Omicron: a Galaxy for reproducible proteogenomics

By

Matthew Chase Chambers

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Biomedical Informatics

August, 2016

Nashville, TN

Approved:

Bing Zhang, Ph.D.

Daniel Liebler, Ph.D.

David Tabb, Ph.D.

Table of Contents

List of Tables

List of Figures

Introduction

Reproducibility is a fundamental tenet of science: it means the ability to read the methods of someone's experiment, repeat the experiment in a new context, and get essentially the same results. In reality, reproducing an analysis is often a difficult task. This is perhaps more true for life sciences than for other fields, and a considerable amount of effort has been aimed at making biomedical research more reproducible. [1]

Experimental and computational methods for biology have advanced tremendously in the last two decades: especially fields ending in "-omics". With these advances came new kinds of data (and much more of it) which needed new analysis methods. Microarray, high-throughput sequencing, and mass spectrometry (for -omics) all developed during this period. Yet it is not unusual for the bench scientists that collect that data to lack the computational expertise needed to make sense of it, especially when considering best practices (which usually are not the simplest approach). Although some of the effort on computational reproducibility has been targeted toward these bench scientists, [2] most of it is targeted at bioinformaticians. [3–7] Even for the latter group, setting up a bioinformatics pipeline system is arduous and the payoff is uncertain. Experimental and computational scientists would be better served by a workflow management system that is easy to set up for their own experiments, easy for others to deploy (reproduce), and performant enough to scale to large data sets. The present study demonstrates a system that works for proteogenomics, but it generalizes to any -omics fields.

Proteomics

Proteomics is the large-scale study of protein expression, structure, or function. Studying the proteome - the entire set of proteins in a biological system - is important because proteins are the most common and diverse worker molecules in biology. They are fundamental to the proper functioning of organisms. Proteins have four main biological functions: as enzymes that catalyse

reactions, as hormones/receptors/channels that permit cell signaling and signal transduction, as antigens/antibodies that comprise the immune system, and as structural elements. Proteins also have roles (though not necessarily causal ones) in most diseases, whether it be from misfolding, [8] underexpression, [9] or overexpression. [10] Thus, proteins are good candidates to be biomarkers of disease [11] and predictors of treatment efficacy. [12]

The most sensitive approach to identify the proteomic content of a sample is “bottom-up shotgun proteomics”. A typical shotgun proteomics protocol starts with samples chemically treated to unfold the sample’s proteins and make them less reactive. Then the proteins are digested by a proteolytic agent – commonly the enzyme trypsin – which cuts the proteins into short sequences of amino acids called peptides. The peptides are separated with a liquid chromatography (LC) protocol such as reverse phase LC. The LC column injects the peptides into a mass spectrometer (MS) via an ion source – typically electrospray – which ionizes the peptides and makes them detectable by MS. In a typical shotgun proteomics instrument method, the MS takes a survey mass spectrum (MS1) every Nth scan (where N depends on the instrument’s scanning speed). It selects the most intense peak from the MS1 and analyzes just that peak by collecting a tandem mass spectrum (MS2 or MS/MS). To collect an MS2, the instrument isolates and tries to fragment (dissociate) the selected peak (also known as the “precursor”), usually by collision-induced dissociation. Then more MS2s are taken for the next N-1 most intense peaks. To improve sensitivity, if the same precursor masses show up repeatedly in survey scans that occur within a certain time of a previous one, they are excluded from selection; this is called “dynamic exclusion”. The basic idea is that MS2s coming from this method will often be of fragmented peptides, and the masses of those fragments can be used to determine the peptides’ amino acid sequences. Shotgun proteomics can also be used to quantify the proteome, but that usually involves other MS methods.

The most common approach to identify peptides in shotgun proteomics spectra is "database" search. This entails software ("search engines") that match peptides to spectra (peptide-spectrum-matches, or PSMs) by modeling the peptides as *in silico* spectra and comparing those theoretical spectra against the observed MS2 spectra. However, many of these PSMs are false positives. Modern proteomic studies have millions of spectra to be identified, so automated methods are used to filter out the false positives. Software can produce a confident PSM list by matching the observed spectra against reversed or randomized versions of the protein sequences along with the real sequences. The reversed sequences are treated as "decoys"; if a spectrum's best match is to a decoy peptide, the software can assume the match is a false positive, and determine a score threshold for PSMs that excludes most of those false positive PSMs.

Even experienced proteomic scientists cannot simply look at a list of peptides and determine which proteins are present in the sample. The peptides must be mapped back to the proteome by protein assembly software. Modern protein assemblers use rules or models to report a parsimonious protein list (a complex topic by itself). A typical parsimony problem solved by protein assembly is protein isoforms sharing many peptides. If no peptides are identified that are unique to one of the isoforms, then all of the isoforms are considered to be a "protein group" with equal evidence.

Considering the many experimental and computational steps of a shotgun proteomics experiment, it is perhaps not surprising that these experiments are problematic to reproduce. Standard operating protocols for sample processing can help to reduce some of the experimental variation - e.g. intra- and inter-lab variability [13] - but computational data processing between labs has resisted widespread standardization. Furthermore, data processing software is being continuously improved to be more effective, and every improvement has the potential to change the software's output. Of course, no one should try to deter the development of better software, but an investigator trying to reproduce an old

experiment should be able to easily download and setup the software used by that experiment. For now, for most published proteomic analyses, it is far from easy.

Proteogenomics

As the name suggests, proteogenomics lies at the intersection of proteomics and genomics. The precise meaning of the term has shifted with changing technology. At first it referred to using six-frame translations of the genome to search for novel forms of translated proteins/peptides. The six-frame translated genome replaced the protein database in the proteomic database search, and the rest of the proteomic pipeline essentially remained the same. This early approach was intended to find proteins for which the genome did not have a gene annotation, or for which the annotations were incorrect. However, because of the large number of false candidate proteins created by six-frame translation, early proteogenomics struggled to pick out real novel peptides from false positives.

The meaning and scope of proteogenomics changed with the advent of second generation sequencing. With RNA-Seq, it became possible to cheaply sequence the RNA content of a sample, and a sample could be treated so that it included only certain kinds of RNA (e.g. mRNA, microRNA). The deep coverage of RNA-Seq (i.e., many overlapping sequences mapped to each nucleotide) led to new methods in genomics and proteogenomics.

Since mRNA transcripts are an integral part of the cellular process of creating proteins, they can theoretically predict which protein sequences may be present in a sample. That is, if an RNA transcript is not found in a sample, excluding its translation from the proteomic search space could improve sensitivity. This approach is not always ideal though, since the "lifespan" of some RNA transcripts may be much shorter than the lifespan of its translated protein; the opposite may sometimes be true as well, in which case the RNA transcript is a better indication of a protein’s presence than the protein itself. There are also issues of durability and detectability: some proteins may not have enough detectable peptides, or its peptides are

especially unstable. Again, these can also be problems for RNA detection. These issues tend to limit the correlation between RNA and protein expression; using both RNA-Seq and shotgun proteomics separately often provides complementary rather than corroborating evidence.

A more promising way to use RNA-Seq data to improve proteomics is to add mRNA mutations (i.e., SNPs, INDELs, and junction changes) to the proteomic search space. That is, a protein database can be created that includes both the reference protein sequences and the various mutated forms (also called “variant” or “aberrant” proteins) detected by RNA-Seq. With this approach, each sample’s proteomic search space can be customized according to its unique mutations. The sample-specific database approach has become quite popular with bioinformaticians, but the wider proteomics community lags behind. [14–25] That may be because no best practices have been developed, or because the tools to run this kind of analysis are difficult to use.

Indeed, it is quite difficult just to reproduce the results of single omics experiments where one kind of instrument is producing one kind of data to be analyzed. In proteogenomics there are two kinds of data and software pipelines to handle. Furthermore, many investigators do not have expertise in both proteomics and genomics. Yet proteogenomics studies are likely to become more common as investigators strive to maximize the biological and clinical information they can extract from their samples and as funding agencies push for more precision medicine. As with proteomics, it should be easy to download the software used for these experiments to reproduce and reuse them, but that is not the case now.

Current approaches for reproducibility

Several factors make it difficult to reproduce a computational analysis: 1) different revisions/versions of the same analysis tool can produce different results; 2) tools and workflows are frequently updated; 3) downloading and configuring older versions of tools is challenging and error-prone [26]; 4) having multiple versions of the same tool on the same

computer can cause obscure errors and make it easy to run the wrong version. These problems should not surprise veterans of the -omics fields, but veterans may be surprised by the significant work toward reproducible data analysis that has already been done. [2,27–35] These projects range in generalizability (specialized for -omics vs. arbitrary tool pipelines) and user-friendliness (command-line vs. graphical). Table 1 summarizes some existing work that has been used to facilitate reproducible analysis. All these technologies have been used to improve reproducibility, but they all have drawbacks.

Technology	Advantages	Disadvantages
Workflow managers • Galaxy • Taverna • Ruffus	• Easy to set up an analysis as series of steps • Runs on one node, HPC cluster, or the cloud	• Must be scripted for new tools • Hard to archive for re-use
Virtual machines • VirtualBox • VMWare • Xen	• Fully isolated from host OS • Usually works across platforms • Easy to archive for re-use	• Lower computational speed • Archives are large (entire OS) • Hard to automate build/test/run • No workflow assistance
Containers • Docker • LXC • Sandboxie	• Partially isolated from host OS • Easy to archive for re-use • Archives are small • Easy to automate build/test/run	• Usually specific to one platform • Security concerns running on shared HPC clusters • No workflow assistance

Table 1. Examples of technologies used to improve reproducibility. "HPC" is high-performance computing.

Fortunately, workflow managers can be combined with VMs or containers to combine their strengths. But unfortunately, few workflow management projects have garnered a large user base, either inside or outside the -omics fields. [36] There are several likely reasons for that: 1) bioinformaticians that develop new tools and workflows devote more effort to new studies than to making old or current ones reproducible; 2) investigators that use 3rd party tools and workflows do not have the expertise to write the scripts and wrappers needed by workflow managers; 3) funding agencies do not require that results be reproducible. This third problem is important because it means investigators are not financially motivated to make their pipelines easily reproducible. However, at least two solutions to the first two problems exist that I believe are well-established enough to improve reproducibility of -omics analysis: Galaxy and Docker.

Galaxy [2,28] is a free, open-source workflow management platform originally designed to improve accessibility and reproducibility of genomic analyses. Over time it has been extended to support arbitrary tools and workflows, although it maintains a focus on -omics. [24,25,27] It functions as a multi-user server to which users can upload their data. Once uploaded, they can analyze those data with arbitrary tools or workflows (chains of tools). Typically that analysis will create new data sets which users can view, download, and further analyze. A major advantage for Galaxy is the many -omics tools that are already available in its “ToolShed” [37] - analogous to Apple’s “App Store”, but for data analysis. Competing workflow managers, e.g. Taverna, [38] Kronos, [31] Ruffus, [34] Bpipe, [33] and Snakemake [32] would require additional scripting or programming to be able to work with many of the -omics tools Galaxy already supports.

By creating and customizing a private Galaxy server, investigators can create “flavors” of Galaxy with their experiment’s specific set of tools, workflows, and tool parameters. Data analysis done this way can be easily reproduced as long as that instance of Galaxy is available. Unfortunately, it is not easy to set up a private Galaxy instance from scratch; it requires considerable knowledge of Linux/Mac server administration and reading Galaxy’s documentation. After an instance is set up, its continuing availability usually depends on an unpredictable funding climate. So the entire system needs to be archivable, and furthermore the server and tools must be isolated and archived together or it will suffer from the same reproducibility problems described above: i.e. how can another investigator easily set up their own identical instance of Galaxy to reproduce the analysis? To solve those problems and still retain the benefits of a workflow manager, a couple solutions are available.

The traditional solution for archiving any collection of software, including a workflow manager like Galaxy, has been to install it all inside a “virtual machine” (VM) and then create a redistributable image of the VM for others to download. [39–42] Starting up a VM image is much easier than installing Galaxy from scratch. The VM image archives the tools, workflows, and parameters used for an experiment, and runs fully isolated from the host operating system.

Unfortunately, that level of isolation has some drawbacks. Since VM images must include an entire OS, they tend to be multiple gigabytes. Downloading and running dozens of VM images, each one corresponding to a different experiment, is not a pleasant task to manage. Software also runs slower on a VM compared to the native operating system. When analyzing large amounts of data, the extra time to run an analysis could be inconvenient or expensive. Although VMs remain a popular way to solve the reproducibility problem, recently another isolation paradigm has gained traction. It provides the archiving and most of the isolation of a VM while making more effective use of storage space and computation time.

That alternative is “containerization”. Containers, as the name implies, contain an isolated set of files needed by some software, but allow the software to run without hardware virtualization. The most widely used and best documented container system is currently “Docker”. [36,43] A particularly attractive feature of Docker is the way containers are composed from layers. Each layer typically adds new files or changes/removes files from earlier layers. This design allows containers to derive basic functionality from another container and then add special functionality on top. If multiple containers (each encapsulating a different analysis) derive from the same base container, the base container is only stored once. Thus, it is feasible to have dozens of analysis pipelines that share the same base container without squandering disk space.

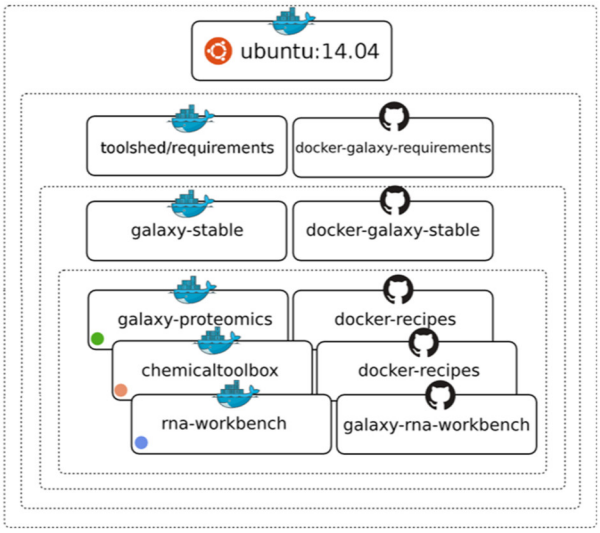

Figure 1. An ecosystem of Docker containers for Galaxy developed by Gruening et al. “Galaxy-stable” is the basic production-ready Galaxy instance. Deriving from that are “flavors” of Galaxy, e.g. “galaxy-proteomics”. Each Docker container is mapped to a GitHub repository.

Gruening et al. [44] have already created a “production-ready” Docker ecosystem for Galaxy (Figure 1). Production-ready means that some auxiliary applications are preinstalled to make

the containerized Galaxy more performant and accessible, including an FTP server, a PostgreSQL database, and a more efficient HTTP server.

Although it is easy to start one of the Galaxy Docker images from Gruening et al. on a single server, it is not trivial to connect them to an HPC cluster. This capability is a practical necessity to analyze large proteogenomic data sets in a reasonable time. Of course, not every investigator has access to an HPC cluster. Fortunately, tools like StarCluster [45] and cfncluster [46] solve both problems by making it relatively simple to set up an ephemeral HPC cluster on a cloud provider such as Amazon Web Services (AWS). Cfncluster uses Amazon's CloudFormation (CFN) to create an AWS "stack" - analogous to a VM image – to manage a set of cloud resources such as network interfaces, security rules, compute instances, and auto-scaling support. Complex CFN stacks can be launched from a single template file.

With a customized CFN template and an AWS account with a payment method, it is possible to launch a containerized flavor of Galaxy with a corresponding scalable compute cluster in just a few mouse clicks. This compares favorably to competing cloud-compatible workflow managers. Kronos [47] and CPFP [48] require potential users to open a command-line or learn AWS' complicated resource configuration interfaces (compared to the CFN stack creation interface).

Research goals

There are 3 necessary qualities for a computational reproducibility platform to be both useful and usable for experimentalists and bioinformaticians, but I found no existing solutions that satisfied them:

1. **Ease of development:** non-technical users can set up their specific analysis
2. **Ease of deployment:** other non-technical users can run that analysis
3. **Scalability:** able to reproduce large data analyses in a reasonably short time

To address this problem I created a proof-of-concept platform, which satisfies all these qualities for a proteogenomic analysis, yet can be easily modified for other multi-omic analyses. My goals for the proof-of-concept were to:

1. Create a custom flavor of Galaxy ("Omicron") with the tools necessary to reproduce the proteogenomic variant analysis described by **Zhang et al.**, [49] where proteomic data was searched with sample-specific, variant-aware protein databases.
2. Make Omicron easy for non-technical investigators to launch.
3. Make it easy to deploy a scalable, production-quality instance of Omicron to AWS.

Methods

Software components

Simply put, Omicron is a flavor of Galaxy customized for a specific proteogenomic experiment. Starting from a vanilla Galaxy installation, I added the extra proteogenomics tools and customizations to reproduce the variant analysis pipeline described by Zhang et al. That pipeline required a certain set of tools:

- R [50]
- Samtools, SAM to BAM, bcftools [51]
- FASTQ Groomer [52]
- FASTA Merge Files and Filter Unique Sequences [53]
- TopHat2/Bowtie2 [54]
- VCFfilter [55]
- **MyriMatch/IDPicker** [56,57]
- **customProDB** [24]
- **proBAMsuite** [58]

Some of these tools relied on reference data, such as the human genome sequences and the TopHat/Bowtie indexes of those sequences. Galaxy manages reference data like this in a centralized fashion so that all users on a Galaxy server can share the same genome sequences and indexes. However, because the reference data was so large and in order to preserve easy

archivability, I did not include the reference data as part of Omicron itself. Instead, the human reference genome, gene annotations, and Bowtie2 indexes were created by “data managers” as part of the pipeline. Omicron's data managers included:

- Reference genome (hg19)
- Bowtie2 index (also generated the TopHat2 index)
- SAM FASTA index
- **CustomProDB annotation**

Most of these tools were already available in the Galaxy Toolshed. In order to be able to use the tools designated in **bold**, I developed new or modified existing “tool wrappers”: XML files that tell Galaxy how to run the tools, what their inputs and outputs are, and what configuration options to show to users. The non-data-manager tool wrappers included automated “functional testing” to ensure that the wrapper could run the tool and produce the expected output for a given input. Tool wrappers allowed running the tools individually, but that was not sufficient to simplify reproducing the Zhang et al. pipeline. For that, I needed workflows.

One of Galaxy’s most compelling features is its workflow editor. The graphical interface makes it simple to chain tools together so that one tool’s output becomes another tool’s input. Based on the methods description of the variant analysis in the Zhang et al. paper and some private correspondence with the authors, I created 6 workflows in Omicron to reproduce that analysis. These workflows had to be run in order, since the input to workflow 2 depended on the output from workflow 1. Because of the way data managers worked in this version of Galaxy and the difference in number of input files and output files between various parts of the pipeline, it could not be wrapped up in a single workflow. For example, the TopHat2 index data manager depended on the reference genome data manager. When both data managers were in one workflow, there was no way to make the TopHat2 manager start after the reference genome data manager finishes.

Variant analysis pipeline

I will describe the workflows in detail, but start with an outline of the 6 workflows:

1. Download reference genome and run customProDB annotation for human (hg19)
2. Index hg19 genome for TopHat2 and samtools
3. Create sample-specific protein FASTAs from RNA-Seq data
4. Merge all FASTAs to capture variant proteins from all samples
5. Run sample-specific MyriMatch search on proteomic data and assemble with IDPicker
6. Merge all IDPicker files into a single assembly and create result tables

Note that workflows 1 and 2 were simply data managers for downloading the reference data.

Workflow 3 - RNA-Seq analysis

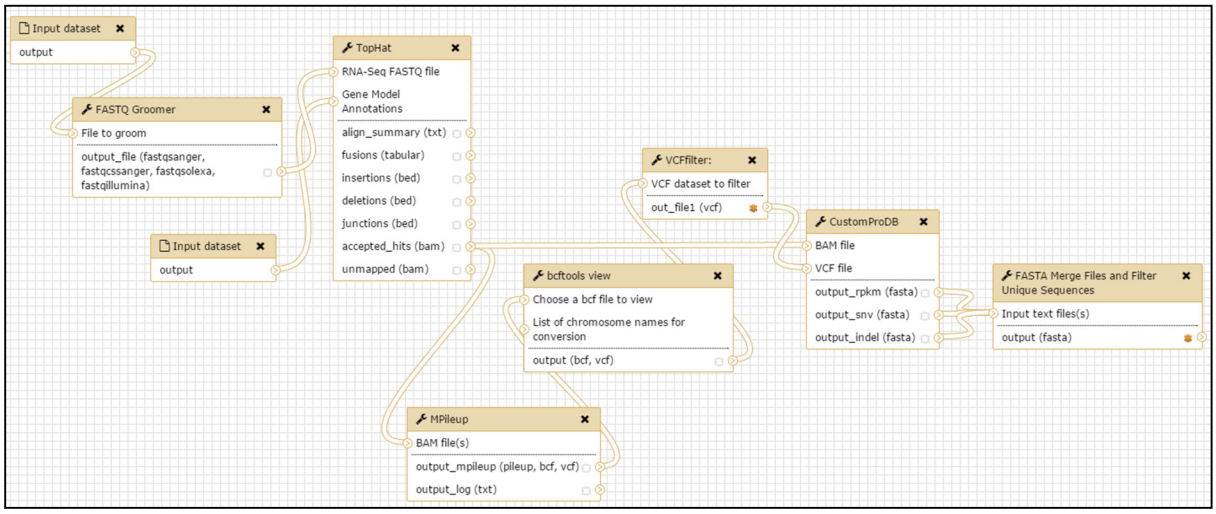

Figure 2. Galaxy workflow view of the Omicron RNA-Seq workflow.

The genomic analysis started at the left with a FASTQ file (raw RNA-Seq reads) for each sample. “FASTQ Groomer” ensured that the input reads were indeed FASTQ format and used Sanger-style quality scores. TopHat2 took the groomed FASTQ and the gene annotations for hg19 and aligned the reads to the genome. The “accepted_hits” output from TopHat2 was used as input to both MPileup (for SNP calling) and customProDB (for filtering by transcript abundance, although in this analysis the filtering is disabled). Bcftools and VCFfilter filtered the MPileup output (variant call file, VCF) to reproduce the filtering criteria described by Zhang et al. The final VCF file was the other input to customProDB, which used the VCF to create variant proteins for each SNP. CustomProDB created a separate FASTA file for reference proteins,

SNP variant proteins, and insertion/deletion variant proteins; the last tool in this workflow combined those FASTA files and filtered out duplicate sequences.

Workflow 4 - simply merged the FASTA files from all samples into a single file. IDPicker needed this combined file to have consistent identifiers for the same proteins.

Workflow 5 - proteomic analysis

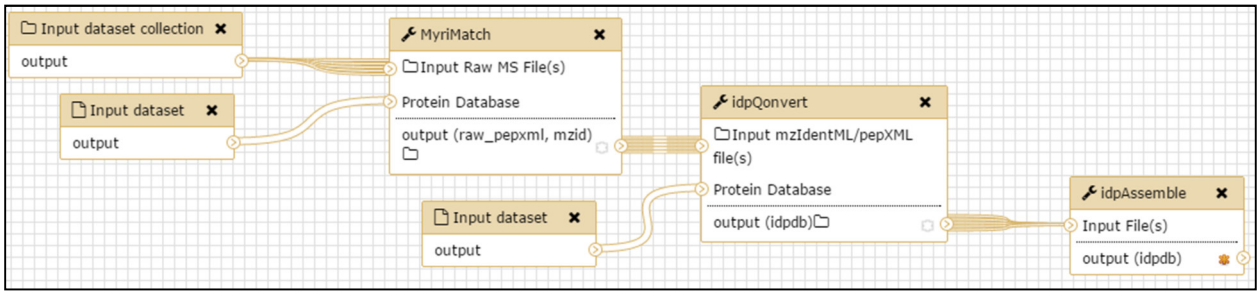

Figure 3. Galaxy workflow view of the Omicron proteomic workflow.

Proteomic tandem mass spectrometry experiments are often fractionated to improve peptide identification in complex samples: each fraction is a separate run on the MS and creates a separate data file. To accommodate this, each sample's fractions were organized into a "dataset collection". Galaxy handles collections as a single unit which improves the user experience when dealing with thousands of files. Each collection was paired with its corresponding protein FASTA from workflow 3. MyriMatch took those pairs as input and matched the tandem mass spectra to peptides in the FASTA. IdpQonvert converted and filtered the peptide-spectrum-matches to a confident subset. As mentioned above, idpQonvert used the combined all-samples FASTA from workflow 4 so that all proteins had consistent identifiers. IdpAssemble merged the idpQonvert output from each fraction (one idpDB file per fraction) so that each sample ended up with a single idpDB for all fractions.

Workflow 6 - pipeline results

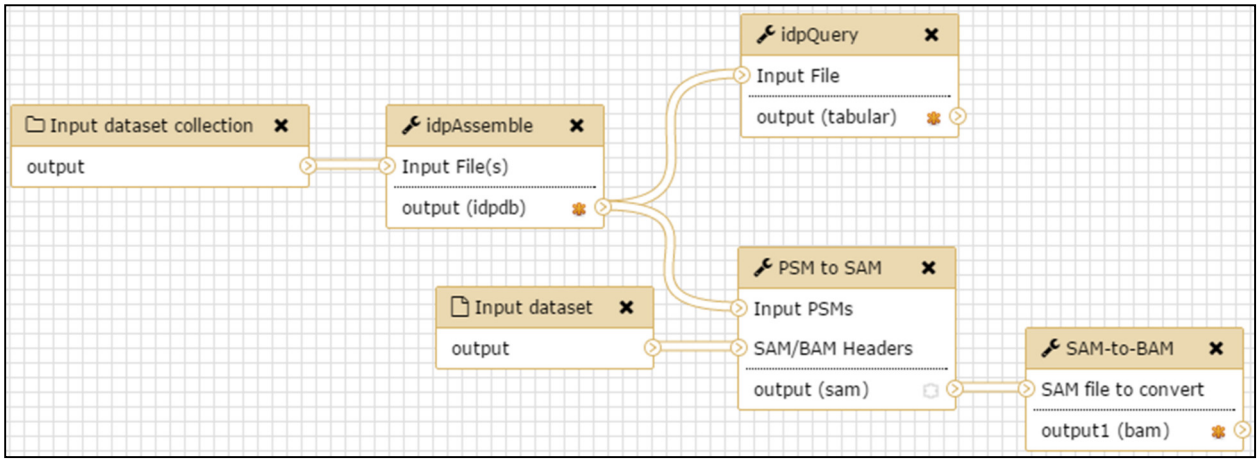

Figure 4. Galaxy workflow view of the Omicron output workflow.

To summarize the results, IdpAssemble merged the collection of idpDBs (one per sample) into a global assembly for the entire dataset. IdpQuery read that assembly to create a table of all peptides with spectral count broken out by sample. IdpQuery could not create a variant-only peptide table, so I used SQL queries outside of Omicron to create that table (Appendix A). I compared the tabular results from this step to the results presented by Zhang et al.

The other branch of this workflow was “PSM to SAM”, which was not part of the original Zhang et a. pipeline. This branch is described below in the **Visualization** section.

Containerized with Docker

Archiving the tools, workflows, and parameters that comprise Omicron was critical to making it easily reproducible. Both containers and VMs could have been used to achieve this archival. I chose Docker because containers were more efficient than VMs - in terms of storage space and computational time - and it allowed me to build on existing work that already made it easy to launch a containerized instance of vanilla Galaxy. Omicron is based on the “galaxy-stable” Docker container developed by Gruening et al. The base container installs a production-ready Galaxy instance: it downloads the latest release of Galaxy (v16.04 for Omicron), sets environment variables, creates a PostgreSQL database, starts an FTP server, and installs

software dependencies. I simply derived Omicron's Dockerfile from the galaxy-stable container and added commands to install the tools for the Zhang et al. pipeline. (Appendix B)

Scalable Omicron cluster

To make it easy for other investigators to launch their own scalable Omicron instances, I created a cluster with AWS' cfncluster tool, extracted its CloudFormation (CFN) template (Appendix D), and simplified the template's parameters to make it easy for non-technical users to launch. It uses an "AutoScalingGroup" to dynamically add and remove new compute nodes according to the number of pending jobs in Galaxy. The template runs a post-install script to install the same packages on the compute nodes that are installed in the Omicron Docker image (e.g., R, customProDB, and PSM to SAM). To allow Galaxy's job system to reliably run thousands of jobs on AWS, I derived another Docker container from the main Omicron container: *Omicron-cfncluster* (Appendix C). This was necessary because the galaxy-stable container had a different version of the SLURM cluster scheduler than the cfncluster template, so it had to be updated for the two components to talk to one another. All of these files are version-controlled on GitHub at: https://github.com/chambm/omicron-galaxy.

Any AWS user can import my template to start an Omicron cluster in just few clicks. Figure 5 shows the sequence to launch Omicron via the CFN template; all the user needs to do is click the blue buttons (but they can tweak some parameters if the defaults are not suitable).

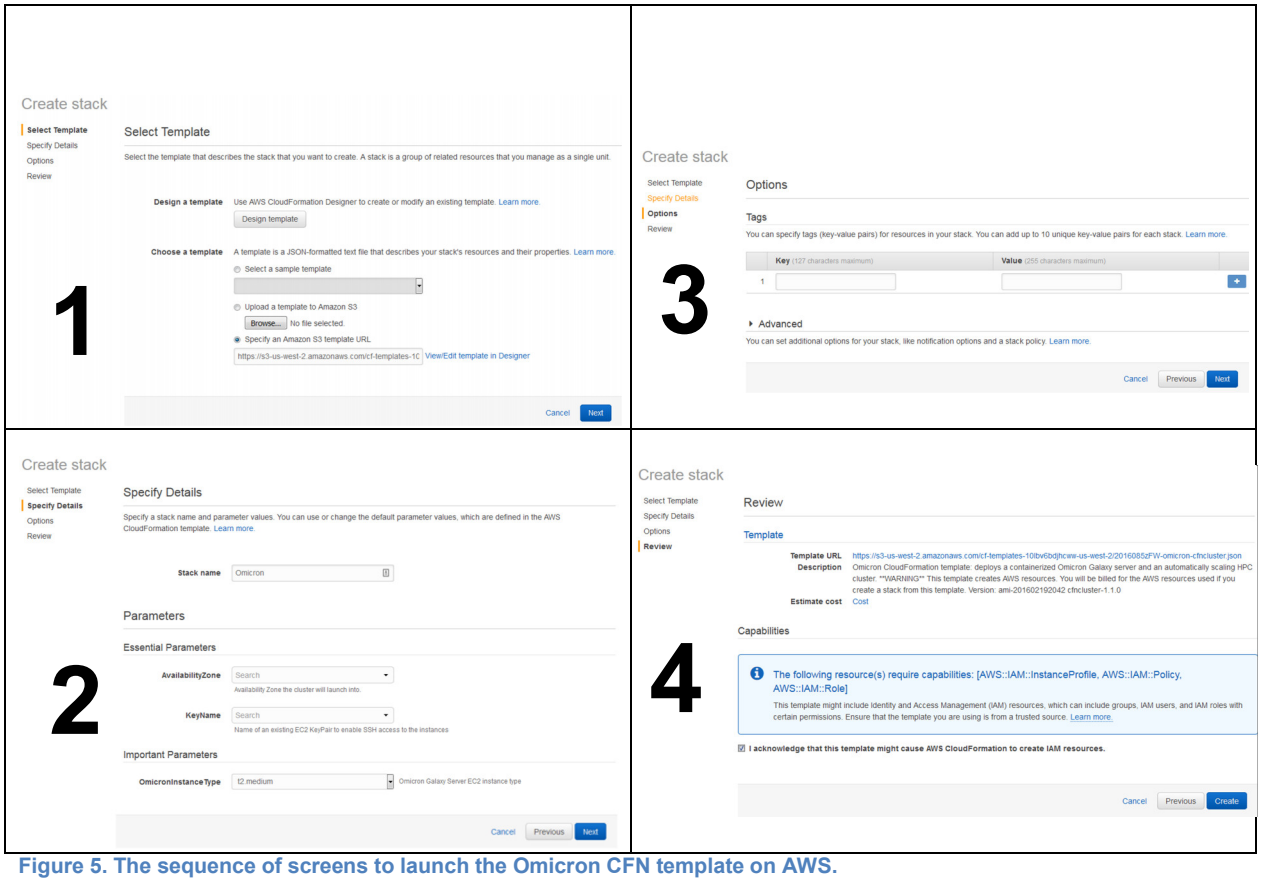

Figure 5. The sequence of screens to launch the Omicron CFN template on AWS.

Figure 6 shows the status screen after a CFN stack is created. An Omicron stack takes about 15 minutes to fully start up. Users would return to this CFN status page to delete the stack when they are done using it. This will, by default, also delete the datasets uploaded to Omicron and generated during analysis. If a user wants to save those datasets, they need to download them locally or detach the volume from the instance.

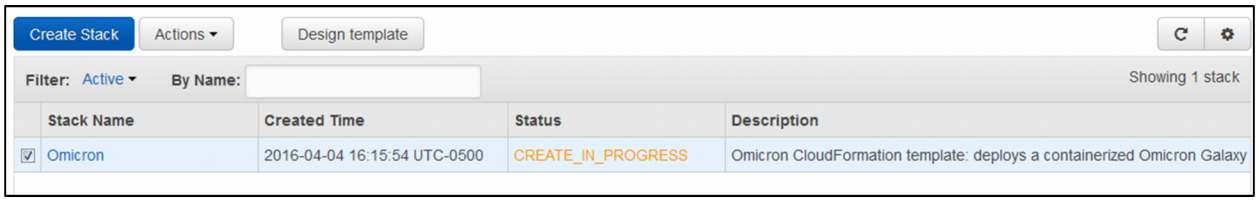

Figure 6. The CFN stack status screen where stacks are created and deleted.

Although several groups (including the Galaxy developers) have created similar ways to simplify deploying a cloud-based Galaxy instance, they either only support a vanilla Galaxy installation or are limited to a single node. [60,61] My approach permits a containerized flavor of Galaxy to run with a scalable cluster of compute nodes “out of the box”.

Data sets

The colorectal cancer proteomic and transcriptomic datasets I used for this study were the same TCGA/CPTAC datasets used by Zhang et al. [49] I downloaded proteomic data for 95 specimens from 90 patients from the CPTAC Data Portal. [62] I downloaded 92 RNA-Seq FASTQ files for 90 patients from a private Zhang Lab file server (genomic data is protected such that users must sign a data use agreement before downloading it from the public repository). Of the 90 patients with transcriptomic data and 90 patients with proteomic data, 4 patients had only proteomic data and 3 patients had only transcriptomic data. I followed the supplemental table from the original paper to choose which duplicate RNA-Seq and proteomic samples to exclude. Thus, like Zhang et al., my pipeline included 86 samples with both data types.

Validation

My primary goal for developing Omicron was to make reproducing an analysis easy for users, so the methods I used for validation are described above. The remaining work was to apply the workflows I developed to reproduce the protein sequence variant analysis described by Zhang et al. I created my Omicron cluster on AWS using the CFN template; Table 2 summarizes the parameters specified for the template.

Parameter Name	Master node	Compute node(s)
Instance type	t2.medium (2 cores, 4 GiB)	m4.xlarge (4 cores, 16 GiB)
Shared volume size	2 TiB	-
Local volume size	100 GiB	100 GiB
Availability zone (AZ)	us-east-1b	-
ScalingEvaluationPeriods	2	-
ScalingPeriod	60	-
ScalingAdjustment (small)	1	-
ScalingThreshold (small)	1	-
ScalingAdjustment (large)	10	-
ScalingThreshold (large)	200	-

Table 2. List of parameters used to start the Omicron validation instance on AWS. Some parameters are global and thus the same between the Master and Compute nodes.

I chose a master node instance type with enough speed and memory to run Galaxy, but cheap enough to run for long periods of time. The compute node type had memory suitable for running genomic tools and enough cores to run multithreaded tools efficiently. The ScalingEvaluationPeriods and ScalingPeriod parameters determined how quickly the Omicron cluster responded to changes in the job queue. The ScalingAdjustment and ScalingThreshold parameters determined the queue lengths which triggered a “scale-up” (i.e., starting up ScalingAdjustment new nodes). There were small and large adjustments. For example, when there were 200 or more pending jobs in the queue for 2 consecutive periods of 60 seconds, the large adjustment was triggered: 10 new nodes were started. When there were between 1 and 200 pending jobs in the queue, the small adjustment would be triggered: 1 new node was started.

I uploaded the 86 RNA-Seq and proteomic samples to the cluster, ran them through my 6 workflows, and downloaded the results. During the analysis I changed from “ondemand” to “spot” instances, which on average were about 10 times cheaper. However, my spot instances were terminated spontaneously if someone else made a higher bid for those resources.

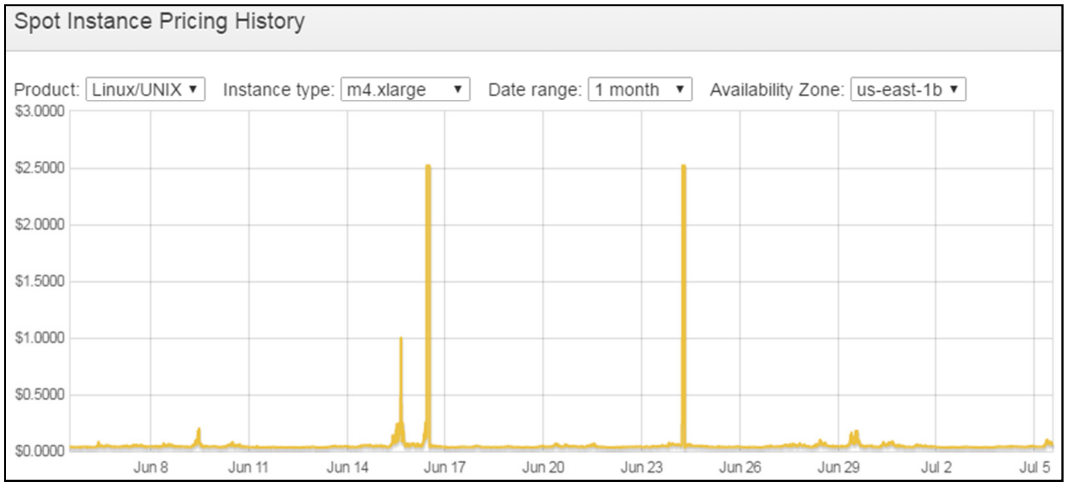

Figure 7. Spot price history for m4.xlarge instances in one availability zone over a 1 month period. The ondemand price of these instances is $0.32/hr, and the average spot price is about $0.05/hr. When using spot pricing, instances would be shut down when the spot price exceeded my maximum bid ($0.42/hr).

I chose $0.42 (per instance-hour) as a reasonably safe bid by evaluating the spot pricing history; if a node was terminated because of a huge bid (Figure 7), Galaxy would requeue the job and run it later when the spot price came down.

Visualization

One output from workflow 6 not included in the original Zhang et al. analysis was a SAM file of the proteomic matches aligned to the genome. Galaxy made it easy export these matches to a genome viewer. The PSM to SAM tool [63] created this SAM file. I modified the PSM to SAM code so that it could read directly from idpDB files. However, my version of PSM to SAM did not include support for mapping variant peptides to the genome - that version is still in development. Nevertheless, I demonstrate this visualization on some non-variant peptides using the Integrative Genome Viewer [59].

Results and discussion

Summary of validation results

The final assembly from Omicron reported 8187 protein groups (a group of proteins evidenced by the same set of peptides) at a false discovery rate (FDR) of 1.3%. It identified

71,842 distinct peptides (ignoring modifications and charge state), 91,593 distinct matches (not ignoring modifications and charge state), and 3,004,484 spectra. Of the 71,842 distinct peptides, 949 only mapped to variant proteins. These variant peptides presented proteomic evidence of the amino acid variations detected in the RNA-Seq data.

Comparison to Zhang et al. variant analysis

Zhang et al. reported similar protein-level numbers: 8352 protein groups with a 1.8% FDR. At the peptide level, results were not as consistent. They identified 74,631 distinct peptides, 95,260 distinct matches, and 3,214,016 spectra. Of the 74,631 distinct peptides, 971 were variant peptides. Table 3 summarizes the comparison:

	Omicron	Zhang et al.
Protein Groups	8187	8352
Filtered Spectra	3,004,484	3,214,016
Distinct Matches	91,593	95,260
Distinct Peptides	71,842	74,631
Variant Peptides	949	971

Table 3. IDPicker summary counts from Omicron and Zhang et al.

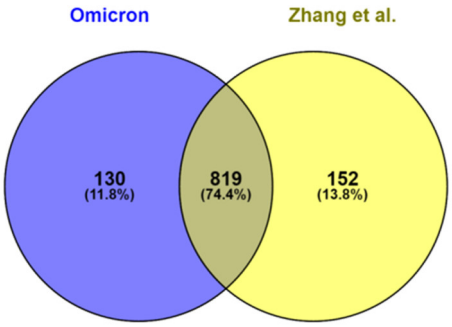

Figure 8. Variant peptide overlap between Omicron and Zhang et al.

By the numbers the variant peptides seem to be a bright spot, but when I compared the actual overlap between the two analyses, only 74% of the peptides were shared (Figure 8).

To understand why there was not as much overlap as expected, I tracked down the IDPicker assembly and protein database created by the Zhang et al. analysis. With access to these, I discovered several reasons for the lack of overlap (Figure 9):

1) Changes to reference protein sequences between 2014 and 2016
   a. 8% found by Zhang et al. but missed by Omicron were in the reference sequences used by Omicron
   b. 1% found by Omicron but missed by Zhang et al. were in the reference sequences used by Zhang et al.
2) Differences in SNP calling from RNA-Seq
   a. 15% found by Zhang et al. but missed by Omicron were not in the variant sequences used by Omicron
   b. 32% found by Omicron but missed by Zhang et al. were not in the variant sequences used by Zhang et al.
3) Differences in proteomic search and assembly
   a. 77% found by Zhang et al. but missed by Omicron were excluded by Omicron's PSM or protein filtering
   b. 67% found by Omicron but missed by Zhang et al. were excluded by the Zhang et al. PSM or protein filtering

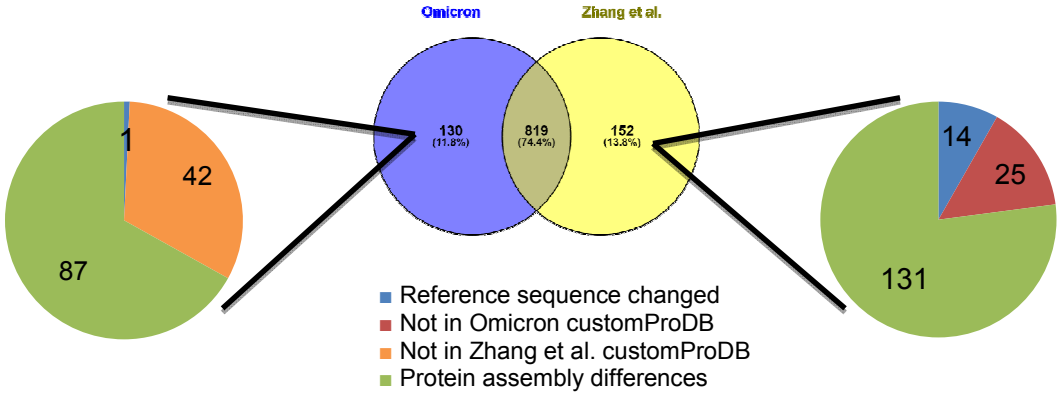

Figure 9. A closer look at the reasons for non-overlapping peptides between Omicron and Zhang et al.

The 15 non-overlapping variant peptides that were missed due to reference sequence changes highlight an important reproducibility problem when an analysis pipeline depends on unversioned reference data downloaded from the Internet (like customProDB does). A paper may report which version of RefSeq and dbSNP were used, but if those versions are no longer available, it may not be possible to faithfully reproduce the analysis: both RefSeq and dbSNP have changed significantly since the Zhang et al. analysis.

A total of 67 missed variant peptides were not in the customProDB variant FASTA due to differences in SNP calling or filtering. These were probably missed because of slight differences between versions of TopHat, samtools, and VCFfilter. For example, the Zhang et al. methods described that many SNP calling parameters were not specified and thus used default values.

Thus, Omicron also left those parameters at default values. If the default values for those parameters changed between tool versions, the results could change.

The majority of the non-overlapping variant peptides were missed due to changes in protein assembly. Exhaustively determining the reasons for these missed peptides was not feasible because the upstream differences (discussed above) influence the protein assembly process. As a simple example, consider if the Zhang et al. analysis had a protein with 2 variant peptides representing 2 different SNPs (and no other peptides). If Omicron missed one of those variant peptides because the SNP was not present in Omicron's customProDB, then that protein would no longer pass the IDPicker filter that requires 2 peptides per protein. Thus, the protein would be filtered out, along with the other variant peptide which actually was identified.

While comparing Omicron's IDPicker assembly to the one from Zhang et al., I discovered several other important differences. The Zhang et al. analysis actually included 91 proteomic samples in its assembly: the 86 that Omicron analyzed, plus the 5 duplicated samples. To eliminate the duplicate samples as a cause for protein assembly differences, I removed them from the Zhang et al. assembly. Table 4 compares the IDPicker summary counts before and after removing the duplicate samples from the Zhang et al. assembly. With the duplicate samples excluded, the summary counts were much closer, but the variant peptide overlap only improved marginally: from 74% to 77% (Figure 10).

	Omicron	Zhang et al. 86 samples	Zhang et al. 91 samples
Protein Groups	8187	8238	8352
Filtered Spectra	3,004,484	3,050,496	3,214,016
Distinct Matches	91,593	92,322	95,260
Distinct Peptides	71,842	72,377	74,631
Variant Peptides	949	942	971

Table 4. IDPicker summary counts before and after removing the duplicate samples.

Comparing the MyriMatch search configurations between the assemblies revealed 2 important differences which would impact the proteomic results. First, Omicron allowed 2 missed tryptic cleavages, while Zhang et al. allowed 4. This meant that the variant peptide GKWERPFEVKDTEEEDFHVDQATTVK, with 3 missed tryptic cleavages, could not have been found by Omicron's MyriMatch search. Second, Omicron allowed 4 dynamic modifications per peptide, while Zhang et al. allowed 3. This meant that the peptide MGQMAMGGAMGINNR, with 4 oxidized methionine sites, could only be found by Omicron. It also meant that the number of peptides Omicron considered for each spectrum (the "search space") would be somewhat larger. However, there were only a couple of variant peptides in the Zhang et al. results with more than 2 missed tryptic cleavages, and no variant peptides at all with 4 dynamic modifications. The difference in search space was probably more significant for peptides with low spectrum counts. In other words, some Zhang et al. variant peptides that only barely passed the IDPicker filters probably did not pass the Omicron IDPicker filters, and vice versa.

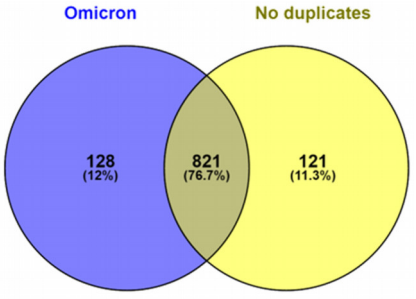

Figure 10. Variant peptide overlap between Omicron and Zhang et al. after removing the duplicate samples.

These results suggest that investigators should be conservative when interpreting a variant analysis. False positive variants could come from flawed reference sequences, random SNP-calling errors (that passed the VCF filtering steps), and random peptide-spectrum-matches (that passed protein assembly filters). Even the true positive variants could be difficult to reproduce if they barely passed the various filters along the pipeline.

Visualization examples

Galaxy has robust integration with genome viewers: users can easily send a BAM file to popular viewers, such as IGV and UCSC, with one click. Figure 11 shows IGV with the RNA-Seq and proteomic BAM alignments shown for the VIM gene.

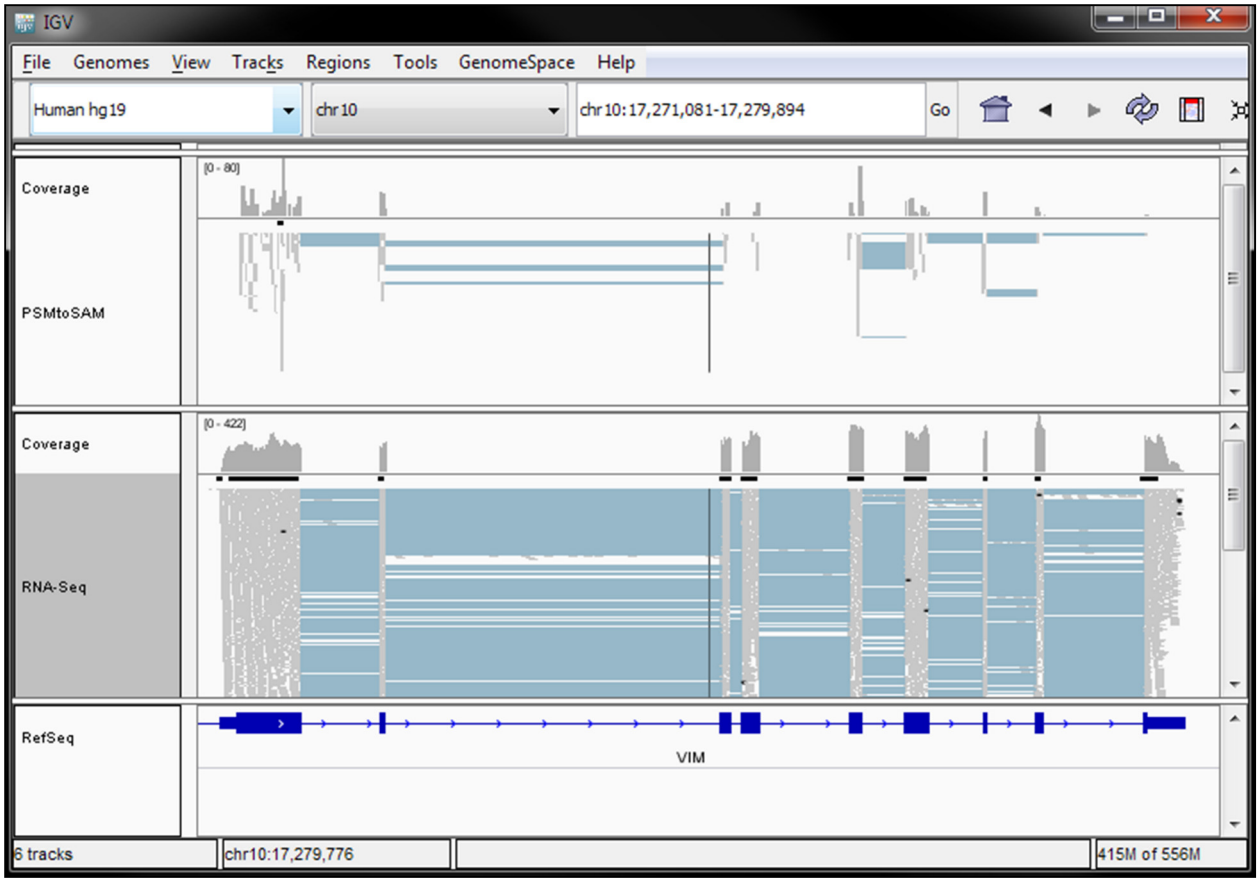

Figure 11. Integrated Genome Viewer showing peptide-spectrum-matches (top) and RNA-Seq reads (bottom) mapping to VIM. This view shows many exon-exon junction peptides.

Cost, performance, and reliability

During the course of running the validation workflow on Omicron-cfncluster, I discovered several reliability and scalability issues. I describe the most significant issues below. Because I fixed these issues as they occurred, I could not completely separate the “development” costs of making those fixes (while running on AWS), from the “production” cost I would expect if the system had been running optimally.

At the beginning of the validation process, the Galaxy job configuration, which tells Galaxy where and how to run jobs (e.g., submitting to a scheduler or running directly on the master node), was not properly configured to deal with certain kinds of job failures. I had not seen those failures in smaller scale tests. When the shared filesystem was overloaded because too many compute nodes were accessing it at the same time, the master node often “timed-out” when

trying to collect the output from completed jobs, and sometimes was not able to even contact the SLURM scheduler to query job status. The default behavior in Galaxy was to treat this kind of job failure as an “ok” result. It did not mark them as errors and did not requeue the job. After changing the job configuration file to treat those events as errors, my workflows proceeded more smoothly. However, this was only a workaround for the root problem.

When accessed by more than about 15 compute nodes (the exact number depended on which tool was running), the cluster’s shared filesystem became unresponsive due to excess load. This was a classic example of a bottleneck in distributed computing. Specialized distributed filesystems have been developed to alleviate this problem such as Lustre [64] and GP-FS [65]. Unfortunately, these solutions were not simple to deploy in an ephemeral cfncluster environment. Instead, I developed custom job scripts for the SLURM scheduler that copied input datasets to compute nodes’ local storage and forced tools to write their output to local storage; after the tool ran, the script copied the output back to the shared volume. Compared to tools directly reading input from and writing their output to the shared filesystem, this approach was significantly more scalable.

Although these setbacks made my validation run suboptimal for estimating cost, I still present the aggregate cost and usage of AWS resources over the 19 day course of the analysis (Figure 12). This chart shows both the total number of instance-hours used per day and the cost per hour. The dramatic shift in cost per hour happened when I switched from on-demand instances to spot instances. The increase in usage happened when all samples were ready to be queued and I increased the maximum number of compute nodes to 90.

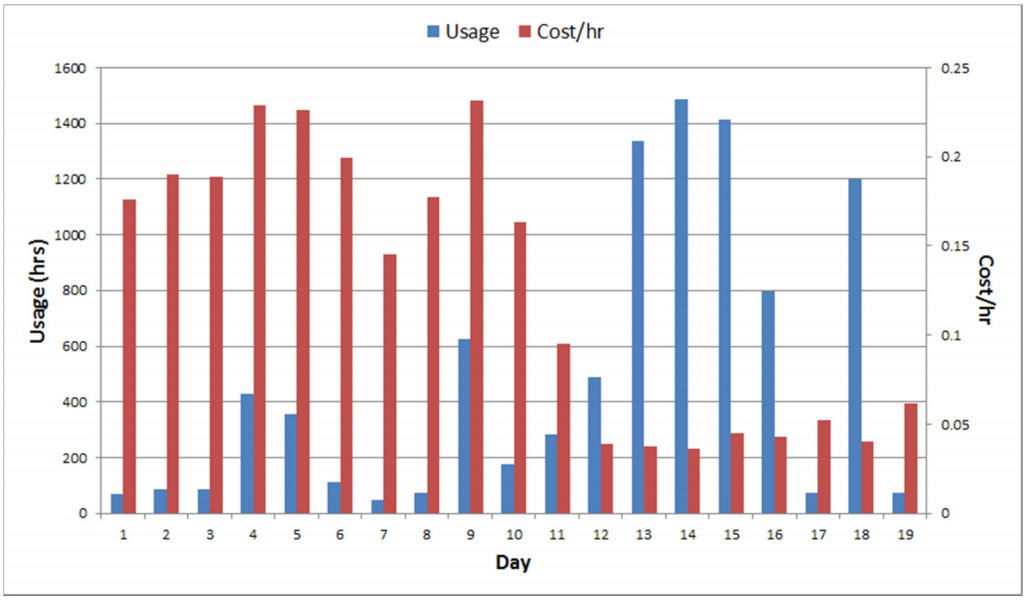

Figure 12. Usage (in hours) and cost per hour during the validation run. Spot pricing makes a huge difference even if some jobs have to be repeated because a node was terminated.

Conclusions, limitations, and future work

Galaxy, Docker, and AWS are standalone technologies that, when combined, allowed me to develop a state-of-the-art solution for reproducing computational analyses for proteogenomics. I admit there are limitations inherent in the implementation. An obvious one is Galaxy's dependence on a POSIX operating system such as Linux or Mac OS X. Tools which only run on Windows and depend on proprietary vendor applications or libraries are problematic for Galaxy to run. The Galaxy team developed a solution (Pulsar), but it requires a separate Windows machine (or VM), and configuring it adds an extra layer of complexity to a Galaxy deployment. If Galaxy was able to run on Windows then the Galaxy node itself could handle those Windows-only jobs, although that configuration would not be scalable.

The proteogenomic workflow that Omicron encapsulates is just an example of what a containerized Galaxy flavor can do. I encourage other investigators doing -omics analyses to adapt Omicron to their own workflows so that anyone can easily reproduce their experiments. To that effect, changing the tools included in Omicron is usually as simple as editing the Dockerfile's list of ToolShed packages. If a tool does not yet have a wrapper in the ToolShed,

then creating a Galaxy tool wrapper for it is not much harder than creating shell scripts or batch files - scripts that -omics investigators (or their students) already routinely make.

The Omicron CloudFormation template makes it relatively simple for any investigator to deploy a scalable Omicron cluster to AWS: much simpler than any existing approach I am aware of. An investigator can create an AWS account, set up their payment information, and launch my template in a few dozen clicks: no command-line or scripting required. The template sets up the AWS resources needed for the Omicron instance to be able to run compute jobs on a variable number of compute node instances. These resources are specific to the investigator's AWS account, so any number of different investigators can use the template at one time. It is also possible to allow multiple users access to an Omicron instance (with some extra configuration).

Although the template allows easy deployment on AWS, it does not necessarily deploy easily to other cloud providers (e.g. Azure, Rackspace, CloudSigma). Several commercial and open projects try to provide vendor-agnostic templates to configure infrastructure-as-a-service clouds (e.g. OpenStack Heat), but I found no mature, open projects that could do both autoscaling and containerization. OpenStack Heat supposedly is backward compatible with CloudFormation templates, but I have not yet tested on an OpenStack-compatible cloud.

There are still some scalability issues with the shared filesystem in my cluster system. The most significant scalability problem that remains is that some Galaxy tools index input before processing it. For example, TopHat indexes gene annotation (GTF) input and MyriMatch indexes protein FASTA input. Because Galaxy jobs are isolated from each other, every job has to recompute these indexes. The combined customProDB protein FASTAs from all samples contains over 200,000 proteins; indexing it takes a few minutes and the resulting index is almost a gigabyte. The appropriate solution to this is to have Galaxy index these files once and copy both the file and its index to the nodes' local storage. Alternatively, AWS recently released their "Elastic Filesystem" (EFS) to the public, which should function similarly to Lustre and GP-FS,

but integrate much more cleanly and easily with other AWS resources. Space on EFS is more expensive than standard AWS disk storage (“Elastic Block Store”), but the extra time saved in running workflows may compensate for that. To test this, I will add EFS support to Omicron in the future.

Ideally, each proteogenomics study published would include an Omicron specifically tailored to run it, with the workflows and tool parameters configured exactly as they were in the study. Unfortunately, the input and reference data for -omics is almost always too big to practically store it in a Docker container (the “small archive” advantage would certainly be lost). In the future, tailored Omicron instances could include downloading the data as the first step of the workflow. Omicron could be configured to download predefined datasets from online repositories, such as the CPTAC Data Portal. However, it would be challenging to make this work for data that is considered to contain protected health information, such as RNA-Seq of human subjects.

The CloudFormation template I created can be easily modified to point at a different containerized Galaxy flavor. Thus, I also encourage other investigators to modify the template to make it easy to launch and reproduce their experiments in the cloud. When a vendor-agnostic cloud template matures, the Omicron template will migrate to that.

References

1. Nekrutenko A, Taylor J. Next-generation sequencing data interpretation: enhancing reproducibility and accessibility. Nat Rev Genet. 2012;13: 667–672.
2. Giardine B, Riemer C, Hardison RC, Burhans R, Elnitski L, Shah P, et al. Galaxy: a platform for interactive large-scale genome analysis. Genome Res. 2005;15: 1451–1455.
3. Goodstadt L. Ruffus: a lightweight Python library for computational pipelines. Bioinformatics. 2010;26: 2778–2779.
4. Köster J, Rahmann S. Snakemake--a scalable bioinformatics workflow engine. Bioinformatics. 2012;28: 2520–2522.
5. Fisch KM, Meißner T, Gioia L, Ducom J-C, Carland TM, Loguercio S, et al. Omics Pipe: a community-based framework for reproducible multi-omics data analysis. Bioinformatics. 2015;31: 1724–1728.
6. Sadedin SP, Pope B, Oshlack A. Bpipe: a tool for running and managing bioinformatics pipelines. Bioinformatics. 2012;28: 1525–1526.
7. Li Y, Wang X, Cho J-H, Shaw TI, Wu Z, Bai B, et al. JUMPg: An Integrative Proteogenomics Pipeline Identifying Unannotated Proteins in Human Brain and Cancer Cells. J Proteome Res. 2016;15: 2309–2320.
8. Gregersen N, Niels G, Lars B, Peter B. Protein Misfolding, Aggregation, and Degradation in Disease. Protein Misfolding and Disease. pp. 3–16.
9. Wang J, Zhang K, Wang J, Wu X, Liu X, Li B, et al. Underexpression of LKB1 tumor suppressor is associated with enhanced Wnt signaling and malignant characteristics of human intrahepatic cholangiocarcinoma. Oncotarget. 2015;6: 18905–18920.
10. Elcheva I, Tarapore RS, Bhatia N, Spiegelman VS. Overexpression of mRNA-binding protein CRD-BP in malignant melanomas. Oncogene. 2008;27: 5069–5074.
11. Abbott A, Alison A. Biomarkers could predict Alzheimer's before it starts. Nature. 2014; doi:10.1038/nature.2014.14834
12. Andersen CY, Yding Andersen C. Levels of Steroid-Binding Proteins and Steroids in Human Preovulatory Follicle Fluid and Serum as Predictors of Success in in Vitro Fertilization-Embryo Transfer Treatment*. J Clin Endocrinol Metab. 1990;71: 1375–1381.
13. Tabb DL, Vega-Montoto L, Rudnick PA, Variyath AM, Ham A-JL, Bunk DM, et al. Repeatability and reproducibility in proteomic identifications by liquid chromatography-tandem mass spectrometry. J Proteome Res. 2010;9: 761–776.
14. Jagtap PD, Johnson JE, Onsongo G, Sadler FW, Murray K, Wang Y, et al. Flexible and accessible workflows for improved proteogenomic analysis using the Galaxy framework. J Proteome Res. 2014;13: 5898–5908.
15. Li J, Su Z, Ma Z-Q, Slebos RJC, Halvey P, Tabb DL, et al. A bioinformatics workflow for variant peptide detection in shotgun proteomics. Mol Cell Proteomics. 2011;10: M110.006536.
16. Wang X, Slebos RJC, Wang D, Halvey PJ, Tabb DL, Liebler DC, et al. Protein identification using customized protein sequence databases derived from RNA-Seq data. J Proteome Res. 2012;11: 1009–1017.
17. Evans VC, Barker G, Heesom KJ, Fan J, Bessant C, Matthews DA. De novo derivation of proteomes from transcriptomes for transcript and protein identification. Nat Methods. 2012;9: 1207–1211.
18. Sheynkman GM, Shortreed MR, Frey BL, Smith LM. Discovery and mass spectrometric analysis of novel splice-junction peptides using RNA-Seq. Mol Cell Proteomics. 2013;12: 2341–2353.
19. Risk BA, Spitzer WJ, Giddings MC. Peppy: Proteogenomic Search Software. J Proteome Res. 2013;12: 3019–3025.
20. Prabakaran S, Hemberg M, Chauhan R, Winter D, Tweedie-Cullen RY, Dittrich C, et al. Quantitative profiling of peptides from RNAs classified as noncoding. Nat Commun. 2014;5: 5429.
21. Woo S, Cha SW, Merrihew G, He Y, Castellana N, Guest C, et al. Proteogenomic database construction driven from large scale RNA-seq data. J Proteome Res. 2014;13: 21–28.

22. Zickmann F, Renard BY. MSProGene: integrative proteogenomics beyond six-frames and single nucleotide polymorphisms. Bioinformatics. 2015;31: i106–15.
23. Ruggles KV, Tang Z, Wang X, Grover H, Askenazi M, Teubl J, et al. An Analysis of the Sensitivity of Proteogenomic Mapping of Somatic Mutations and Novel Splicing Events in Cancer. Mol Cell Proteomics. 2016;15: 1060–1071.
24. Wang X, Zhang B. customProDB: an R package to generate customized protein databases from RNA-Seq data for proteomics search. Bioinformatics. 2013;29: 3235–3237.
25. Sheynkman GM, Johnson JE, Jagtap PD, Shortreed MR, Getiria O, Frey BL, et al. Using Galaxy-P to leverage RNA-Seq for the discovery of novel protein variations. BMC Genomics. 2014;15: 703.
26. Gent IP. The Recomputation Manifesto [Internet]. 12 Apr 2013 [cited 10 Mar 2016]. Available: http://recomputation.org/sites/default/files/Manifesto1_9479.pdf
27. Fan J, Saha S, Barker G, Heesom KJ, Ghali F, Jones AR, et al. Galaxy Integrated Omics: Web-based Standards-Compliant Workflows for Proteomics Informed by Transcriptomics. Mol Cell Proteomics. 2015;14: 3087–3093.
28. Goecks J, Jeremy G, Anton N, James T, The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. 2010;11: R86.
29. Aranguren ME, Wilkinson MD. Enhanced reproducibility of SADI web service workflows with Galaxy and Docker. Gigascience. 2015;4: 59.
30. Boekel J, Chilton JM, Cooke IR, Horvatovich PL, Jagtap PD, Käll L, et al. Multi-omic data analysis using Galaxy. Nat Biotechnol. 2015;33: 137–139.
31. Taghiyar MJ, Jafar Taghiyar M, Jamie R, Diljot G, Bruno G, Radhouane A, et al. Kronos: a workflow assembler for genome analytics and informatics [Internet]. 2016. doi:10.1101/040352
32. Köster J, Rahmann S. Snakemake--a scalable bioinformatics workflow engine. Bioinformatics. 2012;28: 2520–2522.
33. Sadedin SP, Pope B, Oshlack A. Bpipe: a tool for running and managing bioinformatics pipelines. Bioinformatics. 2012;28: 1525–1526.
34. Goodstadt L. Ruffus: a lightweight Python library for computational pipelines. Bioinformatics. 2010;26: 2778–2779.
35. Fisch KM, Meißner T, Gioia L, Ducom J-C, Carland TM, Loguercio S, et al. Omics Pipe: a community-based framework for reproducible multi-omics data analysis. Bioinformatics. 2015;31: 1724–1728.
36. Boettiger C. An introduction to Docker for reproducible research. Oper Syst Rev. 2015;49: 71–79.
37. Blankenberg D, Von Kuster G, Bouvier E, Baker D, Afgan E, Stoler N, et al. Dissemination of scientific software with Galaxy ToolShed. Genome Biol. 2014;15: 403.
38. Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, et al. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Res. 2013;41: W557–61.
39. Virtual machine - Wikipedia, the free encyclopedia [Internet]. [cited 8 Mar 2016]. Available: https://en.wikipedia.org/wiki/Virtual_machine#System_virtual_machines
40. Davidson RL, Weber RJM, Liu H, Sharma-Oates A, Viant MR. Galaxy-M: a Galaxy workflow for processing and analyzing direct infusion and liquid chromatography mass spectrometry-based metabolomics data. Gigascience. 2016;5: 10.
41. Moorhouse MJ, van Zessen D, IJspeert H, Hiltemann S, Horsman S, van der Spek PJ, et al. ImmunoGlobulin galaxy (IGGalaxy) for simple determination and quantitation of immunoglobulin heavy chain rearrangements from NGS. BMC Immunol. 2014;15: 59.
42. Lu H, Papathomas TG, van Zessen D, Palli I, de Krijger RR, van der Spek PJ, et al. Automated Selection of Hotspots (ASH): enhanced automated segmentation and adaptive step finding for Ki67 hotspot detection in adrenal cortical cancer. Diagn Pathol. 2014;9: 216.
43. Anderson C, Charles A. Docker [Software engineering]. IEEE Softw. 2015;32: 102–c3.
44. Bjoern Gruening. bgruening/docker-galaxy-stable. In: GitHub [Internet]. [cited 28 Jan 2016]. Available: https://github.com/bgruening/docker-galaxy-stable

45. STAR: Cluster - Home [Internet]. [cited 23 Mar 2016]. Available: http://star.mit.edu/cluster/
46. CfnCluster. In: Amazon Web Services, Inc. [Internet]. [cited 23 Mar 2016]. Available: http://aws.amazon.com/hpc/cfncluster/
47. Guides — kronos 2.1.0 documentation [Internet]. [cited 23 Mar 2016]. Available: http://kronos.readthedocs.org/en/latest/guides/guides.html#guide-for-deploying-kronos-to-the-cloud
48. Central Proteomics Facilities Pipeline / Code / [f52e86] /CLOUD_INSTALL [Internet]. [cited 23 Mar 2016]. Available: https://sourceforge.net/p/cpfp/code/ci/master/tree/CLOUD_INSTALL
49. Zhang B, Wang J, Wang X, Zhu J, Liu Q, Shi Z, et al. Proteogenomic characterization of human colon and rectal cancer. Nature. 2014;513: 382–387.
50. R Core Team. R: A Language and Environment for Statistical Computing [Internet]. Vienna, Austria: R Foundation for Statistical Computing; 2016. Available: https://www.R-project.org
51. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009;25: 2078–2079.
52. Blankenberg D, Gordon A, Von Kuster G, Coraor N, Taylor J, Nekrutenko A, et al. Manipulation of FASTQ data with Galaxy. Bioinformatics. Oxford University Press; 2010;26: 1783–1785.
53. galaxyproject. galaxyproject/planemo. In: GitHub [Internet]. [cited 24 Mar 2016]. Available: https://github.com/galaxyproject/planemo
54. Kim D, Pertea G, Trapnell C, Pimentel H, Kelley R, Salzberg SL. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. Genome Biol. BioMed Central; 2013;14: 1.
55. vcflib. vcflib/vcflib. In: GitHub [Internet]. [cited 6 Jul 2016]. Available: https://github.com/vcflib/vcflib
56. Tabb DL, Fernando CG, Chambers MC. MyriMatch: highly accurate tandem mass spectral peptide identification by multivariate hypergeometric analysis. J Proteome Res. 2007;6: 654–661.
57. Ma Z-Q, Ze-Qiang M, Surendra D, Chambers MC, Litton MD, Sobecki SM, et al. IDPicker 2.0: Improved Protein Assembly with High Discrimination Peptide Identification Filtering. J Proteome Res. 2009;8: 3872–3881.
58. Wang X, Slebos RJC, Chambers MC, Tabb DL, Liebler DC, Zhang B. proBAMsuite, a Bioinformatics Framework for Genome-Based Representation and Analysis of Proteomics Data. Mol Cell Proteomics. 2016;15: 1164–1175.
59. Thorvaldsdóttir H, Robinson JT, Mesirov JP. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. Brief Bioinform. 2013;14: 178–192.
60. Afgan E, Enis A, Dannon B, Nate C, Brad C, Anton N, et al. Galaxy CloudMan: delivering cloud compute clusters. BMC Bioinformatics. 2010;11: S4.
61. Afgan E, Chapman B, Jadan M, Franke V, Taylor J. Using cloud computing infrastructure with CloudBioLinux, CloudMan, and Galaxy. Curr Protoc Bioinformatics. 2012;Chapter 11: Unit11.9.
62. CPTAC Data Portal [Internet]. [cited 4 Jul 2016]. Available: https://cptac-data-portal.georgetown.edu/cptacPublic/
63. Wang X, Slebos RJC, Chambers MC, Tabb DL, Liebler DC, Zhang B. proBAMsuite, a Bioinformatics Framework for Genome-Based Representation and Analysis of Proteomics Data. Mol Cell Proteomics. ASBMB; 2016;15: 1164–1175.
64. Miller FP, Vandome AF, John M. Lustre (File System). Alphascript Publishing; 2010.
65. Quintero D, Bolinches L, Chaudhary P, Davis W, Duersch S, Fachim CH, et al. IBM Spectrum Scale (formerly GPFS). IBM Redbooks; 2015.

Appendix

A. Variant Peptide SQL statements

```
DROP TABLE IF EXISTS RefSeqProtein;
DROP TABLE IF EXISTS VariantProtein;
CREATE TABLE RefSeqProtein AS SELECT Id, Accession FROM UnfilteredProtein WHERE regexp(".*NP_[0-9]+$", Accession);
CREATE TABLE VariantProtein AS SELECT Id, Accession FROM UnfilteredProtein WHERE regexp(".*NP_[0-9]+_[^>*]+$", Accession);
CREATE INDEX RefSeqProtein_Id ON RefSeqProtein (Id);
CREATE INDEX VariantProtein_Id ON VariantProtein (Id);


DROP TABLE IF EXISTS VariantPeptide;
CREATE TABLE VariantPeptide AS SELECT pep.Id AS Id, pi.Offset AS Offset, IFNULL(SUBSTR(pd.Sequence, pi.Offset+1, pi.Length), pep.DecoySequence) AS Peptide,
GROUP_CONCAT(DISTINCT vp.Accession) AS VariantProteins, GROUP_CONCAT(DISTINCT rsp.Accession) AS RefSeqProteins, GROUP_CONCAT(DISTINCT pro.Accession) AS AllProteins
FROM UnfilteredProtein pro
JOIN UnfilteredPeptideInstance pi ON pro.Id=pi.Protein
LEFT JOIN VariantProtein vp ON pro.Id=vp.Id
LEFT JOIN RefSeqProtein rsp ON pro.Id=rsp.Id
LEFT JOIN ProteinData pd ON pro.Id=pd.Id
JOIN Peptide pep ON pi.Peptide=pep.Id
GROUP BY pep.Id;
CREATE INDEX VariantPeptide_Id ON VariantPeptide (Id);


DELETE FROM PeptideSpectrumMatch WHERE Peptide NOT IN (SELECT Id FROM VariantPeptide vp WHERE RefSeqProteins IS null AND VariantProteins IS NOT null);
DELETE FROM Spectrum WHERE Id NOT IN (SELECT DISTINCT Spectrum FROM PeptideSpectrumMatch);
DELETE FROM Peptide WHERE Id NOT IN (SELECT DISTINCT Peptide FROM PeptideSpectrumMatch);
DELETE FROM PeptideInstance WHERE Peptide NOT IN (SELECT DISTINCT Id FROM Peptide);
DELETE FROM Protein WHERE Id NOT IN (SELECT DISTINCT Protein FROM PeptideInstance);


SELECT pep.Id, IFNULL(SUBSTR(pd.Sequence, pi.Offset+1, pi.Length), pep.DecoySequence) AS Peptide
, GROUP_CONCAT(DISTINCT pi.Offset) AS StartPos
, COUNT(DISTINCT psm.Spectrum) AS FilteredSpectra
, VariantProteins
FROM PeptideSpectrumMatch psm
JOIN PeptideInstance pi ON psm.Peptide=pi.Peptide
JOIN Peptide pep ON psm.Peptide=pep.Id
JOIN Protein pro ON pi.Protein=pro.Id
LEFT JOIN ProteinData pd ON pro.Id=pd.Id
GROUP BY pep.Id
ORDER BY FilteredSpectra DESC
```

B. Omicron base Dockerfile

```
# Galaxy - Omicron
#
# VERSION       Galaxy-central


FROM chambm/docker-galaxy-stable


MAINTAINER Matt Chambers, matt.chambers@vanderbilt.edu


# Install various tool and development requirements
# 1. Add R-dev PPA and its public key (then 'apt-get update')
# 2. Install linuxbrew requirements
# 3. Install python-h5py (for mz5 support in Galaxy)
# 4. Install R-3.2 from R-dev PPA
# 5. Cleanup
RUN codename=$(lsb_release -c -s) && \
   echo "deb http://mirrors.nics.utk.edu/cran/bin/linux/ubuntu $codename/" | tee -a /etc/apt/sources.list && \
   apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E084DAB9 && \
   echo "deb http://ppa.launchpad.net/marutter/rdev/ubuntu $codename main" | tee -a /etc/apt/sources.list.d/marutter-rdev-$codename.list && \
   apt-get -qq update && \
   apt-get install -y --no-install-recommends r-base r-cran-rjson cabextract build-essential curl git m4 ruby texinfo libbz2-dev libcurl4-openssl-dev libxml2-dev
libexpat-dev libncurses-dev zlib1g-dev libhdf5-dev python-h5py libreadline-dev && \
   sudo -u galaxy ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/linuxbrew/go/install)" && \
   echo export PATH=$PATH:/home/galaxy/.linuxbrew/bin >> /home/galaxy/.bashrc && \
   apt-get purge -y software-properties-common && \
   apt-get autoremove -y && apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*


# Install BioConductor and proBAMr
RUN R --vanilla -e 'source("http://bioconductor.org/biocLite.R"); biocLite(ask=F)' && \
   R --vanilla -e 'source("http://bioconductor.org/biocLite.R"); biocLite(c("RGalaxy", "proBAMr", "customProDB"), ask=F)'


# Note: do not install files to volumes from the Dockerfile; they won't persist
ENV GALAXY_CONFIG_TOOL_DEPENDENCY_DIR=/galaxy-central/tool_deps \
   GALAXY_DESTINATIONS_DEFAULT=slurm_cluster


# Install standard Omicron tools into Galaxy
RUN install-repository "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name data_manager_sam_fasta_index_builder" \
                       "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name data_manager_fetch_genome_all_fasta" \
                       "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name samtools_mpileup" \
                       "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name sam_to_bam" \
                       "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name sam_merge"


# Test that the tools are persisting
```

```
RUN stat $GALAXY_ROOT/../shed_tools/*


# Install custom Omicron tools into Galaxy
RUN add-tool-shed --url 'http://testtoolshed.g2.bx.psu.edu/' --name 'Test Tool Shed' && \
   install-repository "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name myrimatch" \
                      "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name idpqonvert" \
                      "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name idpassemble" \
                      "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name idpquery" \
                      "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name msconvert" \
                      "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name psm_to_sam" \
                      "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name custom_pro_db" \
                      "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name custom_pro_db_annotation_data_manager"


RUN install-repository "--url http://testtoolshed.g2.bx.psu.edu/ -o galaxyp --name msgfplus" \
                      "--url http://toolshed.g2.bx.psu.edu/ -o galaxyp --name fasta_merge_files_and_filter_unique_sequences" \
                      "--url http://toolshed.g2.bx.psu.edu/ -o mandorodriguez --name concat_fasta_files" \
                      "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name tophat2" \
                      "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name fastq_groomer" \
                      "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name data_manager_bowtie2_index_builder" \
                      "--url http://toolshed.g2.bx.psu.edu/ -o iuc --name bcftools" \
                      "--url http://toolshed.g2.bx.psu.edu/ -o devteam --name vcffilter"


# The following commands will be executed as user galaxy
USER galaxy


# BASH_ENV tells bash what to do for non-interactive logins
ADD .bashrc /home/galaxy/.bashrc
ENV PATH=$PATH:/home/galaxy/.linuxbrew/bin BASH_ENV=/home/galaxy/.bashrc


# Install planemo (will be installed in linuxbrew's bin directory)
RUN brew tap galaxyproject/tap && brew install planemo --HEAD


ENV GALAXY_CONFIG_BRAND="Omicron" \
   ENABLE_TTS_INSTALL=True


WORKDIR $GALAXY_ROOT
USER root


COPY *.ga /tmp/
COPY import_workflow.sh /usr/bin/import-workflow
RUN import-workflow /tmp/Omicron-Workflow-2_Index_hg19_genome.ga \
                    /tmp/Omicron-Workflow-1_Download_hg19_reference_genome.ga


# Mark folders as imported from the host.
VOLUME ["/export/", "/data/", "/var/lib/docker"]


# Expose port 80 (webserver), 21 (FTP server), 8800 (Proxy), 9009 (toolshed)
EXPOSE :80 :21 :8800 :9009


# Autostart script that is invoked during container start
CMD ["/usr/bin/startup"]
```

C. Omicron cfncluster Dockerfile

```
# Galaxy Omicron - CfnCluster edition
#
# VERSION       Galaxy-central


FROM chambm/omicron-galaxy


MAINTAINER Matt Chambers, matt.chambers@vanderbilt.edu


RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 5E25F516B04C661B A6BFD95994EE84A6 && \
   echo "deb http://ppa.launchpad.net/mertes/slurm/ubuntu trusty main" | tee -a /etc/apt/sources.list.d/slurm-trusty.list && \
   apt-get update -q && \
   apt-get autoremove -y slurm-llnl slurm-drmaa1 && \
   apt-get install -y slurm=15-08-7-1-ppa2+2004 libslurm-dev=15-08-7-1-ppa2+2004 && \
   apt-get autoremove -y && apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* && \
   echo OPTIONS=\"--force\" >> /etc/default/munge


ENV NONUSE=reports,slurmd,slurmctld \
   SLURM_CONF=/opt/slurm/etc/slurm.conf \
   GALAXY_CONFIG_ALLOW_USER_DATASET_PURGE=True \
   GALAXY_CONFIG_ENABLE_QUOTAS=True \
   GALAXY_CONFIG_NGINX_UPLOAD_STORE=/export/nginx_upload_store


RUN mkdir /src && cd /src && curl http://apps.man.poznan.pl/trac/slurm-drmaa/downloads/9 | tar xz && \
   cd /src/slurm-drmaa-1.0.7 && \
   ./configure && make && make install && \
   rm -fr /src


ADD job_conf.xml $GALAXY_CONFIG_JOB_CONFIG_FILE
```

```
ENV DRMAA_LIBRARY_PATH=/usr/local/lib/libdrmaa.so


# 1. Set passive port range for proFTPd (this port range must match the ports opened by the CFN template)
# 2. Add 'Input is VCF' option to the bcftools_view tool
# 3. Change upload directory in nginx.conf
# 4. Make TopHat more tolerant of non-fatal errors during execution; only non-zero exit code is an error
RUN sed -i.bak 's/^\(PassivePorts\).*/\1 30000 30100/' /etc/proftpd/proftpd.conf && \
   sed -i.bak 's/$varsitesonly/$S $varsitesonly/' $GALAXY_ROOT/../shed_tools/toolshed.g2.bx.psu.edu/repos/iuc/bcftools/e5e78d1e09a6/bcftools/bcftools_view.xml && \
   sed -i.bak 's/\(        <param name="input" type="data" format="bcf,vcf" label="Choose a bcf file to view" \/>\)/\1\n        <param name="S" type="boolean"
truevalue="-S" falsevalue="" checked="False" label="Input is VCF" help="-S" \/>/'
$GALAXY_ROOT/../shed_tools/toolshed.g2.bx.psu.edu/repos/iuc/bcftools/e5e78d1e09a6/bcftools/bcftools_view.xml && \
   sed -i.bak "s/upload_store \/tmp\/nginx_upload_store/upload_store \/export\/nginx_upload_store/" /etc/nginx/nginx.conf && \
   sed -i.bak 's/\(\s*\)\(<regex match="Exception|Error" source="both"\) level="fatal" description="Tool execution failed"/\1<exit_code range="1:" level="fatal"
description="Tool execution failed" \/>\n\1\2 level="log" description="Warning"/'
$GALAXY_ROOT/../shed_tools/toolshed.g2.bx.psu.edu/repos/devteam/tophat2/758594ed0364/tophat2/tophat2_wrapper.xml


# Make Galaxy venv relocatable and use copies instead of  links
RUN virtualenv --always-copy --relocatable $GALAXY_VIRTUAL_ENV


VOLUME ["/export/", "/data/", "/vakr/lib/docker"]


# Expose port 80 (webserver), 21 (FTP server), 8800 (Proxy), 9009 (toolshed)
EXPOSE :80 :21 :8800 :9009


# Autostart script that is invoked during container start
CMD ["/usr/bin/startup"]
```

D. Omicron-cfncluster CloudFormation template

```
{
 "AWSTemplateFormatVersion" : "2010-09-09",
 "Description" : "Omicron CloudFormation template: deploys a containerized Omicron Galaxy server and an automatically scaling HPC cluster. **WARNING** This template
creates AWS resources. You will be billed for the AWS resources used if you create a stack from this template. Version: ami-201602192042 cfncluster-1.1.0",
 "Parameters" : {
   "KeyName" : {
     "Description" : "Name of an existing EC2 KeyPair to enable SSH access to the instances",
     "Type" : "AWS::EC2::KeyPair::KeyName"
   },
   "AvailabilityZone" : {
     "Description" : "Availability Zone the cluster will launch into.",
     "Type" : "AWS::EC2::AvailabilityZone::Name"
   },
   "OmicronDockerImage" : {
     "Description" : "Omicron Galaxy Docker Image",
     "Type" : "String",
     "Default" : "chambm/omicron-cfncluster:latest"
   },
   "MasterInstanceType" : {
     "Description" : "EC2 instance type used for the HPC cluster's master node",
     "Type" : "String",
     "Default" : "t2.medium",
     "ConstraintDescription" : "must be a valid EC2 instance type.",
     "AllowedValues" : [
       "cc2.8xlarge",
       "c3.8xlarge",
       "c3.4xlarge",
       "c3.2xlarge",
       "c3.xlarge",
       "c3.large",
       "c4.8xlarge",
       "c4.4xlarge",
       "c4.2xlarge",
       "c4.xlarge",
       "c4.large",
       "r3.8xlarge",
       "r3.4xlarge",
       "r3.2xlarge",
       "r3.xlarge",
       "r3.large",
       "i2.8xlarge",
       "i2.4xlarge",
       "i2.2xlarge",
       "i2.xlarge",
       "cr1.8xlarge",
       "cg1.4xlarge",
       "m3.medium",
       "m3.large",
       "m3.xlarge",
       "m3.2xlarge",
       "hi1.4xlarge",
       "g2.2xlarge",
       "g2.8xlarge",
       "t2.micro",
       "t2.small",
       "t2.medium",
       "t2.large",
       "d2.8xlarge",
       "d2.4xlarge",
       "d2.2xlarge",
       "d2.xlarge",
       "m4.large",
       "m4.xlarge",
       "m4.2xlarge",
       "m4.4xlarge",
       "m4.10xlarge"
     ]
   },
   "ComputeInstanceType" : {
```

```
  "Description" : "EC2 instance type used for the HPC cluster's compute nodes",
  "Type" : "String",
  "Default" : "m4.xlarge",
  "ConstraintDescription" : "must be a valid EC2 instance type.",
  "AllowedValues" : [
    "cc2.8xlarge",
    "c3.8xlarge",
    "c3.4xlarge",
    "c3.2xlarge",
    "c3.xlarge",
    "c3.large",
    "c4.8xlarge",
    "c4.4xlarge",
    "c4.2xlarge",
    "c4.xlarge",
    "c4.1large",
    "r3.8xlarge",
    "r3.4xlarge",
    "r3.2xlarge",
    "r3.xlarge",
    "r3.large",
    "i2.8xlarge",
    "i2.4xlarge",
    "i2.2xlarge",
    "i2.xlarge",
    "cr1.8xlarge",
    "cg1.4xlarge",
    "m3.medium",
    "m3.large",
    "m3.xlarge",
    "m3.2xlarge",
    "hi1.4xlarge",
    "g2.2xlarge",
    "g2.8xlarge",
    "t2.micro",
    "t2.small",
    "t2.medium",
    "t2.large",
    "d2.8xlarge",
    "d2.4xlarge",
    "d2.2xlarge",
    "d2.xlarge",
    "m4.large",
    "m4.xlarge",
    "m4.2xlarge",
    "m4.4xlarge",
    "m4.10xlarge"
  ]
},
"InitialQueueSize" : {
  "Description" : "Inital number of EC2 instances to launch as compute nodes in the cluster.",
  "Type" : "Number",
  "Default" : "1"
},
"MaxQueueSize" : {
  "Description" : "Maximum number of EC2 instances that can be launched in the cluster.",
  "Type" : "Number",
  "Default" : "10"
},
"ScalingThreshold" : {
  "Description" : "Threshold for triggering CloudWatch ScaleUp action",
  "Type" : "String",
  "Default" : "1"
},
"ScalingEvaluationPeriods" : {
  "Description" : "Number of periods consective periods required to trigger the scaling adjustment",
  "Type" : "String",
  "Default" : "2"
},
"ScalingPeriod" : {
  "Description" : "Period in seconds to measure ScalingThreshold",
  "Type" : "String",
  "Default" : "60"
},
"SpotPrice" : {
  "Description" : "Spot price for the SpotComputeFleet",
  "Type" : "Number",
  "Default" : "0.42"
},
"ClusterType" : {
  "Description" : "Type of cluster to launch i.e. ondemand or spot",
  "Type" : "String",
  "Default" : "spot",
  "ConstraintDescription" : "must be a supported cluster type",
  "AllowedValues" : [
    "ondemand",
    "spot"
  ]
},
"ProxyServer" : {
  "Description" : "hostname and port of HTTP proxy server for cfn-init, boto and yum i.e. proxy.example.com:8080",
  "Type" : "String",
  "Default" : "NONE"
},
"VolumeSize" : {
  "Description" : "Size of EBS volume in GB, if creating a new one",
  "Type" : "Number",
  "Default" : "20"
},
"VolumeType" : {
  "Description" : "Type of volume to create either new or from snapshot",
  "Type" : "String",
  "Default" : "gp2",
  "ConstraintDescription" : "must be a supported volume type: standard, io1, gp2",
  "AllowedValues" : [
    "standard",
    "gp2",
    "io1"
  ]
},
```

```
"EBSSnapshotId" : {
  "Description" : "Id of EBS snapshot if using snapshot as source for volume",
  "Type" : "String",
  "Default" : "NONE"
},
"AccessFrom" : {
  "Description" : "Lockdown SSH/HTTP access (default can be accessed from anywhere)",
  "Type" : "String",
  "MinLength" : "9",
  "MaxLength" : "18",
  "Default" : "0.0.0.0/0",
  "AllowedPattern" : "(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",
  "ConstraintDescription" : "must be a valid CIDR range of the form x.x.x.x/x."
},
"MaintainInitialSize" : {
  "Description" : "Boolean flag to set autoscaling group to maintain initial size and scale back",
  "Type" : "String",
  "Default" : "true",
  "ConstraintDescription" : "true/false",
  "AllowedValues" : [
    "true",
    "false"
  ]
},
"UsePublicIps" : {
  "Description" : "Boolean flag to use public IP's for instances. If false, the VPC must be correctly setup to use NAT for all traffic.",
  "Type" : "String",
  "Default" : "true",
  "ConstraintDescription" : "true/false",
  "AllowedValues" : [
    "true",
    "false"
  ]
},
"VolumeIOPS" : {
  "Description" : "Number of IOPS for volume type io1. Not used for standard volumes.",
  "Type" : "Number",
  "Default" : "100"
},
"ComputeWaitConditionCount" : {
  "Description" : "Specific number of instances to wait for while creating the cluster",
  "Type" : "Number",
  "Default" : "1"
},
"S3ReadResource" : {
  "Description" : "S3 resource with read access from cfncluster nodes",
  "Type" : "String",
  "Default" : "NONE"
},
"S3ReadWriteResource" : {
  "Description" : "Addtional policy document to be added to EC2 IAM role created and assigned to all nodes.",
  "Type" : "String",
  "Default" : "NONE"
},
"Placement" : {
  "Description" : "Type of placement requird in cfncluster, it can either be cluster or compute.",
  "Type" : "String",
  "Default" : "cluster",
  "AllowedValues" : [
    "cluster",
    "compute"
  ]
},
"PlacementGroup" : {
  "Description" : "The name of an existing placement group",
  "Type" : "String",
  "Default" : "NONE"
},
"EncryptedEphemeral" : {
  "Description" : "Boolean flag to encrypt local ephemeral drives. The keys are in-memory and non-recoverable.",
  "Type" : "String",
  "Default" : "false",
  "ConstraintDescription" : "true/false",
  "AllowedValues" : [
    "true",
    "false"
  ]
},
"EBSEncryption" : {
  "Description" : "Boolean flag to use EBS encryption for /shared volume. (Not to be used for snapshots)",
  "Type" : "String",
  "Default" : "false",
  "ConstraintDescription" : "true/false",
  "AllowedValues" : [
    "true",
    "false"
  ]
},
"EphemeralDir" : {
  "Description" : "The path/mountpoint for the ephemeral drive",
  "Type" : "String",
  "Default" : "/scratch"
},
"BaseOS" : {
  "Description" : "Base OS type for cluster AMI",
  "Type" : "String",
  "Default" : "alinux",
  "ConstraintDescription" : "must be a supported base OS",
  "AllowedValues" : [
    "centos6",
    "centos7",
    "alinux",
    "ubuntu1404"
  ]
},
"ScalingThreshold2" : {
  "Description" : "Threshold for triggering CloudWatch ScaleUp2 action",
  "Type" : "String",
  "Default" : "200"
},
```

```
    "ScalingCooldown" : {
      "Description" : "Period in seconds to wait before allowing further scaling actions",
      "Type" : "String",
      "Default" : "300"
    },
    "ScalingAdjustment" : {
      "Description" : "Number of instances to add to cluster when the CloudWatch ScaleUp action is called.",
      "Type" : "String",
      "Default" : "1"
    },
    "ScalingAdjustment2" : {
      "Description" : "Number of instances to add to cluster when the CloudWatch ScaleUp2 action is called.",
      "Type" : "String",
      "Default" : "10"
    },
    "SharedDir" : {
      "Description" : "The path/mountpoint for the shared drive",
      "Type" : "String",
      "Default" : "/export"
    },
    "CLITemplate" : {
      "Type" : "String"
    },
    "AdditionalSG" : {
      "Description" : "Additional VPC secuirty group to be added to instances. Defaults to NONE",
      "Type" : "String",
      "Default" : "NONE"
    },
    "CWLRegion" : {
      "Description" : "CloudWatch Logs region",
      "Type" : "String",
      "Default" : "NONE"
    },
    "CWLLogGroup" : {
      "Description" : "CloudWatch Logs LogGroup",
      "Type" : "String",
      "Default" : "NONE"
    },
    "Tenancy" : {
      "Description" : "Type of placement requird in cfncluster, it can either be cluster or compute.",
      "Type" : "String",
      "Default" : "default",
      "AllowedValues" : [
        "default",
        "dedicated"
      ]
    },
    "EBSKMSKeyId" : {
      "Description" : "KMS ARN for customer created master key, will be used for EBS encryption",
      "Type" : "String",
      "Default" : "NONE"
    },
    "EphemeralKMSKeyId" : {
      "Description" : "KMS ARN for customer created master key, will be used for ephemeral encryption",
      "Type" : "String",
      "Default" : "NONE"
    },
    "ClusterReadyScript" : {
      "Description" : "Cluster ready script URL. This is only on the MasterServer, when the cluster reaches CREATE_COMPLETE.",
      "Type" : "String",
      "Default" : "NONE"
    },
    "MasterRootVolumeSize" : {
      "Description" : "Size of MasterServer EBS root volume in GB",
      "Type" : "Number",
      "Default" : "50"
    },
    "ComputeRootVolumeSize" : {
      "Description" : "Size of ComputeFleet EBS root volume in GB",
      "Type" : "Number",
      "Default" : "50"
    },
    "EC2IAMRoleName" : {
      "Description" : "Existing EC2 IAM role name",
      "Type" : "String",
      "Default" : "NONE"
    },
    "VPCSecurityGroupId" : {
      "Description" : "Existing VPC security group Id",
      "Type" : "String",
      "Default" : "NONE"
    },
    "EBSVolumeId" : {
      "Description" : "Existing EBS volume Id",
      "Type" : "String",
      "Default" : "NONE"
    }
  },
  "Conditions" : {
    "UseSpotInstances" : {
      "Fn::Equals" : [{ "Ref" : "ClusterType" },
        "spot"
      ]
    },
    "UseEBSSnapshot" : {
      "Fn::Not" : [{
          "Fn::Equals" : [{ "Ref" : "EBSSnapshotId" },
            "NONE"
          ]
        }
      ]
    },
    "MaintainInitialASGSize" : {
      "Fn::Equals" : [{ "Ref" : "MaintainInitialSize" },
        "true"
      ]
    },
    "MasterPublicIp" : {
      "Fn::Equals" : [{ "Ref" : "UsePublicIps" },
        "true"
```

```
  ]
},
"UseEBSPIOPS" : {
  "Fn::Equals" : [{ "Ref" : "VolumeType" },
    "io1"
  ]
},
"UseS3ReadPolicy" : {
  "Fn::Not" : [{
      "Fn::Equals" : [{ "Ref" : "S3ReadResource" },
        "NONE"
      ]
    }
  ]
},
"UsePlacementGroup" : {
  "Fn::Not" : [{
      "Fn::Equals" : [{ "Ref" : "PlacementGroup" },
        "NONE"
      ]
    }
  ]
},
"UseClusterPlacement" : {
  "Fn::And" : [{
      "Fn::Equals" : [{ "Ref" : "Placement" },
        "cluster"
      ]
    }, {
      "Condition" : "UsePlacementGroup"
    }
  ]
},
"UseEBSEncryption" : {
  "Fn::Equals" : [{ "Ref" : "EBSEncryption" },
    "true"
  ]
},
"UseS3ReadWritePolicy" : {
  "Fn::Not" : [{
      "Fn::Equals" : [{ "Ref" : "S3ReadWriteResource" },
        "NONE"
      ]
    }
  ]
},
"CloudWatchLogs" : {
  "Fn::And" : [{
      "Fn::Not" : [{
          "Fn::Equals" : [{ "Ref" : "CWLRegion" },
            "NONE"
          ]
        }
      ]
    }, {
      "Fn::Not" : [{
          "Fn::Equals" : [{ "Ref" : "CWLLogGroup" },
            "NONE"
          ]
        }
      ]
    }
  ]
},
"AddAdditionalSG" : {
  "Fn::Not" : [{
      "Fn::Equals" : [{ "Ref" : "AdditionalSG" },
        "NONE"
      ]
    }
  ]
},
"UseEBSKMSKey" : {
  "Fn::And" : [{
      "Fn::Not" : [{
          "Fn::Equals" : [{ "Ref" : "EBSKMSKeyId" },
            "NONE"
          ]
        }
      ]
    }, {
      "Condition" : "UseEBSEncryption"
    }
  ]
},
"UseEphemeralKMSKey" : {
  "Fn::And" : [{
      "Fn::Not" : [{
          "Fn::Equals" : [{ "Ref" : "EphemeralKMSKeyId" },
            "NONE"
          ]
        }
      ]
    }, {
      "Fn::Equals" : [{ "Ref" : "EncryptedEphemeral" },
        "true"
      ]
    }
  ]
},
"UseDedicatedTenancy" : {
  "Fn::Equals" : [{ "Ref" : "Tenancy" },
    "dedicated"
  ]
},
"UseEC2IAMRole" : {
  "Fn::Not" : [{
      "Fn::Equals" : [{ "Ref" : "EC2IAMRoleName" },
        "NONE"
      ]
```

```
      }
    ]
  },
  "CreateEC2IAMRole" : {
    "Fn::Equals" : [{ "Ref" : "EC2IAMRoleName" },
      "NONE"
    ]
  },
  "UseExistingSecurityGroup" : {
    "Fn::Not" : [{
        "Fn::Equals" : [{ "Ref" : "VPCSecurityGroupId" },
          "NONE"
        ]
      }
    ]
  },
  "UseExistingEBSVolume" : {
    "Fn::Not" : [{
        "Fn::Equals" : [{ "Ref" : "EBSVolumeId" },
          "NONE"
        ]
      }
    ]
  },
  "CreateEBSVolume" : {
    "Fn::Equals" : [{ "Ref" : "EBSVolumeId" },
      "NONE"
    ]
  },
  "CreateSecurityGroups" : {
    "Fn::Equals" : [{ "Ref" : "VPCSecurityGroupId" },
      "NONE"
    ]
  },
  "CreatePlacementGroup" : {
    "Fn::And" : [{
        "Fn::Equals" : [{ "Ref" : "PlacementGroup" },
          "DYNAMIC"
        ]
      }, {
        "Condition" : "UsePlacementGroup"
      }
    ]
  }
},
"Mappings" : {
  "AWSInstanceType2Capabilites" : {
    "cc2.8xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "False"
    },
    "cr1.8xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "False"
    },
    "g2.2xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "g2.8xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "m3.medium" : {
      "Arch" : "64HVM",
      "EBSOpt" : "False"
    },
    "m3.large" : {
      "Arch" : "64HVM",
      "EBSOpt" : "False"
    },
    "m3.xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "m3.2xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "c3.8xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "False"
    },
    "c3.4xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "c3.2xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "c3.xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "c3.large" : {
      "Arch" : "64HVM",
      "EBSOpt" : "False"
    },
    "c4.8xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "c4.4xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
    },
    "c4.2xlarge" : {
      "Arch" : "64HVM",
      "EBSOpt" : "True"
```

```
  },
  "c4.xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "c4.large" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "r3.8xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "r3.4xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "r3.2xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "r3.xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "r3.large" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "i2.8xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "i2.4xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "i2.2xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "i2.xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "i2.large" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "cg1.4xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "t2.micro" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "t2.small" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "t2.medium" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "d2.8xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "d2.4xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "d2.2xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "d2.xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "t2.large" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  },
  "m4.10xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "m4.4xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "m4.2xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "m4.xlarge" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "m4.large" : {
    "Arch" : "64HVM",
    "EBSOpt" : "True"
  },
  "t2.nano" : {
    "Arch" : "64HVM",
    "EBSOpt" : "False"
  }
},
```

```
"AWSRegionOS2AMI" : {
  "eu-west-1" : {
    "centos6" : "ami-b30a8ec0",
    "centos7" : "ami-2d0e8a5e",
    "alinux" : "ami-6a0e8a19",
    "ubuntu1404" : "ami-7b088c08"
  },
  "us-west-2" : {
    "centos6" : "ami-e5b25985",
    "centos7" : "ami-148b6074",
    "alinux" : "ami-5ab15a3a",
    "ubuntu1404" : "ami-f2bf5492"
  },
  "eu-central-1" : {
    "centos6" : "ami-1ee90f71",
    "centos7" : "ami-1fe90f70",
    "alinux" : "ami-48f71127",
    "ubuntu1404" : "ami-38e80e57"
  },
  "sa-east-1" : {
    "centos6" : "ami-12b8347e",
    "centos7" : "ami-20b63a4c",
    "alinux" : "ami-24bf3348",
    "ubuntu1404" : "ami-97b73bfb"
  },
  "ap-northeast-1" : {
    "centos6" : "ami-3b809555",
    "centos7" : "ami-f284919c",
    "alinux" : "ami-5c839632",
    "ubuntu1404" : "ami-5d839633"
  },
  "us-east-1" : {
    "centos6" : "ami-a8535cc2",
    "centos7" : "ami-7c555a16",
    "alinux" : "ami-1d585777",
    "ubuntu1404" : "ami-7a585710"
  },
  "us-west-1" : {
    "centos6" : "ami-6ac2b00a",
    "centos7" : "ami-18d8aa78",
    "alinux" : "ami-1dd8aa7d",
    "ubuntu1404" : "ami-0bdeac6b"
  },
  "ap-southeast-2" : {
    "centos6" : "ami-20b99943",
    "centos7" : "ami-cfba9aac",
    "alinux" : "ami-9fb999fc",
    "ubuntu1404" : "ami-c7ba9aa4"
  },
  "ap-southeast-1" : {
    "centos6" : "ami-2b945e48",
    "centos7" : "ami-87945ee4",
    "alinux" : "ami-ef955f8c",
    "ubuntu1404" : "ami-72945e11"
  },
  "us-gov-west-1" : {
    "centos6" : "ami-cd14a8ac",
    "centos7" : "ami-eb15a98a",
    "alinux" : "ami-c016aaa1",
    "ubuntu1404" : "ami-ee15a98f"
  },
  "ap-northeast-2" : {
    "centos6" : "ami-0ea26b60",
    "centos7" : "ami-79a06917",
    "alinux" : "ami-85a069eb",
    "ubuntu1404" : "ami-30a36a5e"
  }
},
"OSFeatures" : {
  "centos6" : {
    "User" : "centos",
    "RootDevice" : "/dev/sda1"
  },
  "centos7" : {
    "User" : "centos",
    "RootDevice" : "/dev/sda1"
  },
  "alinux" : {
    "User" : "ec2-user",
    "RootDevice" : "/dev/xvda"
  },
  "ubuntu1404" : {
    "User" : "ubuntu",
    "RootDevice" : "/dev/sda1"
  }
},
"CfnClusterVersions" : {
  "default" : {
    "cfncluster" : "cfncluster-1.2.1",
    "cookbook" : "cfncluster-cookbook-1.2.2",
    "chef" : "12.8.1",
    "ridley" : "4.5.0",
    "berkshelf" : "4.3.0",
    "ami" : "201603231645"
  }
},
"AWSRegion2Capabilites" : {
  "eu-west-1" : {
    "arn" : "aws"
  },
  "us-east-1" : {
    "arn" : "aws"
  },
  "ap-northeast-1" : {
    "arn" : "aws"
  },
  "us-west-2" : {
    "arn" : "aws"
  },
  "sa-east-1" : {
```

```
      "arn" : "aws"
    },
    "us-west-1" : {
      "arn" : "aws"
    },
    "ap-southeast-1" : {
      "arn" : "aws"
    },
    "ap-southeast-2" : {
      "arn" : "aws"
    },
    "eu-central-1" : {
      "arn" : "aws"
    },
    "us-gov-west-1" : {
      "arn" : "aws-us-gov"
    },
    "ap-northeast-2" : {
      "arn" : "aws"
    }
  }
},
"Resources" : {
  "SQS" : {
    "Type" : "AWS::SQS::Queue",
    "Properties" : {},
    "Metadata" : {

    }
  },
  "SQSPolicy" : {
    "Type" : "AWS::SQS::QueuePolicy",
    "Properties" : {
      "PolicyDocument" : {
        "Id" : "MyQueuePolicy",
        "Statement" : [{
            "Sid" : "Allow-SendMessage-From-AS-SNS-Topic",
            "Effect" : "Allow",
            "Principal" : {
              "AWS" : "*"
            },
            "Action" : [
              "sqs:SendMessage"
            ],
            "Resource" : "*",
            "Condition" : {
              "ArnEquals" : {
                "aws:SourceArn" : { "Ref" : "SNS" }
              }
            }
          }
        ]
      },
      "Queues" : [{ "Ref" : "SQS" }
      ]
    },
    "Metadata" : {

    }
  },
  "SNS" : {
    "Type" : "AWS::SNS::Topic",
    "Properties" : {
      "Subscription" : [{
          "Endpoint" : {
            "Fn::GetAtt" : [
              "SQS",
              "Arn"
            ]
          },
          "Protocol" : "sqs"
        }
      ]
    },
    "Metadata" : {

    }
  },
  "DynamoDBTable" : {
    "Type" : "AWS::DynamoDB::Table",
    "Properties" : {
      "AttributeDefinitions" : [{
          "AttributeName" : "instanceId",
          "AttributeType" : "S"
        }
      ],
      "KeySchema" : [{
          "AttributeName" : "instanceId",
          "KeyType" : "HASH"
        }
      ],
      "ProvisionedThroughput" : {
        "ReadCapacityUnits" : "5",
        "WriteCapacityUnits" : "5"
      }
    },
    "Metadata" : {

    }
  },
  "RootRole" : {
    "Type" : "AWS::IAM::Role",
    "Properties" : {
      "AssumeRolePolicyDocument" : {
        "Version" : "2012-10-17",
        "Statement" : [{
            "Effect" : "Allow",
            "Principal" : {
              "Service" : [
                "ec2.amazonaws.com"
```

```
            ]
          },
          "Action" : [
            "sts:AssumeRole"
          ]
        }
      ]
    },
    "Path" : "/"
  },
  "Condition" : "CreateEC2IAMRole",
  "Metadata" : {

  }
},
"RootInstanceProfile" : {
  "Type" : "AWS::IAM::InstanceProfile",
  "Properties" : {
    "Path" : "/",
    "Roles" : [{ "Ref" : "RootRole" }
    ]
  },
  "Condition" : "CreateEC2IAMRole",
  "Metadata" : {

  }
},
"CfnClusterPolicies" : {
  "Type" : "AWS::IAM::Policy",
  "Properties" : {
    "PolicyName" : "cfncluster",
    "PolicyDocument" : {
     "Statement" : [{
          "Sid" : "EC2",
          "Action" : [
            "ec2:AttachVolume",
            "ec2:DescribeInstanceAttribute",
            "ec2:DescribeInstanceStatus",
            "ec2:DescribeInstances"
          ],
          "Effect" : "Allow",
          "Resource" : [
            "*"
          ]
        }, {
          "Sid" : "DynamoDBList",
          "Action" : [
            "dynamodb:ListTables"
          ],
          "Effect" : "Allow",
          "Resource" : [
            "*"
          ]
        }, {
          "Sid" : "SQSQueue",
          "Action" : [
            "sqs:SendMessage",
            "sqs:ReceiveMessage",
            "sqs:ChangeMessageVisibility",
            "sqs:DeleteMessage",
            "sqs:GetQueueUrl"
          ],
          "Effect" : "Allow",
          "Resource" : [{
              "Fn::GetAtt" : [
                "SQS",
                "Arn"
              ]
            }
          ]
        }, {
          "Sid" : "Autoscaling",
          "Action" : [
            "autoscaling:DescribeAutoScalingGroups",
            "autoscaling:TerminateInstanceInAutoScalingGroup",
            "autoscaling:SetDesiredCapacity"
          ],
          "Effect" : "Allow",
          "Resource" : [
            "*"
          ]
        }, {
          "Sid" : "CloudWatch",
          "Action" : [
            "cloudwatch:PutMetricData"
          ],
          "Effect" : "Allow",
          "Resource" : [
            "*"
          ]
        }, {
          "Sid" : "DynamoDBTable",
          "Action" : [
            "dynamodb:PutItem",
            "dynamodb:Query",
            "dynamodb:GetItem",
            "dynamodb:DeleteItem",
            "dynamodb:DescribeTable"
          ],
          "Effect" : "Allow",
          "Resource" : [{
              "Fn::Join" : [
                "",
                [
                  "arn:", {
                    "Fn::FindInMap" : [
                      "AWSRegion2Capabilites", { "Ref" : "AWS::Region" },
                      "arn"
                    ]
                  },
```

```
                ":dynamodb:", { "Ref" : "AWS::Region" },
                ":", { "Ref" : "AWS::AccountId" },
                ":table/", { "Ref" : "DynamoDBTable" }
              ]
            ]
          }
        ]
      }, {
        "Sid" : "SQSList",
        "Action" : [
          "sqs:ListQueues"
        ],
        "Effect" : "Allow",
        "Resource" : [
          "*"
        ]
      }, {
        "Sid" : "CloudWatchLogs",
        "Action" : [
          "logs:*"
        ],
        "Effect" : "Allow",
        "Resource" : [{
            "Fn::Join" : [
              "",
              [
                "arn:", {
                  "Fn::FindInMap" : [
                    "AWSRegion2Capabilites", { "Ref" : "AWS::Region" },
                    "arn"
                  ]
                },
                ":logs:*:*:*"
              ]
            ]
          }
        ]
      }
    ]
  },
  "Roles" : [{ "Ref" : "RootRole" }
  ]
},
"Condition" : "CreateEC2IAMRole",
"Metadata" : {

}
},
"S3ReadRolePolicies" : {
  "Type" : "AWS::IAM::Policy",
  "Properties" : {
    "PolicyName" : "S3Read",
    "PolicyDocument" : {
      "Version" : "2012-10-17",
      "Statement" : [{
          "Sid" : "S3Read",
          "Effect" : "Allow",
          "Action" : [
            "s3:Get*",
            "s3:List*"
          ],
          "Resource" : [{ "Ref" : "S3ReadResource" }
          ]
        }
      ]
    },
    "Roles" : [{ "Ref" : "RootRole" }
    ]
  },
  "Condition" : "UseS3ReadPolicy",
  "Metadata" : {

  }
},
"S3ReadWriteRolePolicies" : {
  "Type" : "AWS::IAM::Policy",
  "Properties" : {
    "PolicyName" : "S3ReadWrite",
    "PolicyDocument" : {
      "Version" : "2012-10-17",
      "Statement" : [{
          "Sid" : "S3ReadWrite",
          "Effect" : "Allow",
          "Action" : [
            "s3:*"
          ],
          "Resource" : [{ "Ref" : "S3ReadWriteResource" }
          ]
        }
      ]
    },
    "Roles" : [{ "Ref" : "RootRole" }
    ]
  },
  "Condition" : "UseS3ReadWritePolicy",
  "Metadata" : {

  }
},
"MasterServer" : {
  "Type" : "AWS::EC2::Instance",
  "Properties" : {
    "InstanceType" : { "Ref" : "MasterInstanceType" },
    "BlockDeviceMappings" : [{
        "DeviceName" : "/dev/xvdba",
        "VirtualName" : "ephemeral0"
      }, {
        "DeviceName" : "/dev/xvdbb",
        "VirtualName" : "ephemeral1"
      }, {
```

```
      "DeviceName" : "/dev/xvdbc",
      "VirtualName" : "ephemeral2"
    }, {
      "DeviceName" : "/dev/xvdbd",
      "VirtualName" : "ephemeral3"
    }, {
      "DeviceName" : "/dev/xvdbe",
      "VirtualName" : "ephemeral4"
    }, {
      "DeviceName" : "/dev/xvdbf",
      "VirtualName" : "ephemeral5"
    }, {
      "DeviceName" : "/dev/xvdbg",
      "VirtualName" : "ephemeral6"
    }, {
      "DeviceName" : "/dev/xvdbh",
      "VirtualName" : "ephemeral7"
    }, {
      "DeviceName" : "/dev/xvdbi",
      "VirtualName" : "ephemeral8"
    }, {
      "DeviceName" : "/dev/xvdbj",
      "VirtualName" : "ephemeral9"
    }, {
      "DeviceName" : "/dev/xvdbk",
      "VirtualName" : "ephemeral10"
    }, {
      "DeviceName" : "/dev/xvdbl",
      "VirtualName" : "ephemeral11"
    }, {
      "DeviceName" : "/dev/xvdbm",
      "VirtualName" : "ephemeral12"
    }, {
      "DeviceName" : "/dev/xvdbn",
      "VirtualName" : "ephemeral13"
    }, {
      "DeviceName" : "/dev/xvdbo",
      "VirtualName" : "ephemeral14"
    }, {
      "DeviceName" : "/dev/xvdbp",
      "VirtualName" : "ephemeral15"
    }, {
      "DeviceName" : "/dev/xvdbq",
      "VirtualName" : "ephemeral16"
    }, {
      "DeviceName" : "/dev/xvdbr",
      "VirtualName" : "ephemeral17"
    }, {
      "DeviceName" : "/dev/xvdbs",
      "VirtualName" : "ephemeral18"
    }, {
      "DeviceName" : "/dev/xvdbt",
      "VirtualName" : "ephemeral19"
    }, {
      "DeviceName" : "/dev/xvdbu",
      "VirtualName" : "ephemeral20"
    }, {
      "DeviceName" : "/dev/xvdbv",
      "VirtualName" : "ephemeral21"
    }, {
      "DeviceName" : "/dev/xvdbw",
      "VirtualName" : "ephemeral22"
    }, {
      "DeviceName" : "/dev/xvdbx",
      "VirtualName" : "ephemeral23"
    }, {
      "DeviceName" : {
        "Fn::FindInMap" : [
          "OSFeatures", { "Ref" : "BaseOS" },
          "RootDevice"
        ]
      },
      "Ebs" : {
        "VolumeSize" : { "Ref" : "MasterRootVolumeSize" },
        "VolumeType" : "gp2"
      }
    }
  ],
  "KeyName" : { "Ref" : "KeyName" },
  "Tags" : [{
      "Key" : "Application",
      "Value" : { "Ref" : "AWS::StackName" }
    }, {
      "Key" : "Name",
      "Value" : "Master"
    }
  ],
  "NetworkInterfaces" : [{
      "NetworkInterfaceId" : { "Ref" : "MasterENI" },
      "DeviceIndex" : "0"
    }
  ],
  "ImageId" : {
      "Fn::FindInMap" : [
          "AWSRegionOS2AMI", { "Ref" : "AWS::Region" }, { "Ref" : "BaseOS" }
      ]
  },
  "EbsOptimized" : {
    "Fn::FindInMap" : [
      "AWSInstanceType2Capabilites", { "Ref" : "MasterInstanceType" },
      "EBSOpt"
    ]
  },
  "IamInstanceProfile" : {
    "Fn::If" : [ "UseEC2IAMRole", { "Ref" : "EC2IAMRoleName" }, { "Ref" : "RootInstanceProfile" } ]
  },
  "PlacementGroupName" : {
    "Fn::If" : [
      "UseClusterPlacement", {
        "Fn::If" : [ "CreatePlacementGroup", { "Ref" : "DynamicPlacementGroup" }, { "Ref" : "PlacementGroup" } ]
```

```
            }, { "Ref" : "AWS::NoValue" }
          ]
        },
        "Tenancy" : { "Ref" : "Tenancy" },
        "UserData" : {
          "Fn::Base64" : {
            "Fn::Join" : [
              "",
              [
                "#!/bin/bash -x\n\n",
                "function error_exit\n",
                "{\n",
                "  cfn-signal ${proxy_args} --exit-code=1 --reason=\"$1\" --stack=", { "Ref" : "AWS::StackName" },
                " --resource=MasterServer --region=", { "Ref" : "AWS::Region" },
                "\n",
                "  exit 1\n",
                "}\n",
                "function bootstrap_instance\n",
                "{\n",
                "  which yum 2>/dev/null; yum=$?\n",
                "  which apt-get 2>/dev/null; apt=$?\n",
                "  if [ \"$yum\" == \"0\" ]; then\n",
                "    yum -y groupinstall development && yum -y install curl wget\n",
                "  fi\n",
                "  if [ \"$apt\" == \"0\" ]; then\n",
                "    apt-cache search build-essential; apt-get clean; apt-get update; apt-get -y install build-essential curl wget apt-transport-https ca-certificates\n",
                "  fi\n",
                "  which cfn-init 2>/dev/null || ( curl -s -L -o /tmp/aws-cfn-bootstrap-latest.tar.gz https://s3.amazonaws.com/cloudformation-examples/aws-cfn-bootstrap-
latest.tar.gz; easy_install -U /tmp/aws-cfn-bootstrap-latest.tar.gz)\n",
                "  mkdir -p /etc/chef && chown -R root:root /etc/chef\n",
                "  curl -L https://www.chef.io/chef/install.sh | bash -s -- -v $chef_version\n",
                "  /opt/chef/embedded/bin/gem install --no-rdoc --no-ri ridley:$ridley_version berkshelf:$berkshelf_version\n",
                "  mkdir /opt/cfncluster && echo $cfncluster_version | tee /opt/cfncluster/.bootstrapped\n",
                "}\n",
                "proxy=", { "Ref" : "ProxyServer" },
                "\n",
                "custom_cookbook=\"NONE\"\n",
                "if [ \"$proxy\" != \"NONE\" ]; then\n",
                "  proxy_args=\"--http-proxy=${proxy} --https-proxy=${proxy}\"\n",
                "  proxy_host=$(echo \"$proxy\" | awk -F/ '{print $3}' | cut -d: -f1)\n",
                "  proxy_port=$(echo \"$proxy\" | awk -F/ '{print $3}' | cut -d: -f2)\n",
                "  export http_proxy=$proxy; export https_proxy=$http_proxy\n",
                "  export HTTP_PROXY=$proxy; export HTTPS_PROXY=$http_proxy\n",
                "  echo -e \"[Boto]\nproxy = ${proxy_host}\nproxy_port = ${proxy_port}\n\" >/etc/boto.cfg\n",
                "else\n",
                "  proxy_args=\"\"\n",
                "fi\n",
                "if [ \"$custom_cookbook\" != \"NONE\" ]; then\n",
                "  cookbook_url=$custom_cookbook\n",
                "else\n",
                "  if [ \"", { "Ref" : "AWS::Region" },
                "\" == \"us-east-1\" ]; then\n",
                "    s3_prefix=s3\n",
                "  else\n",
                "    s3_prefix=s3-", { "Ref" : "AWS::Region" },
                "\n",
                "  fi\n",
                "  cookbook_url=https://${s3_prefix}.amazonaws.com/cfncluster-", { "Ref" : "AWS::Region" },
                "/cookbooks/", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "cookbook"
                  ]
                },
                ".tgz\n",
                "fi\n",
                "export PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/opt/aws/bin\n",
                "export cfncluster_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "cfncluster"
                  ]
                },
                "\n",
                "export cookbook_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "cookbook"
                  ]
                },
                "\n",
                "export chef_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "chef"
                  ]
                },
                "\n",
                "export ridley_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "ridley"
                  ]
                },
                "\n",
                "export berkshelf_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "berkshelf"
                  ]
                },
                "\n",
                "if [ -f /opt/cfncluster/.bootstrapped ]; then\n",
                "  installed_version=$(cat /opt/cfncluster/.bootstrapped)\n",
```

```
              "  if [ \"$cfncluster_version\" != \"$installed_version\" ]; then\n",
              "    bootstrap_instance\n",
              "  fi\n",
              "else\n",
              "  bootstrap_instance\n",
              "fi\n",
              "mkdir /tmp/cookbooks\n",
              "cd /tmp/cookbooks\n",
              "curl -s -L -o cookbook.tar.gz $cookbook_url\n",
              "tar -xzf cookbook.tar.gz\n",
              "cd /tmp\n",
              "# Call CloudFormation\n",
              "cfn-init ${proxy_args} -s ", { "Ref" : "AWS::StackName" },
              " -v -c default -r MasterServer --region ", { "Ref" : "AWS::Region" },
              " || error_exit 'Failed to run cfn-init. If --norollback was specified, check /var/log/cfn-init.log and /var/log/cfncluster.log.'\n",

              "cfn-signal ${proxy_args} --exit-code=0 --reason=\"MasterServer setup complete\" --stack=", { "Ref" : "AWS::StackName" }, " --resource=MasterServer --
region=", { "Ref" : "AWS::Region" },
              "\n",
              "# End of file\n"
            ]
          ]
        }
      }
    },
    "Metadata" : {
      "Comment" : "cfncluster Master server",
      "AWS::CloudFormation::Init" : {
        "configSets" : {
          "default" : [
            "deployConfigFiles",
            "getCookbooks",
            "chefPrepEnv",
            "shellRunPreInstall",
            "chefConfig",
            "shellRunPostInstall",
            "shellForkClusterReadyInstall"
          ]
        },
        "deployConfigFiles" : {
          "files" : {
              "/tmp/dna.json" : {
              "mode" : "000644",
              "owner" : "root",
              "group" : "root",
              "content" : {
                "cfncluster" : {
                  "stack_name" : { "Ref" : "AWS::StackName" },
                  "cfn_postinstall" : "https://s3.amazonaws.com/omicron-galaxy/omicronPostInstall.sh",
                  "cfn_region" : { "Ref" : "AWS::Region" },
                  "cfn_volume" : {
                    "Fn::If" : [ "UseExistingEBSVolume", { "Ref" : "EBSVolumeId" }, { "Ref" : "SharedVolume" } ]
                  },
                  "cfn_scheduler" : "slurm",
                  "cfn_encrypted_ephemeral" : { "Ref" : "EncryptedEphemeral" },
                  "cfn_ephemeral_dir" : { "Ref" : "EphemeralDir" },
                  "cfn_shared_dir" : { "Ref" : "SharedDir" },
                  "cfn_proxy" : { "Ref" : "ProxyServer" },
                  "cfn_node_type" : "MasterServer",
                  "cfn_cluster_user" : {
                    "Fn::FindInMap" : [
                      "OSFeatures", { "Ref" : "BaseOS" },
                      "User"
                    ]
                  },
                  "cfn_ddb_table" : { "Ref" : "DynamoDBTable" },
                  "cfn_sqs_queue" : {
                    "Fn::GetAtt" : [
                      "SQS",
                      "QueueName"
                    ]
                  }
                },
                "run_list" : "recipe[cfncluster::slurm_config]"
              }
              },
              "/etc/chef/client.rb" : {
                  "mode" : "000644",
                  "owner" : "root",
                  "group" : "root",
                  "content" : {
                    "Fn::Join" : [
                      "",
                      [
                        "cookbook_path ['/etc/chef/cookbooks']"
                      ]
                    ]
                  }
              },
              "/tmp/extra.json" : {
                  "mode" : "000644",
                  "owner" : "root",
                  "group" : "root",
                  "content" : "{}"
              }
          },
          "commands" : {
            "mkdir" : {
              "command" : "mkdir -p /etc/chef/ohai/hints"
            },
            "touch" : {
              "command" : "touch /etc/chef/ohai/hints/ec2.json"
            },
            "jq" : {
              "command" : "/usr/local/bin/jq --argfile f1 /tmp/dna.json --argfile f2 /tmp/extra.json -n '$f1 + $f2 | .cfncluster = $f1.cfncluster + $f2.cfncluster' >
/etc/chef/dna.json || ( echo \"jq not installed\"; cp /tmp/dna.json /etc/chef/dna.json )"
            }
          }
        },
        "getCookbooks" : {
```

```
          "commands" : {
            "berk" : {
              "command" : "for d in `ls /tmp/cookbooks`; do cd /tmp/cookbooks/$d;LANG=en_US.UTF-8 /opt/chef/embedded/bin/berks vendor /etc/chef/cookbooks; done ",
              "cwd" : "/tmp/cookbooks",
              "env" : {
                "HOME" : "/tmp"
              }
            }
          }
        },
        "chefPrepEnv" : {
          "commands" : {
            "chef" : {
              "command" : "chef-client --local-mode --config /etc/chef/client.rb --log_level auto --force-formatter --no-color --chef-zero-port 8889 --json-attributes
/etc/chef/dna.json --override-runlist cfncluster::_prep_env",
              "cwd" : "/etc/chef"
            }
          }
        },
        "shellRunPreInstall" : {
          "commands" : {
            "runpreinstall" : {
              "command" : "/opt/cfncluster/scripts/fetch_and_run -preinstall"
            }
          }
        },
        "chefConfig" : {
          "commands" : {
            "chef" : {
              "command" : "chef-client --local-mode --config /etc/chef/client.rb --log_level auto --force-formatter --no-color --chef-zero-port 8889 --json-attributes
/etc/chef/dna.json",
              "cwd" : "/etc/chef"
            }
          }
        },
        "shellRunPostInstall" : {
          "commands" : {
            "runpostinstall" : {
              "command" : "/opt/cfncluster/scripts/fetch_and_run -postinstall"
            }
          }
        },
        "shellForkClusterReadyInstall" : {
          "commands" : {
            "clusterreadyinstall" : {
              "command" : "/opt/cfncluster/scripts/fetch_and_run -clusterreadyinstall"
            }
          }
        }
      }
    },
    "CreationPolicy" : {
      "ResourceSignal" : {
        "Count" : "1",
        "Timeout" : "PT30M"
      }
    }
  },
  "ComputeFleet" : {
    "Type" : "AWS::AutoScaling::AutoScalingGroup",
    "Properties" : {
      "MaxSize" : { "Ref" : "MaxQueueSize" },
      "AvailabilityZones" : [{ "Ref" : "AvailabilityZone" }
      ],
      "VPCZoneIdentifier" : [{ "Ref" : "ComputeSubnet" }
      ],
      "LaunchConfigurationName" : { "Ref" : "ComputeServerLaunchConfig" },
      "MinSize" : {
        "Fn::If" : [
          "MaintainInitialASGSize", { "Ref" : "InitialQueueSize" },
          "0"
        ]
      },
      "DesiredCapacity" : { "Ref" : "InitialQueueSize" },
      "NotificationConfiguration" : {
        "TopicARN" : { "Ref" : "SNS" },
        "NotificationTypes" : [
          "autoscaling:EC2_INSTANCE_TERMINATE"
        ]
      },
      "Tags" : [{
          "Key" : "Name",
          "Value" : "Compute",
          "PropagateAtLaunch" : "true"
        }
      ],
      "PlacementGroup" : {
        "Fn::If" : [
          "UsePlacementGroup", {
            "Fn::If" : [ "CreatePlacementGroup", { "Ref" : "DynamicPlacementGroup" }, { "Ref" : "PlacementGroup" } ]
          }, { "Ref" : "AWS::NoValue" }
        ]
      }
    },
    "DependsOn" : "MasterServer",
    "CreationPolicy" : {
      "ResourceSignal" : {
        "Timeout" : "PT30M",
        "Count" : { "Ref" : "ComputeWaitConditionCount" }
      }
    },
    "Metadata" : {

    }
  },
  "ComputeServerLaunchConfig" : {
    "Type" : "AWS::AutoScaling::LaunchConfiguration",
    "Properties" : {
      "SecurityGroups" : [{
          "Fn::If" : [ "CreateSecurityGroups", { "Ref" : "ComputeSecurityGroup" }, { "Ref" : "AWS::NoValue" } ]
```

```
  }, {
    "Fn::If" : [ "AddAdditionalSG", { "Ref" : "AdditionalSG" }, { "Ref" : "AWS::NoValue" } ]
  }, {
    "Fn::If" : [ "UseExistingSecurityGroup", { "Ref" : "VPCSecurityGroupId" }, { "Ref" : "AWS::NoValue" } ]
  }
],
"InstanceType" : { "Ref" : "ComputeInstanceType" },
"KeyName" : { "Ref" : "KeyName" },
"IamInstanceProfile" : {
  "Fn::If" : [ "UseEC2IAMRole", { "Ref" : "EC2IAMRoleName" }, { "Ref" : "RootInstanceProfile" } ]
},
"SpotPrice" : {
  "Fn::If" : [ "UseSpotInstances", { "Ref" : "SpotPrice" }, { "Ref" : "AWS::NoValue" } ]
},
"ImageId" : {
    "Fn::FindInMap" : [
        "AWSRegionOS2AMI", { "Ref" : "AWS::Region" }, { "Ref" : "BaseOS" }
      ]
},
"InstanceMonitoring" : "false",
"BlockDeviceMappings" : [{
    "DeviceName" : "/dev/xvdba",
    "VirtualName" : "ephemeral0"
  }, {
    "DeviceName" : "/dev/xvdbb",
    "VirtualName" : "ephemeral1"
  }, {
    "DeviceName" : "/dev/xvdbc",
    "VirtualName" : "ephemeral2"
  }, {
    "DeviceName" : "/dev/xvdbd",
    "VirtualName" : "ephemeral3"
  }, {
    "DeviceName" : "/dev/xvdbe",
    "VirtualName" : "ephemeral4"
  }, {
    "DeviceName" : "/dev/xvdbf",
    "VirtualName" : "ephemeral5"
  }, {
    "DeviceName" : "/dev/xvdbg",
    "VirtualName" : "ephemeral6"
  }, {
    "DeviceName" : "/dev/xvdbh",
    "VirtualName" : "ephemeral7"
  }, {
    "DeviceName" : "/dev/xvdbi",
    "VirtualName" : "ephemeral8"
  }, {
    "DeviceName" : "/dev/xvdbj",
    "VirtualName" : "ephemeral9"
  }, {
    "DeviceName" : "/dev/xvdbk",
    "VirtualName" : "ephemeral10"
  }, {
    "DeviceName" : "/dev/xvdbl",
    "VirtualName" : "ephemeral11"
  }, {
    "DeviceName" : "/dev/xvdbm",
    "VirtualName" : "ephemeral12"
  }, {
    "DeviceName" : "/dev/xvdbn",
    "VirtualName" : "ephemeral13"
  }, {
    "DeviceName" : "/dev/xvdbo",
    "VirtualName" : "ephemeral14"
  }, {
    "DeviceName" : "/dev/xvdbp",
    "VirtualName" : "ephemeral15"
  }, {
    "DeviceName" : "/dev/xvdbq",
    "VirtualName" : "ephemeral16"
  }, {
    "DeviceName" : "/dev/xvdbr",
    "VirtualName" : "ephemeral17"
  }, {
    "DeviceName" : "/dev/xvdbs",
    "VirtualName" : "ephemeral18"
  }, {
    "DeviceName" : "/dev/xvdbt",
    "VirtualName" : "ephemeral19"
  }, {
    "DeviceName" : "/dev/xvdbu",
    "VirtualName" : "ephemeral20"
  }, {
    "DeviceName" : "/dev/xvdbv",
    "VirtualName" : "ephemeral21"
  }, {
    "DeviceName" : "/dev/xvdbw",
    "VirtualName" : "ephemeral22"
  }, {
    "DeviceName" : "/dev/xvdbx",
    "VirtualName" : "ephemeral23"
  }, {
    "DeviceName" : {
      "Fn::FindInMap" : [
        "OSFeatures", { "Ref" : "BaseOS" },
        "RootDevice"
      ]
    },
    "Ebs" : {
      "VolumeSize" : { "Ref" : "ComputeRootVolumeSize" },
      "VolumeType" : "gp2"
    }
  }
],
"PlacementTenancy" : {
  "Fn::If" : [ "UseDedicatedTenancy", { "Ref" : "Tenancy" }, { "Ref" : "AWS::NoValue" } ]
},
"UserData" : {
  "Fn::Base64" : {
```

```
            "Fn::Join" : [
              "",
              [
                "#!/bin/bash -x\n\n",
                "function error_exit\n",
                "{\n",
                "  cfn-signal ${proxy_args} --exit-code=1 --reason=\"$1\" --stack=", { "Ref" : "AWS::StackName" },
                " --resource=ComputeFleet --region=", { "Ref" : "AWS::Region" },
                "\n",
                "  exit 1\n",
                "}\n",
                "function bootstrap_instance\n",
                "{\n",
                "  which yum 2>/dev/null; yum=$?\n",
                "  which apt-get 2>/dev/null; apt=$?\n",
                "  if [ \"$yum\" == \"0\" ]; then\n",
                "    yum -y groupinstall development && yum -y install curl wget libcurl-devel\n",
                "  fi\n",
                "  if [ \"$apt\" == \"0\" ]; then\n",
                "    apt-cache search build-essential; apt-get clean; apt-get update; apt-get -y install build-essential curl wget libcurl4-openssl-dev\n",
                "  fi\n",
                "  which cfn-init 2>/dev/null || ( curl -s -L -o /tmp/aws-cfn-bootstrap-latest.tar.gz https://s3.amazonaws.com/cloudformation-examples/aws-cfn-bootstrap-
latest.tar.gz; easy_install -U /tmp/aws-cfn-bootstrap-latest.tar.gz)\n",
                "  mkdir -p /etc/chef && chown -R root:root /etc/chef\n",
                "  curl -L https://www.chef.io/chef/install.sh | bash -s -- -v $chef_version\n",
                "  /opt/chef/embedded/bin/gem install --no-rdoc --no-ri ridley:$ridley_version berkshelf:$berkshelf_version\n",
                "  mkdir /opt/cfncluster && echo $cfncluster_version | tee /opt/cfncluster/.bootstrapped\n",
                "}\n",
                "proxy=", { "Ref" : "ProxyServer" },
                "\n",
                "custom_cookbook=\"NONE\"\n",
                "if [ \"$proxy\" != \"NONE\" ]; then\n",
                "  proxy_args=\"--http-proxy=${proxy} --https-proxy=${proxy}\"\n",
                "  proxy_host=$(echo \"$proxy\" | awk -F/ '{print $3}' | cut -d: -f1)\n",
                "  proxy_port=$(echo \"$proxy\" | awk -F/ '{print $3}' | cut -d: -f2)\n",
                "  export http_proxy=$proxy; export https_proxy=$http_proxy\n",
                "  export HTTP_PROXY=$proxy; export HTTPS_PROXY=$http_proxy\n",
                "  echo -e \"[Boto]\nproxy = ${proxy_host}\nproxy_port = ${proxy_port}\n\" >/etc/boto.cfg\n",
                "else\n",
                "  proxy_args=\"\"\n",
                "fi\n",
                "if [ \"$custom_cookbook\" != \"NONE\" ]; then\n",
                "  cookbook_url=$custom_cookbook\n",
                "else\n",
                "  if [ \"", { "Ref" : "AWS::Region" },
                "\" == \"us-east-1\" ]; then\n",
                "    s3_prefix=s3\n",
                "  else\n",
                "    s3_prefix=s3-", { "Ref" : "AWS::Region" },
                "\n",
                "  fi\n",
                "  cookbook_url=https://${s3_prefix}.amazonaws.com/cfncluster-", { "Ref" : "AWS::Region" },
                "/cookbooks/", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "cookbook"
                  ]
                },
                ".tgz\n",
                "fi\n",
                "export PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/opt/aws/bin\n",
                "export cfncluster_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "cfncluster"
                  ]
                },
                "\n",
                "export cookbook_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "cookbook"
                  ]
                },
                "\n",
                "export chef_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "chef"
                  ]
                },
                "\n",
                "export ridley_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "ridley"
                  ]
                },
                "\n",
                "export berkshelf_version=", {
                  "Fn::FindInMap" : [
                    "CfnClusterVersions",
                    "default",
                    "berkshelf"
                  ]
                },
                "\n",
                "if [ -f /opt/cfncluster/.bootstrapped ]; then\n",
                "  installed_version=$(cat /opt/cfncluster/.bootstrapped)\n",
                "  if [ \"$cfncluster_version\" != \"$installed_version\" ]; then\n",
                "    bootstrap_instance\n",
                "  fi\n",
                "else\n",
                "  bootstrap_instance\n",
                "fi\n",
```

```
          "mkdir /tmp/cookbooks\n",
          "cd /tmp/cookbooks\n",
          "curl -s -L -o cookbook.tar.gz $cookbook_url\n",
          "tar -xzf cookbook.tar.gz\n",
          "cd /tmp\n",


          "echo manual | sudo tee /etc/init.d/httpd.override\n",
          "service httpd stop\n",


          "which yum 2>/dev/null; yum=$?\n",
          "which apt-get 2>/dev/null; apt=$?\n",
          "if [ \"$yum\" == \"0\" ]; then\n",
          "  yum -y install libcurl-devel\n",
          "fi\n",
          "if [ \"$apt\" == \"0\" ]; then\n",
          "  apt-get update; apt-get -y install libcurl4-openssl-dev\n",
          "fi\n",

          "useradd -u 1450 galaxy\n",
          "ln -s /export/galaxy-central /galaxy-central\n",
          "ln -s /export/shed_tools /shed_tools\n",
          "ln -s /export/galaxy_venv /galaxy_venv\n",

          "# Call CloudFormation\n",
          "cfn-init ${proxy_args} -s ", { "Ref" : "AWS::StackName" },
          " -v -c default -r ComputeServerLaunchConfig --region ", { "Ref" : "AWS::Region" },
          " || error_exit 'Failed to run cfn-init. If --norollback was specified, check /var/log/cfn-init.log and /var/log/cfncluster.log.'\n",


          "cfn-signal ${proxy_args} --exit-code=0 --reason=\"MasterServer setup complete\" --stack=", { "Ref" : "AWS::StackName" },
          " --resource=ComputeFleet --region=", { "Ref" : "AWS::Region"},
          "\n",
          "# End of file\n"
        ]
      ]
    }
  }
},
"Metadata" : {
  "Comment" : "cfncluster Compute server",
  "AWS::CloudFormation::Init" : {
    "configSets" : {
      "default" : [
        "deployConfigFiles",
        "getCookbooks",
        "chefPrepEnv",
        "shellRunPreInstall",
        "chefConfig",
        "shellRunPostInstall",
        "shellForkClusterReadyInstall",
        "signalComputeReady"
      ]
    },
    "deployConfigFiles" : {
      "files" : {
        "/tmp/dna.json" : {
          "mode" : "000644",
          "owner" : "root",
          "group" : "root",
          "content" : {
            "cfncluster" : {
              "stack_name" : { "Ref" : "AWS::StackName" },
              "cfn_postinstall" : "https://s3.amazonaws.com/omicron-galaxy/omicronPostInstall.sh",
              "cfn_region" : { "Ref" : "AWS::Region" },
              "cfn_scheduler" : "slurm",
              "cfn_encrypted_ephemeral" : { "Ref" : "EncryptedEphemeral" },
              "cfn_ephemeral_dir" : { "Ref" : "EphemeralDir" },
              "cfn_shared_dir" : { "Ref" : "SharedDir" },
              "cfn_proxy" : { "Ref" : "ProxyServer" },
              "cfn_sqs_queue" : { "Ref" : "SQS" },
              "cfn_master" : {
                "Fn::GetAtt" : [
                  "MasterServer",
                  "PrivateDnsName"
                ]
              },
              "cfn_node_type" : "ComputeFleet",
              "cfn_cluster_user" : {
                "Fn::FindInMap" : [
                  "OSFeatures", { "Ref" : "BaseOS" },
                  "User"
                ]
              }
            },
            "run_list" : "recipe[cfncluster::slurm_config]"
          }
        },
        "/etc/chef/client.rb" : {
          "mode" : "000644",
          "owner" : "root",
          "group" : "root",
          "content" : {
            "Fn::Join" : [
              "",
              [
                "cookbook_path ['/etc/chef/cookbooks']"
              ]
            ]
          }
        },
        "/tmp/extra.json" : {
          "mode" : "000644",
          "owner" : "root",
          "group" : "root",
          "content" : "{}"
        }
      },
```

```
          "commands" : {
            "mkdir" : {
              "command" : "mkdir -p /etc/chef/ohai/hints"
            },
            "touch" : {
              "command" : "touch /etc/chef/ohai/hints/ec2.json"
            },
            "jq" : {
              "command" : "/usr/local/bin/jq --argfile f1 /tmp/dna.json --argfile f2 /tmp/extra.json -n '$f1 + $f2 | .cfncluster = $f1.cfncluster + $f2.cfncluster' >
/etc/chef/dna.json || ( echo \"jq not installed\"; cp /tmp/dna.json /etc/chef/dna.json )"
            }
          }
        },
        "getCookbooks" : {
          "commands" : {
            "berk" : {
              "command" : "for d in `ls /tmp/cookbooks`; do cd /tmp/cookbooks/$d;LANG=en_US.UTF-8 /opt/chef/embedded/bin/berks vendor /etc/chef/cookbooks; done ",
              "cwd" : "/tmp/cookbooks",
              "env" : {
                "HOME" : "/tmp"
              }
            }
          }
        },
        "chefPrepEnv" : {
          "commands" : {
            "chef" : {
              "command" : "chef-client --local-mode --config /etc/chef/client.rb --log_level auto --force-formatter --no-color --chef-zero-port 8889 --json-attributes
/etc/chef/dna.json --override-runlist cfncluster::_prep_env",
              "cwd" : "/etc/chef"
            }
          }
        },
        "shellRunPreInstall" : {
          "commands" : {
            "runpreinstall" : {
              "command" : "/opt/cfncluster/scripts/fetch_and_run -preinstall"
            }
          }
        },
        "chefConfig" : {
          "commands" : {
            "chef" : {
              "command" : "chef-client --local-mode --config /etc/chef/client.rb --log_level auto --force-formatter --no-color --chef-zero-port 8889 --json-attributes
/etc/chef/dna.json",
              "cwd" : "/etc/chef"
            }
          }
        },
        "shellRunPostInstall" : {
          "commands" : {
            "runpostinstall" : {
              "command" : "/opt/cfncluster/scripts/fetch_and_run -postinstall"
            }
          }
        },
        "shellForkClusterReadyInstall" : {
          "commands" : {
            "clusterreadyinstall" : {
              "command" : "/opt/cfncluster/scripts/fetch_and_run -clusterreadyinstall"
            }
          }
        },
        "signalComputeReady" : {
          "commands" : {
            "compute_ready" : {
              "command" : "/opt/cfncluster/scripts/compute_ready"
            }
          }
        }
      }
    }
  },
  "ScaleUpPolicy2" : {
    "Type" : "AWS::AutoScaling::ScalingPolicy",
    "Properties" : {
      "Cooldown" : { "Ref" : "ScalingCooldown" },
      "ScalingAdjustment" : { "Ref" : "ScalingAdjustment2" },
      "AdjustmentType" : "ChangeInCapacity",
      "AutoScalingGroupName" : { "Ref" : "ComputeFleet" }
    },
    "Metadata" : {

    }
  },
  "AddCapacityAlarm2" : {
    "Type" : "AWS::CloudWatch::Alarm",
    "Properties" : {
      "Threshold" : { "Ref" : "ScalingThreshold2" },
      "Period" : { "Ref" : "ScalingPeriod" },
      "EvaluationPeriods" : { "Ref" : "ScalingEvaluationPeriods" },
      "Statistic" : "Sum",
      "AlarmActions" : [{ "Ref" : "ScaleUpPolicy2" }
      ],
      "Namespace" : "cfncluster",
      "ComparisonOperator" : "GreaterThanOrEqualToThreshold",
      "Dimensions" : [{
          "Name" : "Stack",
          "Value" : { "Ref" : "AWS::StackName" }
        }
      ],
      "MetricName" : "pending"
    },
    "Metadata" : {

    }
  },
  "ScaleUpPolicy" : {
    "Type" : "AWS::AutoScaling::ScalingPolicy",
    "Properties" : {
```

```
    "Cooldown" : { "Ref" : "ScalingCooldown" },
    "ScalingAdjustment" : { "Ref" : "ScalingAdjustment" },
    "AdjustmentType" : "ChangeInCapacity",
    "AutoScalingGroupName" : { "Ref" : "ComputeFleet" }
  },
  "Metadata" : {

  }
},
"AddCapacityAlarm" : {
  "Type" : "AWS::CloudWatch::Alarm",
  "Properties" : {
    "Threshold" : { "Ref" : "ScalingThreshold" },
    "Period" : { "Ref" : "ScalingPeriod" },
    "EvaluationPeriods" : { "Ref" : "ScalingEvaluationPeriods" },
    "Statistic" : "Average",
    "AlarmActions" : [{ "Ref" : "ScaleUpPolicy" }
    ],
    "Namespace" : "cfncluster",
    "ComparisonOperator" : "GreaterThanOrEqualToThreshold",
    "Dimensions" : [{
        "Name" : "Stack",
        "Value" : { "Ref" : "AWS::StackName" }
      }
    ],
    "MetricName" : "pending"
  },
  "Metadata" : {

  }
},
"OmicronVPC" : {
  "Type" : "AWS::EC2::VPC",
  "Properties" : {
    "CidrBlock" : "10.0.0.0/16",
    "EnableDnsSupport" : "true",
    "EnableDnsHostnames" : "true",
    "InstanceTenancy" : "default"
  }
},
"OmicronIGW" : {
  "Type" : "AWS::EC2::InternetGateway"
},
"OmicronIGWA" : {
  "Type" : "AWS::EC2::VPCGatewayAttachment",
  "Properties" : {
    "InternetGatewayId" : { "Ref" : "OmicronIGW" },
    "VpcId" : { "Ref" : "OmicronVPC" }
  }
},
"MasterSubnet" : {
  "Type" : "AWS::EC2::Subnet",
  "Properties" : {
    "VpcId" : { "Ref" : "OmicronVPC" },
    "CidrBlock" : "10.0.1.0/20",
    "AvailabilityZone" : { "Ref" : "AvailabilityZone" },
    "MapPublicIpOnLaunch" : { "Ref" : "UsePublicIps" }
  }
},
"ComputeSubnet" : {
  "Type" : "AWS::EC2::Subnet",
  "Properties" : {
    "VpcId" : { "Ref" : "OmicronVPC" },
    "CidrBlock" : "10.0.16.0/20",
    "Tags" : [{
        "Key" : "Network",
        "Value" : "ComputeSubnet"
      }
    ],
    "AvailabilityZone" : { "Ref" : "AvailabilityZone" }
  },
  "Metadata" : {

  }
},
"MasterRouteTable" : {
  "Type" : "AWS::EC2::RouteTable",
  "Properties" : {
    "VpcId" : { "Ref" : "OmicronVPC" },
    "Tags" : [{
        "Key" : "Application",
        "Value" : { "Ref" : "AWS::StackName" }
      }, {
        "Key" : "Network",
        "Value" : "ComputeSubnet"
      }
    ]
  }
},
"MasterSubnetRouteTableAssociation" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "MasterSubnet" },
    "RouteTableId" : { "Ref" : "MasterRouteTable" }
  }
},
"MasterRoute" : {
  "Type" : "AWS::EC2::Route",
  "DependsOn" : ["OmicronIGW"],
  "Properties" : {
    "RouteTableId" : { "Ref" : "MasterRouteTable" },
    "DestinationCidrBlock" : "0.0.0.0/0",
    "GatewayId" : { "Ref" : "OmicronIGW" }
  }
},
"ComputeRouteTable" : {
  "Type" : "AWS::EC2::RouteTable",
  "Properties" : {
    "VpcId" : { "Ref" : "OmicronVPC" },
    "Tags" : [{
```

```
        "Key" : "Application",
        "Value" : { "Ref" : "AWS::StackName" }
      }, {
        "Key" : "Network",
        "Value" : "ComputeSubnet"
      }
    ]
  },
  "Metadata" : {

  }
},
"ComputeSubnetRouteTableAssociation" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "ComputeSubnet" },
    "RouteTableId" : { "Ref" : "ComputeRouteTable" }
  },
  "Metadata" : {

  }
},
"ComputeRoute" : {
  "Type" : "AWS::EC2::Route",
  "Properties" : {
    "RouteTableId" : { "Ref" : "ComputeRouteTable" },
    "DestinationCidrBlock" : "0.0.0.0/0",
    "NetworkInterfaceId" : { "Ref" : "MasterENI" }
  },
  "Metadata" : {

  }
},
"MasterSecurityGroup" : {
  "Type" : "AWS::EC2::SecurityGroup",
  "Properties" : {
    "GroupDescription" : "Enable access to the Master host",
    "VpcId" : { "Ref" : "OmicronVPC" },
    "SecurityGroupIngress" : [
        {
            "IpProtocol" : "tcp",
            "FromPort" : "22",
            "ToPort" : "22",
            "CidrIp" : { "Ref" : "AccessFrom" }
        },
        {
            "IpProtocol" : "tcp",
            "FromPort" : "80",
            "ToPort" : "80",
            "CidrIp" : { "Ref" : "AccessFrom" }
        },
        {
            "IpProtocol" : "tcp",
            "FromPort" : "21",
            "ToPort" : "21",
            "CidrIp" : { "Ref" : "AccessFrom" }
        },
        {
            "IpProtocol" : "tcp",
            "FromPort" : "30000",
            "ToPort" : "30100",
            "CidrIp" : { "Ref" : "AccessFrom" }
        }
    ]
  },
  "Condition" : "CreateSecurityGroups",
  "Metadata" : {

  }
},
"ComputeSecurityGroup" : {
  "Type" : "AWS::EC2::SecurityGroup",
  "Properties" : {
    "GroupDescription" : "Allow access to resources in subnets behind front",
    "VpcId" : { "Ref" : "OmicronVPC" },
    "SecurityGroupIngress" : [{
        "SourceSecurityGroupId" : { "Ref" : "MasterSecurityGroup" },
        "IpProtocol" : "-1",
        "FromPort" : "0",
        "ToPort" : "65535"
      }
    ]
  },
  "Condition" : "CreateSecurityGroups",
  "Metadata" : {

  }
},
"ComputeToComputeSecurityGroupIngress" : {
  "Type" : "AWS::EC2::SecurityGroupIngress",
  "Properties" : {
    "IpProtocol" : "-1",
    "FromPort" : "0",
    "ToPort" : "65535",
    "SourceSecurityGroupId" : { "Ref" : "ComputeSecurityGroup" },
    "GroupId" : { "Ref" : "ComputeSecurityGroup" }
  },
  "Condition" : "CreateSecurityGroups"
},
"ComputeToMasterSecurityGroupIngress" : {
  "Type" : "AWS::EC2::SecurityGroupIngress",
  "Properties" : {
    "IpProtocol" : "-1",
    "FromPort" : "0",
    "ToPort" : "65535",
    "SourceSecurityGroupId" : { "Ref" : "ComputeSecurityGroup" },
    "GroupId" : { "Ref" : "MasterSecurityGroup" }
  },
  "Condition" : "CreateSecurityGroups",
  "Metadata" : {
```

```
    }
  },
  "MasterToMasterSecurityGroupIngress" : {
    "Type" : "AWS::EC2::SecurityGroupIngress",
    "Properties" : {
      "IpProtocol" : "-1",
      "FromPort" : "0",
      "ToPort" : "65535",
      "SourceSecurityGroupId" : {  "Ref" : "MasterSecurityGroup" },
      "GroupId" : {  "Ref" : "MasterSecurityGroup" }
    },
    "Condition" : "CreateSecurityGroups",
    "Metadata" : {

    }
  },
  "MasterENI" : {
    "Type" : "AWS::EC2::NetworkInterface",
    "Properties" : {
      "Description" : "cfncluster Master Server",
      "SubnetId" : { "Ref" : "MasterSubnet" },
      "SourceDestCheck" : "false",
      "GroupSet" : [{
          "Fn::If" : [ "CreateSecurityGroups", { "Ref" : "MasterSecurityGroup" }, { "Ref" : "AWS::NoValue" } ]
        }, {
          "Fn::If" : [ "AddAdditionalSG", { "Ref" : "AdditionalSG" }, { "Ref" : "AWS::NoValue" } ]
        }, {
          "Fn::If" : [ "UseExistingSecurityGroup", { "Ref" : "VPCSecurityGroupId" }, { "Ref" : "AWS::NoValue" } ]
        }
      ]
    },
    "Metadata" : {

    }
  },
  "SharedVolume" : {
    "Type" : "AWS::EC2::Volume",
    "Properties" : {
      "AvailabilityZone" : { "Ref" : "AvailabilityZone" },
      "VolumeType" : { "Ref" : "VolumeType" },
      "Size" : {
        "Fn::If" : [ "UseEBSSnapshot", { "Ref" : "AWS::NoValue" }, { "Ref" : "VolumeSize" } ]
      },
      "SnapshotId" : {
        "Fn::If" : [ "UseEBSSnapshot", { "Ref" : "EBSSnapshotId" }, { "Ref" : "AWS::NoValue" } ]
      },
      "Iops" : {
        "Fn::If" : [ "UseEBSPIOPS", { "Ref" : "VolumeIOPS" }, { "Ref" : "AWS::NoValue" } ]
      },
      "Encrypted" : {
        "Fn::If" : [ "UseEBSEncryption", { "Ref" : "EBSEncryption" }, { "Ref" : "AWS::NoValue" } ]
      },
      "KmsKeyId" : {
        "Fn::If" : [ "UseEBSKMSKey", { "Ref" : "EBSKMSKeyId" }, { "Ref" : "AWS::NoValue" } ]
      }
    },
    "Condition" : "CreateEBSVolume",
    "Metadata" : {

    }
  },
  "DynamicPlacementGroup" : {
    "Type" : "AWS::EC2::PlacementGroup",
    "Properties" : {
      "Strategy" : "cluster"
    },
    "Condition" : "CreatePlacementGroup",
    "Metadata" : {

    }
  }
},
"Outputs" : {
  "MasterPrivateIP" : {
    "Description" : "Private IP Address of the Master host",
    "Value" : {
      "Fn::GetAtt" : [
        "MasterServer",
        "PrivateIp"
      ]
    }
  },
  "MasterPublicIP" : {
    "Description" : "Public IP Address of the Master host",
    "Value" : {
      "Fn::GetAtt" : [
        "MasterServer",
        "PublicIp"
      ]
    },
    "Condition" : "MasterPublicIp"
  },
  "GangliaPrivateURL" : {
    "Description" : "Private URL to access Ganglia",
    "Value" : {
      "Fn::Join" : [
        "",
        [
          "http://", {
            "Fn::GetAtt" : [
              "MasterServer",
              "PrivateIp"
            ]
          },
          "/ganglia/"
        ]
      ]
    }
  },
```

```
    "GangliaPublicURL" : {
      "Description" : "Public URL to access Ganglia",
      "Value" : {
        "Fn::Join" : [
          "",
          [
            "http://", {
              "Fn::GetAtt" : [
                "MasterServer",
                "PublicIp"
              ]
            },
            "/ganglia/"
          ]
        ]
      },
      "Condition" : "MasterPublicIp"
    }
  },
  "Metadata" : {
    "AWS::CloudFormation::Interface" : {
      "ParameterGroups" : [{
          "Label" : {
            "default" : "Essential Parameters"
          },
          "Parameters" : ["AvailabilityZone", "KeyName"]
        }, {
          "Label" : {
            "default" : "Important Parameters"
          },
          "Parameters" : ["OmicronDockerImage", "MasterInstanceType", "ComputeInstanceType"]
        }, {
          "Label" : {
            "default" : "Advanced Network Parameters"
          },
          "Parameters" : ["VPCSecurityGroupId", "AccessFrom", "UsePublicIps"]
        }, {
          "Label" : {
            "default" : "Advanced EC2 Parameters"
          },
          "Parameters" : ["BaseOS", "EC2IAMRoleName", "Tenancy"]
        }, {
          "Label" : {
            "default" : "Advanced Cluster Parameters"
          },
          "Parameters" : ["ClusterType", "S3ReadResource", "S3ReadWriteResource", "SharedDir", "EphemeralDir", "InitialQueueSize", "MaxQueueSize", "MaintainInitialSize",
            "ScalingPeriod", "ScalingEvaluationPeriods", "ScalingThreshold", "ScalingAdjustment", "ScalingThreshold2", "ScalingAdjustment2", "ScalingCooldown"]
        }
      ]
    }
  }
}
```