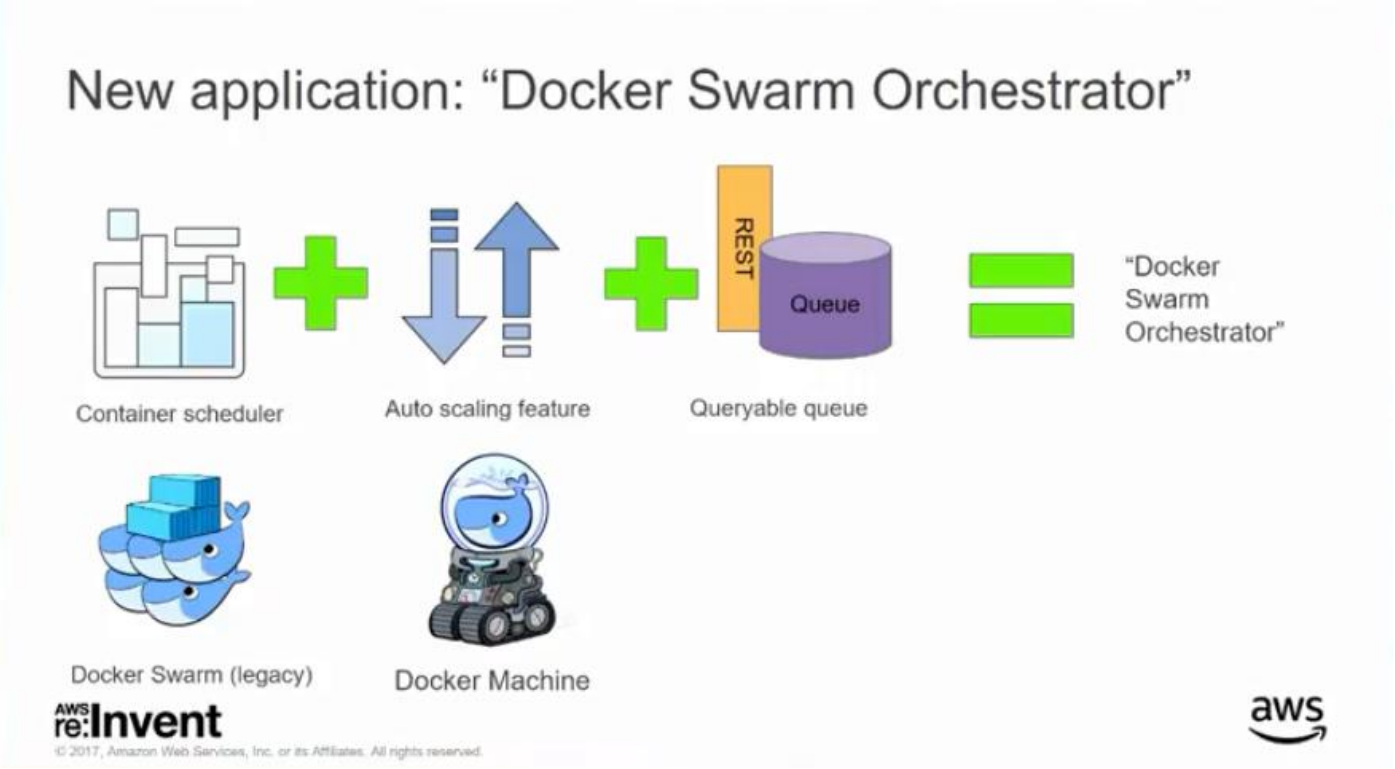




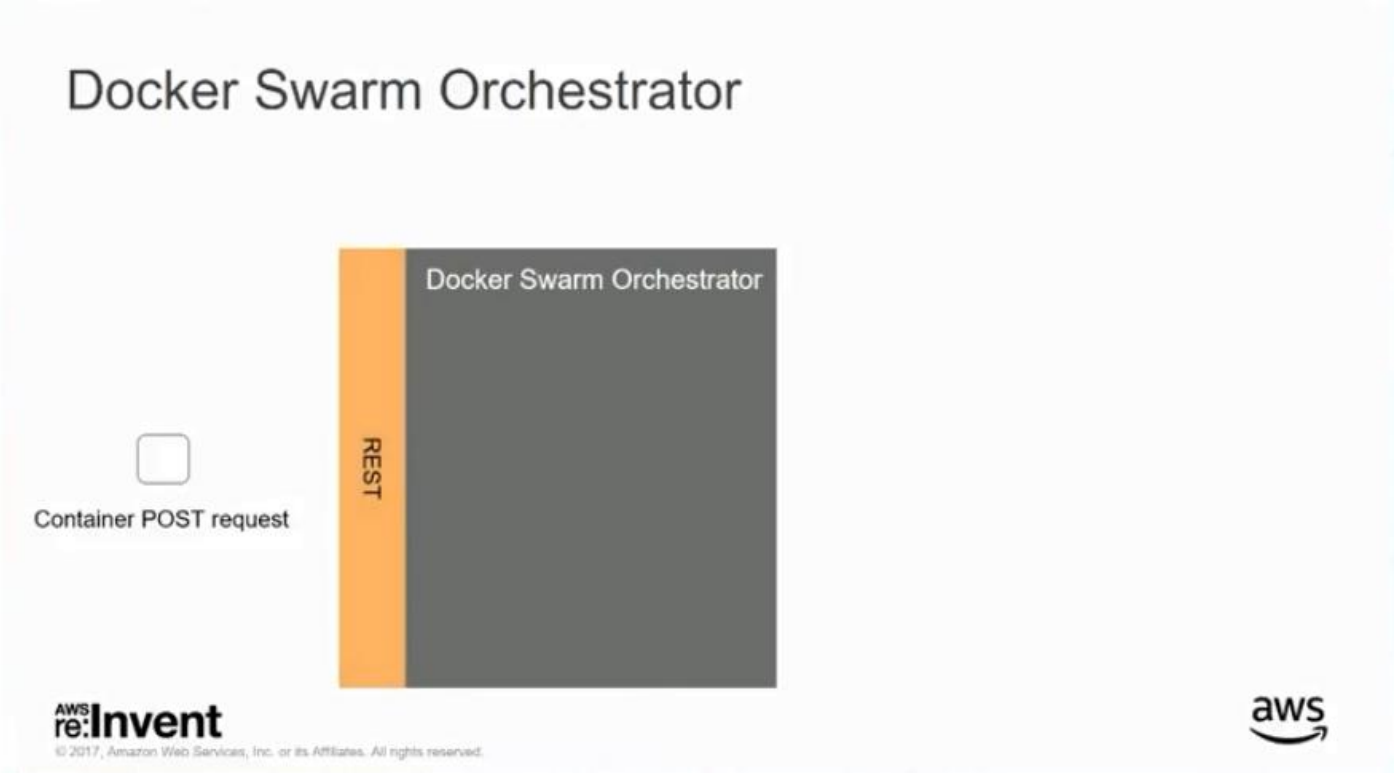




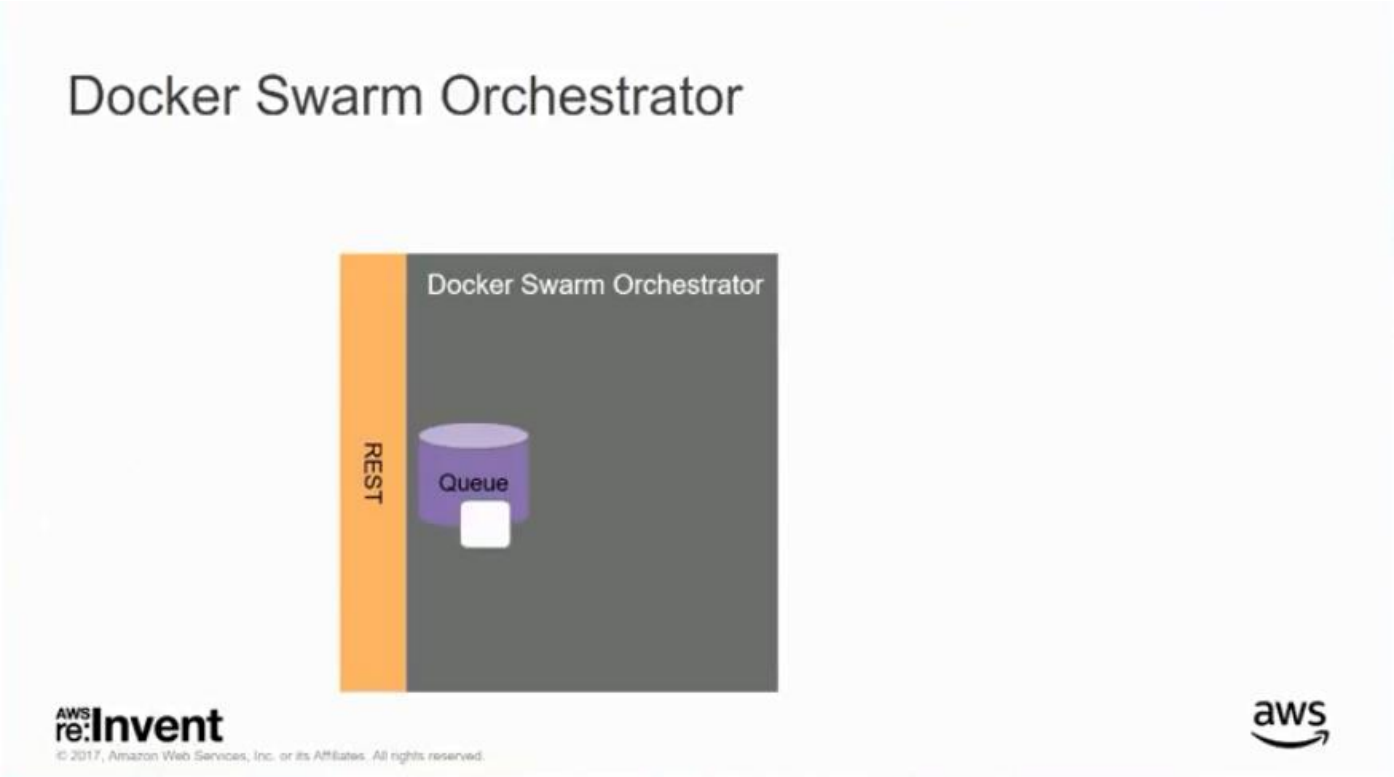
We need the above requirements fulfilled when eHive creates/schedules its jobs for the workers to pick up and run. This meant that we had to build our own tool that will have all features above.



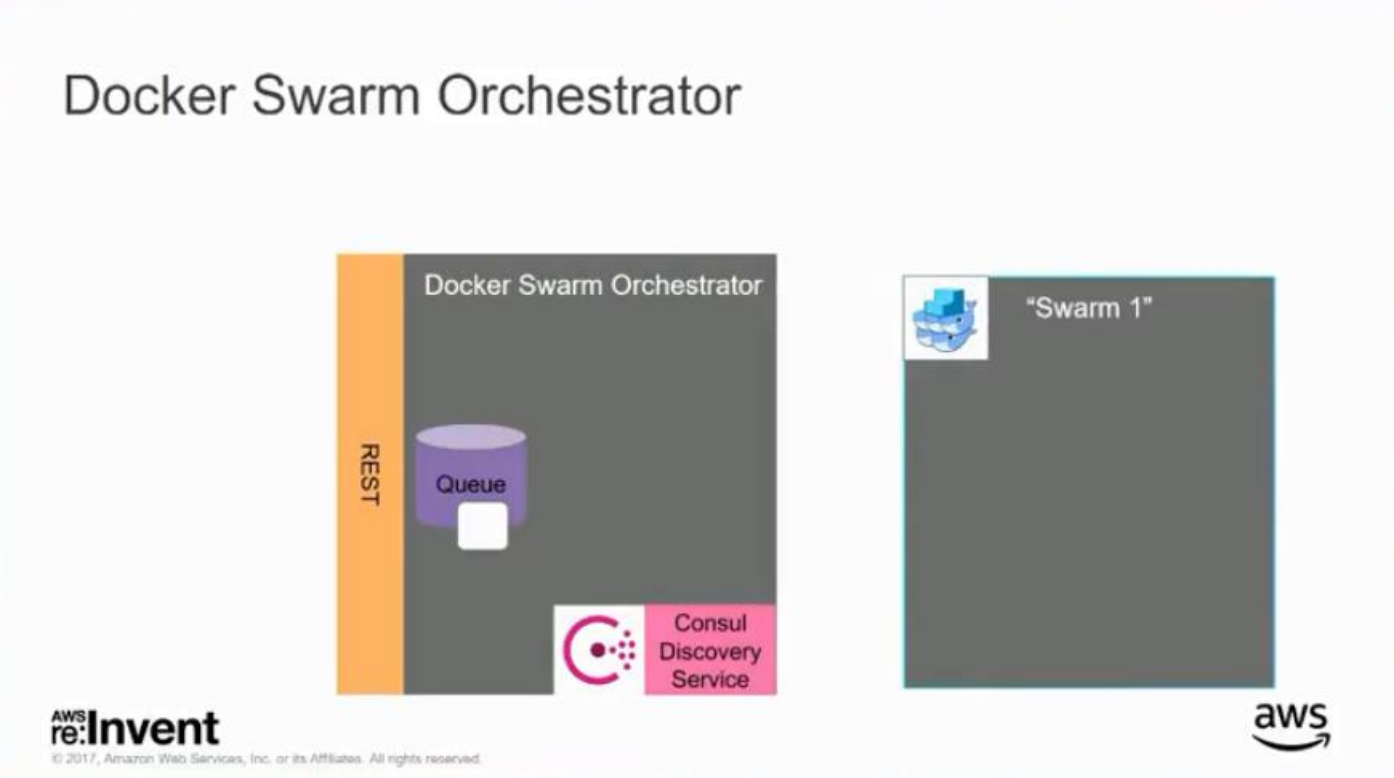
We used Docker Swarm as our orchestrator, Docker Machine API for autoscaling the swarm to create or delete instances depending on available jobs in the queue, and we used RabbitMQ for the queue.



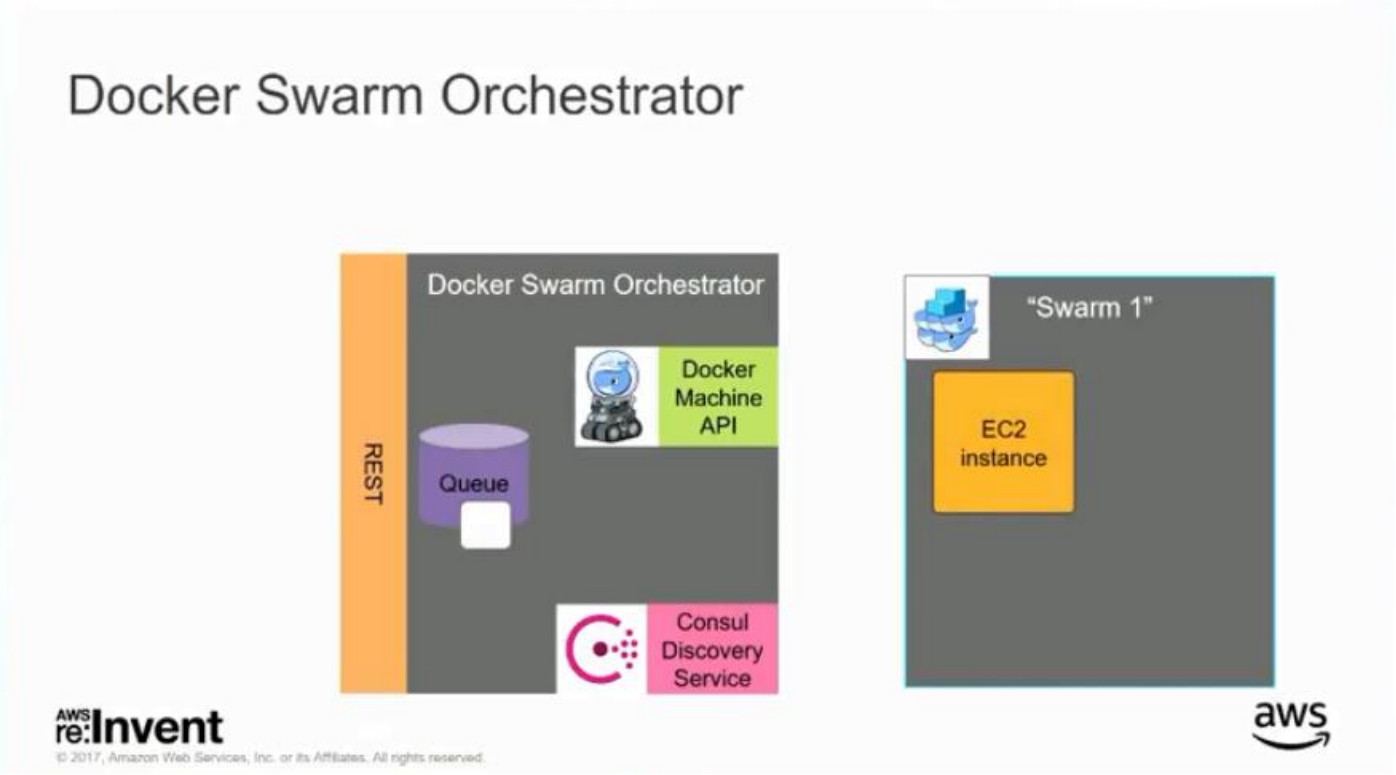
When we make a container POST request to the REST interface of the Docker Swarm Orchestrator DSO,



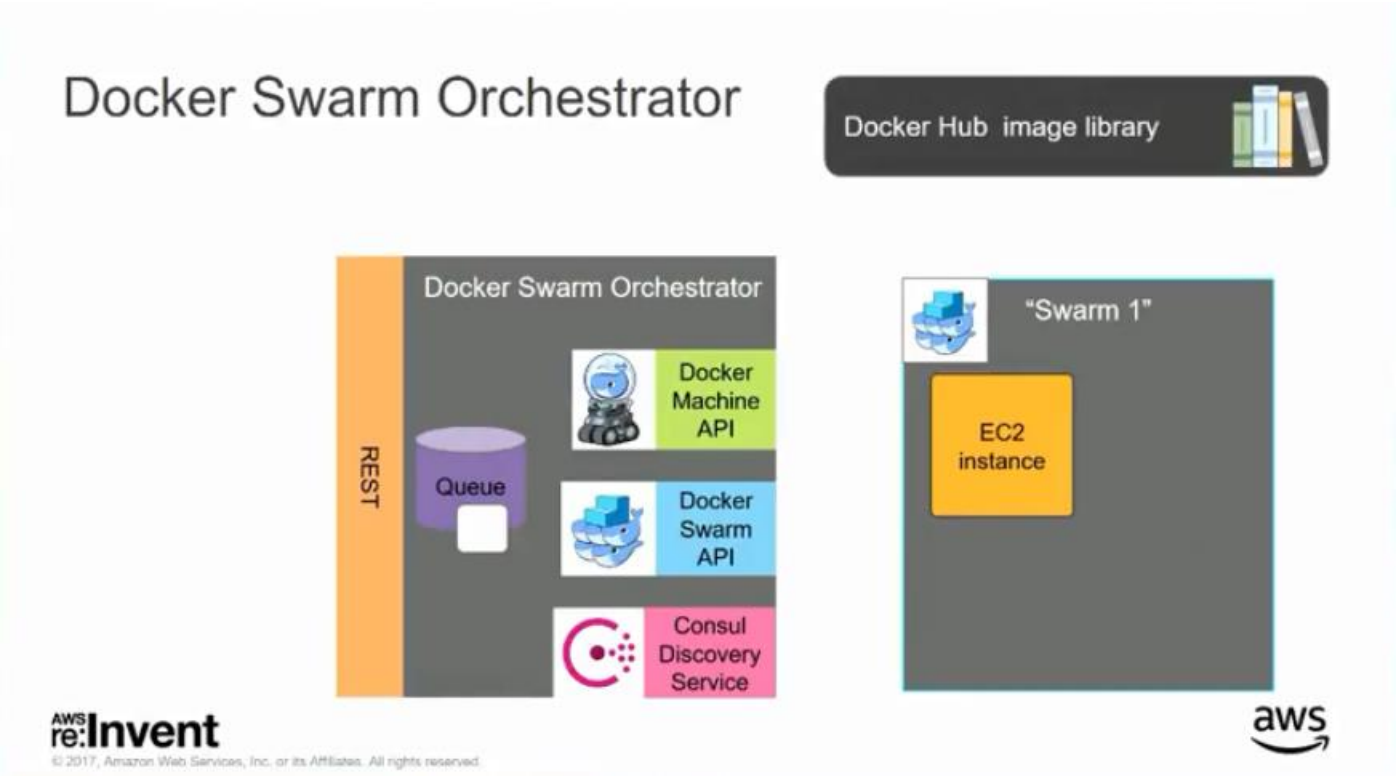
The DSO then puts the request into the queue



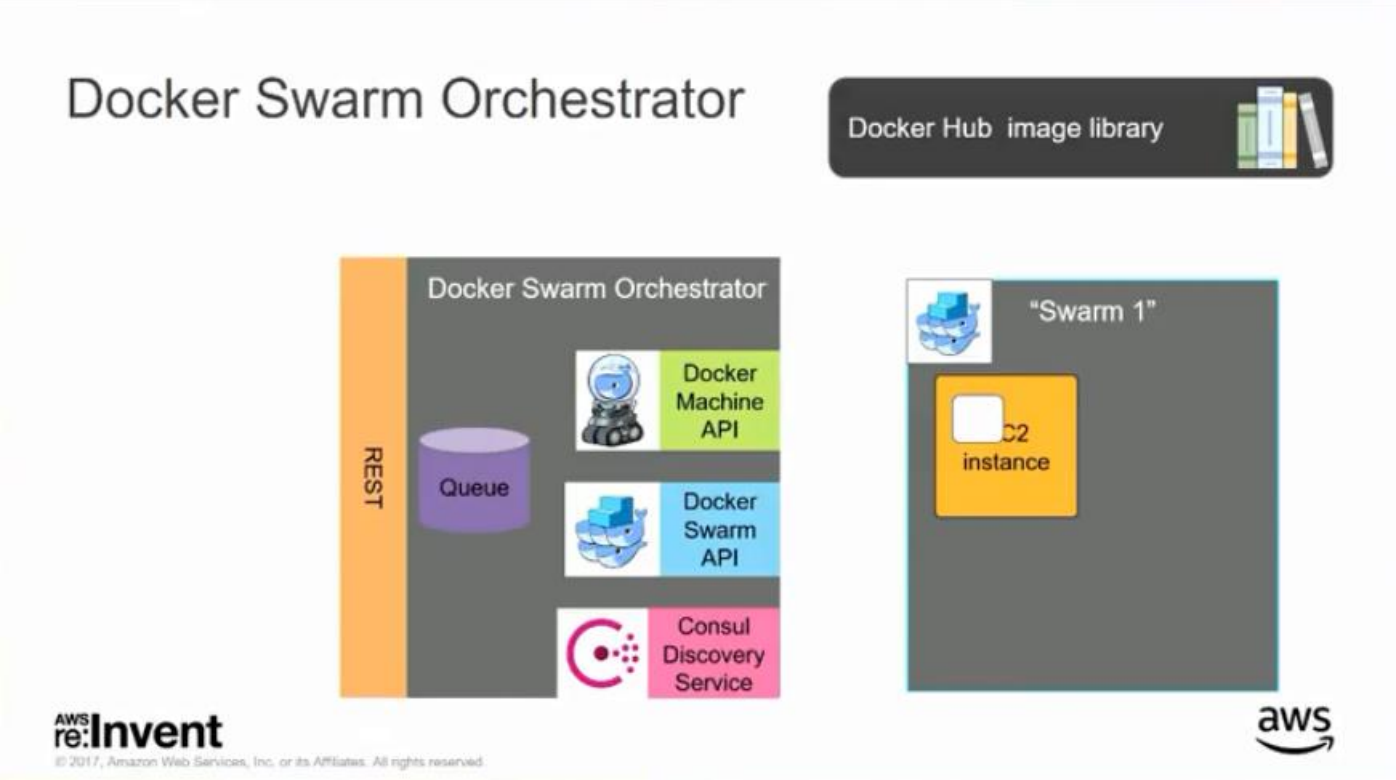
We then create a new Docker swarm that uses a discovery service like Consul to register this new swarm



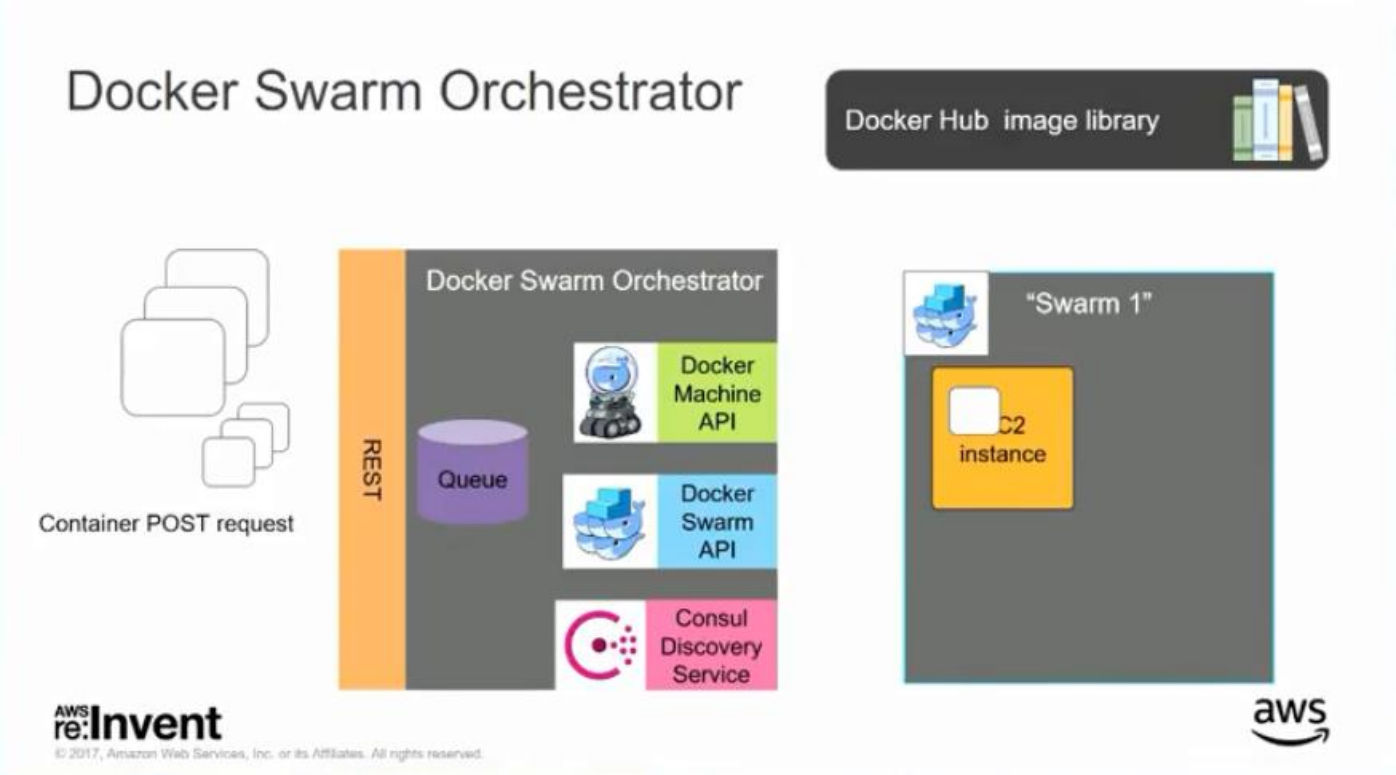
We then use the Docker Machine API to add instances into the swarm when we have jobs in the queue



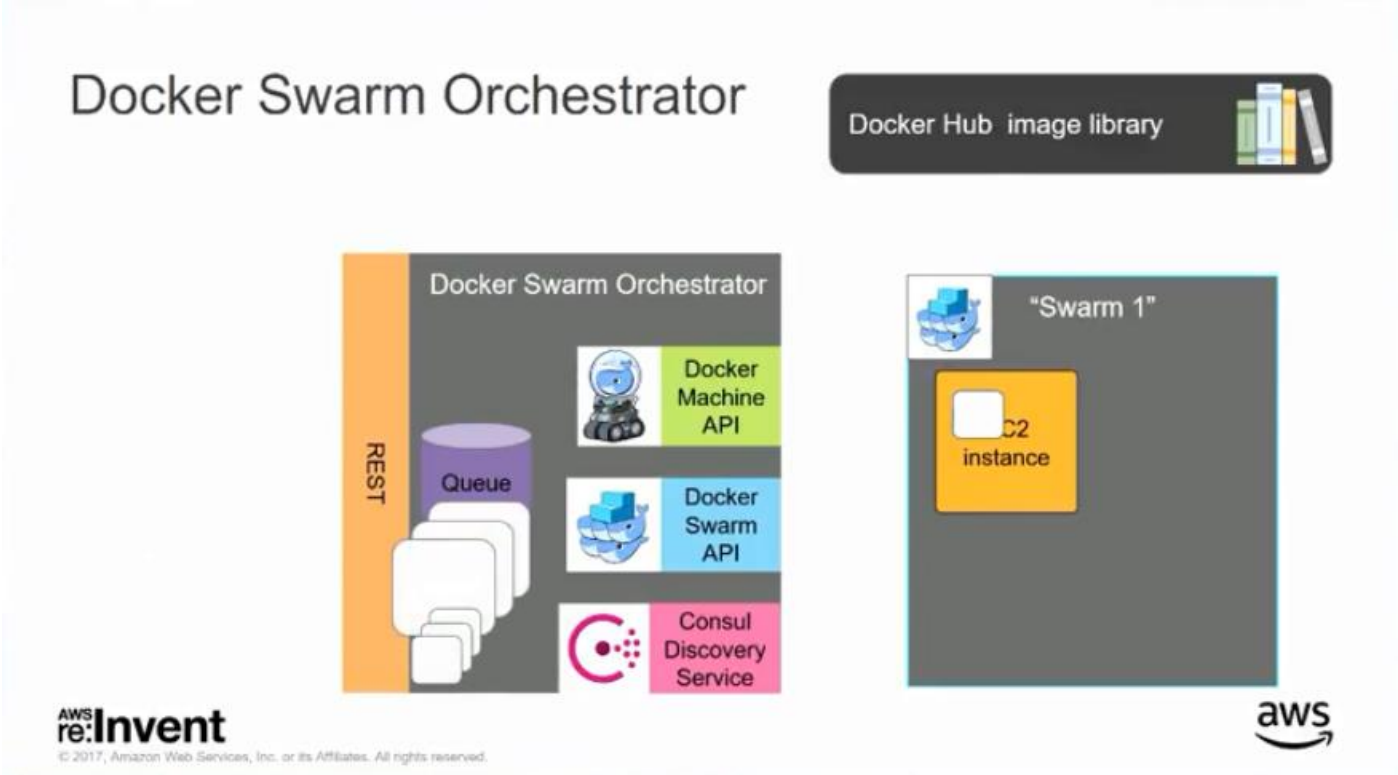
The instance is now registered in Docker Swarm and Docker Swarm can then pull the needed image to run from Docker Hub



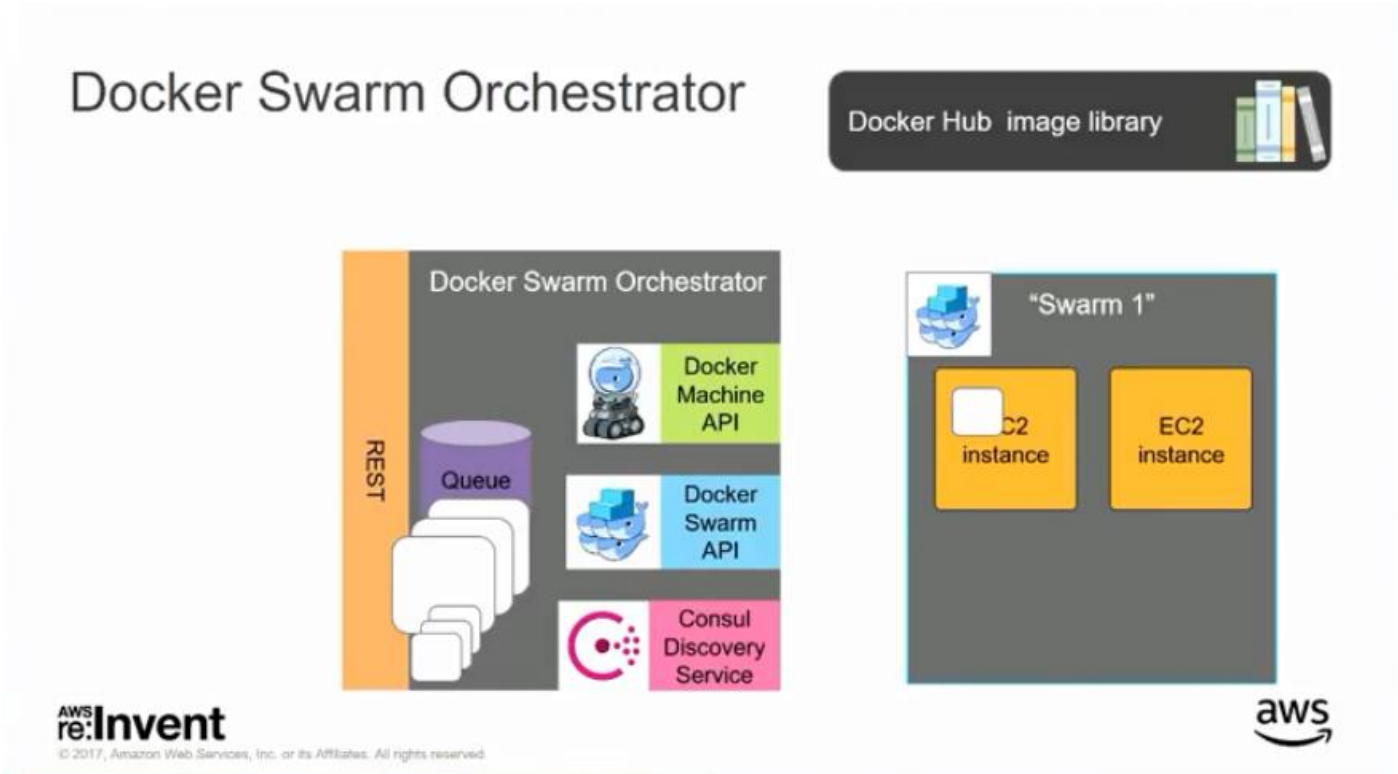
It then pulls the job details and starts running the job in the container

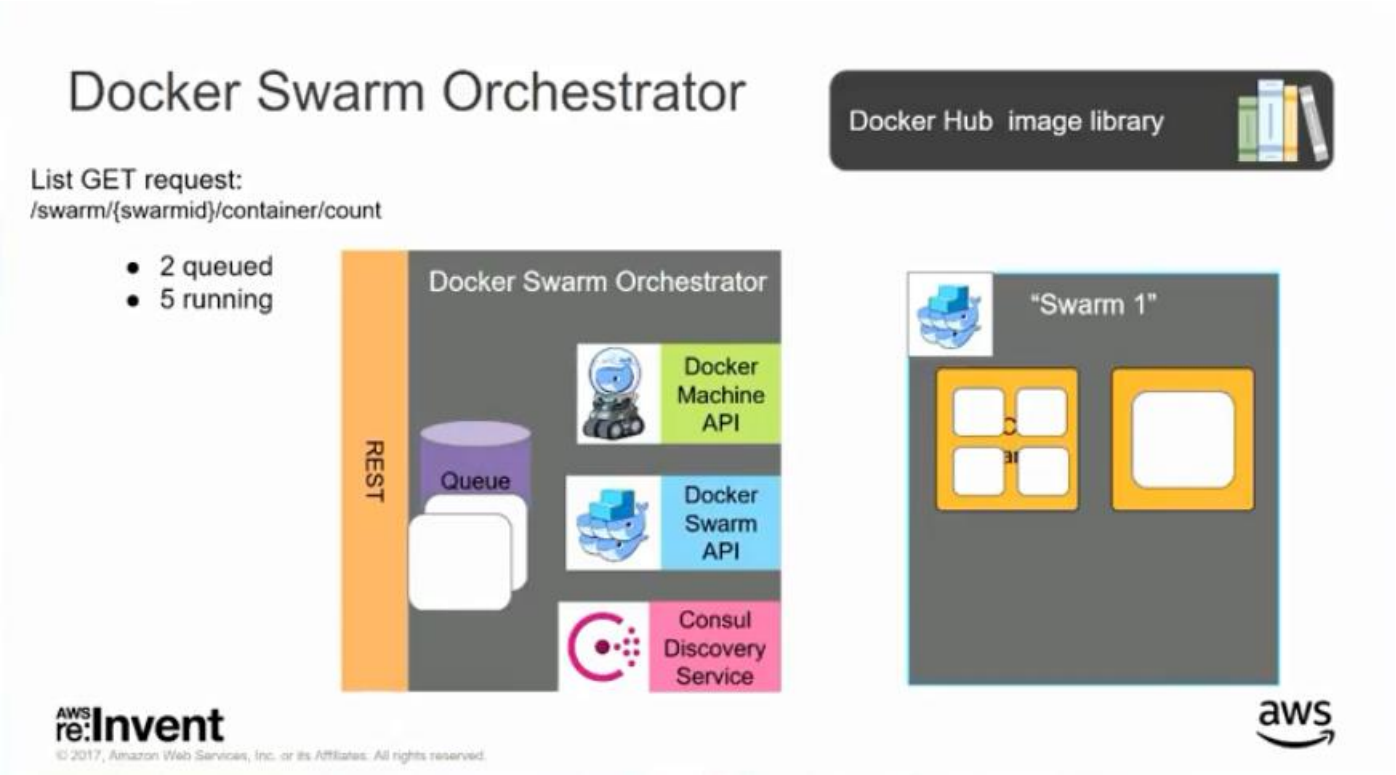


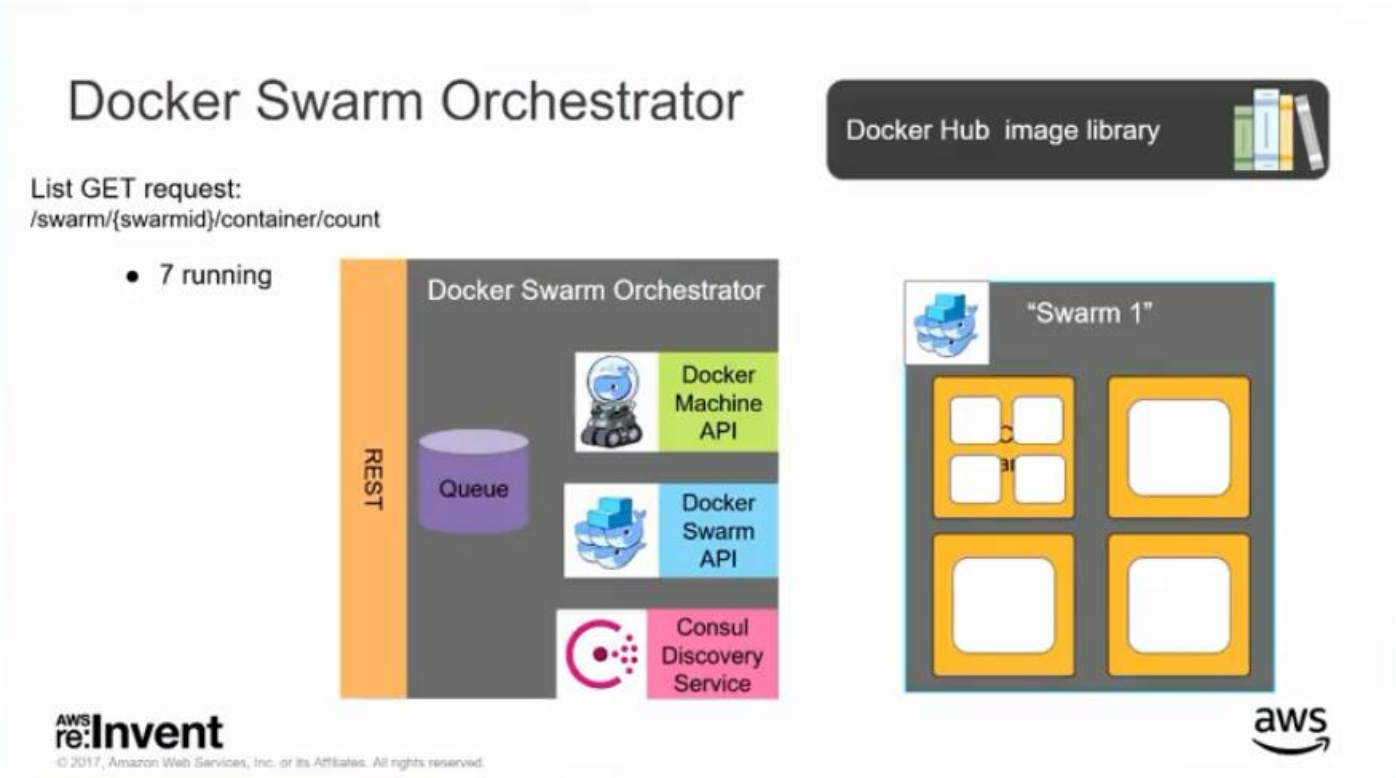
You can post any number of jobs into the queue with different memory requirements



The DSO starts the containers using available resources as available







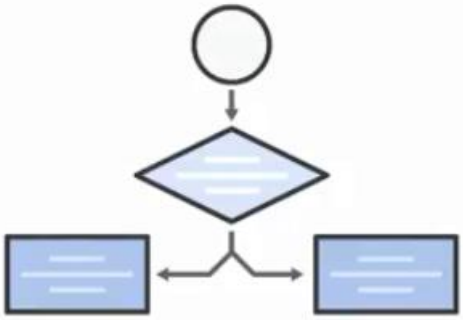
The DSO then dynamically starts some more instances to run the jobs in the queue

From eHive to e[hive]: Stage 3

Putting the **e[hive]** pieces together

How e[hive] runs a containerised workflow in the AWS cloud using the Docker Swarm Orchestrator

Workflow steps and resource requirements



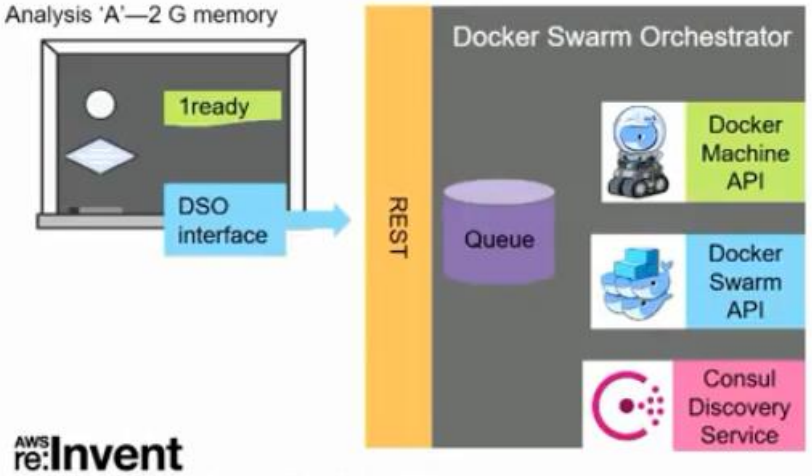
Analysis 'A'—2 G memory

Analysis 'B'—7.5 G memory

Analysis 'C'—100 G memory

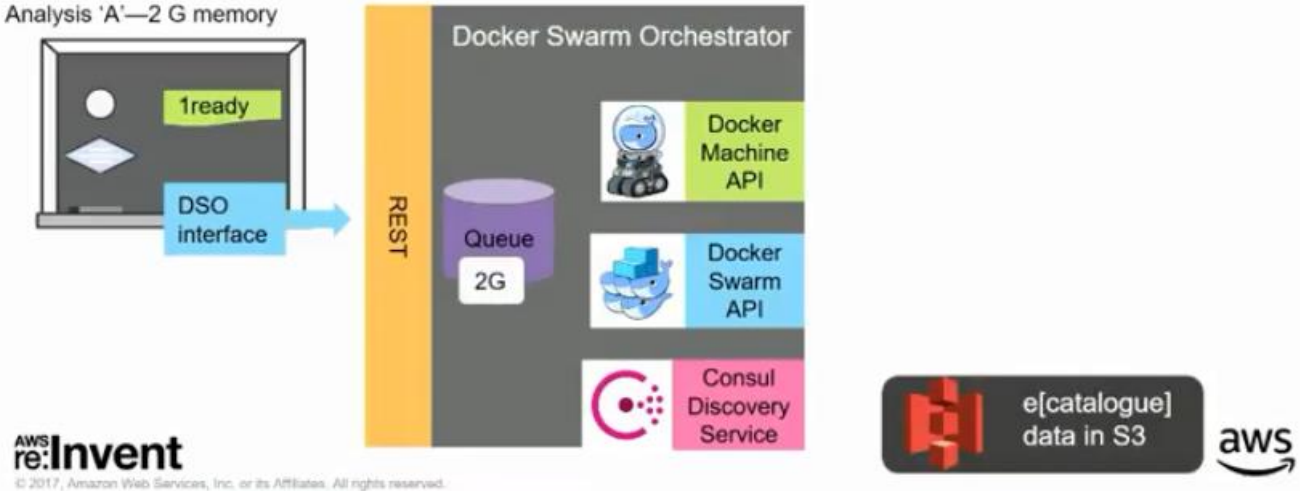
Assembling the **e[hive]** pieces

Docker Hub image library

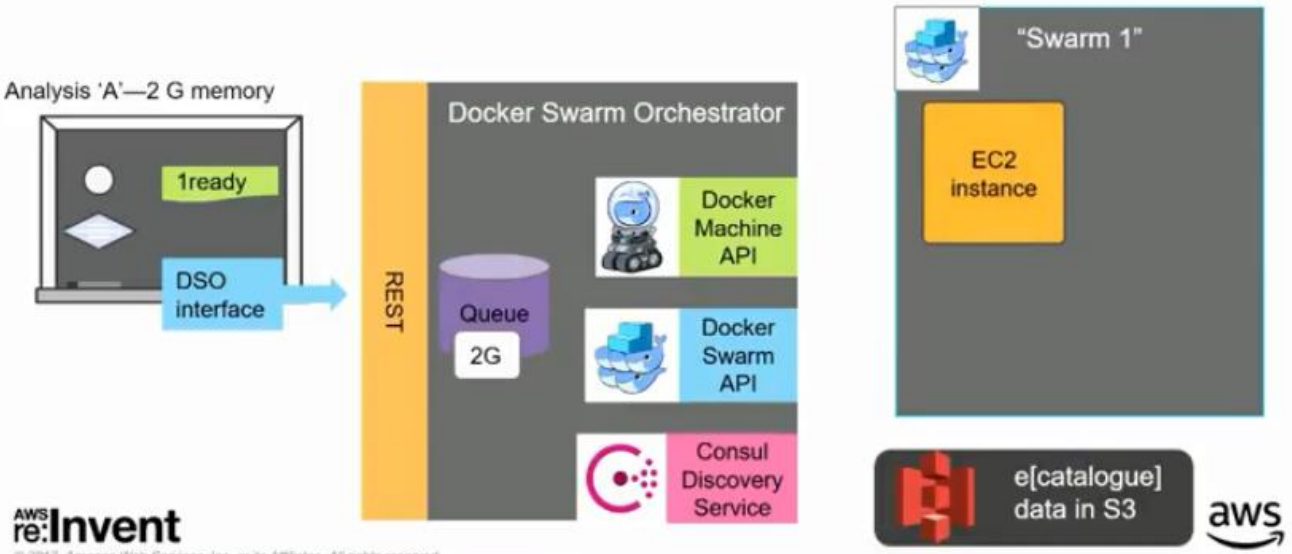


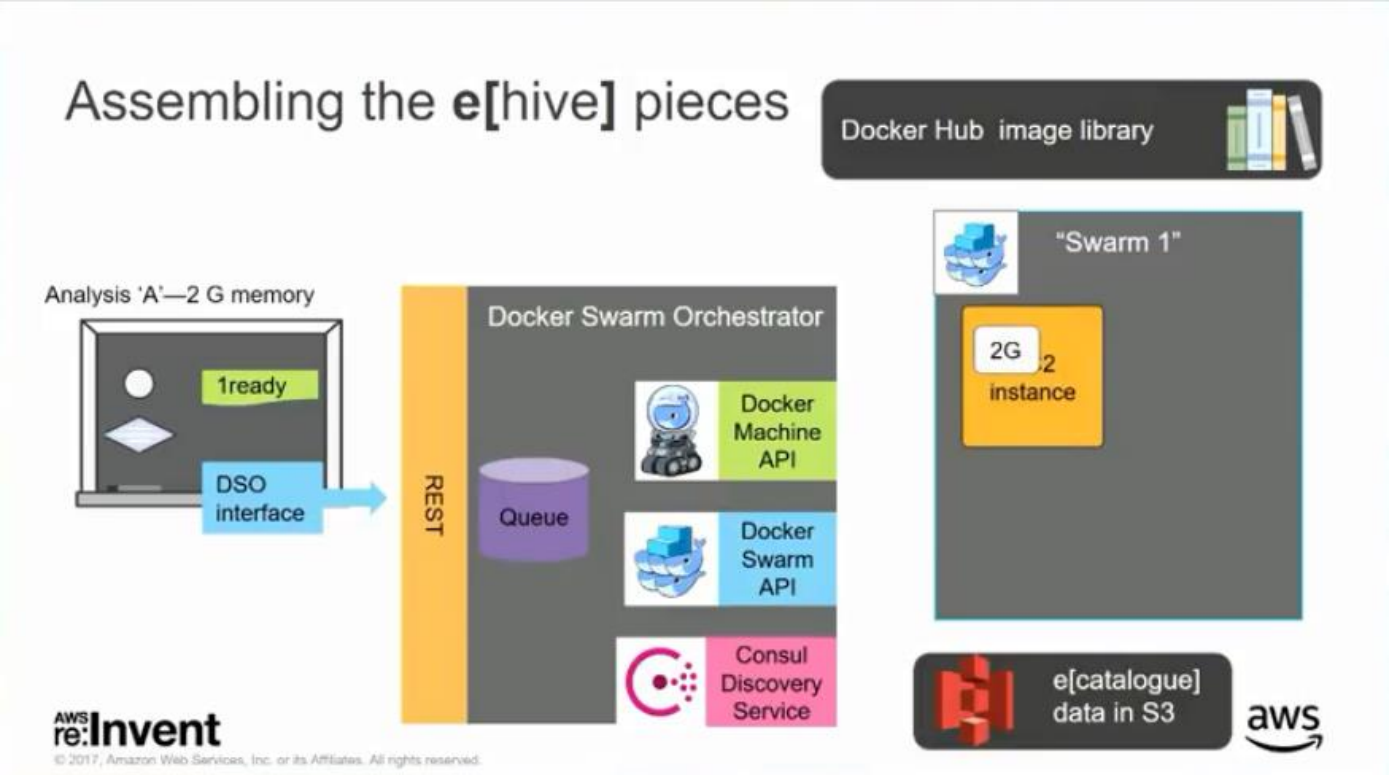
e[catalogue] data in S3

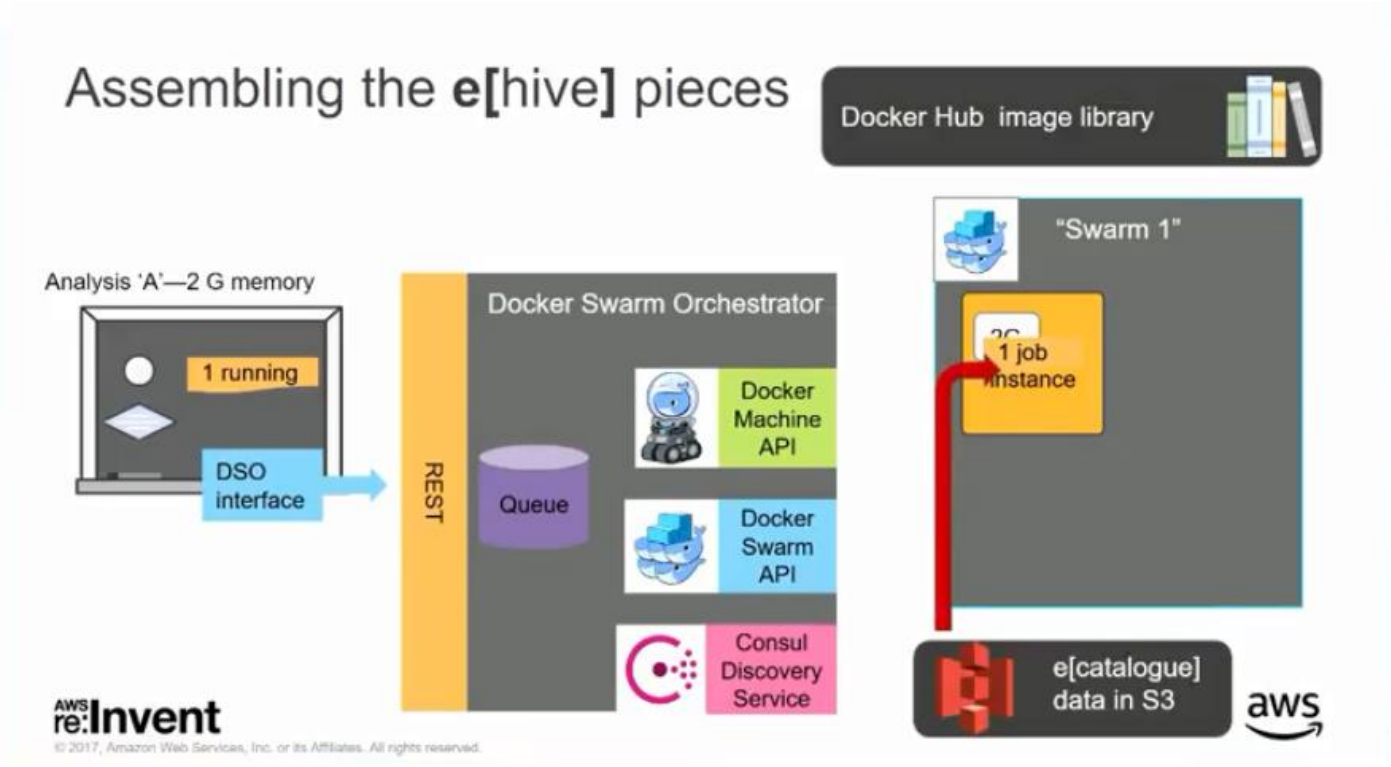
Assembling the e[hive] pieces



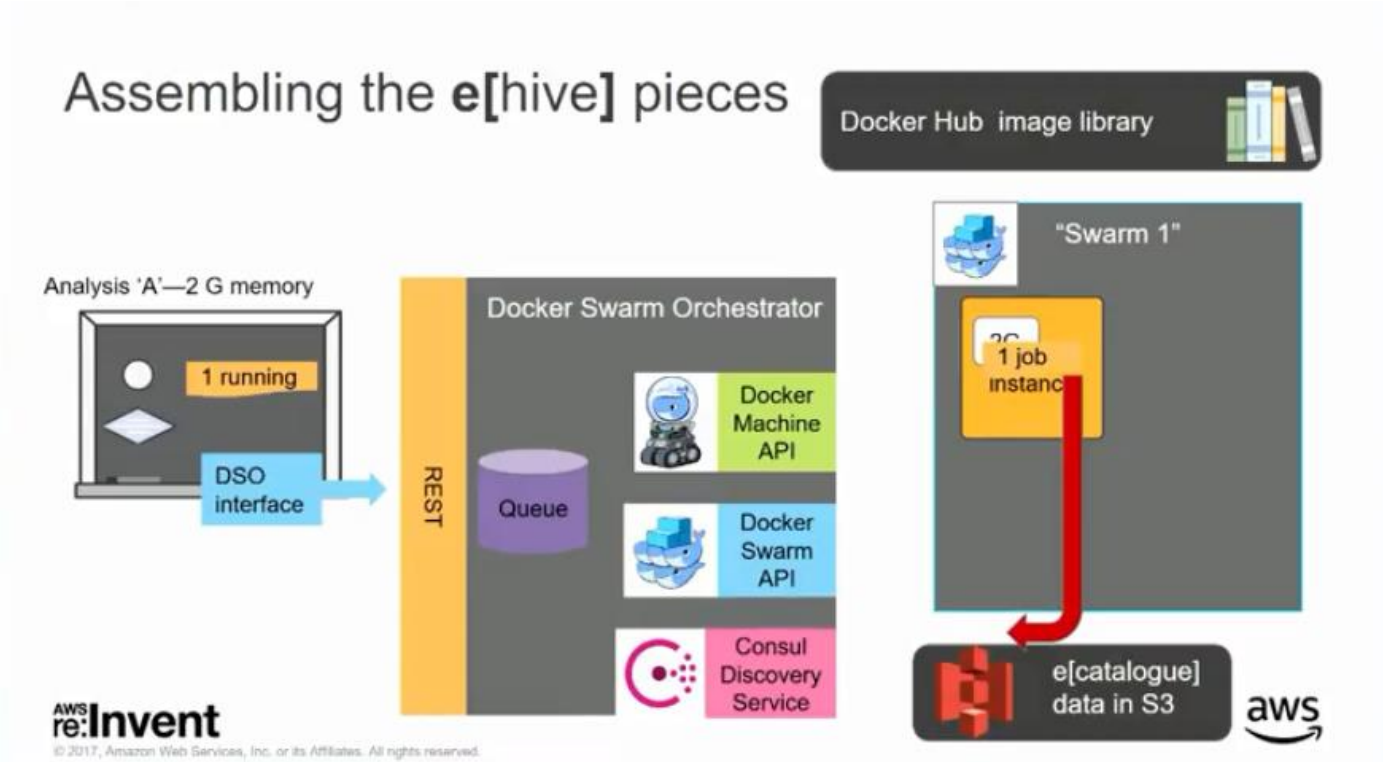
Assembling the e[hive] pieces



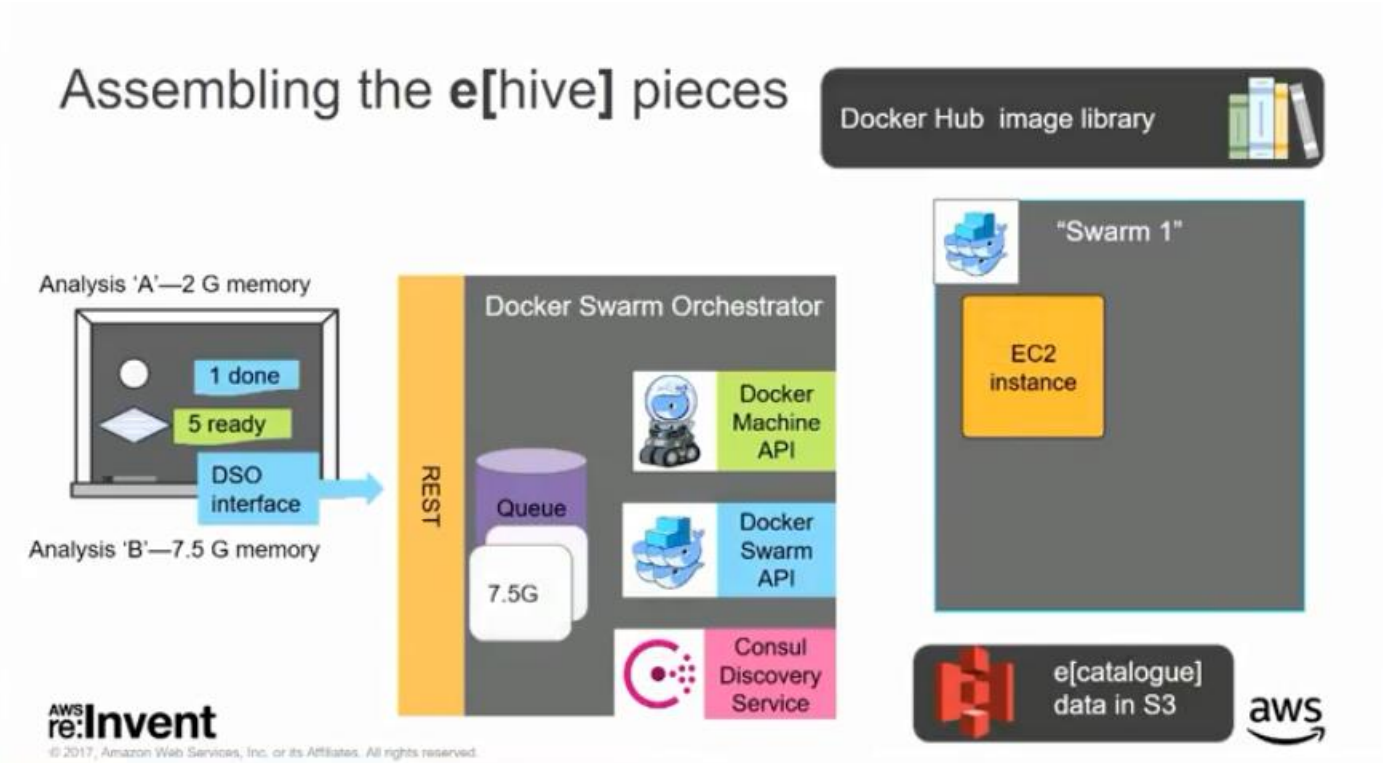




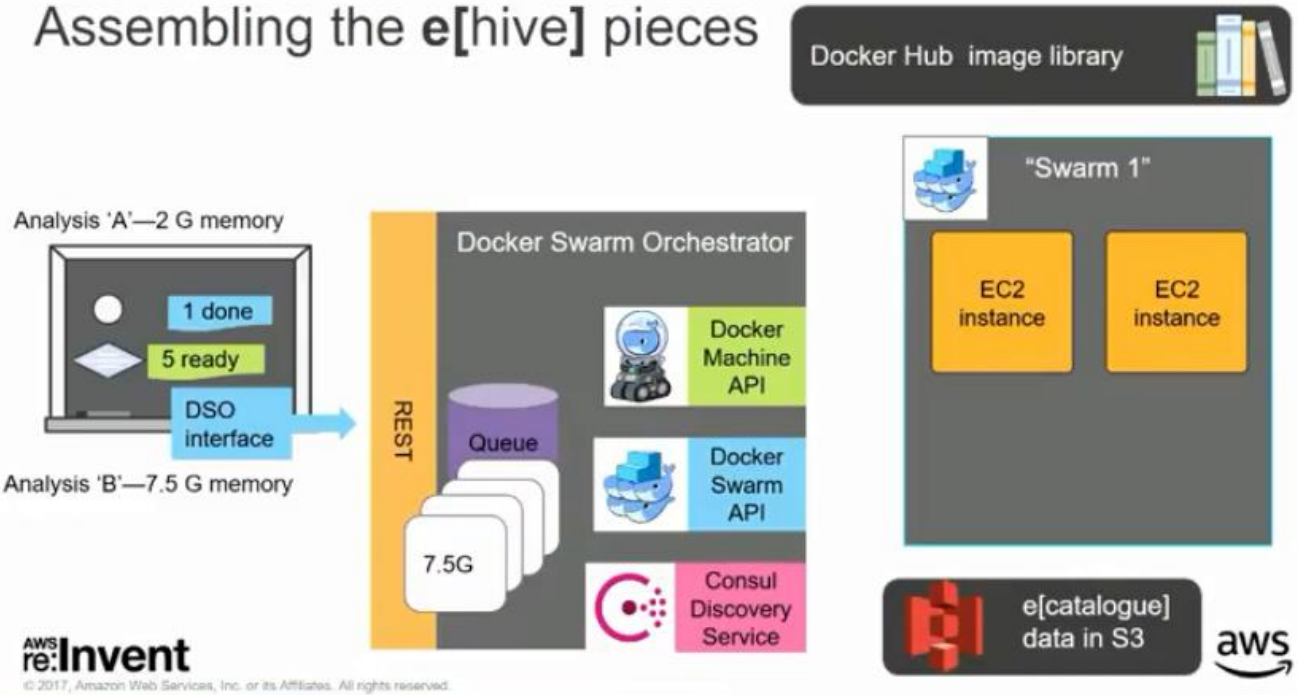
The running instance pulls the job details from the eCatalogue bucket in S3 and starts running the job



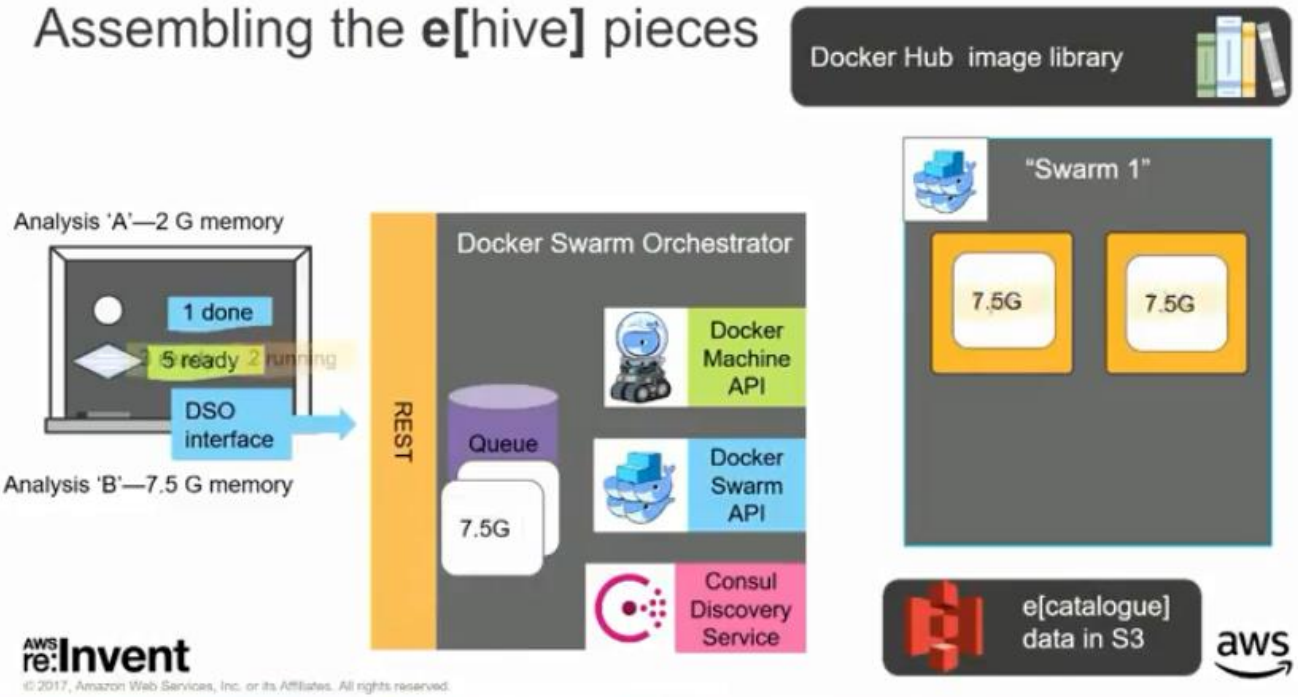
It also writes the outputs of the running job back to the eCatalogue.



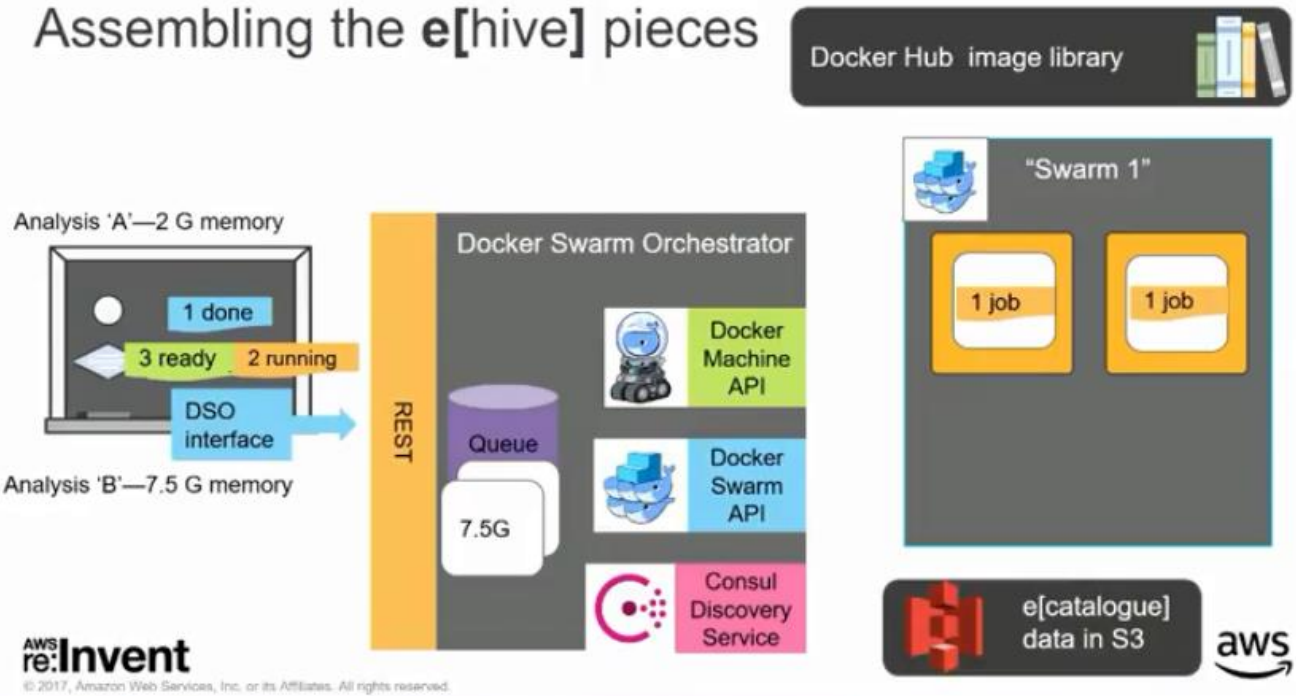
Assembling the e[hive] pieces



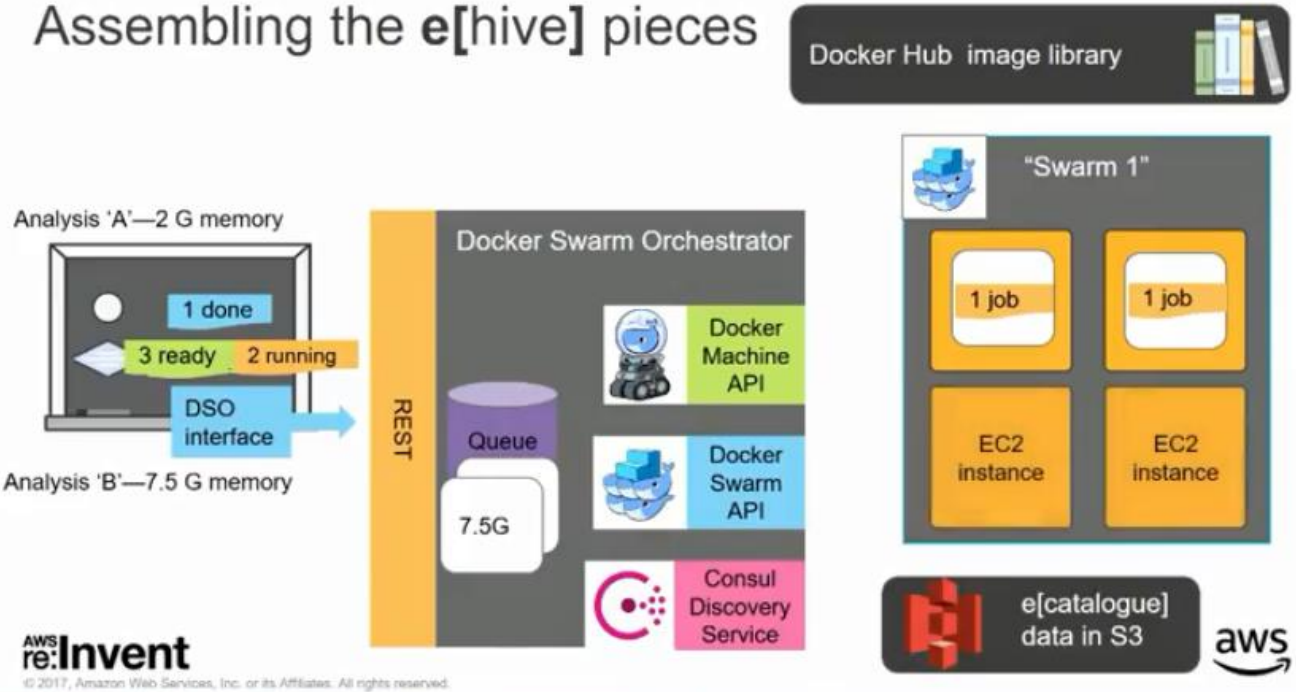
Assembling the e[hive] pieces



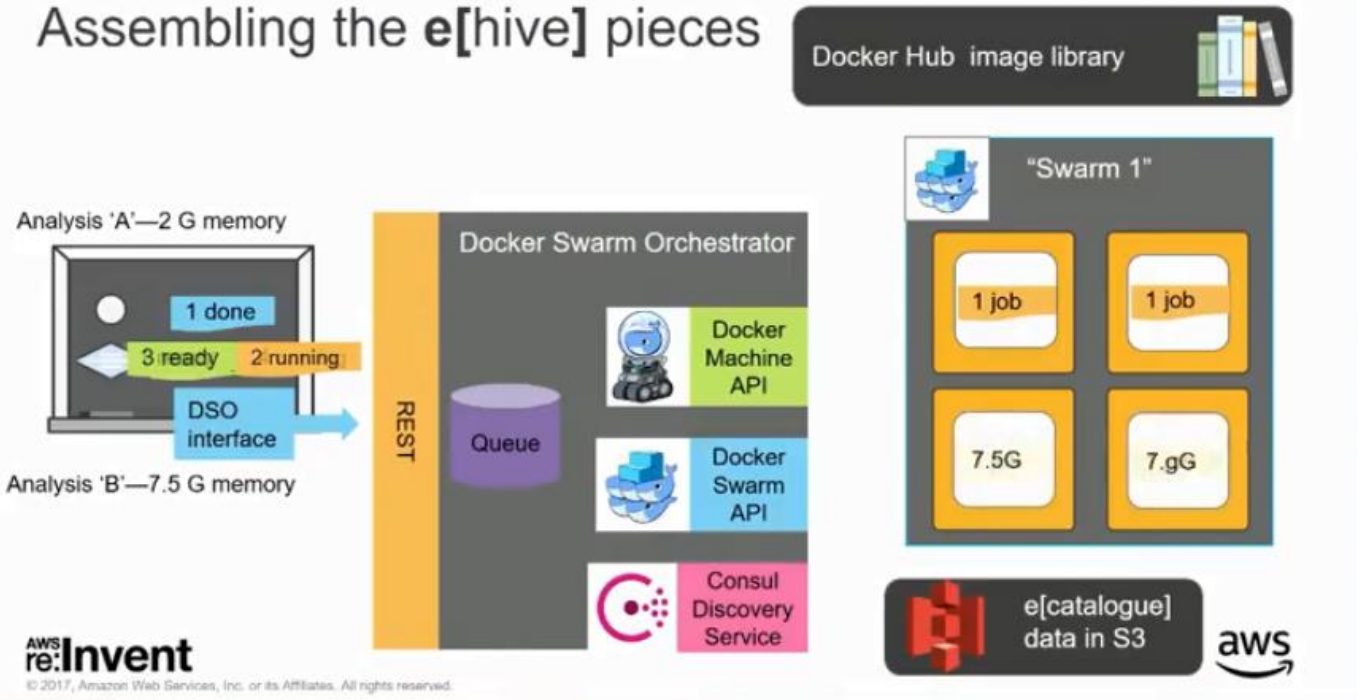
Assembling the e[hive] pieces



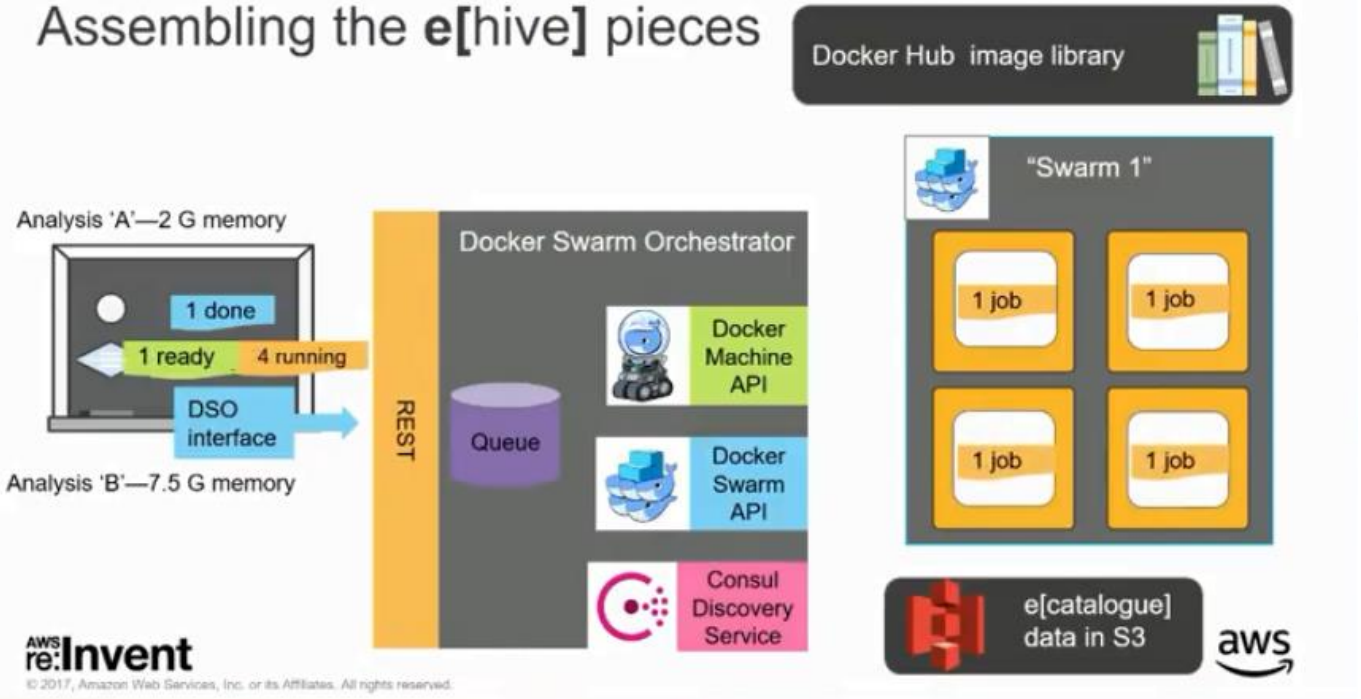
Assembling the e[hive] pieces



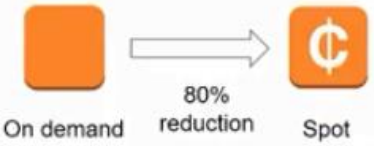
Assembling the e[hive] pieces

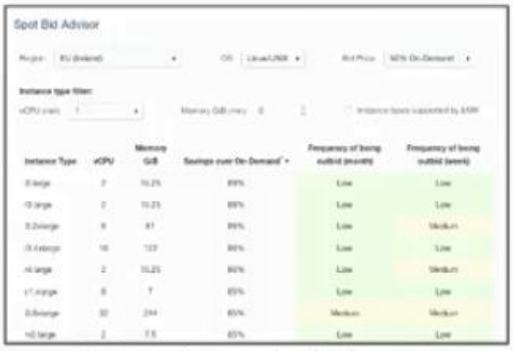


Assembling the e[hive] pieces

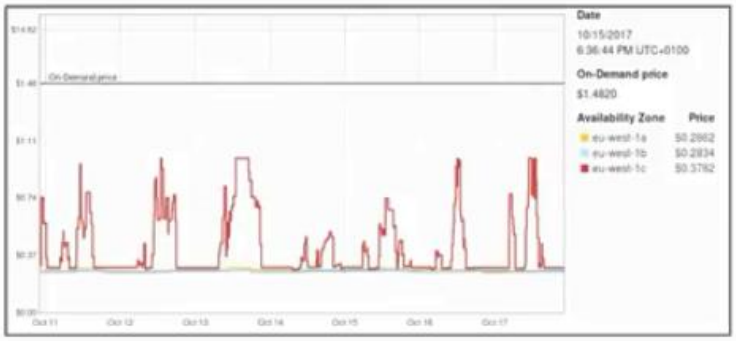


Use spot instances!





Instance types and reductions vary - choose carefully



Availability Zone important

Spot instances and reliability for SLA

- Spot instances have risk
- Multi-layered fault tolerance mitigates against this
  - eHive and job/worker monitoring
  - DSO and container queue
  - DSO and Docker Swarm

Putting e[hive] into action

Optimising resource usage with **e[**hive**]**

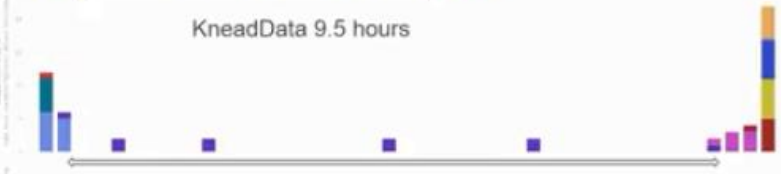
Using metrics to optimise analysis resource usage

Making use of the metrics: Optimising resource usage

7.5 G container, bowtie2 options: very sensitive

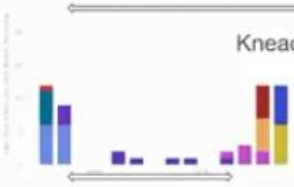
KneadData 9.5 hours

30G container, bowtie2 options: very sensitive



KneadData 18.5 hours

7.5 G container, bowtie2 options: very fast



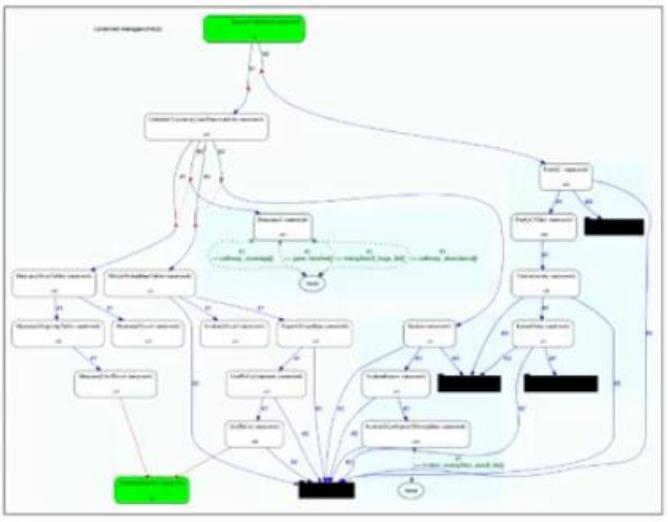
KneadData 5 hours

Time and cost of KneadData analysis

Options used	Average time per job (sample)	Cost per sample (on demand, r3.xlarge $0.371 hourly)	Cost per 1000 sample (on demand, r3.xlarge $0.371 hourly)	Cost per 1000 sample (Spot instances, r3.xlarge $0.05 hourly)
7.5G container, bowtie2 options : very-sensitive	6.7 hr	2.4857	2485.7	335
30G container, bowtie2 options : very-sensitive	4.3 hr	1.5953	1595.3	215
7.5G container, bowtie2 options : very-fast	3.5 hr	1.2985	1298.5	175
KneadData with 30G memory container, bowtie2 options: 4 CPUs, --very-sensitive	1.1 hr	0.4081	408.1	55
KneadData with 30G memory container, bowtie2 options: 4 CPUs, --very-fast	50.2 min	0.308	308	41.5

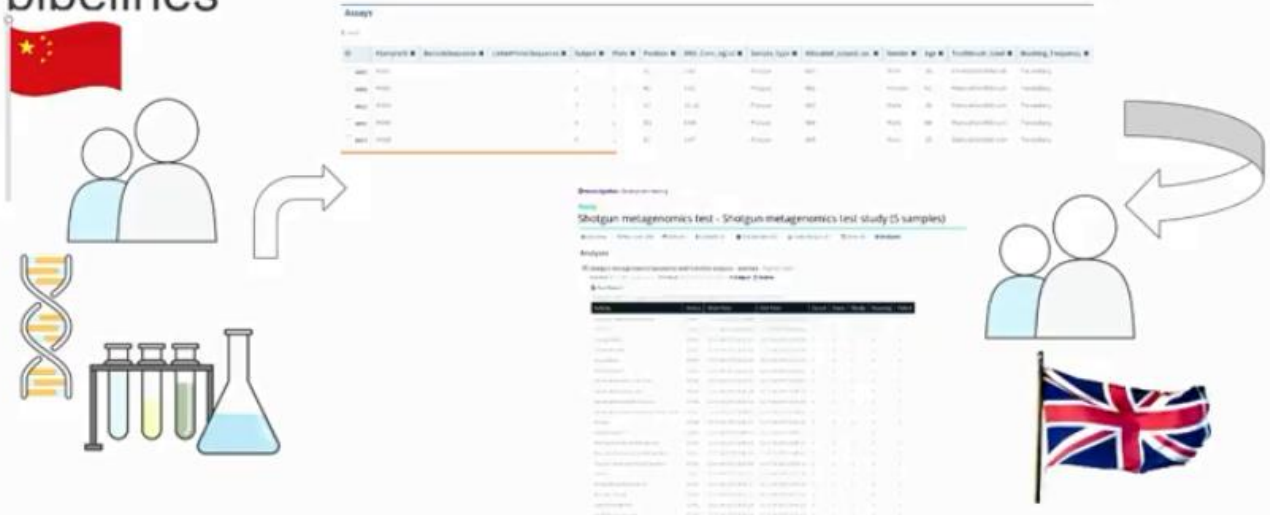
Metagenomics workflow performance

- Identify function and taxonomy for all sequences
- Input: ~1.7 billion read pairs (each read 125 base pairs)
- Took about 30 hours across ~50 instances (200 CPUs)
- Total cost < $50

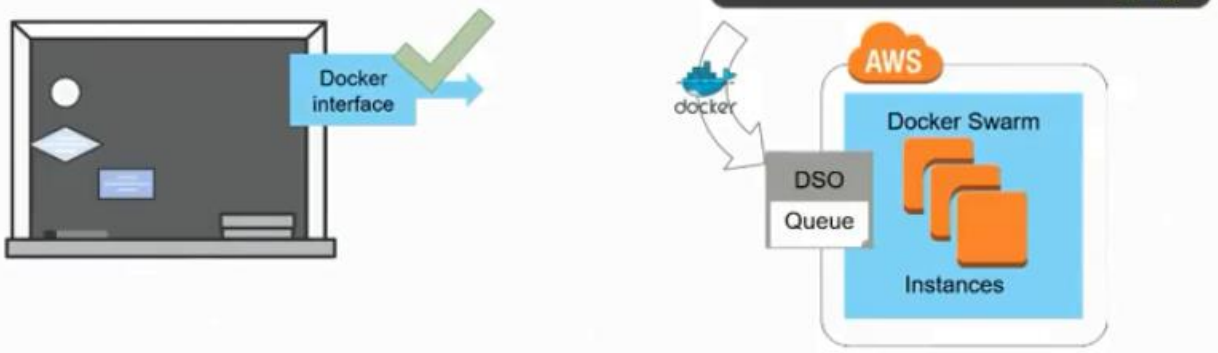


Conclusions and perspective

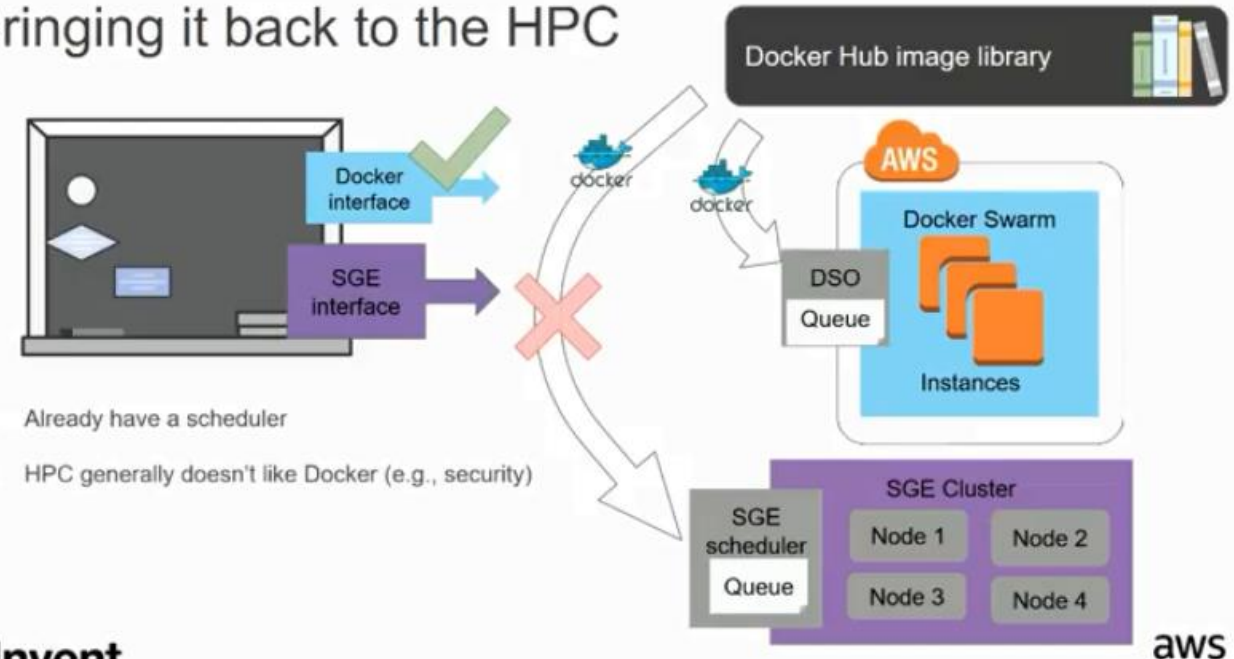
Scientists curate data in **e[**catalog**]** and run pipelines



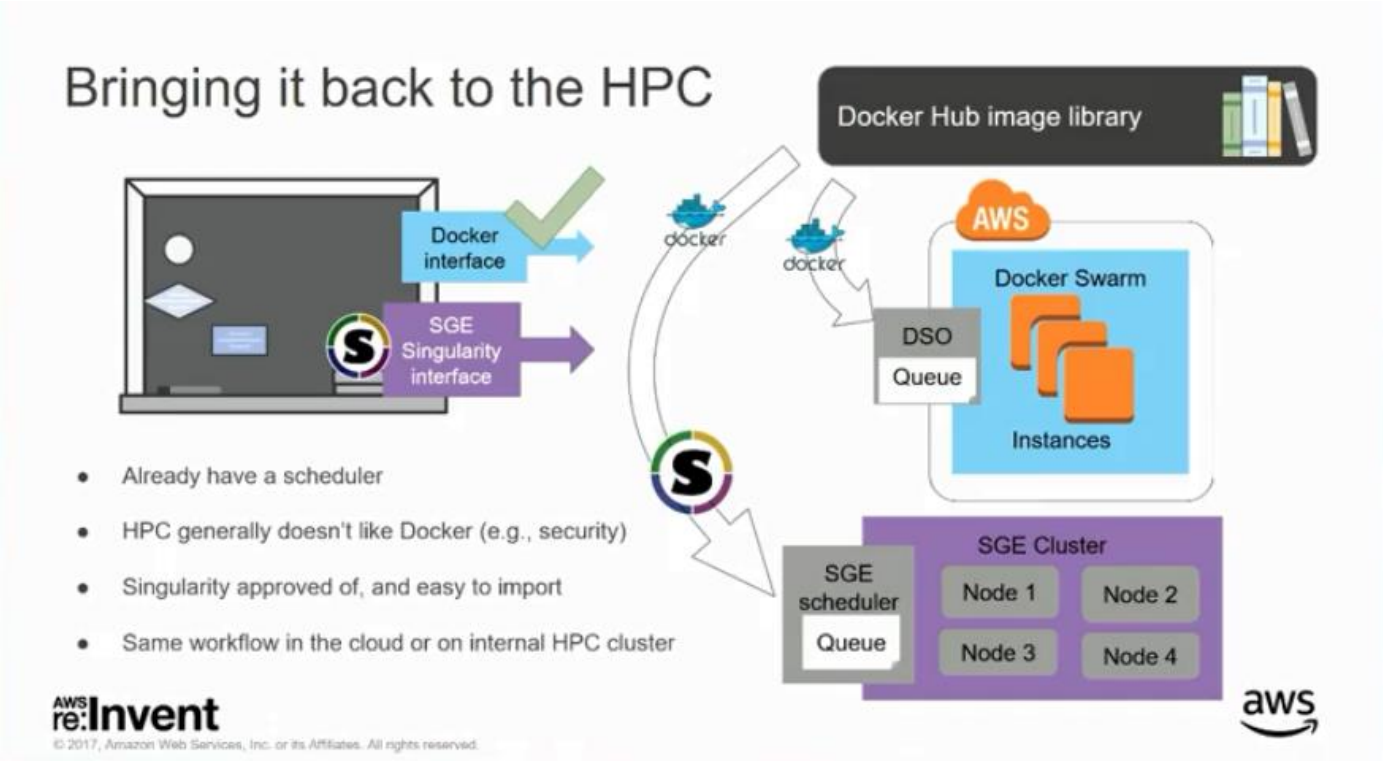
Bringing it back to the HPC



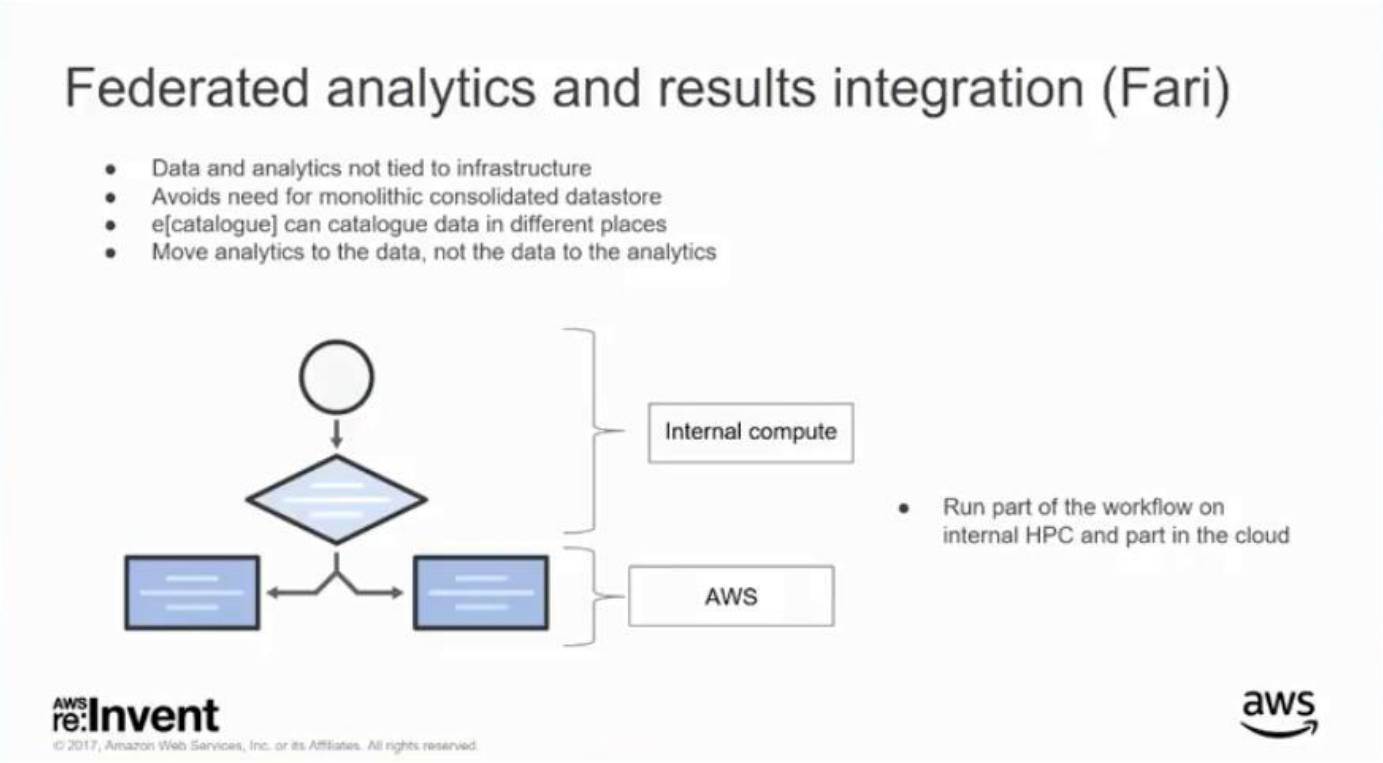
Bringing it back to the HPC



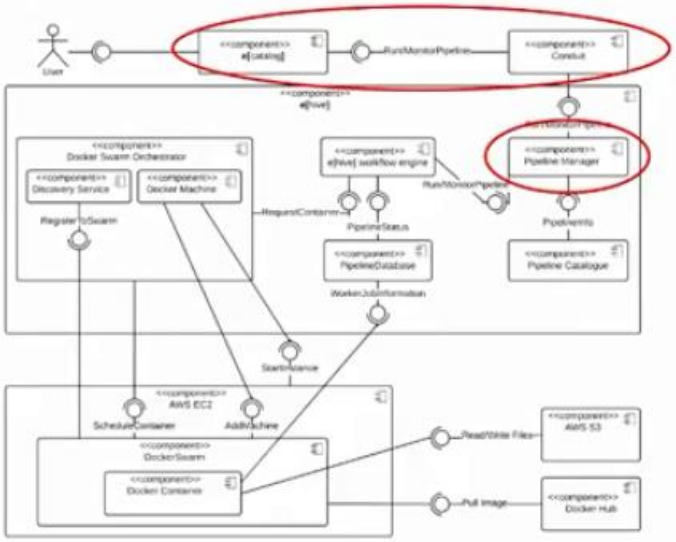
- Already have a scheduler
- HPC generally doesn't like Docker (e.g., security)



But we can now use another container type called Singularity that is well suited for within internal SGE clusters. It can translate our Docker image into a Singularity image and run that within the SGE cluster.



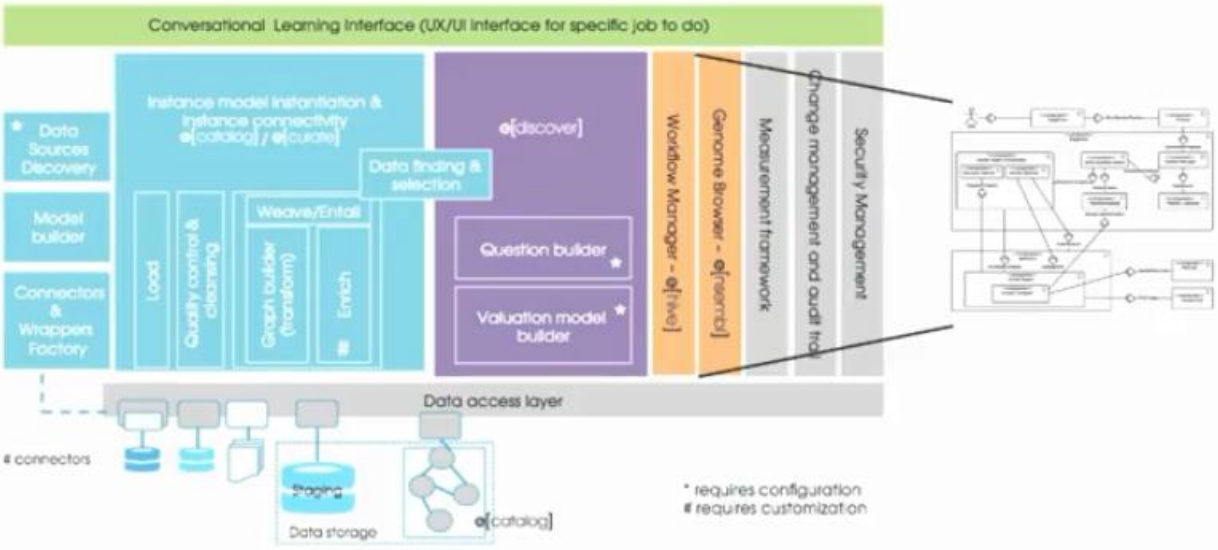
Putting it together: Enabling the end user



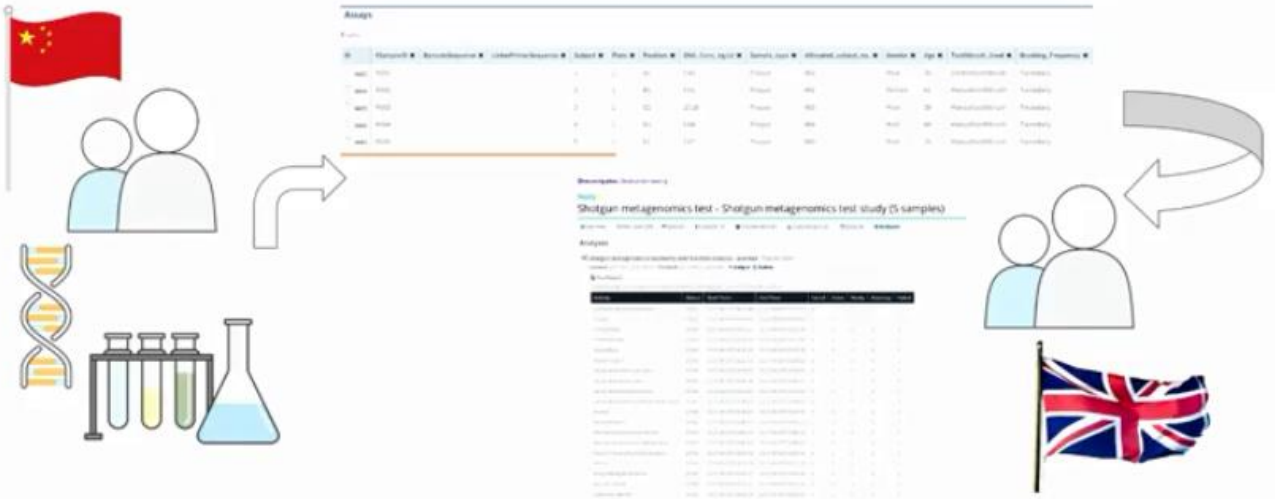
e[catalog] is separate application

Implemented RESTful pipeline manager

Eagle Genomics platform functional architecture



Unilever productivity & scale increased 20 times using this workflow



Thank you!