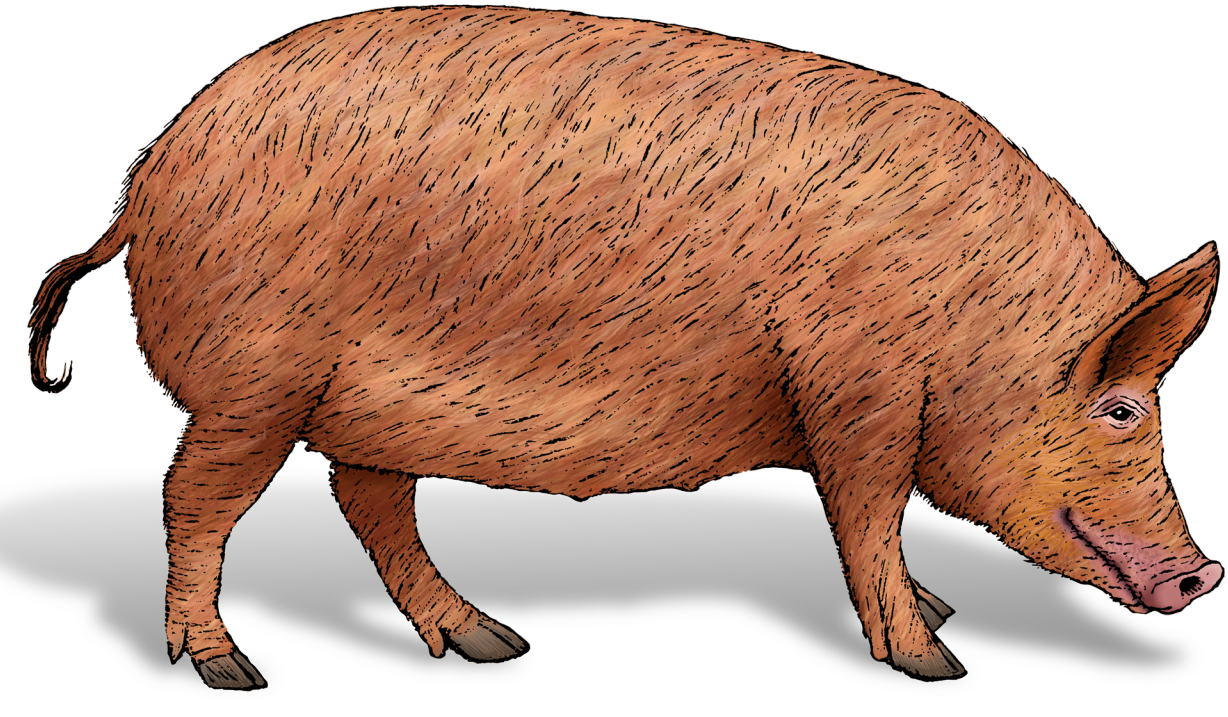
O'REILLY®

Genomics in the Azure Cloud

Scaling Your Bioinformatics Workloads Using Enterprise-Grade Solutions



Colby T. Ford

O'REILLY®

Genomics in the Azure Cloud

Bridge the gap between general cloud computing architecture in Microsoft Azure and scientific computing for bioinformatics and genomics. With this practical guide, technical, computational, and research personnel will learn how to apply Azure's architecture patterns and services in your company's bioinformatics practice.

Author Colby T. Ford provides code examples that you can reuse for your specific needs. You'll learn how Azure cloud provides a perfect vehicle to scale, automate, and collaborate on bioinformatics analyses, far beyond what is possible in restrictive, on-premises computing environments. And you'll get plenty of concrete examples that illustrate how to use a given service in a bioinformatics context.

This book provides valuable advice on how to:

- **Use enterprise platform services to easily scale your bioinformatics workloads**
- **Organize, query, and analyze genomic data at scale**
- **Build a genomics data lake and accompanying data warehouse**
- **Use Azure Machine Learning to scalably train machine learning models, track model performance, and deploy winning models**
- **Orchestrate and automate processing pipelines using Azure Data Factory and Databricks**
- **Cloudify your organization's existing bioinformatics pipelines by moving your workflows to Azure's high-performance compute services**

"Dr. Ford makes taking control of your bioinformatics pipelines and storage easier than ever. The approach of this book is sensible, straightforward, and can easily function as a desktop reference for any life science professional."

—**CJ Knapp**
Healthcare & Life Sciences Cloud Solution Director, 3Cloud, LLC

Dr. Colby T. Ford is a cloud AI and genomics architect with a diverse background in data science and bioinformatics. He is the founder of Tuple, a Microsoft and Databricks partner consulting firm that builds cloud solutions in the life sciences space. Colby is a researcher in infectious diseases and human genomics and teaches cloud computing at the University of North Carolina at Charlotte.

DATA

US $69.99 CAN $87.99

ISBN: 978-1-098-13904-9

9 781098 139049 56999

Twitter: @oreillymedia
linkedin.com/company/oreilly-media
youtube.com/oreillymedia

Genomics in the Azure Cloud

Scaling Your Bioinformatics Workloads Using Enterprise-Grade Solutions

Colby T. Ford

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Genomics in the Azure Cloud
by Colby T. Ford

Copyright © 2023 Tuple, LLC. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Michelle Smith
Development Editor: Shira Evans
Production Editor: Ashley Stussy
Copyeditor: nSight, Inc
Proofreader: Kim Wimpsett
Indexer: nSight, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

November 2022: First Edition

Revision History for the First Edition
2022-11-11: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098139049* for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Genomics in the Azure Cloud*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-13904-9

[LSI]

Table of Contents

Preface ..... **vii**

1. Essentials of Cloud Architecture **1**
Cloud Horsepower 1
Considerations for the Cloud 2
Three Benefits of the Cloud 4
Types of Cloud Services 5
Infrastructure Services 5
Platform Services 6
Software Services 8
Azure Environment Organization 9
Getting an Azure Account 12
Welcome to the Azure Portal 14
Setting Up a Resource Group 14
Creating Resources 18
Free Services 22
Basics of the Bioinformatics Workflow 24
Primary Analysis 24
Secondary Analysis 26
Tertiary Analysis 29
Other Analyses 30
Other File Formats 30

2. Organizing Genomics Data with Data Lakes **37**
Organizing Your Genomics Data 38
Going for Bronze, Silver, and Gold 38
Letting Your Bioinformatics Workflow Dictate Your Data Lake Organization 40

Planning for -omics and Non-omics Data Together 42
Creating a Data Lake with Azure Storage 43
Blob Storage Versus Data Lake Storage 48
Balancing Costs Versus Performance in Data Storage 49
The Goldilocks Method of Storage Tiers 49
Genomics Data Lifecycle 50
Managing Access Inside the Lake 58
Role-Based Access Control 59
Access-Control Lists 61
Azure Open Datasets for Genomics 63

3. Querying Variant Data in SQL. 69
Building a Genomics Data Warehouse 71
Example: Lab Results 71
Data Warehouse Architecture for Genomics 72
Azure Synapse Analytics 77
Creating an Azure Synapse Analytics Workspace 77
Registering Services in Subscriptions 81
Getting to Work in the Synapse Workspace 84
Using Open Row Sets 87
Creating External Tables 89
Did Someone Say "Pool Party"? 93
Connecting to More Data Sources 96
Azure SQL DB 100
Creating a Database in Azure SQL DB 100
Relaxing at Your Genomics Data Lakehouse 108
Efficient File Formats 109

4. Orchestrating Data Movement and Transformation. 113
Creating Your Data Factory 114
Getting Started with Data Movement 119
Getting Data into Your Data Lake Using the Copy Data Tool 119
Linking to NCBI's FTP Server 121
Transforming Data Using Data Flows 130
Building and Triggering Pipelines for Automation 147

5. Azure Databricks (and Apache Spark). 149
Introduction to Apache Spark and Databricks 149
Setting Up an Azure Databricks Workspace 153
Connecting Databricks to Your Data Lake 162
Processing Variant Data with the Glow Package 165
Exploring DataFrames 168

Automating Variant Data Processing 170
Orchestrating a Databricks Notebook from Data Factory 173
A Brief Interlude About Distributed File Formats 185
Using Other Tools in Databricks 187
Single-Node Bioinformatics Tools 188
Koalas 189
Hail 190

6. Azure Machine Learning. 195
How to Scale Machine Learning Tasks 195
Creating an Azure Machine Learning Workspace 197
Training a Drug Sensitivity Model 200
Creating a Compute Instance in Azure Machine Learning Studio 201
Datastores and Datasets 203
Experimenting with Cluster-Based Training 209
Automating Model Training with AutoML 213
Explainable Machine Learning 216
Using Azure Machine Learning Not for Machine Learning 218
Performing Alignment in a Notebook 218
Custom Docker Images for Bioinformatics 219

7. High-Performance Computing and Other Compute Services. 227
Bring Your Own Pipeline (BYOP) 228
Why Azure for HPC? 228
Azure Batch 230
Scaling Workloads with Cromwell 231
Azure CycleCloud 238
Setting Up CycleCloud Clusters 239
Microsoft Genomics 258
Alignment and Variant Calling with the msgen Package 258

8. Deployment, Security, Compliance, and Potpourri. 265
Automating the Deployment of Cloud Resources 265
Dev, Staging, and Prod 266
Lifting Your Deployment with ARMs and Biceps 266
Security Planning 275
Azure Active Directory 275
Role-Based Access Controls and Access-Control Lists 278
Compliance 280
HIPAA, HITECH, and HITRUST 281
Azure Blueprints 284
Cost Considerations 286

Azure Pricing Calculator 286
Retail Pricing Versus Enterprise Agreements 288
Budgeting Examples 289
Quota Problems 293
Please, Sir, Can I Have Some More (vCPUs)? 295
Getting General Support 297

Conclusion **303**

Index **307**

Preface

Genomics in the Azure Cloud is a book that attempts to bridge the gap between general cloud computing architecture in Microsoft Azure and scientific computing for bioinformatics and genomics.[1] This is meant to be a practical guide to architecting performant solutions in Azure for a variety of bioinformatics use cases.

Who Should Read This Book

This book is written for people with experience in bioinformatics and genomics but not necessarily cloud computing. This book can also be valuable for those who are cloud engineers looking to get a better understanding of how cloud architecture should work for bioinformatics.

In many university programs, bioinformatics students are exposed to technical concepts to analyze genetic data. Then, they may also be taught how to use on-premises high-performance computing (HPC) environments (*clusters*) to run larger workloads. These skills will be valuable when transitioning to the cloud. So don't worry, all the skills you learned in grad school aren't for naught.

Many online resources for cloud-enabled bioinformatics focus only on human genomics and, more specifically, how to use the Broad Institute's set of tools (such as GATK, Picard, Cromwell, etc.) in the cloud. I'd like to generalize this book a bit more since, in my experience, people use bits and pieces of software from all over the place, combining them to make a patchwork of tools to fit their specific needs. Also, as an infectious disease guy, I have used lots of different tools in the past that are never mentioned by the human-focused online resources. So all that to say, this book will not be focused on any specific species or suite of tools. I want this book to be useful to

1 See "A Brief Guide to Genomics," National Human Genome Research Institute, accessed September 14, 2022, *https://oreil.ly/xERCI*.

scientists no matter if they work on humans or viruses or elephants or plants or whatever…

Genomics in the Azure Cloud will teach readers how to use enterprise platform services to scale their bioinformatics workloads with ease. Plus, the book starts off by covering how to organize and query the vast amounts of genomics data an organization may have by building a genomics data lake and accompanying data warehouse.

The code examples in this book should serve as a reference point or a practical recommendation of an approach that has worked for me in the past. Each of the code snippets can be thought of as part of a recipe that can be edited to fit your specific needs.

After reading this book, you should have a solid understanding of the services that are offered in Azure and how they might be used in your bioinformatics practice. You'll be familiar with some standard architecture patterns, including how to store and organize data and how to analyze it at scale. You'll also understand how to cloudify your organization's existing bioinformatics pipelines by converting the workflows to work in compute services in Azure.

Some technical things that may be helpful to know:

- Proficiency with Python (or R) is a good idea since this book covers bioinformatics programming and machine learning, which usually require one of these languages.
- Familiarity with database concepts, including some knowledge of SQL, may be useful when we talk about how to store and manage tons of -omics data.
- Knowledge of primary, secondary, and tertiary analysis pipelines will help you brainstorm what steps in your workflow will benefit most from being in the cloud.
- Understanding of the esoteric file formats seen in bioinformatics will be a plus. So, if any of the following formats are unfamiliar, take a second to look them up: *.fasta/fastq*, *.sam/bam*, *.vcf*, *.bgen*, *.bed*, *.gtf*, *.gff*.

How the Book Is Organized

This book is structured more like a reference manual than a novel. It is organized by an overarching task, adding in the service components to make an all-encompassing architecture by the end of the book. The bulk of each chapter will be information about the individual services and the considerations for making certain selections. Then there will often be a concrete example to showcase how a given service is used in a bioinformatics context.

We'll start by getting you set up in Azure and acquainted with general concepts of the cloud. Then we'll peruse storing data, moving data about, and analyzing it. We'll finish the book by covering how to deploy these cloud services, and then I'll share some considerations surrounding security and managing costs (your IT team will thank me later).

While this book is focused on the Microsoft Azure cloud, almost all of the concepts in this book can be translated to Amazon Web Services or the Google Cloud Platform. Individual services names may differ, but the ideas stated here will remain mostly intact.

Software and Hardware Requirements

All of the activities in this book will be done in Microsoft Azure. So you'll need an Azure account and subscription, which I'll show you how to create in the first chapter.

I would recommend that you access Azure using a modern browser. Also, having a second screen, while not a requirement, can be helpful when we're working in two Azure Portal windows at the same time.

There really isn't any required software since everything can be accessed on the cloud using your browser. However, there are a few applications that I find helpful:

Azure Storage Explorer
: This tool allows you to copy data to/from your Azure Storage account on your local machine. It also handles large file uploads fairly well. Get it here.

Azure Command Line Interface (Azure CLI)
: This command-line application allows you to create and manage Azure resources. It's helpful when deploying services outside the Portal UI, which we'll do a lot of in this book. Get it here.

Visual Studio Code (VS Code)
: Unless you're in love with another text editor, I find VS Code really useful for interacting with the Azure CLI or just simply editing and running code. Plus, with its huge library of add-ons, it doubles as a great IDE for lots of tasks. Get it here.

Azure Data Studio
: If you're going to be querying cloud databases often, Azure Data Studio is a nice UI-based tool that was built to help you query, design, and manage databases. Get it here.

Code Conventions and Downloads

Source code for the examples in the book, presented in a `mono-spaced typeface like this`, is available for download on GitHub.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/colbyford/genomics-in-azure-code*.

If you have a technical question or a problem using the code examples, please send email to *bookquestions@oreilly.com*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Genomics in the Azure Cloud* by Colby T. Ford (O'Reilly). Copyright 2023 Tuple, LLC, 978-0-596-13904-9."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/genomics-in-the-azure-cloud-1e*.

Email *bookquestions@oreilly.com* to comment or ask technical questions about this book.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*.

Follow us on Twitter: *https://twitter.com/oreillymedia*.

Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgments

This book is dedicated to my grandmother, Freda (or "Meme," as I called her), who passed away during the writing of this book. She was my biggest supporter from day one and always pushed me to learn more about anything I found interesting. Whether it was the piano, math, music, programming, or science, she was always there to cheer me on. She battled breast cancer, cardiovascular disease, kidney disease, and more, all with the goal to be there for us, her family. Unfortunately, she isn't here to ask about the animal on the cover or to read the book from cover to cover and then excitedly call me to ask questions for hours on end. But she lives on in the work I do every day. I can never thank her enough.

Thank you to my former colleagues-turned-friends, C. J. Knapp and Larry Baker, for all of your help and critiques. Also, thank you to the genomics team members at Microsoft Research, Erdal Cosgun, Roberto Lleras, and Mutlu Dogruel, who helped give valuable insights into the future advancements of the Azure platform for genomics. A special thank-you to William Brandler, formerly of the Databricks genomics team, for all your -omics wisdom and Spark support.

In addition, I would like to thank my publisher-assigned reviewers, Gustavo Patino, Jonah Andersson, Kerrie Holley, and the 10+ other anonymous reviewers. Peer review is a thankless task, but it is super helpful in making this work readable and useful for a broader set of eyes.

Last but not least, I would like to thank my partner, Brandon. Thank you for the never-ending support that you give me on all of my crazy ideas. I know I always take on too much and our relationship suffers at times due to my ongoing desire to take on more and more (to my own detriment). Even if I rarely say it, just know that I love you and that your support does not go unnoticed. Since this book is out now, we can go on vacation, right?

Thank you all a million times over.

CHAPTER 1

Essentials of Cloud Architecture

Microsoft Azure was publicly announced in October 2008 and started to take off in the mid-2010s. Over the last decade, Azure has seen a ton of growth and is now the second-largest public cloud, second only to Amazon Web Services. With Azure, creating scalable solutions for genomics is a great use of all the cloud has to offer. From data lakes to machine learning, you can store, process, and analyze huge quantities of genomics data using Azure services with ease.

In this chapter, I'll introduce you to the basics of Azure along with some core cloud concepts. We'll then cover some of the basics about how your Azure environment is organized and what the different categories of services are. Plus, I'll walk through how to set up an Azure account and how to navigate the Azure Portal. This chapter speaks generally about cloud concepts, but we'll focus on the Azure cloud specifically for the entire book. Next, we'll go over a bit of background about the cloud, as this will help you understand how parts of the bioinformatics workflow fit within a cloud-based environment. At the end of the chapter, we'll also cover some basics of the bioinformatics workflow just to get everyone up to speed with file formats and analyses common in this field.

Cloud Horsepower

In the late 19th century, as people transitioned their method of transportation from horses to cars, an obvious benefit was an increase in speed from point A to point B. While slow speed wasn't always considered a problem with horse-based transport, care for these pre-automobile transport pets was. Feeding the horses, caring for them, having enough land to keep them, and dealing with the crap (literally) were all reduced or eliminated with the onset of the car.

This is a perfect analogy for the cloud-based genomics. The major cloud vendors often talk about the cloud as a means to scale your workloads—that is, speed them up to do what you do today but bigger, better, and faster. This is true with cloud-based genomics, but this is just the tip of the benefits iceberg. In addition to speed, the cloud reduces the need to manage the nitty-gritty components of the infrastructure, security, and segmentation of resources while giving you the ability to automate, collaborate, and centralize your workloads.

So, to belabor this analogy further, you could always better your genomics processes by buying equipment that adds horsepower locally, or you could rent a fleet of machines to meet the ever-changing and ever-growing needs of your organization.

Before I move on to describe each of the different types of cloud services in more detail, let's first discuss some considerations for the cloud, followed by some of Azure's main benefits.

Considerations for the Cloud

As a consultant, I've heard a ton of questions about the cloud, how it works, and why you should or shouldn't use it. The real answer is: it depends. The "cloud" is a blanket term that describes renting compute time and resources from another organization (i.e., "renting someone else's computer"). Though I hate to boil the cloud down to such a curt statement, it is a good concept to keep in mind as I try to demystify the cloud for you in this book. By paying for cloud resources for data storage and compute rather than buying your own, the cloud decreases the management effort of these items through the ability to template and automate. Plus, larger enterprise clouds, like Microsoft Azure, give you global data access and scalability.

Next, I'll explain why the following three statements are common misconceptions.

"I have to move everything to the cloud at once."

I've heard this fear from IT teams that focus on on-premises (on-prem) architecture. They feel like they need to migrate everything to the cloud at once for the cloud to be useful and adopted at the organization. This simply isn't true. Your organization can have a "hybrid cloud" environment where some things are housed on-prem and some things are housed on the cloud. This is especially true when talking about organizations that have many legacy systems that would be very cumbersome to migrate. Moving data and systems to the cloud can be time-consuming, challenging, and expensive, so there is a need to plan accordingly before biting off more than you can chew.

I often recommend that organizations start small and with a pain point. For example, if you have an analysis that takes too long to run, work on migrating it and only the required data for it to complete. Then you can start to evaluate the effectiveness of the

cloud for your organization, get buy-in from teammates, and demonstrate the capabilities on a small scale. An added benefit of this is that a smaller starter project is easier to estimate in terms of costs and scope. If the smaller project is successful, it will help to create momentum and executive buy-in. If you start small, you spend small.

"The cloud is always cheaper/more expensive."

When large organizations buy on-prem hardware (like a computing cluster), they can easily spend hundreds of thousands to millions of dollars on physical equipment and then tons more money on software licenses. But once this initial capital expense is done, they can use this equipment without worrying about a monthly bill (besides electricity and internet and people costs, of course).

With the cloud, you're effectively paying for what you use. So being smart when architecting solutions is key to cost savings. Using the appropriate-sized cloud services and turning things off when they're not in use are the two most basic principles for cost management. From an accounting perspective, cloud services are an operational expense that are expected to be incurred monthly.

If you like analogies, this is similar to buying a home versus renting. You get more freedom when you buy, but you carry more of the risks and responsibilities along with it. You can rent an apartment and not have to mow the lawn or replace the dishwasher, but you don't own anything at the end of the day. Neither option is best for all people in all situations.

In some cases, owning is cost prohibitive due to people, hardware, and software costs. In contrast, the cloud provides a more pay-as-you-go feel, with less need to manage infrastructure and pay for individual software licenses.

"Our IT security team can manage security better."

In short, I doubt it. All of the major cloud vendors, including Microsoft, spend billions of dollars on cybersecurity. It behooves them to never have any security incidents, so they do everything they can to ensure that doesn't happen. This doesn't mean that cyberattacks and security incidents don't happen in the cloud, but Azure provides a state-of-the-art barrier to protect their cloud and gives users an easier way to manage and monitor their specific resources.

Also, this is kind of the wrong way to think about it. The cloud security stuff doesn't replace your cybersecurity team; it augments and enhances their abilities. They'll still get tons of control over security in your cloud environment, and it'll be backed by best-in-class cloud security services. The Azure cloud comes with tons of features like automatic threat detection, automated cloud audits, DDoS protection, and more. In fact, many of these features are free or very cheap in the cloud, which is something you don't necessarily get out of the box using on-prem appliances.

Azure also provides standards-based compliance for organizations that must comply with regulatory standards. Whether it's HIPAA, HITRUST, SOC2, NIST, or some ISO standard, Azure provides blueprints to help you implement these controls and meet whatever compliance standard necessary.

I will also cover compliance considerations more in depth in "Compliance" on page 280.

Three Benefits of the Cloud

If you're used to doing bioinformatics works using your local workstation or your group's computing equipment, you've likely experienced limitations in what you can accomplish. Data is segmented or difficult to find, you have to wait on other people's analyses to finish before you can run your stuff, and collaborating is limited to everyone crowding around a bench in the lab.

From the various research groups I've helped move to the cloud, here are some of the common benefits I've seen.

Collaboration

In creating a central location for your genomics data, you'll be improving the accessibility of the data to your team and organization. This helps to reduce redundant and old versions of data and cuts down on data silos.

Certain services, like Azure Machine Learning and Databricks, also support collaboration by providing shared workspaces where individuals can contribute together. Outside of the Azure cloud services, tools like Microsoft 365 with Teams and Azure DevOps or GitHub (git-based code repository services) also support collaboration while integrating well with the overall cloud development lifecycle.

Scalability

Scalability is a term used to describe how your performance changes as you try to do more work (such as process more data). Architectures that are not scalable may cause bottlenecks when the amount of data or requests grow above a certain threshold. In contrast (and in a perfect world), scalable architectures handle workload fluctuations with ease and grow to fit the demand in an attempt to keep the overall performance (i.e., speed or efficiency) stable.

Using distributed computing services like Apache Spark in Databricks or Azure Batch, you can reduce time to insight by elastically scaling compute contexts for genomics. Specifically, you can reduce processing times for secondary and tertiary analyses and machine learning by harnessing the power of scalable computing services in the cloud.

Automation

Using cloud services like Azure Data Factory, you can orchestrate and automate processing pipelines. This means that workflows can be triggered when new data becomes available, kicking off and processing your data without the need for a bioinformatician to click "Run."

In addition, you can streamline the availability of data to other systems. This is perfect for keeping dashboards and reports up-to-date with the latest results.

In this section, we've discussed some of the benefits and considerations for using the Azure cloud. Designing and planning your cloud environment will be crucial in ensuring you achieve the best benefits while mitigating costs and friction in its usage.

Types of Cloud Services

Within the cloud, there are different types of services. Why is this important to know? Well, different service types are charged differently to your subscription, as shown in Figure 1-1. They can have large cost implications if you know how to use the different services appropriately.

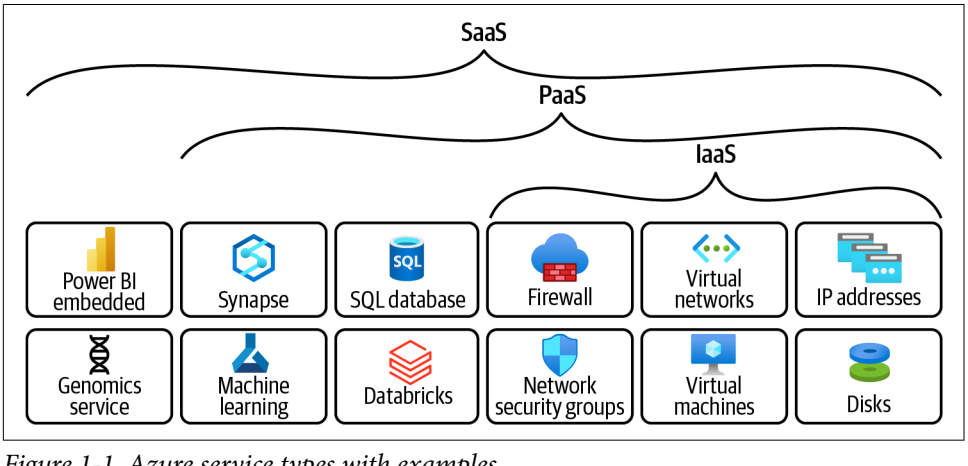


Figure 1-1. Azure service types with examples

Infrastructure Services

The first and most basic type of services are called infrastructure as a service (IaaS). These are usually underlying services that power the more abstracted services that we'll talk about in a second. IaaS services include things like networking, firewall, and virtual machines.

With IaaS services, Azure will spin up the infrastructure, but you can install, configure, and manage the services as you wish. This offers the highest level of customization, but at the expense of having to manage the services more intently.

IaaS options are often charged by the minute within the billing period. So you're paying for them while they are in use. For example, let's say you have a virtual machine running in Azure. While that machine is on, your subscription is being charged by the minute. However, once you turn it off, the charges stop (except for any disk storage or reserved IP addresses that you're using).

Example: Genomics Data Science Virtual Machine

One example of a useful IaaS offering in Azure for genomics is the Genomics Data Science Virtual Machine (GDSVM). This is a custom virtual machine image that includes many software packages that you as a bioinformatician (or a data scientist) may find useful. You can deploy this as either an Ubuntu 18.04 or Windows Server 2016 machine. Once the machine is deployed in your subscription, you'll be able to remote into the machine (using SSH or RDP), where you'll find all the popular data science tools like Jupyter, RStudio, Keras, Tensorflow, etc., along with bioinformatics tools like Bioconductor and GATK.

To deploy the GDSVM for yourself, visit Microsoft Research's GitHub page for the deployment template.

Platform Services

Platform as a service (PaaS) refers to services that supply a more on-demand environment without the need to manage all of the underlying infrastructure. Common Azure PaaS options include databases as a service (like Azure SQL DB, Synapse Analytics, and Cosmos DB) or machine learning and Databricks. These are often simply scalable cloud versions of platforms that could, in theory, be installed and managed locally.

With a PaaS service, Azure will manage the underlying infrastructure (such as the storage, compute, and networking parts), but you can often customize the performance tier that you'd like to use. These services are usually very easy to deploy and get started but still offer a lot of flexibility in their usage and scale. In addition, many Azure services offer "serverless"[1] options that allow you to use a particular tool, such as a database, without managing the underlying server infrastructure at all.

PaaS options are often charged at two levels: 1) a smaller fee for using the platform; and 2) charges for the underlying infrastructure. For example, with Databricks, you'll incur costs for the underlying virtual machines (VMs) that get spun up as part of a cluster plus a "Databricks Unit (DBU)" fee on top of the VM cost per hour. With this consumption-based pricing, you simply pay for the size, performance tier, or usage as you go.

Example: Azure database for PostgreSQL

Let's say, for example, you want to use PostgreSQL to house data for a particular application you're building. You have a couple of options in Azure: (1) spin up a VM and then install PostgreSQL on it (the IaaS option); or (2) use the PaaS option, which is to deploy the Azure Database for PostgreSQL directly.

If you opt for the Azure Database for PostgreSQL option, you'll have a couple choices with regard to how you want to use the service, as shown in Figure 1-2.

Note how there are a couple general size options for database. Once you select one of these, you won't have to manage the underlying machines on which the database runs. You'll just be able to scale the database from the Portal UI and interact with it using SQL as always. For this service, your subscription will be charged by the hour (basically just the VM cost since this is an open source/free application).

1 See "Azure serverless," Azure, accessed September 15, 2022, *https://oreil.ly/9zf6L*.

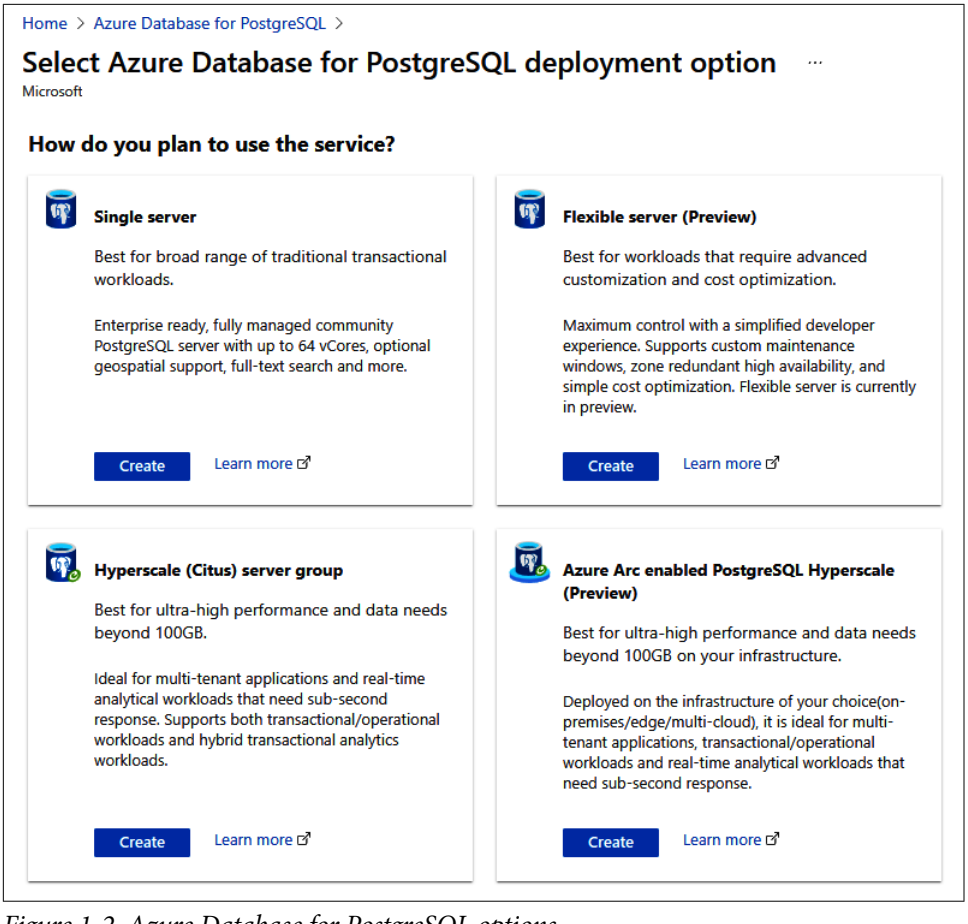


Figure 1-2. Azure Database for PostgreSQL options

Software Services

Software as a service (SaaS) refers to software applications that are available directly in the cloud without managing any underlying infrastructure or installing the application at all. In fact, you've probably used more SaaS applications without even realizing it. Cloud tools like Google Docs or Word Online are basically just SaaS versions of word processing software. The same idea applies in Azure with tools that are made available via APIs or web-based portals. For example, common SaaS offerings in Azure include the Cognitive Services (pre-trained AI models as APIs) or, more specific to bioinformatics, the Azure Genomics service (which we'll cover later).

With an SaaS service, Azure will completely manage the underlying infrastructure, and there is often little opportunity to customize anything.

From a cost perspective, API-based SaaS options are usually charged by the API call (*transaction*). So this means a flat fee for a bundle of API requests. For other application-based SaaS options, you may be charged licensing and then a stationary fee per day/month.

Now that you understand the different types of services you might spin up in Azure, we'll move on to understanding how Azure is organized. It will help to know where you can place all the services as you create them.

Azure Environment Organization

To avoid future confusion, let's walk through the different levels of organization that occur in Azure. This will help you to understand how services are organized, how billing occurs, and how access is controlled later.

Tenant

The tenant is the top-level grouping—aligned to an Azure Active Directory (set of accounts)—for an organization:

- Usually, organizations have only a single tenant for the entire company.
- There are uncommon instances where an organization has more than one tenant to delineate accounts as much as possible. For example, an academic medical center may have a tenant for the university and a separate tenant for the medical arm of the organization.

Subscriptions

Subscriptions are the billable units within a tenant. Subscriptions come in a few different plan types, and it's a little opaque, but I'll try to cover a few of the common ones here:

- Pay-as-You-Go (PAYG)
  - This plan is available without a long-term commitment and provides access to Azure services at full retail pricing. You can pay for this type of subscription using a credit card only. This is the best type of subscription for teams who are just getting started on Azure. There is no difference in pricing for development workloads versus production workloads with this type of subscription.

- Enterprise Agreement
  - This plan usually requires a 12-month commitment and is common among larger organizations with higher Azure usage. This type of subscription provides special pricing for certain services if the organization commits to a certain financial spend on Azure services. Usually, organizations use this subscription type for production workloads as it offers the best service-level agreements and support.
- Enterprise Dev/Test
  - This plan is similar to the Enterprise Agreement plan but offers much lower costs for development and testing workloads on Azure. Organizations use subscriptions of this type as a cost-saving measure when teams are working on developing a new product, platform, or architecture.[2]

Resource Groups

Resource groups are like logical containers for your resources. Often, resource groups hold services that are all related/used for a given solution. For example, you may choose to house all of the services related to your enterprise data warehouse (such as a storage account, a data factory instance, and a Synapse Analytics workspace) within the same resource group. Resource groups allow you to delete a set of resources in Azure, making it easier to manage sets of resources. Also, within the billing breakdown under a subscription, you can also see costs by resource group, which provides an easy way to understand how much a set of resources for a given purpose are costing you. In addition to this, you can set budgets by resource group or subscription.

Regions

Resource groups and individual services are placed in a region. This is a geographical location of an Azure data center where your services and data are hosted. Generally, you pick a region that is closest to you (or your customers), which should reduce latency in getting access to your applications or data. You can select from more than 40 regions across the globe. Most regions have access to most Azure services, but there are some regions that may not have some of the brand-new services (yet). There are also special regions that meet specific security and compliance needs, like the Azure government cloud regions, which are for government entities such as the United States Department of Defense.

If you're just working in your own personal account, this is less important since it's quite simple: you have a single tenant of one user (you) that has a single subscription (the one that Azure makes for you) with one or more resource groups of services.

2 To learn more about all of the current Azure subscription offer types, visit azure.microsoft.com/en-us/support/legal/offer-details/.

If you're using your organization account (from work or school), the tenant is usually managed by IT and includes all the users in the organization. Then the tenant will have multiple subscriptions: usually one for each group in the organization (or three if the organization separates development, staging/testing, and production levels)—or whatever makes sense from a billing/budget perspective. Finally, each subscription will have one or more resource groups of services within them. See Figure 1-3.

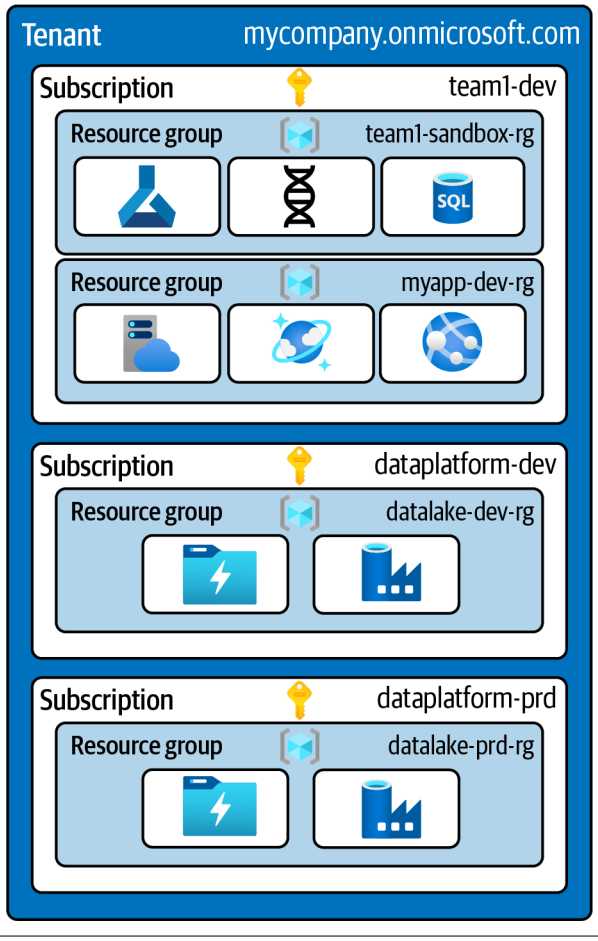


Figure 1-3. Organization of resources within an Azure cloud environment

Note how, in Figure 1-3, you can organize your tenant by making subscriptions for dev/test or production purposes for various applications or teams. In addition, for business-critical applications, it's common to see resource groups mirrored between development and production environments. This allows for a more controlled

promotion of applications into a production environment once the application has been deemed stable and ready for wider organizational use.

Now that you have the basics of Azure under your belt, let's start setting up your very own Azure account.

Getting an Azure Account

If you work for an organization that is already using Azure, you don't necessarily need to create your own account. Just ask your IT department (or whoever manages your Azure environment) to create a sandbox subscription for you to play in.

However, if you want to have your own personal Azure account to play around in, I'll show you how:

1. Navigate to *https://azure.microsoft.com/en-us/free*. (If you are a student, navigate to *https://azure.microsoft.com/en-us/free/students* instead.)
2. If you'd like to get $200 in free credits, click the "Start free" button, shown in Figure 1-4. Otherwise, click "Pay as you go" and then "Get started."[3]

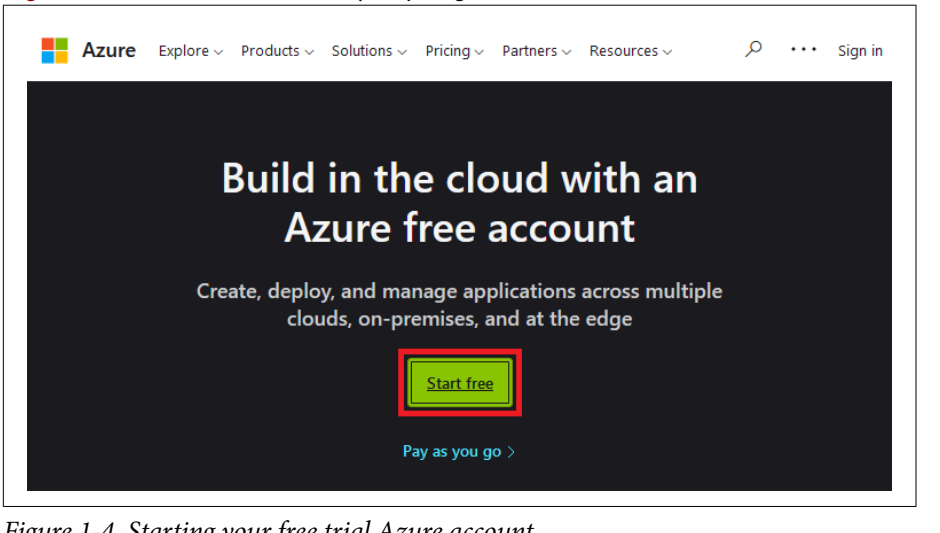


Figure 1-4. Starting your free trial Azure account

3 What's the difference? The "free credits" option doesn't allow you to spin up all of the Azure services, so you'll have to convert to a PAYG subscription later anyway once the credits expire or if you need to spin up services that aren't covered by the credits.

3. Either option will take you to a login screen, as shown in Figure 1-5, where you'll need a Microsoft account. If you already have one (including a GitHub account), go ahead and sign in. Otherwise, create one. (If you are a student, use your university email.)

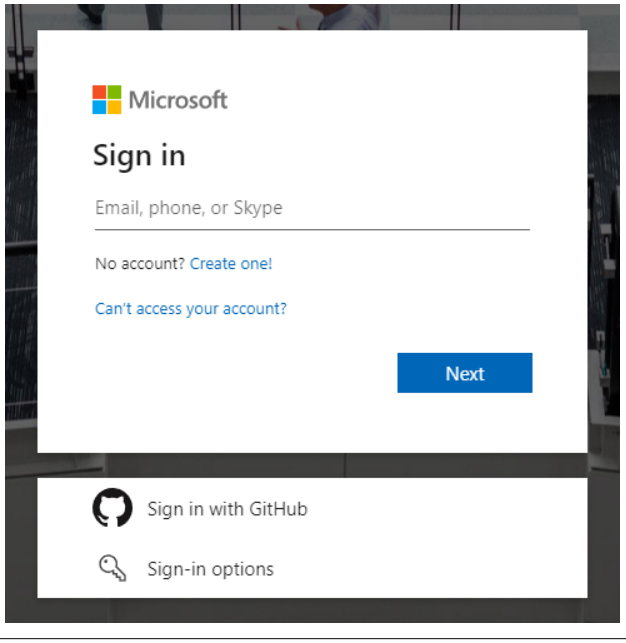


Figure 1-5. Signing in with your Microsoft account

4. In Figure 1-6, you'll input some information to set up a subscription. If you're opting to get the $200 in free credits, this process will give you a free subscription that you will convert to a PAYG subscription later. If you're just going straight for the PAYG option, you won't get the credits, but you won't have to convert the subscription, and you'll be able to spin up any Azure service immediately.

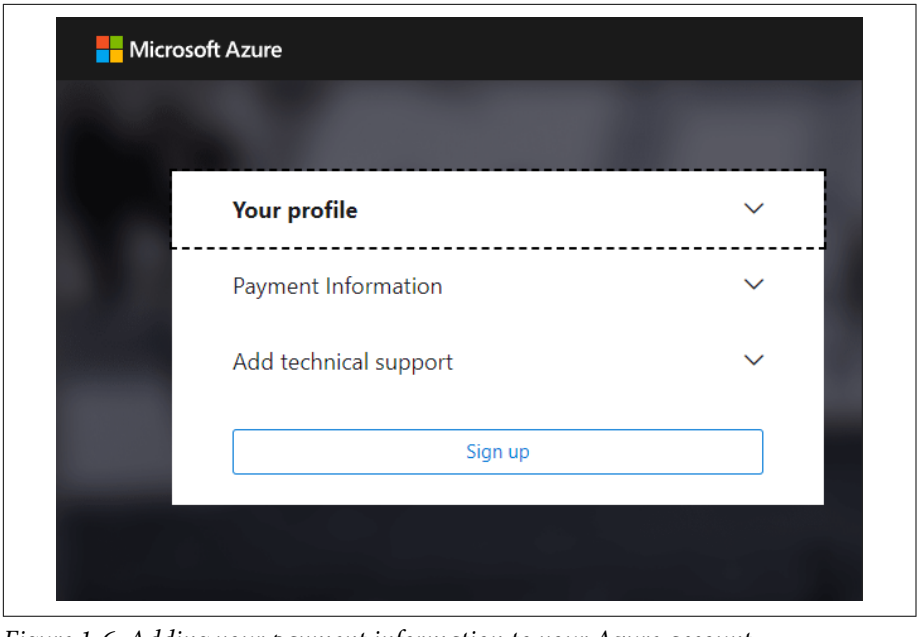


Figure 1-6. Adding your payment information to your Azure account

5. Once you fill out this form and click "Sign up," wait for an email stating that your subscription has been set up. (This could take a little while.) You will then be able to log in to the Azure Portal.

Welcome to the Azure Portal

The Azure Portal is the main entry point to interact with your cloud environment. Once logged in, you can begin spinning up resources as you wish.

Setting Up a Resource Group

Let's walk through a few facets of the Portal and get some things set up first:

1. The Azure Portal will automatically show the commonly accessed resources right from the main screen. The first time that you log in, though, it will look something like Figure 1-7.

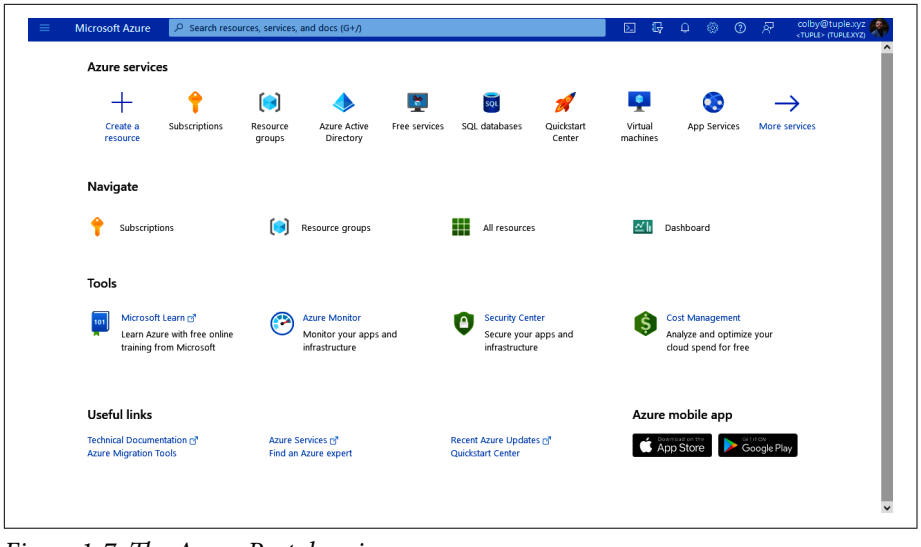


Figure 1-7. The Azure Portal main screen

2. Clicking the gear icon in the top bar will open "Portal settings," as shown in Figure 1-8. You may wish to customize the color theme or whether to show the recent items on the startup page or show a dashboard instead.

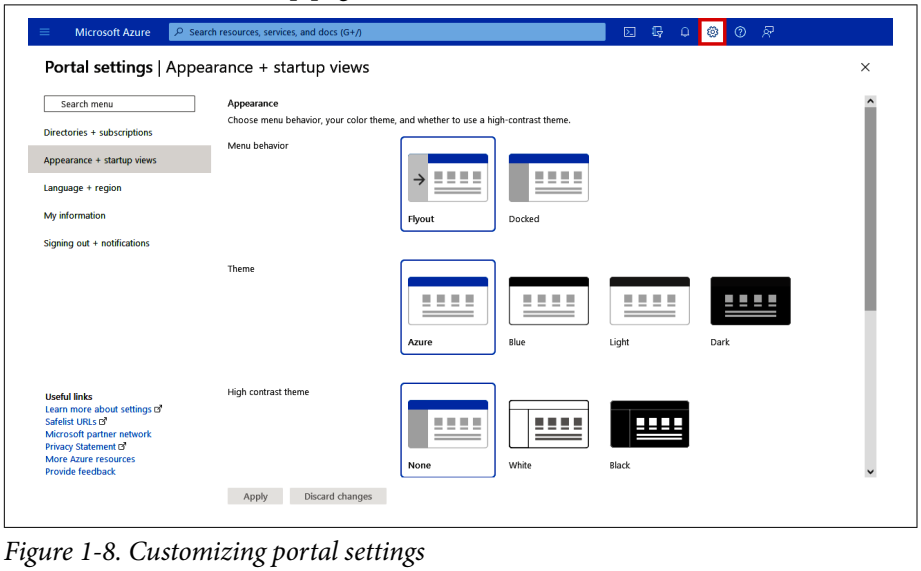


Figure 1-8. Customizing portal settings

3. From the main screen, click "Resource groups" and then click the "+ Create" button to set up your first resource group (see Figure 1-9).

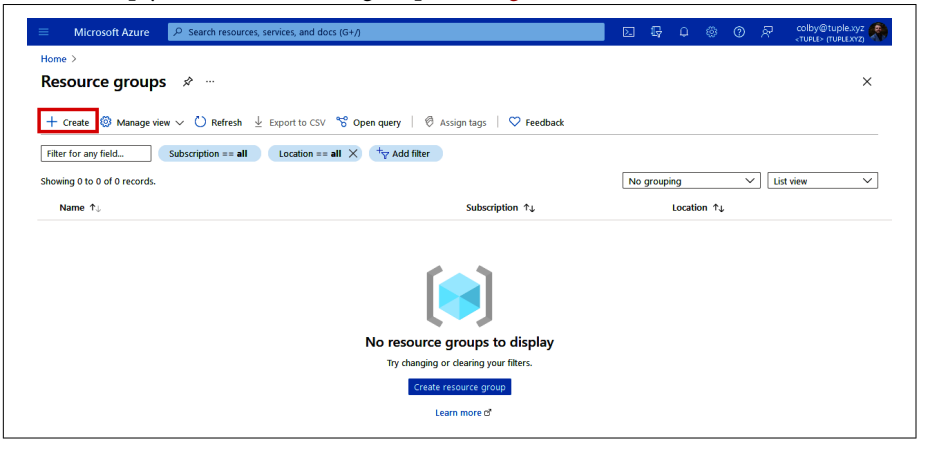


Figure 1-9. Listing resource groups in the Azure Portal

4. A new blade (this is the name for these panels that open in the portal) will open up where you can name the resource group and pick the region. Generally, I would recommend adding *-rg* to the end of your desired name. You can also add *-dev* or *-test* or *-prod* to specify if the resource group is to be used for development, testing, or production purposes, respectively. Finally, pick the region that's closest to you (if you're using your personal account) or pick the region where your organization keeps its other Azure resources (if you're using your work or school account). See Figure 1-10.

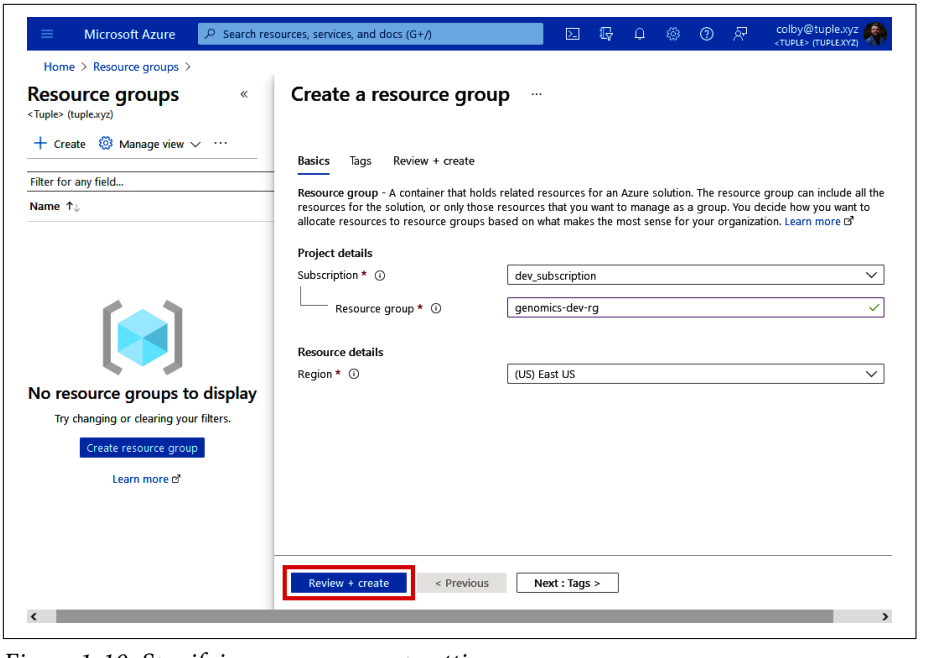


Figure 1-10. Specifying resource group settings

5. Once you've specified the name and region for the resource group, click the "Review + create" button. This will perform a quick check to make sure the resource group can be created. Finally, click the "Create" button, as shown in Figure 1-11, to have Azure start the process to create the resource group. This will take a minute or so.

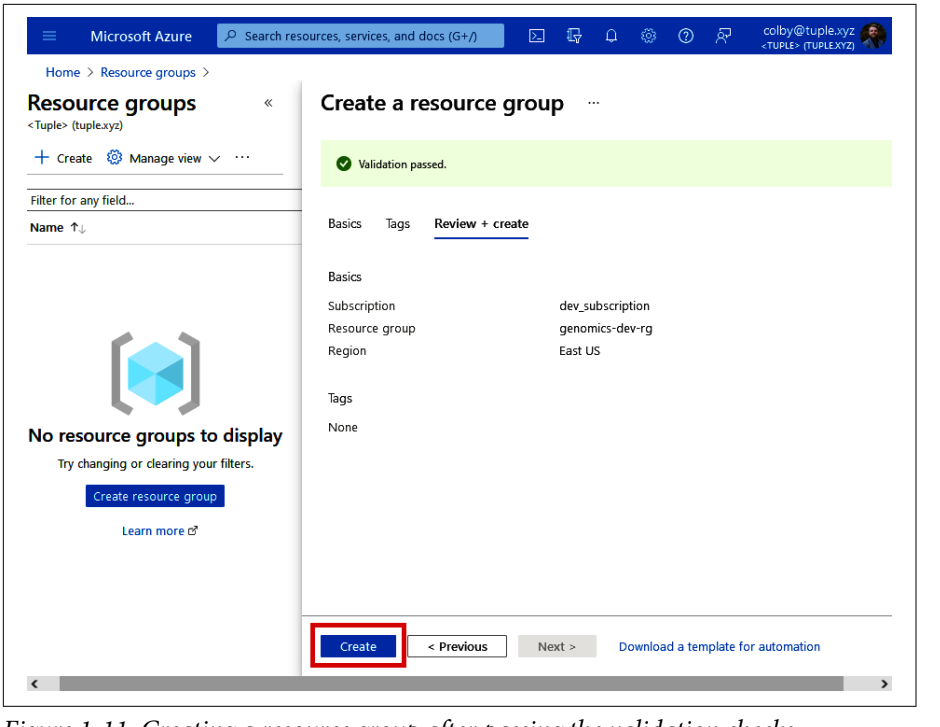


Figure 1-11. Creating a resource group after passing the validation checks

Nice! You now have a resource group to work with. Shall we put something in it?

Creating Resources

1. Back on the main portal screen, click the "+ Create a resource" button. This will open the "Create a resource" menu, as shown in Figure 1-12, where you can search for whatever product you'd like to provision.

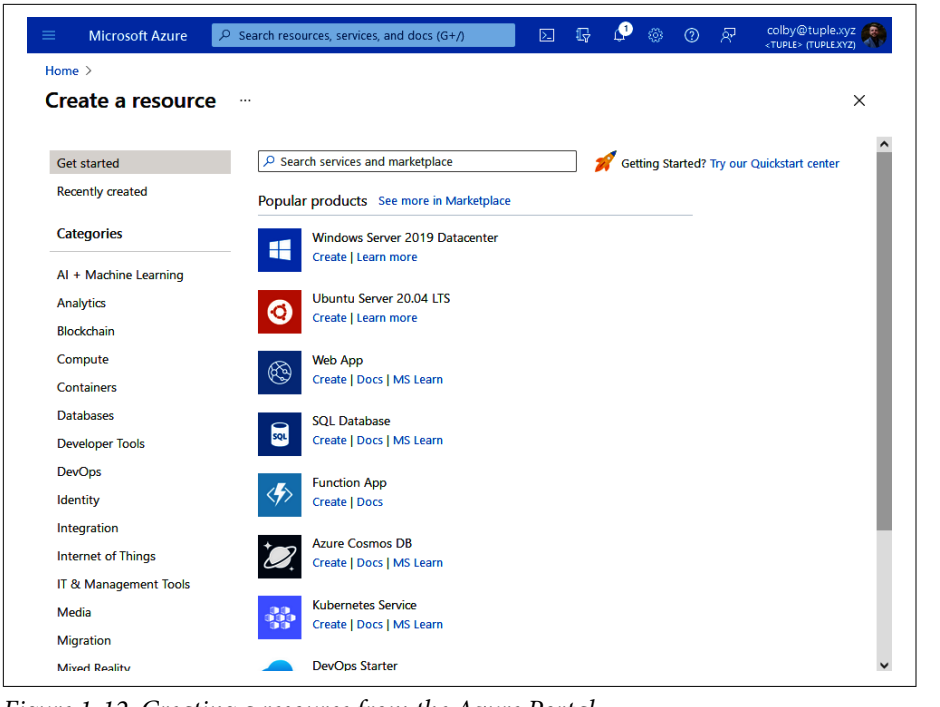


Figure 1-12. Creating a resource from the Azure Portal

2. You can click any of the popular products that show up on the main list, browse any of the categories on the left menu, or simply search for what you're looking for. As an example, let's create a storage account.
3. In the search box, type "storage" and press the Enter key. This will return a list of results that match this query and, ideally, the first option is "Storage account," as shown in Figure 1-13. Click it and click the "Create" button on the next screen.

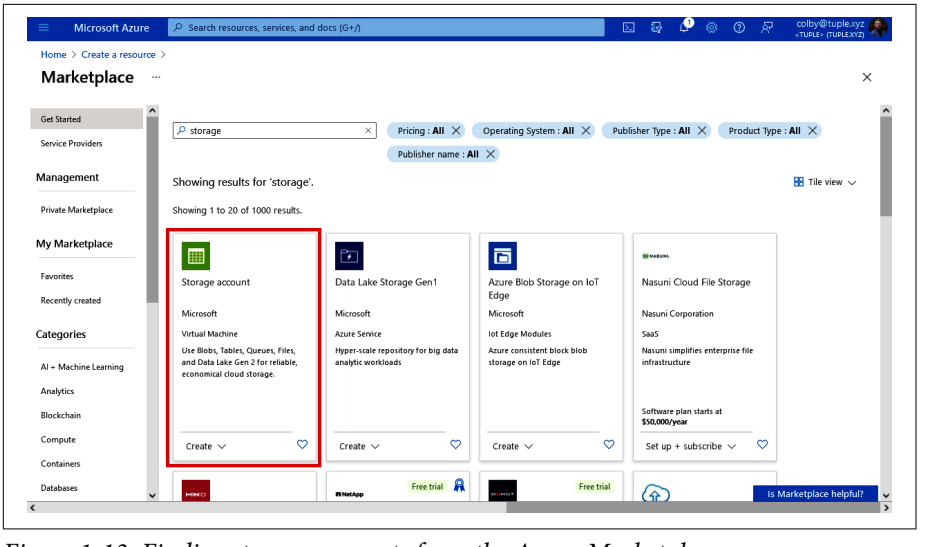


Figure 1-13. Finding storage accounts from the Azure Marketplace

4. For any resource that you want to provision, there are often lots of options that you must specify. For a basic storage account, you'll just need to provide a name for it and a resource group and region to place it in. Once you've specified these things, you can click over to the "Review + create" tab and click the "Create" button (once it has passed the validation, of course). See Figure 1-14.

There are plenty more options to tweak with Azure Storage. Later in this book, we'll walk through how to provision a data lake with geo-redundancy and premium performance, which is a much fancier version of what we're doing now.

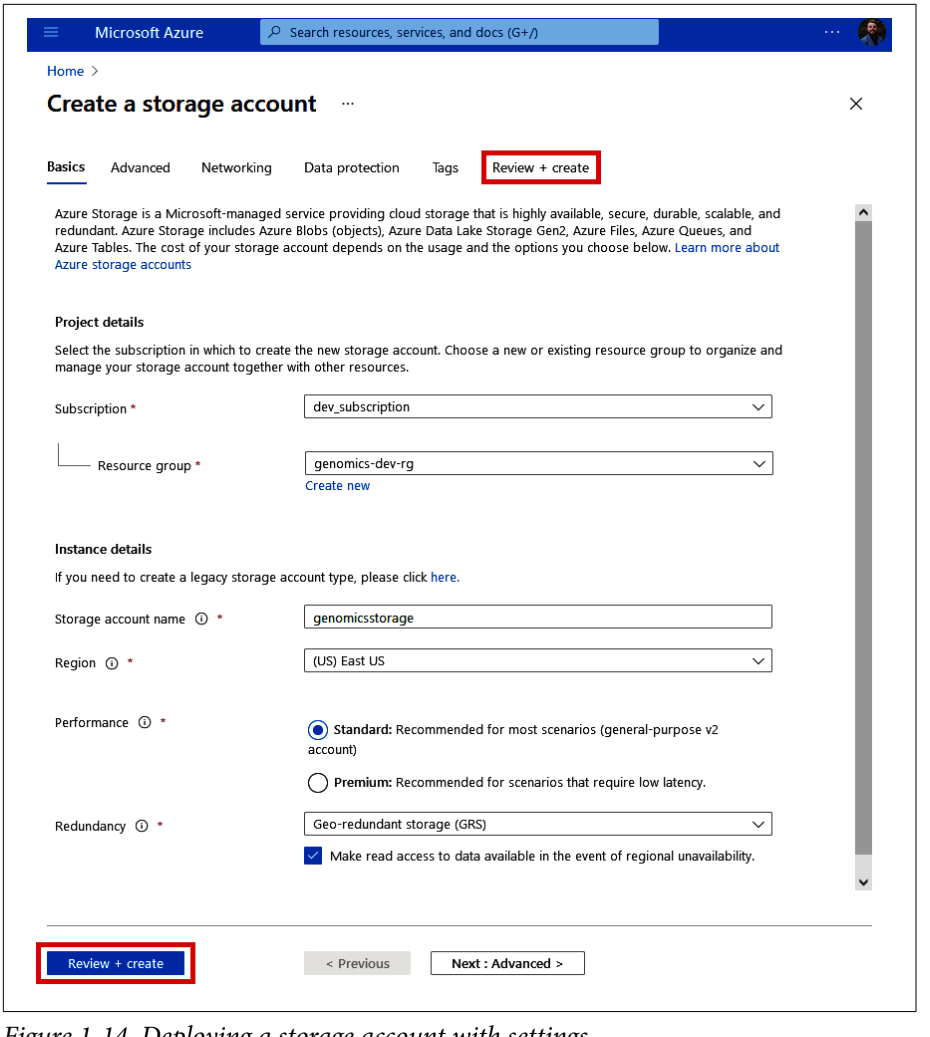


Figure 1-14. Deploying a storage account with settings

5. Once this storage account has finished being deployed, you should then see it on your main portal page under "Recent resources," as shown in Figure 1-15. You can also navigate back to your resource group from before, and you'll see it there as well.

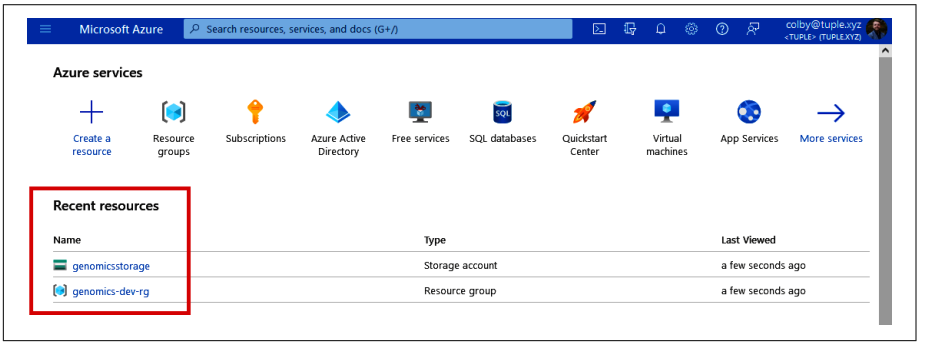


Figure 1-15. Recently used or provisioned resources shown on the Azure Portal

Congrats! You've just created your first resource in Azure.

In this book, we'll be spinning up multiple types of services, but the process is quite similar to what you've just done in this example. Feel free to refer to this section anytime you need a refresher on how to create a service in Azure. In addition to creating services using the Azure Portal (the easiest way), experienced developers, cloud engineers, or IT professionals may find it useful to create resources using the Azure CLI or PowerShell or using infrastructure as code tools (which we'll talk about in Chapter 8).

Free Services

If this is your first foray into Azure or into the cloud in general, you can play with plenty of services for free.[4] This is perfect for learning your way around or prototyping some solutions on the cheap.

From the Azure Portal, you may have a tile at the top under "Azure services" that says "Free services." Otherwise, you can search for "Free services" under "All services" on the side menu, as shown in Figure 1-16.

4 See "Explore free Azure services," Azure, accessed September 15, 2022, *https://oreil.ly/9Xoac*.

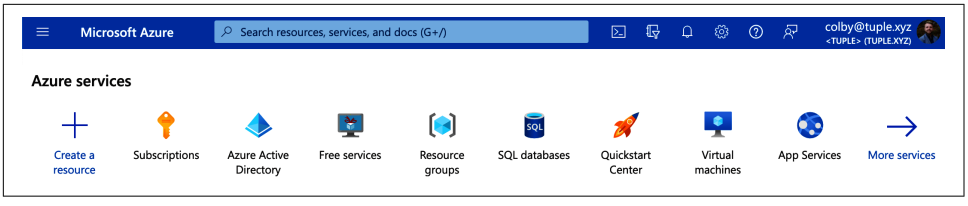


Figure 1-16. Free services from the Azure Portal main screen

This will bring up a list of services that are available at no cost. Notice that some are free for a certain tier for a limited number of hours, like B-tier VMs, or for a specific amount of usage, like Blob storage or some databases. Many of the APIs, such as the Azure Cognitive Services, are free for a certain number of transactions. See Figure 1-17 for some examples of free services you can use in Azure.

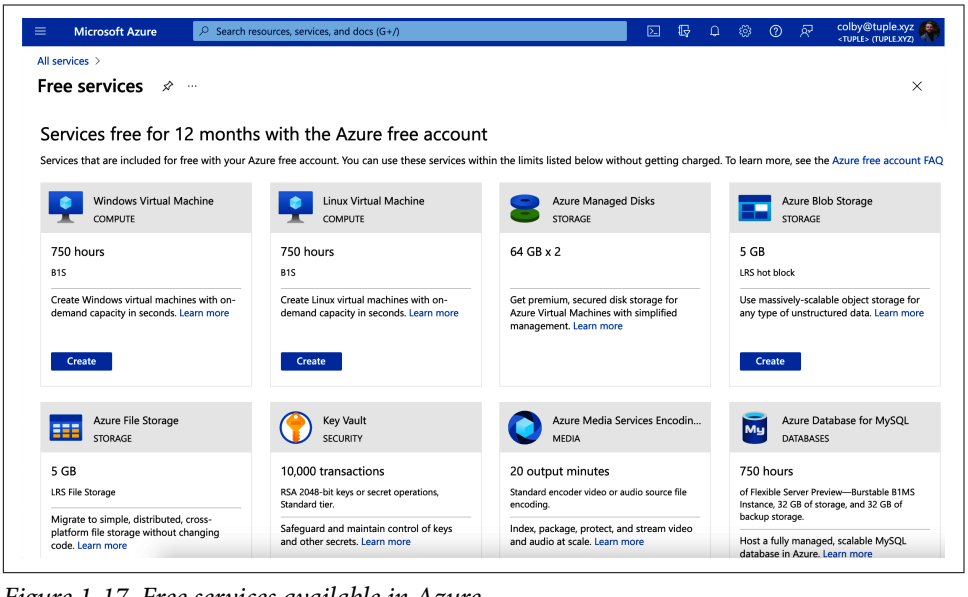


Figure 1-17. Free services available in Azure

You can spin up these services right from this screen. You can also upgrade many of these services once your free time is up or if you need a higher tier of service as you grow.

Also, you don't necessarily have to spin up services from here. For example, if you want to provision the GDSVM from the section "Infrastructure Services," you can simply select the "B1s" VM tier, which is free for 750 hours, and give it a try.

Basics of the Bioinformatics Workflow

If you're reading this book, I'm assuming you know a bit about bioinformatics and genomics already. However, I will provide a quick refresher on some of the basics since the terminology I use in this book may differ from what you've learned in the past.

In bioinformatics, it's common to lay out your analysis steps as steps in a pipeline. Each of these steps processes the input data and converts the information into a more usable form. Beyond complex pipelines, another aspect that makes bioinformatics challenging from a computing perspective is the esoteric file types you may encounter. So, let's walk through the standard steps of a bioinformatics pipeline at a high level (shown in Figure 1-18) to understand the forms the data takes at each stage.

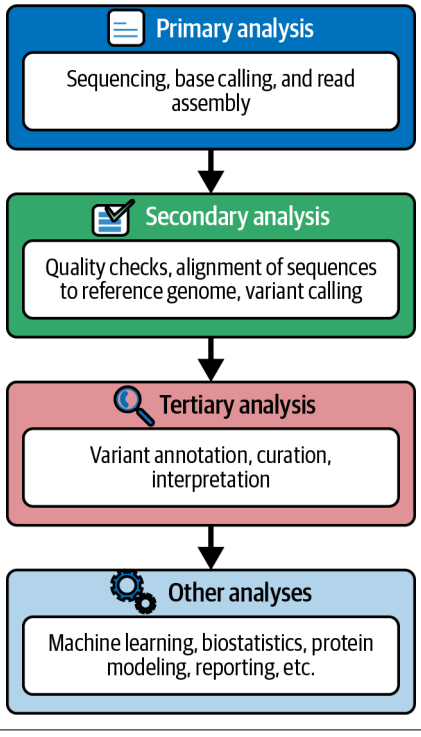


Figure 1-18. General bioinformatics analysis steps

Primary Analysis

Once a sample has been collected from a specimen (such as human blood, a bacterial swab, or plant matter), lab scientists can prepare this sample and load it into a

sequencing machine. These machines will attempt to read the DNA (or RNA) of the sample and perform "base calling," converting the biochemical signals from the machine into A's, C's, T's (or U's), and G's. This sequencing process is called the *primary analysis*.

Usually, the output files of this process are raw sequence data in FASTA or FASTQ format.

FASTA

FASTA files are human readable, meaning they can be opened and read using any text editor on your computer. Each new sequence contains a metadata line that begins with a ">" character, usually followed by a sample ID and then some information about the sample. It's also common for the metadata to be delimited by some other character (such as a "|"), but there is no standard for this.

For example, the following FASTA is part of the reference nucleotide sequence of SARS-CoV-2 from Wuhan, Hubei, China:[5]

```
>NC_045512.2 | SARS-CoV-2 isolate | Wuhan-Hu-1
ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAA
CGAACTTTAAAATCTGTGTGGCTGTCACTCGGCTGCATGCTTAGTGCACTCACGCAGTATAATTAATAAC
TAATTACTGTCGTTGACAGGACACGAGTAACTCGTCTATCTTCTGCAGGCTGCTTACGGTTTCGTCC...
```

FASTQ

FASTQ files are very similar to FASTA, except they also include some information about the quality (or confidence) of the base being reported at a given position. Each new sequence starts with an "@" symbol, followed by metadata. Then, the next few lines are 1) the nucleotide sequence, 2) a line with a "+" sign, and 3) a sequence of quality characters equal to the length of the nucleotide line:

```
@ERR1717084.2 HS29_09589:8:1302:3496:6755#1/1
ATTAACTCTGTGAATATATTTTCAAAATATGCATTTGTTTTTTCTATAGCATCTCGGTCAATGGTTA...
+
@D?EBF@FFG<FEHAGGB8GFEHFHD>H>6H@=HGFF@GHHHFFCG>DDEGHFGCG>2FHEFH;?E?...
```

The quality line might look a bit obscure, but it ranges from a "!" character as the lowest score to a "~" as the highest. The full range of ASCII characters (from worst quality to best) is as follows:

```
!"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
```

5 See Fan Wu et al., "A New Coronavirus Associated with Human Respiratory Disease in China," *Nature* 579 (2020): 265–269, *https://doi.org/10.1038/s41586-020-2008-3*.

Secondary Analysis

Once you have your sequence information, you can then derive more insights by aligning those sequences to reference sequences (or whole reference genomes) and then understand how your samples vary from that reference in a process called *variant calling*.

For many types of research studies, you may wish to align a whole genome (or exome) to a reference and then determine all of the variants the organism has. This is important in human genetics to understand how particular mutations relate to diseases. The pipelines that support these tasks often have multiple steps and may take a while to run given the amount of data being processed.

In other branches of genomics, you may not have nearly as much sequence information, so the process is much easier or more targeted. For example, when studying viruses (like SARS-CoV-2), the entire genome is quite small in comparison to humans or other organisms. Plus, you may care only about particular genes and how they've mutated. In these situations, the secondary analysis pipelines may be shorter and quicker as there's less work to be done.

SAM (and BAM)

The sequence alignment map (SAM) format is a human-readable file type that stores the alignment results of an input sequence against a provided reference. Here are some example lines of a SAM file:

```
@SQ SN:chr1 LN:249250621
@SQ SN:chr2 LN:243199373
@SQ SN:chr3 LN:198022430
...
SOLEXA-1GA-2_220EMB:5:251:979:328 0   chr1   10145  25
      10M AACCCCTAAC hhhhHcWhhH     NM:i:1  X1:i:1  MD:Z:33A2
SOLEXA-1GA-2_220EMB:5:102:214:278 0   chr1   10148  25
      10M CCCTAACCCT hbfhhhXUYh     NM:i:0  X0:i:1  MD:Z:36
SOLEXA-1GA-2_220EMB:5:195:284:685 16  chr1   10149  25
      10M CCAAACACTA ><>B@>?>?D     NM:i:1  X1:i:1  MD:Z:29A3A2
SOLEXA-1GA-2_220EMB:5:35:583:827  0   chr1   10150  25
      10M CTAACCCTAA hhW]MXNO NM:i:1 X1:i:1 MD:Z:10A24A0
SOLEXA-1GA-2_220EMB:5:248:130:724 0   chr1   10152  25
      10M AACCCTAACC hchPhc__  NM:i:1 X1:i:1 MD:Z:17C18
SOLEXA-1GA-2_220EMB:5:236:644:107 16  chr1   10154  25
      10M CCCTAACCCT ICF=A@BHGE NM:i:0 X0:i:1 MD:Z:36
SOLEXA-1GA-2_220EMB:5:165:628:70  16  chr1   10155  25
      10M CCTAACCCTA >D?BAEAA?E NM:i:1 X1:i:1 MD:Z:11A2T1T19
```

The lines at the top of the file (beginning with an "@" symbol) contain metadata about the alignment. In the previous example, the @SQ lines provide information about the reference sequences that were used, their names (e.g., `chr1`), and their

length (denoted by the LN). The metadata lines can also hold information about the file itself, any read group information, etc.

The body of a SAM file contains a standard set of fields as shown in Table 1-1.

Table 1-1. SAM format column specifications

Column	Name	Description	Data type
1	QNAME	Query NAME of the read or the read pair	String
2	FLAG	Bitwise FLAG (pairing, strand, mate strand, etc.)	Integer
3	RNAME	Reference sequence NAME	String
4	POS	1-based leftmost POSition of clipped alignment	Integer
5	MAPQ	MAPping Quality (phred-scaled)	Integer
6	CIGAR	Extended CIGAR string	String
7	MRNM (or RNEXT)	Mate Reference NaMe or Reference name of the NEXT read ('=' if same as RNAME)	String
8	MPOS (or PNEXT)	1-based leftmost mate POSition or position of the NEXT read	Integer
9	ISIZE (or TLEN)	Inferred Insert SIZE or observed template LENgth	Integer
10	SEQ	Query SEQuence on the same strand as the reference	String
11	QUAL	Query QUALity (ASCII-33=phred base quality)	String

However, your file may not have the mate columns 7 and 8 if you don't have information about the next mapped read. Also, information can be added starting in column 12 if there's more data to include.

You can read more about the SAM format specifications at *https://oreil.ly/OFAj9*.

Binary alignment map (BAM) is simply the binary version of a SAM file. These are quite common as they save on storage space and are quicker to read using computers, but they are not human readable without a piece of software (usually Samtools).[6]

VCF

Variant calling is one of the most important parts of secondary analysis as this is the step that distills the information from the alignment down to the more useful bits of information: positions where your sample's sequence varies from the reference. Variant calling is used to provide a list of mutations by position for a given sample, which is useful in understanding diseases, evolution, and more.

6 See "Samtools," accessed September 15, 2022, *https://www.htslib.org*.

The most common format you'll encounter with variant data is the variant call format file (VCF), which looks like this:

```
##fileformat=VCFv4.0 ##fileDate=20090805 ##source=myImputationProgramV3.1
##reference=1000GenomesPilot-NCBI36 ##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description='Number of Samples With Data'>
##INFO=<ID=AN,Number=1,Type=Integer,Description='Total number of alleles in
    called
    genotypes'> ##INFO=<ID=AC,Number=.,Type=Integer,Description='Allele count in
    genotypes'> ##INFO=<ID=DP,Number=1,Type=Integer,Description='Total Depth'>
##INFO=<ID=AF,Number=.,Type=Float,Description='Allele Frequency'>
##INFO=<ID=AA,Number=1,Type=String,Description='Ancestral Allele'>
##INFO=<ID=DB,Number=0,Type=Flag,Description='dbSNP membership, build 129'>
##INFO=<ID=H2,Number=0,Type=Flag,Description='HapMap2 membership'>
##FILTER=<ID=q10,Description='Quality below 10'>
##FILTER=<ID=s50,Description='Less than 50% of samples have data'>
##FORMAT=<ID=GT,Number=1,Type=String,Description='Genotype'>
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description='Genotype Quality'>
##FORMAT=<ID=DP,Number=1,Type=Integer,Description='Read Depth'>
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description='Haplotype Quality'>
##ALT=<ID=DEL:ME:ALU,Description='Deletion of ALU element'>
##ALT=<ID=CNV,Description='Copy number variable region'>
#CHROM POS ID REF ALT QUAL   FILTER INFO FORMAT NA00001 NA00002 NA00003
19 111 .  A  C  9.6 . . GT:HQ 0|0:10,10 0|0:10,10 0/1:3,3
19 112 . A G 10 . . GT:HQ 0|0:10,10 0|0:10,10 0/1:3,3
20 14370 rs6054257 G A 29 PASS NS=3;DP=14;AF=0.5;DB;H2 GT:GQ:DP:HQ
   0|0:48:1:51,51 1|0:48:8:51,51 1/1:43:5:.,.
20 17330 . T A 3 q10 NS=3;DP=11;AF=0.017 GT:GQ:DP:HQ
   0|0:49:3:58,50 0|1:3:5:65,3 0/0:41:3:.,.
20 1110696 rs6040355 A G,T 67 PASS NS=2;DP=10;AF=0.333,0.667;AA=T;DB
   GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2...
20 1230237 . T . 47 PASS NS=3;DP=13;AA=T GT:GQ:DP:HQ
   0|0:54:.:56,60 0|0:48:4:51,51 0/0:61:2:.,.
20 1234567 microsat1 G GA,GAC 50 PASS NS=3;DP=9;AA=G;AN=6;AC=3,1
   GT:GQ:DP 0/1:.:4 0/2:17:2 1/1:40:3
20 1235237 . T . . . . GT 0/0 0|0 ./.
```

The header lines of a VCF file (denoted by ##) contain metadata information about the file. This usually includes information about the software that generated the VCF file, what reference was used, etc. In addition, these lines may contain additional information about certain ALT, INFO, and FORMAT attributes.

In general, the body of the VCF file contains the columns described in Table 1-2.

Table 1-2. VCF file column specifications

Column	Name	Description
1	CHROM	The name of the sequence, usually an integer referencing a chromosome or a string such as "`chr1`," "`chrX`," "`MT`," etc.
2	POS	The 1-based position of the variation on the given sequence
3	ID	The identifier of the variation, often a dbSNP rs identifier (or a "." if unknown)

Column	Name	Description
4	REF	The reference base (or bases if describing an indel) at the given position
5	ALT	The alternative allele(s) at this position
6	QUAL	A quality score of the inference of the given alleles
7	FILTER	A flag indicating which of a given set of filters the variation has failed (or "PASS" if all the filters were passed)
8	INFO	A list of key-value pairs (attributes) describing the variation
9	FORMAT	A list of fields for describing the samples under which the VCF file is describing (optional)
10+	SAMPLEs	For each sample described in the file, the values of the fields listed in FORMAT

The VCF format is quite flexible, which can sometimes make it a pain to use if you have VCFs that were generated by different tools and contain lots of different attributes. Specifically, VCFs can describe one sample, multiple samples, one chromosome, multiple chromosomes, a whole genome, etc., and contain a wide range of INFO attributes, which are usually added in the tertiary analysis step.

Tertiary Analysis

The final step of the main bioinformatics workflow is tertiary analysis, which usually contains the processes to annotate the variants discovered in the previous step. It is important to understand not only how your sample may vary from a reference sequence but also what this variation may mean functionally. To answer this question, you can annotate the variants to provide information on the clinical or functional significance on the organism.

Some common annotation tools include `SnpEff`, `VEP`, and `ANNOVAR`. Each annotation tool adds useful information about the variant and may predict the functional effect of a given mutation. This gets appended to the INFO column of your VCF file for each line after the previous attributes from the variant calling process:

```
#CHROM POS     ID  REF  ALT  QUAL FILTER  INFO
chr1   123456  .   A    T    .    .       NS=3;DP=14;AF=0.0005;ANN=T|...
chr1   123457  .   C    G    .    .       AF=0.0005;ANN=C-chr1:123456_A>T|...
```

This annotation ("ANN") attribute may tell you the type of variant, the level of impact this variant has (LOW, MODERATE, HIGH, etc.), other ancillary information about the variant's position (the gene ID or transcript ID), and the respective cDNA nucleotide change and protein amino acid change. Here's an example of what an ANN attribute for a single VCF line might look like after annotation using `SnpEff`:

```
ANN=T|missense_variant|MODERATE|CCT8L2|ENSG00000198445|transcript|
ENST00000359963|protein_coding|1/1|c.1406G>A|p.Gly469Glu|1666/2034|
1406/1674|469/557||,T|downstream_gene_variant|MODIFIER|FABP5P11|
ENSG00000240122|transcript|ENST00000430910|processed_pseudogene||
n.*397G>A|||||3944|
```

Most of the time annotation tools are simply performing a database lookup of the mutation and position from each record in your VCF. This means that these tools will simply add attributes to the INFO field, which are usually delimited by a different character. Notice that, in the previous example, the annotations are pipe-delimited ("|").

Other Analyses

Outside of the traditional bioinformatics workflow, there are plenty of other tasks you may find yourself doing. Here are some other analyses that you may perform on the results of one of the previous steps or other data altogether:

Machine Learning
: Machine learning models are trained to predict various items. For example, you could predict drug sensitivity based on gene expression or disease status based on diagnostic test results. (Chapters 5 and 6 will cover the tools perfect for machine learning in Azure.)

Biostatistics
: Need to understand correlation (which is not equal to causation) or the statistical significance between the differences of groups? Biostatistical analyses are important in quantifying the confidence.

Protein Structure Modeling
: Outside of the normal nucleotide analysis, you can also look into amino acid sequences and the role proteins play in an organism.

Reporting and Visual Analytics
: Usually as the final step, you need to summarize your results and make them digestible for others in reports. Visualizing the outputs of your analyses helps us to convey your results in an understandable way.

Other File Formats

In addition to the formats described, you may see other file formats that help to run one of the analyses or serve as a different view of the same data.

GEN (and BGEN)

The GEN format is a file format developed by researchers at Oxford University that mainly holds the genotype information for a sample. This is similar to the VCF format in that it contains single nucleotide polymorphisms (SNPs), an ID for the variant, and the alleles seen in the sample. The binary form of this format, called BGEN, is useful due to its very small file size and fast readability by computers (as compared to human-readable formats like VCF).

GFF

The general feature format (GFF) is one of the most broadly purposed formats you may come across. This format was designed to house many types of features for nucleic acid and protein sequences. This can include features such as microRNAs, open reading frames (ORFs), binding domains, exons, and more. The latest version of the GFF is called GFF3, which looks like this:

```
##gff-version 3.1.26
##sequence-region ctg123 1 1497228
ctg123 . gene 1000 9000 . + . ID=gene00001;Name=EDEN
ctg123 . TF_binding_site 1000 1012 . + . ID=tfbs00001;Parent=gene00001
ctg123 . mRNA 1050 9000 . + . ID=mRNA00001;Parent=gene00001;Name=EDEN.1 ctg123 .
mRNA     1050 9000 . + . ID=mRNA00002;Parent=gene00001;Name=EDEN.2\
ctg123 . mRNA 1300 9000 . + . ID=mRNA00003;Parent=gene00001;Name=EDEN.3
```

The standard specification of columns for GFF3 is shown in Table 1-3.

Table 1-3. GFF3 column specifications

Column	Name	Description
1	Sequence ID	The identifier given to the feature's sequence. Could be an accession number or any unambiguous ID.
2	Source	Describes the algorithm or procedure that generated this feature (e.g., GenBank, Protein Homology, EMBL).
3	Feature Type	The type of feature being described (e.g., mRNA, domain, exon).
4	Feature start	The 1-based starting coordinates of the feature.
5	Feature end	The ending coordinates of the feature.
6	Score	A score for sequence similarity (E-values) or predictions (P-values).
7	Strand	The strand from which the feature was derived.
8	Phase	Indicates where the feature begins with reference to the reading frame, commonly used in CDS features.
9	Attributes	A list of key-value pairs, delimited by a semicolon, that provides additional information about each feature.

PDB

If you're a researcher who works in protein analysis, you'll likely come across Protein Data Bank (PDB) files. These files contain the three-dimensional coordinates of the residues in a protein structure.

Here are some sample lines from the SARS-CoV-2 Omicron variant spike structure PDB file:[7]

```
HEADER    VIRAL PROTEIN  19-DEC-21 7T9J TITLE    CRYO-EM STRUCTURE OF THE
    SARS-COV-2 OMICRON SPIKE PROTEIN
...
SOURCE    MOL_ID: 1;
SOURCE    2 ORGANISM_SCIENTIFIC: SEVERE ACUTE RESPIRATORY SYNDROME CORONAVIRUS
SOURCE    3 2;
SOURCE    4 ORGANISM_TAXID: 2697049;
SOURCE    5 GENE: S, 2;
SOURCE    6 EXPRESSION_SYSTEM: HOMO SAPIENS; SOURCE     7 EXPRESSION_SYS-
TEM_TAXID:
9606
KEYWDS    SARS-COV-2, GLYCOPROTEIN, FUSION PROTEIN, VIRAL PROTEIN EXPDTA
    ELECTRON MICROSCOPY
AUTHOR         X.ZHU,D.MANNAR,J.W.SAVILLE,S.S.SRIVASTAVA,A.M.BEREZUK,K.S.TUTTLE,
AUTHOR    2 S.SUBRAMANIAM
REVDAT    1    29-DEC-21 7T9J    0
JRNL      AUTH D.MANNAR,J.W.SAVILLE,X.ZHU,S.S.SRIVASTAVA,A.M.BEREZUK, JRNL
AUTH 2
K.S.TUTTLE,C.MARQUEZ,I.SEKIROV,S.SUBRAMANIAM
JRNL      TITL SARS-COV-2 OMICRON VARIANT: ACE2 BINDING, CRYO-EM STRUCTURE JRNL
    TITL 2 OF SPIKE PROTEIN-ACE2 COMPLEX AND ANTIBODY EVASION
JRNL      REF  TO BE PUBLISHED
JRNL      REFN
```

Note that there are many different line entry types that can be added in a PDB file. This includes the structure's title and author information, the journal article (if it's been published in a paper), and source organism metadata:

```
HEADER    VIRAL PROTEIN  19-DEC-21 7T9J TITLE    CRYO-EM STRUCTURE OF THE SARS-
COV-2
OMICRON SPIKE PROTEIN
...
SOURCE    MOL_ID: 1;
SOURCE    2 ORGANISM_SCIENTIFIC: SEVERE ACUTE RESPIRATORY SYNDROME CORONAVIRUS
SOURCE 3 2;
SOURCE    4 ORGANISM_TAXID: 2697049;
SOURCE    5 GENE: S, 2;
SOURCE    6 EXPRESSION_SYSTEM: HOMO SAPIENS; SOURCE     7 EXPRESSION_SYS-
TEM_TAXID:
9606
KEYWDS    SARS-COV-2, GLYCOPROTEIN, FUSION PROTEIN, VIRAL PROTEIN EXPDTA
    ELECTRON MICROSCOPY
AUTHOR         X.ZHU,D.MANNAR,J.W.SAVILLE,S.S.SRIVASTAVA,A.M.BEREZUK,K.S.TUTTLE,
AUTHOR    2 S.SUBRAMANIAM
REVDAT    1    29-DEC-21 7T9J    0
```

7 See "7T9J: Cryo-EM Structure of the SARS-CoV-2 Omicron Spike Protein," RCSB PDB, accessed September 15, 2022, *https://www.rcsb.org/structure/7T9J*.

```
JRNL      AUTH D.MANNAR,J.W.SAVILLE,X.ZHU,S.S.SRIVASTAVA,A.M.BEREZUK, JRNL
AUTH 2
K.S.TUTTLE,C.MARQUEZ,I.SEKIROV,S.SUBRAMANIAM
JRNL      TITL SARS-COV-2 OMICRON VARIANT: ACE2 BINDING, CRYO-EM STRUCTURE JRNL
    TITL 2 OF SPIKE PROTEIN-ACE2 COMPLEX AND ANTIBODY EVASION
JRNL      REF  TO BE PUBLISHED
JRNL      REFN
```

In addition, it's common for a PDB file to have a ton of REMARK lines, which usually describe the process of determining the structure, quality metrics, and really anything else that the author wants to share:

```
REMARK       2
REMARK       2 RESOLUTION.  2.79 ANGSTROMS.
...
REMARK       245   ELECTRON MICROSCOPE SAMPLE
REMARK       245     SAMPLE TYPE                    :  PARTICLE
REMARK       245     PARTICLE TYPE                  :  POINT
REMARK       245     NAME OF SAMPLE                 :  SARS-COV-2 OMICRON SPIKE
REMARK       245                                       PROTEIN
REMARK       245     SAMPLE CONCENTRATION (MG ML-1) :  NULL
REMARK       245     SAMPLE SUPPORT DETAILS         :  NULL
REMARK       245     SAMPLE VITRIFICATION DETAILS   :  NULL
REMARK       245     SAMPLE BUFFER                  :  NULL
REMARK       245     PH                             :  8.00
REMARK       245     SAMPLE DETAILS                 :  NULL
REMARK       245
...
REMARK       247   ELECTRON MICROSCOPY
REMARK       247     THE COORDINATES IN THIS ENTRY WERE GENERATED FROM ELECTRON
REMARK       247     MICROSCOPY DATA. PROTEIN DATA BANK CONVENTIONS REQUIRE
REMARK       247     THAT CRYST1 AND SCALE RECORDS BE INCLUDED, BUT THE VALUES
REMARK       247     ON THESE RECORDS ARE MEANINGLESS EXCEPT FOR THE CALCULATION
REMARK       247     OF THE STRUCTURE FACTORS.
...
```

PDB files also often contain a SEQRES section that lists the amino acid sequence of the structure. This is useful if you need the sequence for other bioinformatics analyses such as protein sequence alignment, etc.:

```
SEQRES 1 A 1285 MET PHE VAL PHE LEU VAL LEU LEU PRO LEU VAL SER SER
SEQRES 2 A 1285 GLN CYS VAL ASN LEU THR THR ARG THR GLN LEU PRO PRO
SEQRES 3 A 1285 ALA TYR THR ASN SER PHE THR ARG GLY VAL TYR TYR PRO
```

The most important section includes the lines that start with ATOM, which describe the position of the individual atoms of the amino acid residues:

```
ATOM  1   N    GLN  A  14  167.229  153.151  261.437  1.00179.65  N
ATOM  2   CA   GLN  A  14  166.857  151.813  260.994  1.00179.65  C
ATOM  3   C    GLN  A  14  167.456  151.510  259.625  1.00179.65  C
ATOM  4   O    GLN  A  14  167.347  152.316  258.703  1.00179.65  O
```

```
ATOM  5   CB   GLN  A  14  165.336  151.671  260.947  1.00179.65  C
ATOM  6   CG   GLN  A  14  164.672  151.610  262.310  1.00179.65  C
ATOM  7   CD   GLN  A  14  163.179  151.365  262.217  1.00179.65  C
ATOM  8   OE1  GLN  A  14  162.554  151.648  261.194  1.00179.65  O
ATOM  9   NE2  GLN  A  14  162.598  150.838  263.288  1.00179.65  N
ATOM  10  N    CYS  A  15  168.062  150.331  259.493  1.00181.02  N
ATOM  11  CA   CYS  A  15  168.663  149.888  258.244  1.00181.02  C
ATOM  12  C    CYS  A  15  168.213  148.468  257.935  1.00181.02  C
ATOM  13  O    CYS  A  15  168.011  147.653  258.839  1.00181.02  O
ATOM  14  CB   CYS  A  15  170.199  149.930  258.304  1.00181.02  C
ATOM  15  SG   CYS  A  15  170.926  151.538  258.689  1.00181.02  S
...
```

The last section I'll mention includes the positions of atoms that are not part of the protein molecule. These lines are denoted by HETATM. In the next example, this is describing a small molecule ligand called 2-acetamido-2-deoxy-beta-D-glucopyranose (or NAG for short):

```
HETATM21152 C1 NAG D    1    173.940 145.791 256.091 1.00189.94       C
HETATM21153 C2 NAG D    1    174.055 145.911 257.627 1.00189.94       C
HETATM21154 C3 NAG D    1    175.262 146.769 258.009 1.00189.94       C
HETATM21155 C4 NAG D    1    176.533 146.234 257.359 1.00189.94       C
HETATM21156 C5 NAG D    1    176.341 146.159 255.845 1.00189.94       C
HETATM21157 C6 NAG D    1    177.522 145.560 255.117 1.00189.94       C
```

Both the ATOM and HETATM sections have a standard columnar format that includes the columns described in Table 1-4.

Table 1-4. PDB ATOM and HETATM column specification

Column	Positions	Description
1	1–6	"ATOM" for this section
2	7–11	Atom number
3	13–16	Atom name
4	17	Alternate location indicator
5	18–20	Residue name
6	22	Chain identifier
7	23–26	Residue sequence number
8	27	Code for insertion of residues
9	31–38	Orthogonal coordinates for X in Angstroms
10	39–46	Orthogonal coordinates for Y in Angstroms
11	47–54	Orthogonal coordinates for Z in Angstroms
12	55–60	Occupancy percentage
13	61–66	Temperature ("B") factor (Default = 0.0)

Column	Positions	Description
14	73–76	Segment identifier
15	77–78	Element symbol
16	79–80	Charge on the atom

To render a PDB file as a 3D image, researchers often use desktop tools such as `PyMol` or web-based tools like `3Dmol.js` and `JSmol`. For the previous file, we will generate something like Figure 1-19.

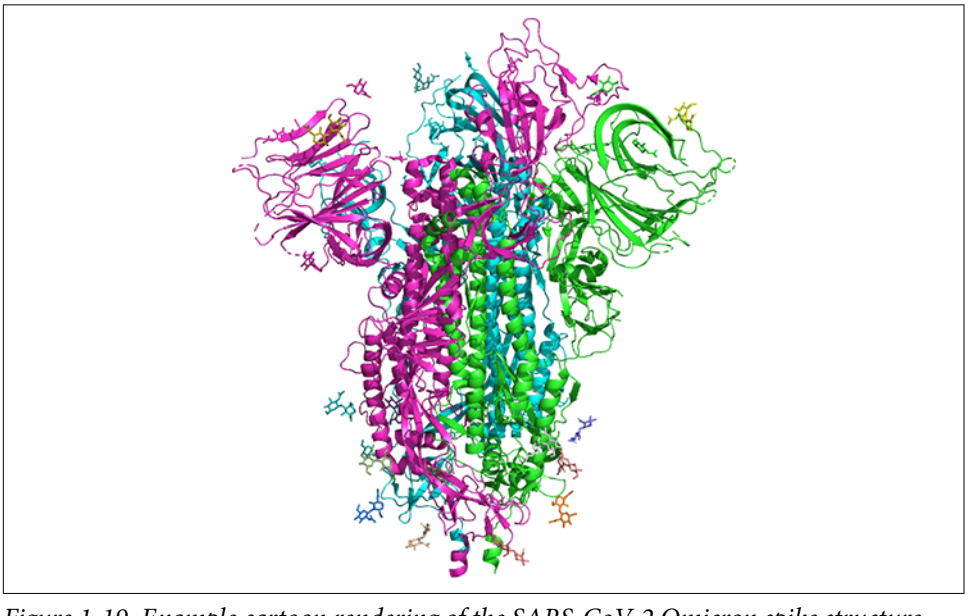


Figure 1-19. Example cartoon rendering of the SARS-CoV-2 Omicron spike structure (PDB: 7T9J)

While there are many other file formats I could cover in this section, the ones mentioned are the most common. So I won't drone on about file formats any longer, though I hope you will take the time to get familiar with the ins and outs of formats that you often use. It'll help you when you need to come up with logic to process that data with various tools.

Now that you have your own Azure environment ready to go and you're all up to speed on bioinformatics terminology, the next chapter will cover how to start setting up a data lake to house your genomics data. Care to take a dive in the lake?

In summary, the Microsoft Azure cloud, similar to other common enterprise cloud providers (Amazon Web Services and Google Cloud Platform), provides a variety of different types of services including IaaSs like virtual machines; networking and platform services (PaaS), such as Azure Machine Learning and Synapse Analytics; and

software services (SaaS), such as hosted applications or APIs. The top level of cloud organization is the tenant, under which you'll have one or more subscriptions. These subscriptions are what get billed when you use Azure services. In a given subscription, you may have one or more resource groups, which organize the services that you provision. Resource groups are often created for a given use case or a group of users.

There are lots of considerations for moving to the cloud. The cloud isn't *always* cheaper or faster, but you don't have to migrate everything to the cloud at once. One of the biggest pros of using the cloud is that it reduces the need to manage the underlying infrastructure as much, including things like security and networks. While Azure is used across the globe in finance, healthcare, retail, government, etc., the use of its enterprise-grade services has long been overlooked for genomics. In this book, we're going to change that.

CHAPTER 2

Organizing Genomics Data with Data Lakes

The concept of a data lake may seem quite foreign at first. In the business world, you may be used to storing data in structured locations like a database. In the research world (or in academia), you may have put your data on a network storage device or an FTP server. And in your personal life, you've probably used cloud storage services like Dropbox, Google Drive, or Microsoft OneDrive. There are concepts from each of these types of data storage that will make data lakes seem more familiar in time.

You can think of a data lake as a general location to store your data. It is basically organized in directories like the files on your hard drive or in your personal cloud storage. It supports structured data, like CSV files, and unstructured data, like images. Plus, using a data lake in Azure allows you to grab your data from any of the other services that you want to use. Finally, you can share or secure different parts of your lake so that not everyone can swim wherever they please.

Here are some benefits of data lakes:

- Data is kept at its various stages and forms and stored in various "zones," organized by the level of maturity, cleanliness, and utility.
- Data lakes are more flexible than databases or data warehouses because they can store virtually *any* type of data: structured, semistructured, or unstructured—the data lake can handle it.
- Users can browse a data lake in much the same way as they'd browse files on their computer.

A big plus of using data lakes in Azure is that they can offer "limitless" scale. So whether you're storing a few files or petabytes of genomes, your performance stays the same. Plus, with many Azure services having the ability to directly connect to your data lake, this will drastically reduce redundant copies of the data being spread

all around. While data retention policies can be defined to delete old/unused data, it's common practice for a data lake to continually keep multiple versions of data. Thus, the data lake serves as the central location for all types of data and all of the versions of an individual data asset. Further, data lakes are proving to be extremely beneficial with big data analytics when coupled with other analytical tools such as Azure Synapse, which you'll hear all about in the next chapter.

Data lakes are designed to be flexible, so you get to formulate your own organization strategy. I do, however, have some general thoughts around storing genomics data in Azure.

In this chapter, we'll cover how to set up and use data lakes for genomics use cases. This will help set you up for success ahead of time such that you don't drown in a messy pond. Rather, you'll be able to swim freely.

Organizing Your Genomics Data

Data lakes are perfect for genomics data because I feel that the organization falls into place naturally based on the normal maturation of data through a bioinformatics workflow. In addition, many tools allow for us to mount an Azure storage account as part of the filesystem, making it simple for us to grab data and analyze it with ease.

Data lakes are organized by containers, which are similar to collections of directories on a filesystem. You can have unlimited containers under which you can have unlimited directories of files. Usually, you'll have one container for the actual data lake data along with one or more other containers that house other, non–data lake data. You'll see other containers being used in Chapter 3, where we'll create another container to store your data warehouse metadata.

One key capability of data lakes is their flexibility. If your organization chooses to have a central data lake for the entire organization, different parts of the data lake can be organized to fit individual needs. In this book, I'll focus on the organization of data for genomics, but the same principles apply to finance data, HR data, or other data your organization may have.

Going for Bronze, Silver, and Gold

If you've read any articles about data lakes before reading this book, you may have come across the concept of bronze, silver, and gold zones. This means zones for data in its raw, staging (intermediate), and curated forms, respectively.

This zone-based approach is a conceptual way to think about your data such that it is organized by its level of cleanliness, maturity, and utility.

For genomics, these thoughts around data maturity and its utility to your overall work fit perfectly with the various types of data that you see (and, subsequently, matches to the flow of data through bioinformatics pipelines). See Figure 2-1.

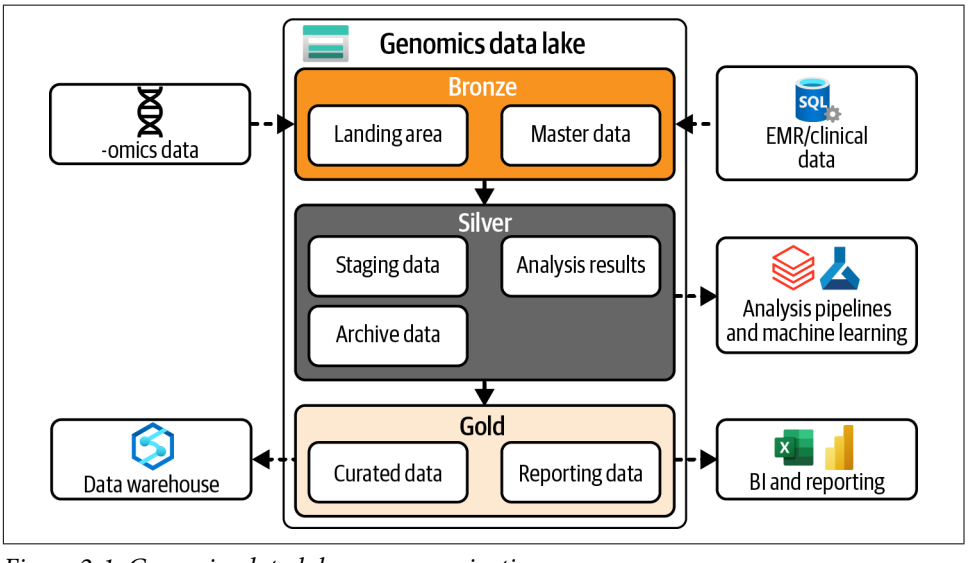


Figure 2-1. Genomics data lake zone organization

Bronze (raw)

The first zone of your data lake should contain data in its rawest form. This usually includes data sourced from other systems that are placed in a "landing" area. In addition, you'll also often see the bronze zone used as a landing area for "mastered" data. Examples include patient data from an electronic medical record (EMR) system, a list of clinical trial participants, or reference genomes. You use this zone to house data that isn't quite ready for use in reporting or machine learning or bioinformatics analyses. This isn't because the raw data isn't useful. Rather, this data serves as the input for other processes to further cleanse and prepare the data for use.

It's common to use this zone to house raw sequence data. As you can see, it's not that this sequence data isn't important but that it requires further processing to be useful.

A key reason for this zone to exist is because it allows for us to change out processing pipelines and rerun analyses from this source data. In bioinformatics, you may choose to rerun some sequences through an updated pipeline (for example, with a new reference genome or a new variant caller). Keeping the raw form of your data allows us to dynamically process the same data in new ways as you grow and evolve.

Silver (staging/intermediate)

Often, you'll see this middle zone referred to as *staging* (a term that I don't love, as I feel it limits the potential of this zone). This refers to the original intention of this zone that falls into the extract-transform-load (ETL) paradigm of data movement, where data is "staged" as it's being transformed into its final form. The silver zone will house data that is ready for use in analysis. Thus, it contains data that has been cleansed (as opposed to its raw form) but not really further modified or aggregated.

Despite the name, this isn't just a transient zone for data as you're manipulating it. I like to think of this zone as the place for your data once it's in a usable form. In practice, this is the largest zone that will often serve as the source for your data if you're a bioinformatician or data scientist. This is the home for most of your training data for machine learning or the results from your bioinformatics pipelines (such as variant data).

Gold (curated)

The last zone houses your data in its most refined and usable form. This data may include cleansed/final analysis results, aggregated findings, or data that has been joined together (or enriched) with other data. Usually, the criteria for something being considered "curated" should be stringent. Thus, the gold zone is often much smaller than silver as it only contains data that is at its most cleansed, curated, and reliable form.

Many times, you use the gold zone as the place to serve up data to other systems. For example, you might keep data in the gold zone that is ready for use in a Power BI dashboard or data that is ready to be directly loaded into data warehouse tables.

Letting Your Bioinformatics Workflow Dictate Your Data Lake Organization

Every organization is different, so none of these organization examples are meant to be "one size fits all." My biggest suggestion for organizing a data lake for genomics is to let your standard bioinformatics workflow guide the organization strategy. For example, take a look at Figure 2-2.

In this workflow, you can see that your raw data comes in the form of sequence files (like FASTQs), which you store in the bronze zone. You then commingle this input data with reference data (genomes) through your secondary and tertiary bioinformatics pipelines. This results in cleaner files, like VCFs, that are now more usable downstream. You place the VCFs and other distilled data in your silver zone. You can use this zone as the data repository for other analyses like machine learning or maybe some biostatistics tasks. You can even pull data from here for analyses like

genome-wide association studies (GWAS), which require that you have both genotype data as well as input phenotype information for each sample.

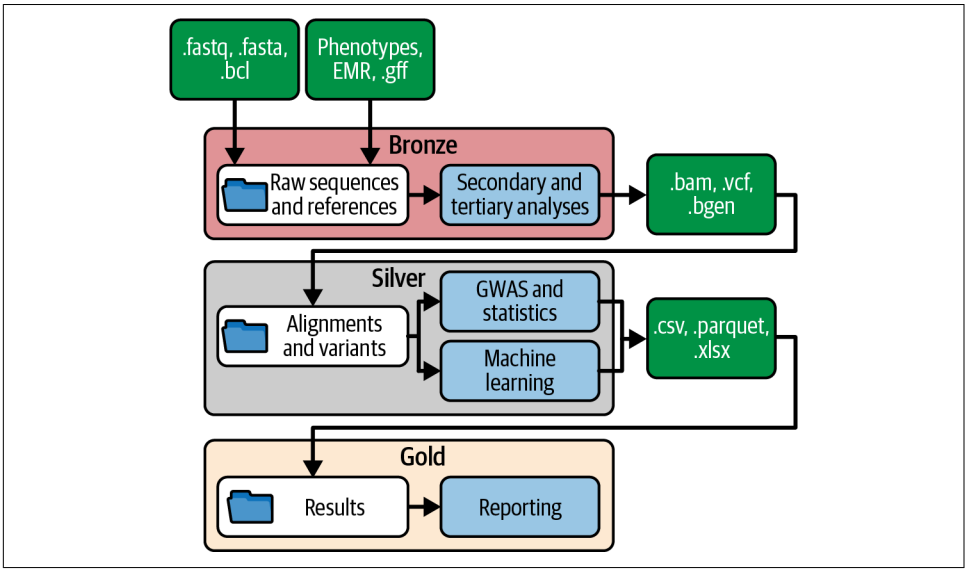


Figure 2-2. Example bioinformatics workflow, showing where data is housed at each stage

These analyses then result in the most curated output, such as insights into important genes, statistics, or aggregated results of the samples in your study. This data is housed in the gold zone, which serves as the data location to feed into your data warehouse or for reporting tools like Power BI or Tableau.

As you can see, the organization of this example data lake is dictated by the set of tasks that take data from raw sequences to distilled insights. Your bioinformatics workflow may be different, which is expected. Think of your data in three levels of maturity: raw, cleansed, and curated. This will help you put data in the appropriate place in your data lake.

Organizing your data lake will always be a work in progress, but starting on the right foot with a plan in mind will help you manage your data as your data lake grows. Plus, organization is key for managing costs and security going forward.

The aforementioned "zones" can be more theoretical concepts rather than physical separations of data. This zoning will help you to categorize data by level of maturity, cleanliness, and utility. So keep that in mind as I walk through them. I'll cover physical organization next.

Planning for -omics and Non-omics Data Together

At your organization, you may use various types of non-omics data to do your work. For example, if you work for a pharmaceutical company, you may have clinical trial data, phenotype information, or maybe some lab results.

These types of data also need to get housed in the data lake with the rest of your genomics data. As shown in Figure 2-1, data from other source systems such as an electronic medical record system (like Epic or Cerner) can land in the bronze zone. You'll then have some data workflows that take this data from its landing location through the staging area in the silver zone and on into the curated data areas in the gold zone. In other reference architectures, you may see directories named "Bronze," "Raw," etc. The concept of delineating your data is more of a guide than a rule, so feel free to stray from this concept to make your data lake work for you.

Study, subject, and sample directories

It's common practice to organize data lakes in a hierarchy based on overarching projects. For example, if you work for a biotech company that has clinical trials, your data like might have a directory structure similar to Figure 2-3.

This structure is flexible enough to house nongenomics data and genomics data in the same place, all related to samples collected in a study.

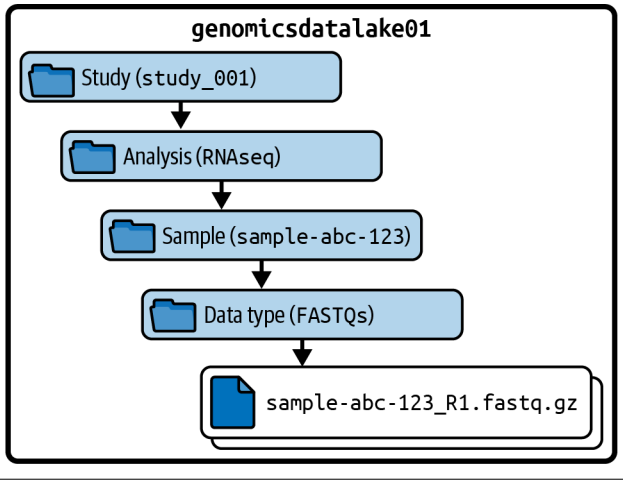


Figure 2-3. Example directory structure for organizing by study

In this example, your directory structure would be organized in such a way that makes logical sense but also allows you to get to your data efficiently. For example, if I wanted to get all RNA-seq FASTQs in `study_001`, the directory structure repeats by sample. This will allow you use wildcards like so:

```
study_001/RNAseq/*/FASTQs/*.fastq.gz
```

This will grab all the files of interest very easily. We'll cover this more in "Creating Your Data Factory" on page 114.

As with the overall data lake zone organization, your directory structure will likely look different than this basic example. I recommend thinking about how you logically organize the workloads at your organization as this will help you plan for the right directory hierarchy.

If you work for a pharma company that segregates workloads by drug product, your directories should reflect that. If you work at a plant research organization that splits trials by species, that might be the highest level of organization in the directories of your data lake. It's certainly not a bad idea to find a whiteboard and map out the types of data your organization uses and generates. This will help you plan accordingly and avoid future data lake reorganizations.

Creating a Data Lake with Azure Storage

Following the same steps as in "Creating Resources" on page 18, let's create a data lake in Azure Storage. From the Azure Portal, click "Create a resource" and search for "storage." Find the "Storage account option" and click "Create," as shown in Figure 2-4.

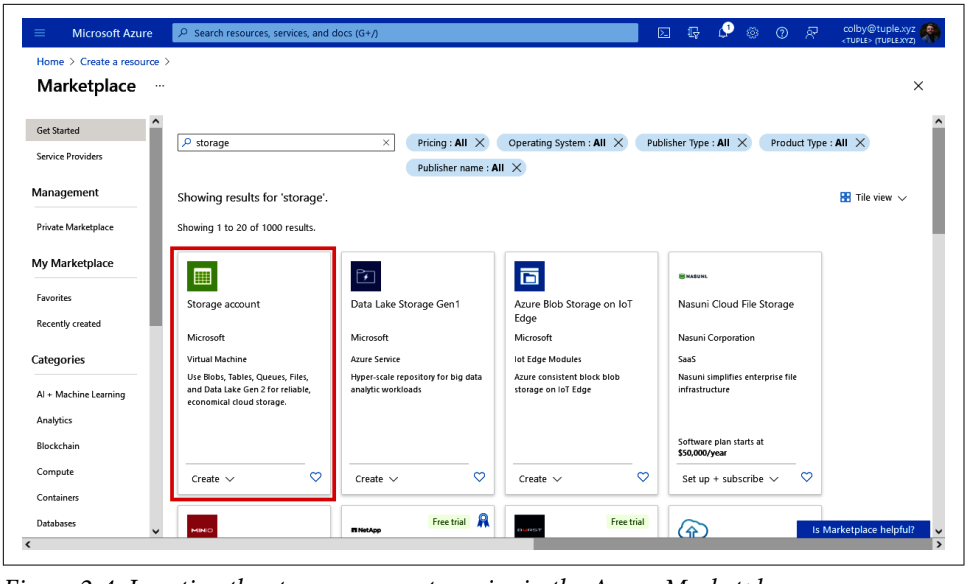


Figure 2-4. Locating the storage account service in the Azure Marketplace

Next, give your data lake a name like "genomicsdatalake01."

These names are globally unique, so you might have to change the numbers at the end if someone else on the planet has given their data lake the same name.

You can choose the premium performance tier[1] if you like. I wouldn't recommend it for now, but your enterprise data lake may opt for the "page blobs" premium performance option if consistently low latency is crucial.

There are also some choices about redundancy. For now, you can leave it as the default "locally redundant storage (LRS)." For business-critical data lakes, better redundancy can provide protection against data loss in the event of a disaster. See Figure 2-5.

1 To read more about premium storage, see the Microsoft documentation.

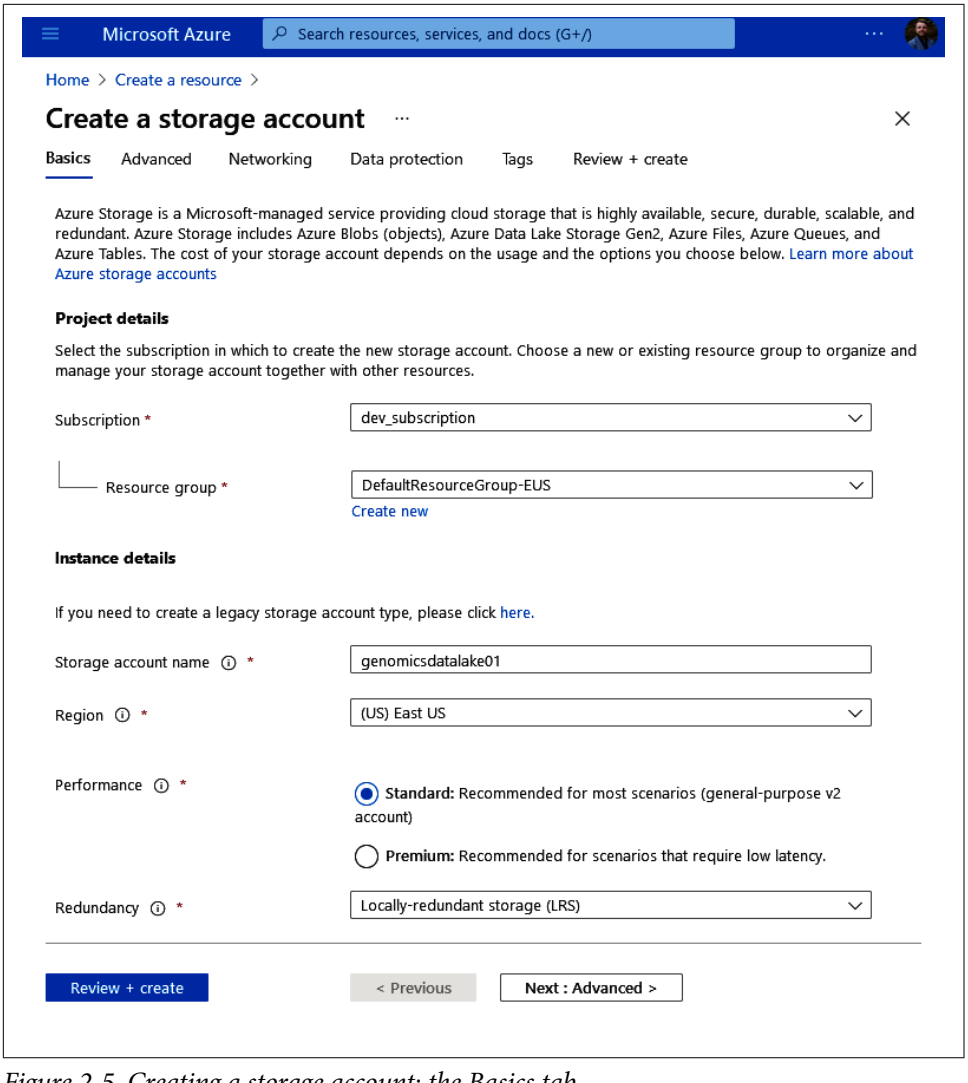


Figure 2-5. Creating a storage account: the Basics tab

Figure 2-6 shows the magic step that changes your storage account from a Blob storage to a data lake. On the "Advanced" tab, look under "Data Lake Storage Gen2" and simply click the checkbox beside "Enable hierarchical namespace." Voilà!

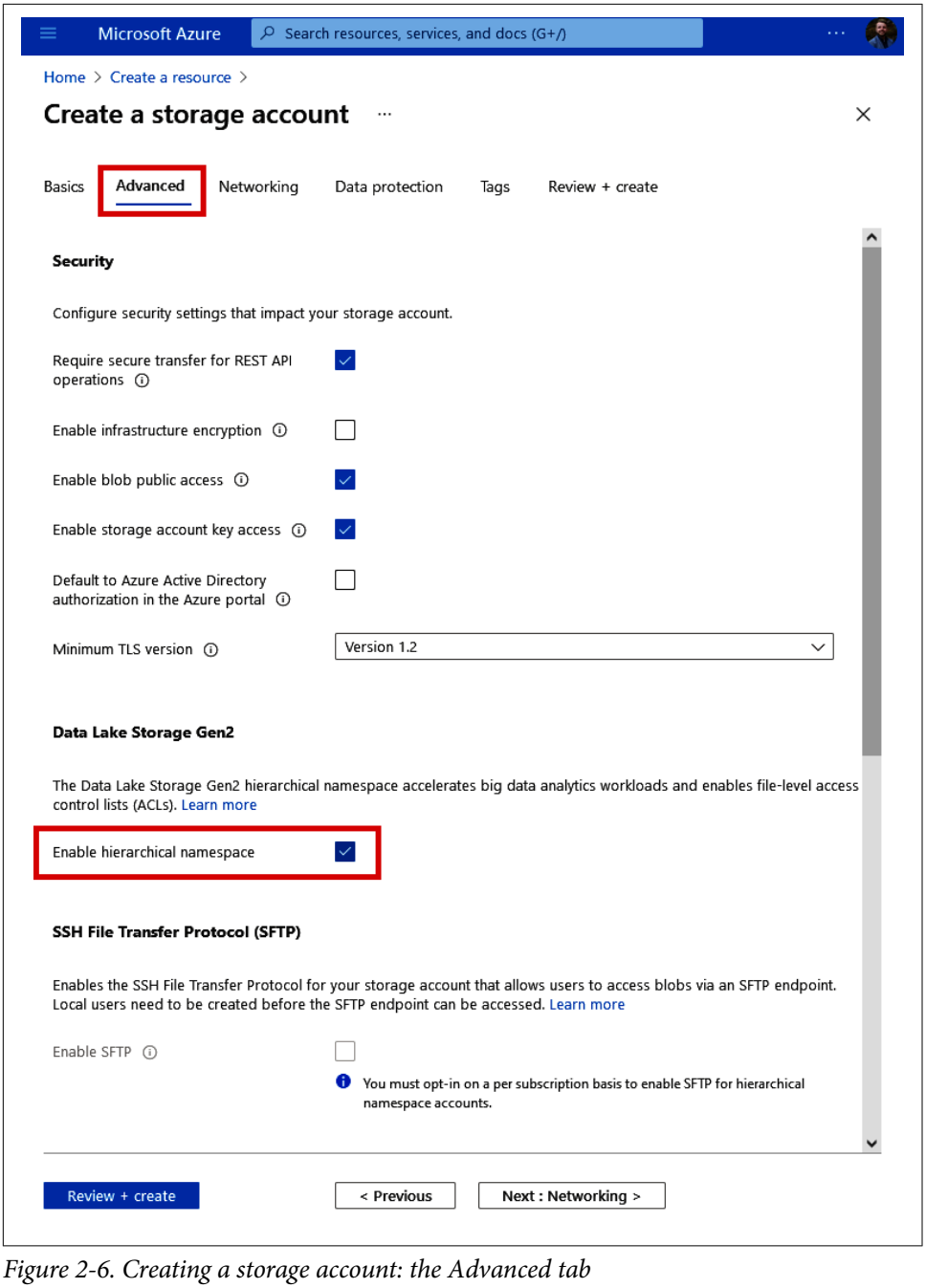


Figure 2-6. Creating a storage account: the Advanced tab

Finally, navigate to the "Review + create" tab and, after Azure validates that your selections are OK, click the "Create" button. See Figure 2-7.

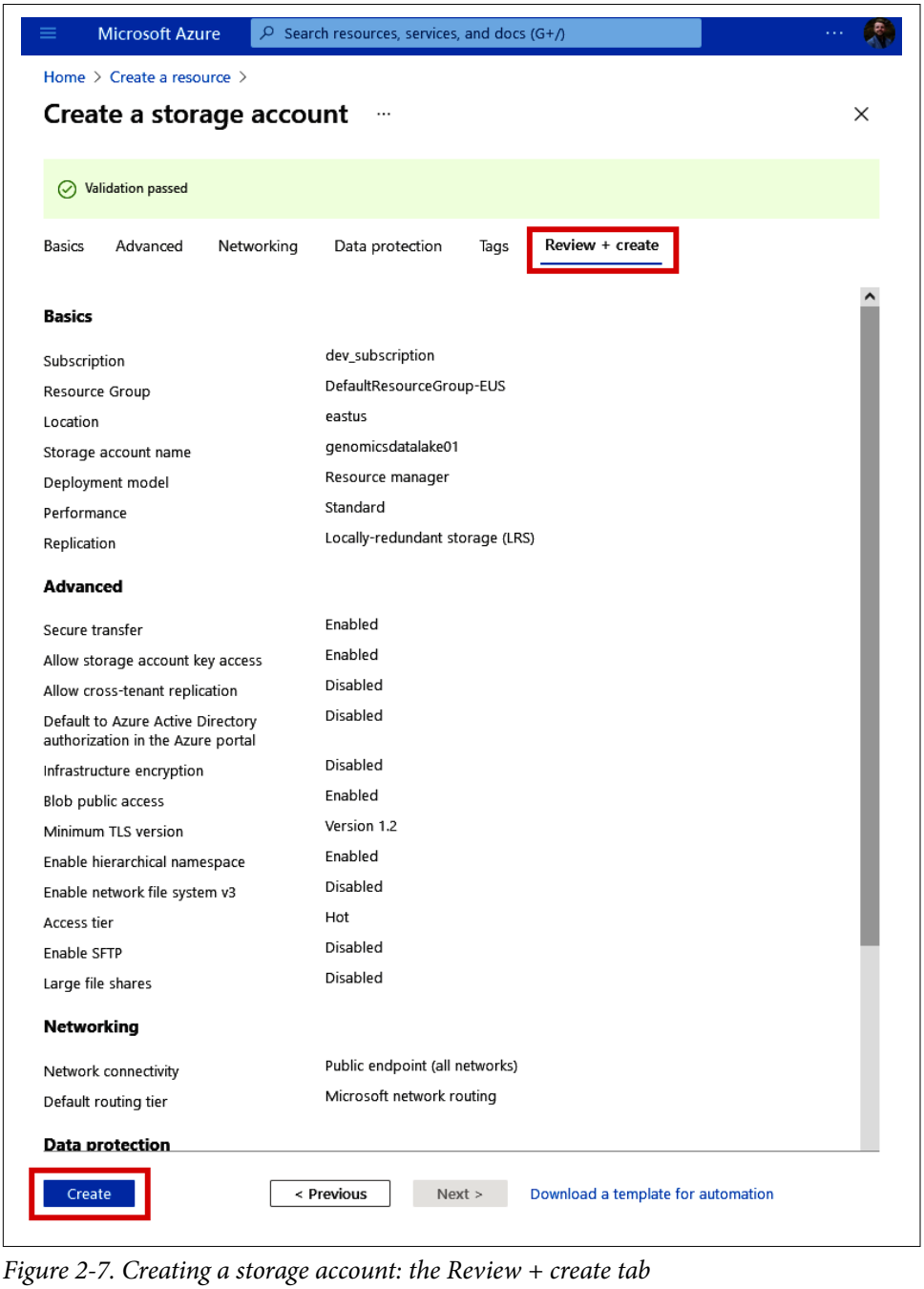


Figure 2-7. Creating a storage account: the Review + create tab

While your data lake is being created, let's talk about how the features differ in the data lake version of an Azure storage account.

Blob Storage Versus Data Lake Storage

Azure Storage is the main storage service for modern data architectures. This one platform contains multiple services to store data for a variety of purposes:

Blobs
: The main object store that provides scalable storage for nonstructured, semi-structured, and structured data (like images, logs, or binary data). (This is the most common storage we'll use in the cloud.)

Files
: File shares using the Server Message Block (SMB) protocol or Network File System (NFS) protocol. Perfect for replacing cloud or on-premises file server systems.

Queue
: A messaging store used for creating a queue of tasks or messages in JSON.

Tables
: NoSQL-based store for schemaless storage of structured data.

Disks
: Cloud-based storage volumes (usually used for virtual machines).

Within the blob service, you can enable the hierarchical namespace, which turns on the data lake storage features for enterprise big data analytics. Data lake storage is built on top of Azure Blob storage but with some added capabilities.

Data lake storage offers streamlined performance with Hadoop-compatible access to data. You can manage and access data in the same way you would with a Hadoop Distributed File System (HDFS). This includes the dedicated ABFS driver, which can be used to access data with Apache Hadoop-based environments such as Azure HDInsight, Azure Databricks, and Azure Synapse Analytics. In short, this means that many tools will be able to connect to the data lake without much fuss, making it easy to retrieve your data from the data lake, analyze it, and write your results back out to the data lake in a different place.

About the hierarchical namespace

The hierarchical namespace of data lake storage improves upon the flat namespace of Blob storage and provides better performance. This allows for better management of directories. While Blob storage allows you to organize files in directories, the directories are only virtual and don't actually exist. In contrast, the hierarchical namespace provides real, atomic organization of data.

The main benefit of data lakes over blob storage for larger organizations is that the hierarchical namespace allows for fine-grained security on individual directories or files. These are implemented using POSIX permissions. This means that you can define who can see which files and directories using Azure Active Directory. The hierarchical namespace is very important if your organization is subject to regulatory standards (HITRUST, NIST 800-171, etc.) since this granular control over data allows you to define, manage, and report on access much easier, enabling you to get to compliance with these standards faster.

Also, given that the cost difference between the flat and hierarchical namespaces is negligible, I would explicitly recommend everyone use the hierarchical namespace.

Speaking of costs, let's look at the different cost considerations of Azure Storage and how they affect performance.

Balancing Costs Versus Performance in Data Storage

Given that most of us are used to having our data readily accessible any time we want it, the idea of having to trade performance for costs may seem a bit foreign. You may be used to backing up data to an external hard drive to free up some space, but you don't really think about the "costs" of access data whenever you want. But in Azure Storage, there are multiple tiers from which you can select to meet various performance needs.

The Goldilocks Method of Storage Tiers

The two main considerations for picking a storage tier are (1) How frequently do you need to access the files? and (2) How long do you plan on holding on to the file?

You'll need to think about how individual types of data are used in your organization, by whom (or which systems), and how often.

Premium
: Premium isn't really a tier but a different type of storage altogether. This provides access to block blobs with consistently low latency, which is perfect for high-frequency data transactions. This carries the highest price tag.

Hot
: Hot is pretty much the default tier. It provides the best performance for retrieving files quickly and often but carries the highest price tag. This is used for files that you plan on consistently accessing, reading, or updating.

Cool

This tier is similar to Hot, though I think it goes underused. This tier is better for important files that you might need to access but only every once in a while. It carries the same performance as Hot but at a lower storage cost (and a higher data access cost). This tier is recommended for files that you may only access every month or so.

Archive

You can think of this tier as the backup version of the other two. This tier has very low storage costs, but it'll take a few hours to access the data when you want it. Archive is recommended for files you access every six months or so.

Cost breakdown

Storage and transaction costs are shown in Table 2-1. As you can see, as you move down in tier "coolness," your storage costs drop, but your transactional costs increase.

Table 2-1. Example costs for data operations in Azure Storage by tier. (This assumes a hierarchical namespace in the East US region with LRS.) Retail pricing from December 2021.

Tier	Storage cost per GB	Cost for 10k write operations	Cost for 10k read operations	Data retrieval per GB
Premium	$0.15	$0.0228	$0.0019	$0.00
Hot	$0.02	$0.065	$0.005	$0.00
Cool	$0.015	$0.13	$0.013	$0.01
Archive	$0.00099	$0.13	$6.50	$0.02

Genomics Data Lifecycle

Let's say you have a standard variant-calling pipeline that takes in pairs of FASTQ files, aligns them, outputs a BAM file of the alignment, and then variant-calls the BAM file to generate a VCF file. Each of these file types is important to the overall analysis, but some are important only at a certain stage and then rarely used again. See Figure 2-8.

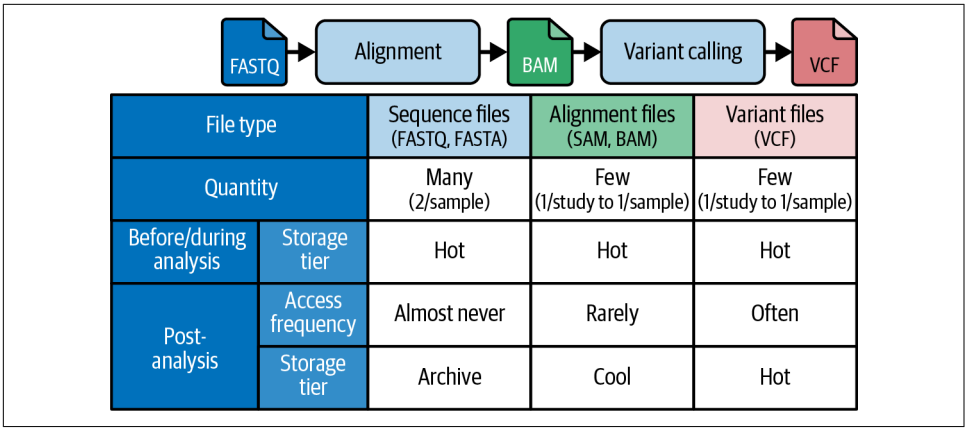


Figure 2-8. Example tier changes based on genomics data type and analysis usage

Each of these types of files may start off in the Hot tier, which is perfect for when you need the data to be readily accessible for analysis.

FASTQ files are often used only at the beginning of the bioinformatics pipeline, and then they aren't touched again unless someone needs to rerun the analysis, for example, against a different reference genome. The same goes for BAM alignment files. You may rereference them only if you need to look at alleles in a population or if you want to run a new type of variant analysis. This means that you can move these files down to Archive or Cool tiers, respectively, to save on costs since you won't be needing them often.

As for VCF files, you may want to continually refer to them for reporting or other downstream tasks. Thus, they can stay in the Hot tier most of the time.

When you go to upload data into your data lake, you can select what tier you'd like the file(s) to be uploaded to. By default, they're uploaded to the Hot tier. For example, let's say I want to upload a FASTA file of the SARS-CoV-2 reference sequence. If I know I'm going to be using this frequently over the next few weeks, I can leave the tier selection as Hot, as shown in Figure 2-9.

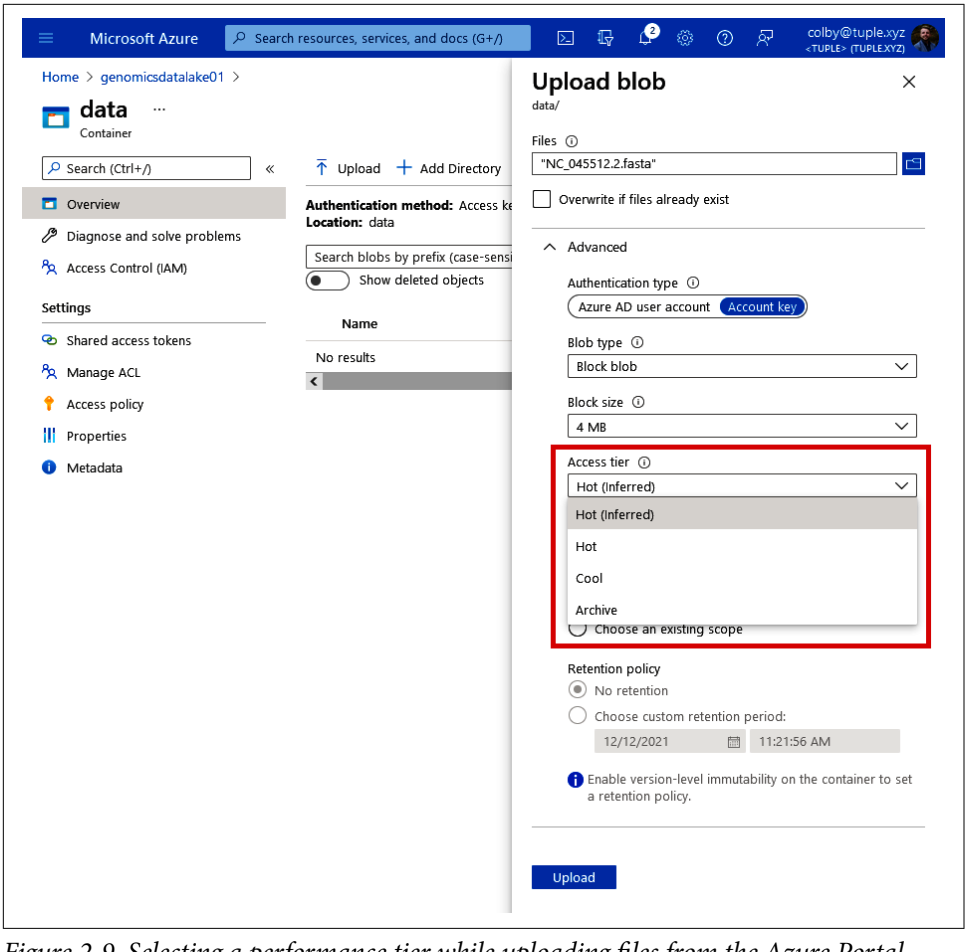


Figure 2-9. Selecting a performance tier while uploading files from the Azure Portal

Not sure which to choose? You can always change this later. Simply find any file and click it; then click "Change tier." See Figure 2-10.

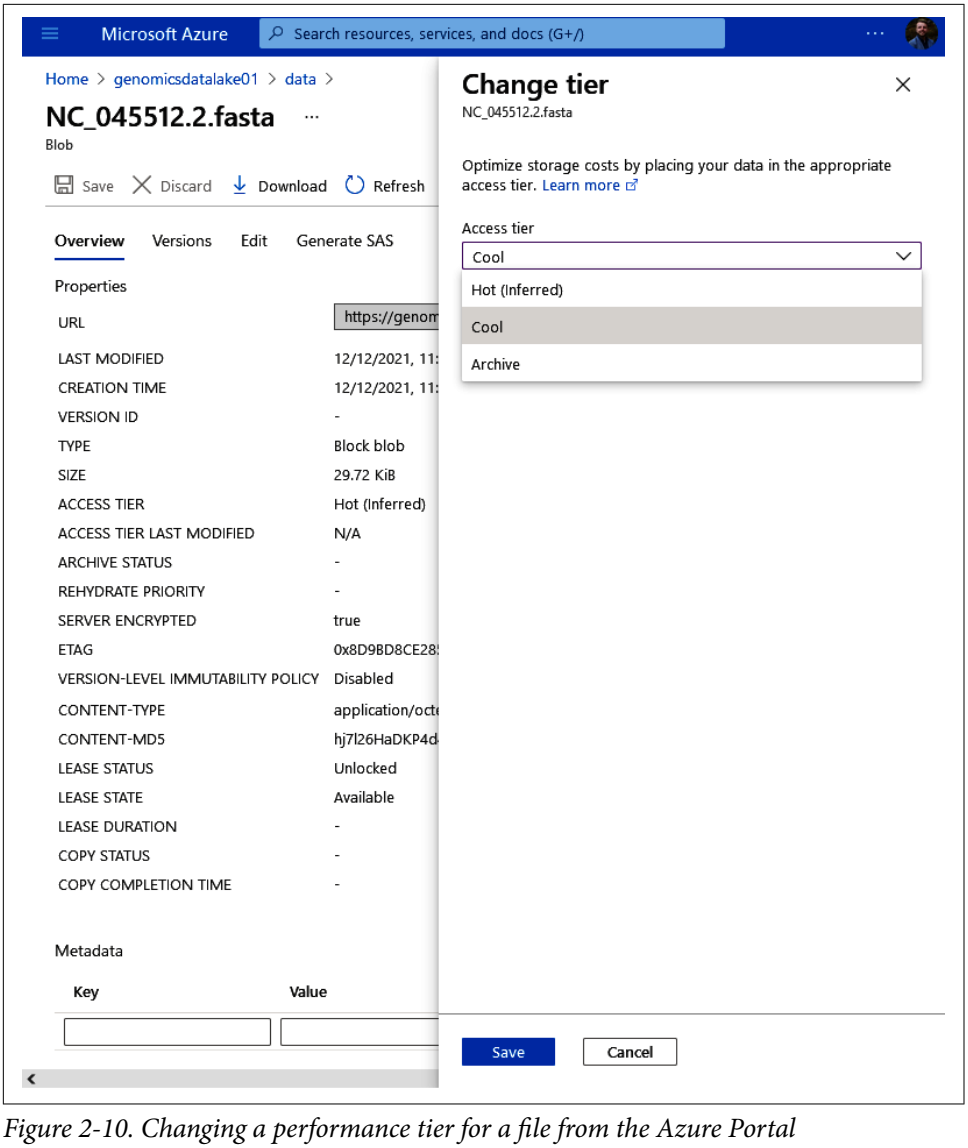


Figure 2-10. Changing a performance tier for a file from the Azure Portal

Using Azure Storage Explorer

If you're familiar with using File Explorer on your Windows machine (or Finder on a Mac), let me introduce you to a simple tool that makes interacting with your data lake feel like File Explorer: Azure Storage Explorer.

While Azure Storage Explorer isn't required for exploring a data lake, it's a commonly used (free) tool that allows you to traverse the directory storage of your data lake without your browser, similar to how you'd traverse the folders on the hard drive of your computer. Plus, I think it is a useful tool for managing data lakes when you have bulk actions you need to perform.

In the Azure Portal, I can specify performance tiers for individual files, but in Azure Storage Explorer, I can change the tier of entire directories. For example, I have a container in my data lake with data from 1000 Genomes,[2] as shown in Figure 2-11. I have a folder of VCF files that I have converted to Parquet format for use in a data warehouse. So I no longer really need my VCF files in the Hot tier in the data lake.

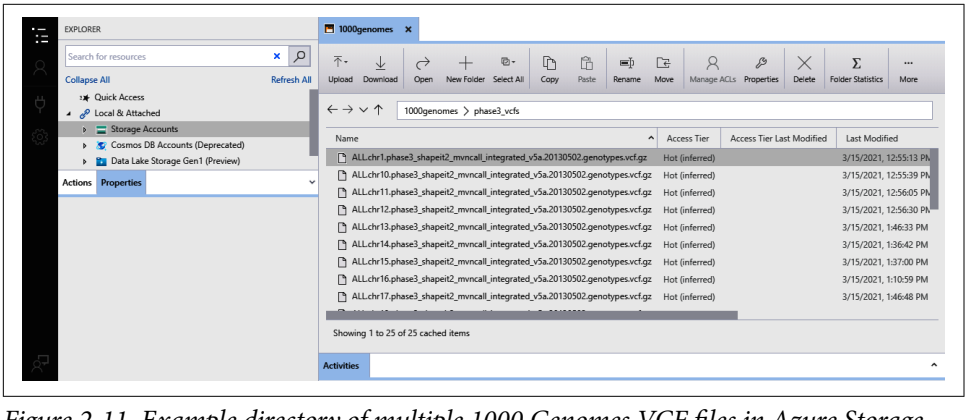


Figure 2-11. Example directory of multiple 1000 Genomes VCF files in Azure Storage Explorer

2 The 1000 Genomes Project set out to sequence 1,000 human genomes between 2008 and 2015. It is now managed by the at International Genome Sample Resource (IGSR).

You can simply right-click the entire directory and click "Change Access Tier…," which will open a window for you to choose the new performance tier for the directory. See Figure 2-12.

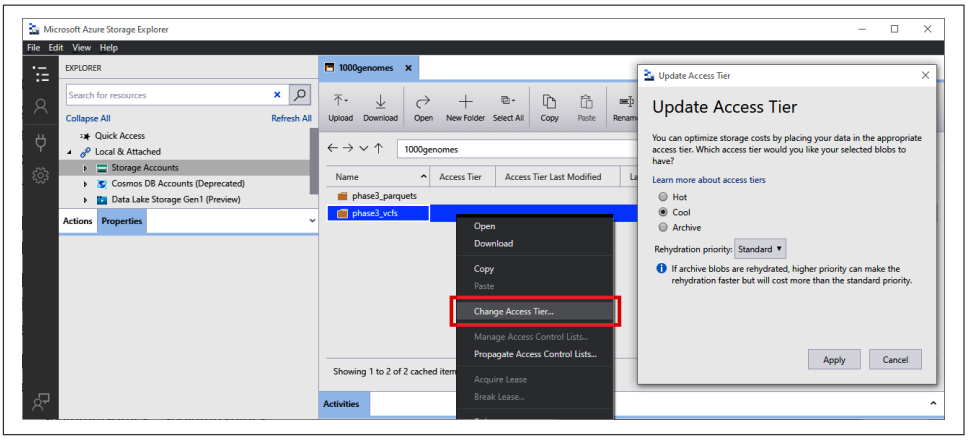


Figure 2-12. Changing a performance tier for a directory from Azure Storage Explorer

Lifecycle rules

If this seems like an overly manual process, you can also specify lifecycle rules to automatically move certain files to a different tier after a specified amount of time. Click the "Lifecycle management" item in the side menu (under "Data management"). This will open a blade that will show all the rules that have been defined for your storage account.

To create a new rule, click "Add a rule."

As an example, let's say you want to set a rule that moves FASTQ files from the Hot tier to the Archive tier after two weeks. First, name your rule "FASTQsToArchive." Select the radio button to "Limit blobs with filters" (as you want to limit this rule to a certain set of directories). You can leave the blob type and blob subtype options as defaults. See Figure 2-13.

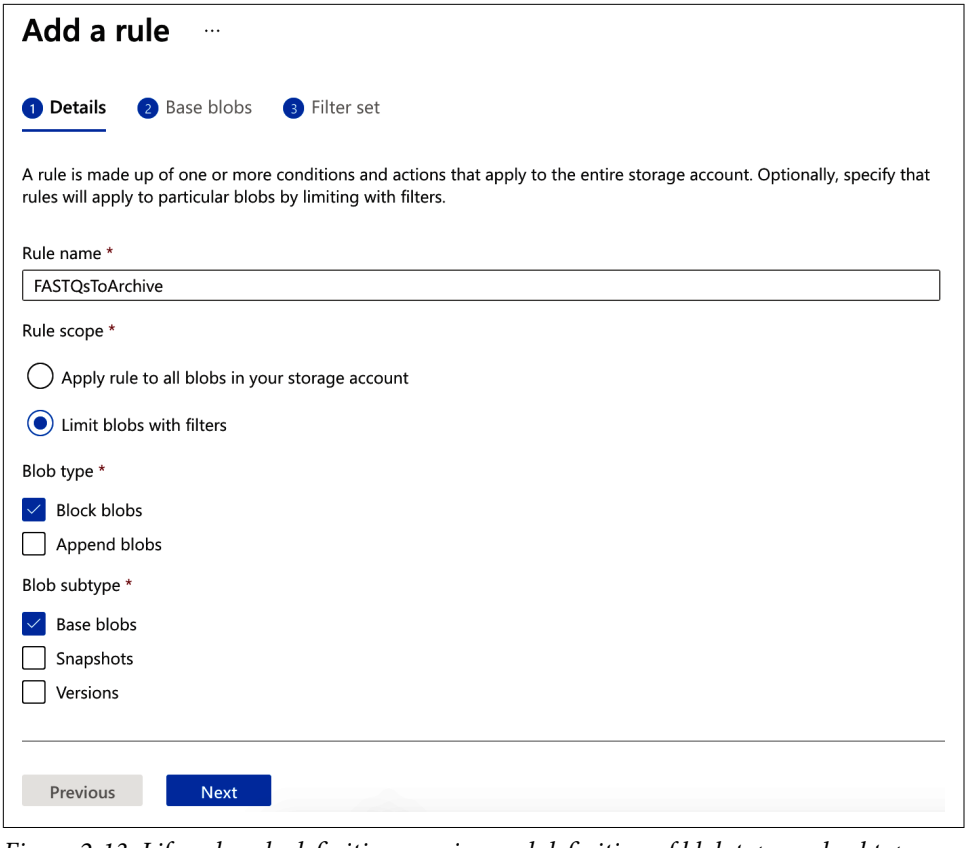


Figure 2-13. Lifecycle rule definition naming and definition of blob type and subtype

Next, specify the duration that passes before the rule is invoked. In this example, 14 days have passed since the blobs were last modified. Also, select the action, which is to "Move to archive storage." See Figure 2-14.

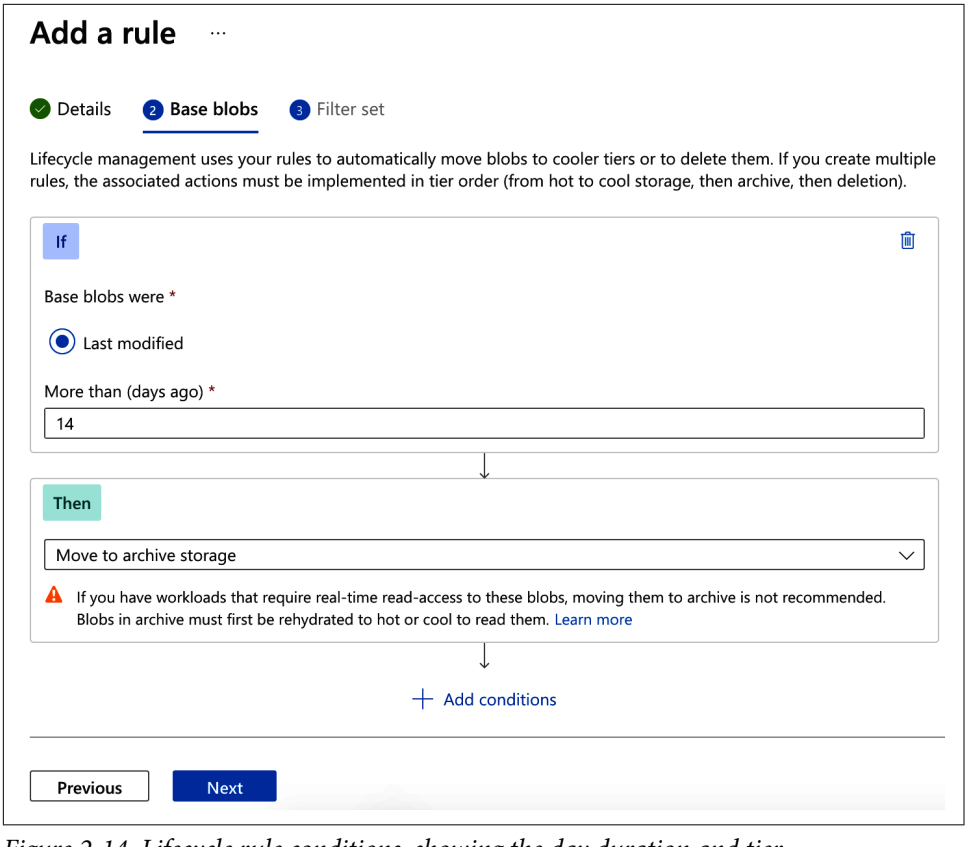


Figure 2-14. Lifecycle rule conditions, showing the day duration and tier

On the last step, specify the directory (called the *Blob prefix*) to which the rule should apply. The pattern is *<container>/<directories>*. As an example, I have specified that all blobs in the *fastq* folder in the *sequencedata* container should be subject to this rule. See Figure 2-15.

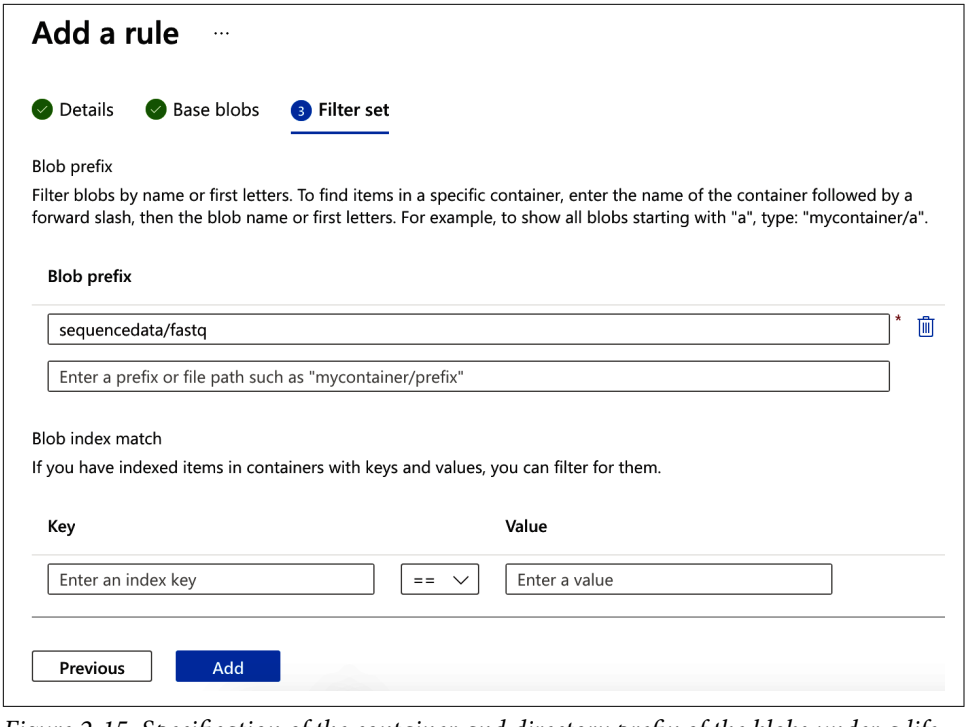


Figure 2-15. Specification of the container and directory prefix of the blobs under a life-cycle rule

In three straightforward steps, we've created a rule that will move the FASTQ data from the Hot tier to Archive after a couple weeks. This ensures that you save on costs if the activity on FASTQ files becomes stagnant once the required processing on them is complete.

Managing Access Inside the Lake

Earlier in this chapter, I talked a lot about security and managing data access within your data lake. This is something that is extremely important at any organization, to not only protect your data from the outside world but also ensure that individuals inside your organization can see only the data that they're allowed to see.

In Azure Data Lake Storage, there are two main levels that help you to manage access by people or other applications: role-based access control (RBAC) and access-control lists (ACLs).

In addition to these two permission-based access controls, you may also use shared keys or shared access signatures (SAS tokens) to grant anonymous but controlled

access. For more information on Azure Storage's authorization procedures, visit the Microsoft documentation page at *https://oreil.ly/moBao*.

Role-Based Access Control

Role-based access control (RBAC) is the broadest level of access management. It relies on your Azure Active Directory (AAD) to assign users or groups to the Azure Storage account. RBAC is considered an industry best practice that, if implemented properly, will really help in maintaining any compliance standards later. Since many of us don't want to have to focus on security, basic RBAC takes care of a lot of the worry up front.

From the Azure Portal, click your Azure storage account (data lake) that you created earlier and find "Access Control (IAM)" in the side menu. See Figure 2-16.

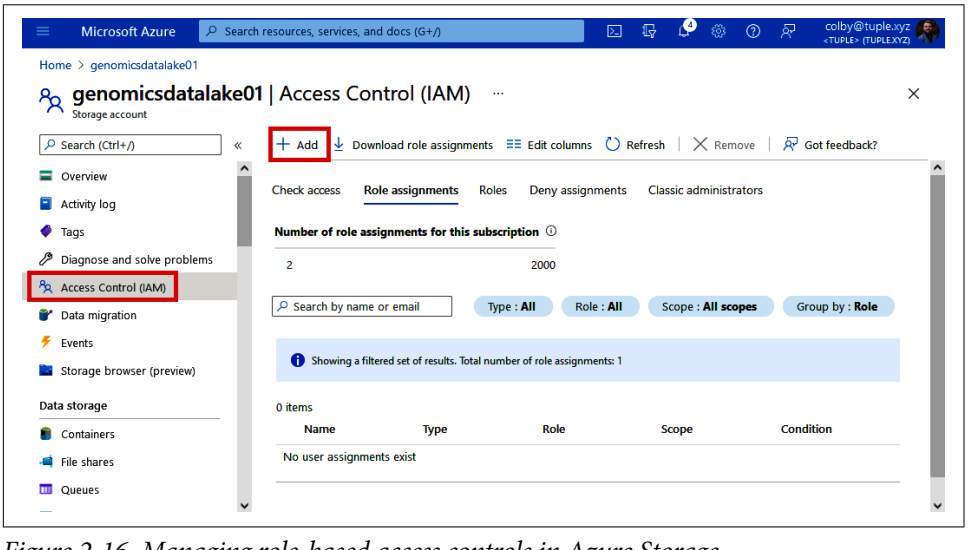


Figure 2-16. Managing role-based access controls in Azure Storage

From this Access Control (IAM) window, you can add role assignments to your storage account, check who currently has access, and more. Click the "+ Add" button at the top and click "Add role assignment." This will show you a list of the different roles a user can have in the data lake, as shown in Figure 2-17. Most of the time, general users that can read and write to the data lake will have the "Contributor" role. To assign a role, click the role name and then click the "Members" tab at the top. You'll then search for users or groups to which to apply the role.

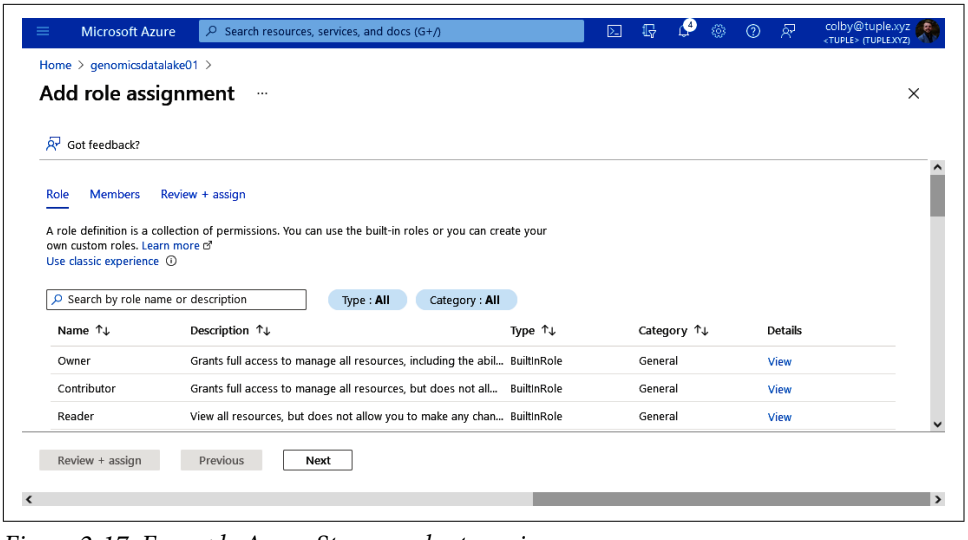


Figure 2-17. Example Azure Storage roles to assign

Note that it's common for IT organizations to create groups in AAD and then assign roles to an entire group. For example, you could create a group for the `study_001` study and give the entire group general Contributor access to the data lake. See Figure 2-18.

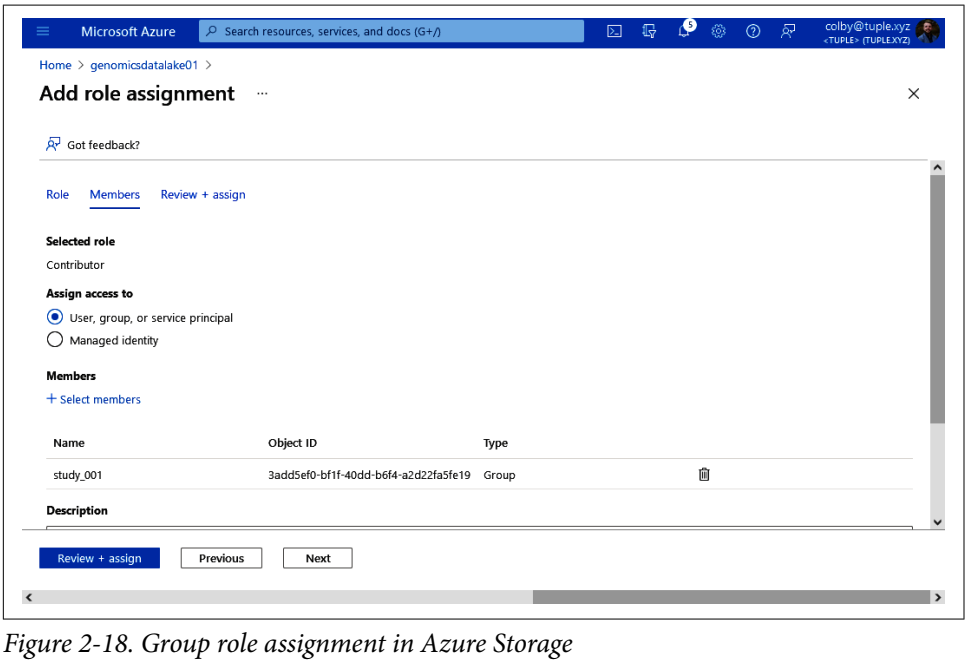


Figure 2-18. Group role assignment in Azure Storage

Access-Control Lists

The second level of access control in Azure Data Lake Storage are called *access-control lists* (ACLs). ACLs allow for very fine-grained access control on individual directories.

From within a given container or directory in your data lake, click "Manage ACL" in the side menu, as shown in Figure 2-19. This will allow you to assign access permissions to individuals or groups.

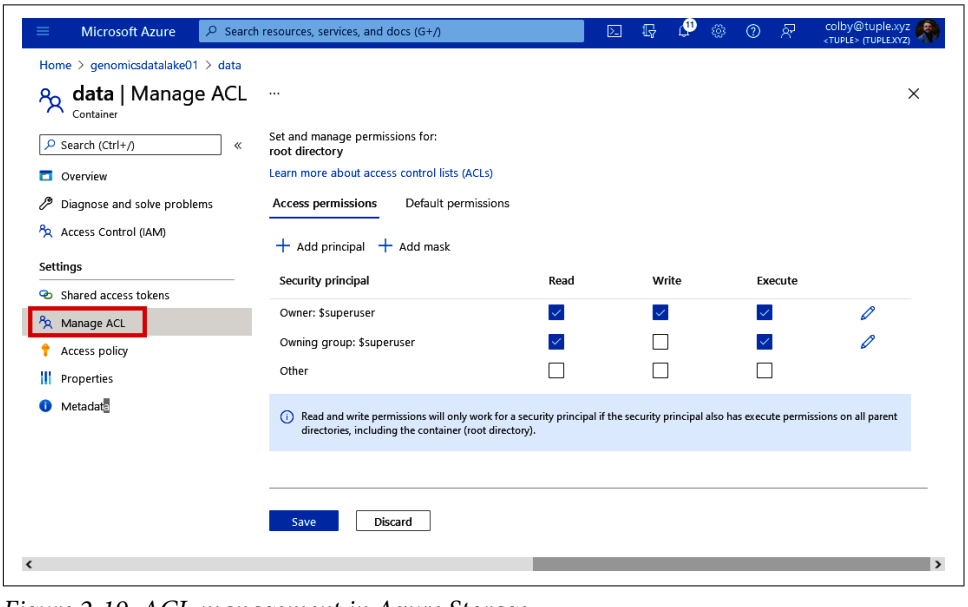


Figure 2-19. ACL management in Azure Storage

As shown in Figure 2-20, click "Add principal" and search for the user or group of interest and click "Select." Then specify read, write, and execute permissions for this user or group. Once you click "Save," the ACL will be updated and will allow or disallow the user or group from accessing data in this directory.

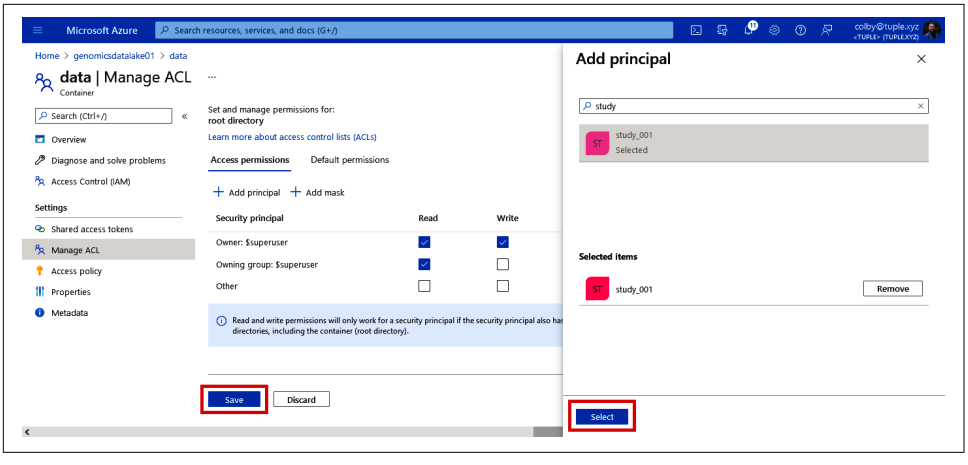


Figure 2-20. Updating the security principal in an ACL in Azure Storage

You can also update an ACL in Azure Storage Explorer by clicking the file or directory you'd like to manage and then clicking "Manage ACLs" in the toolbar at the top of the screen. See Figure 2-21.

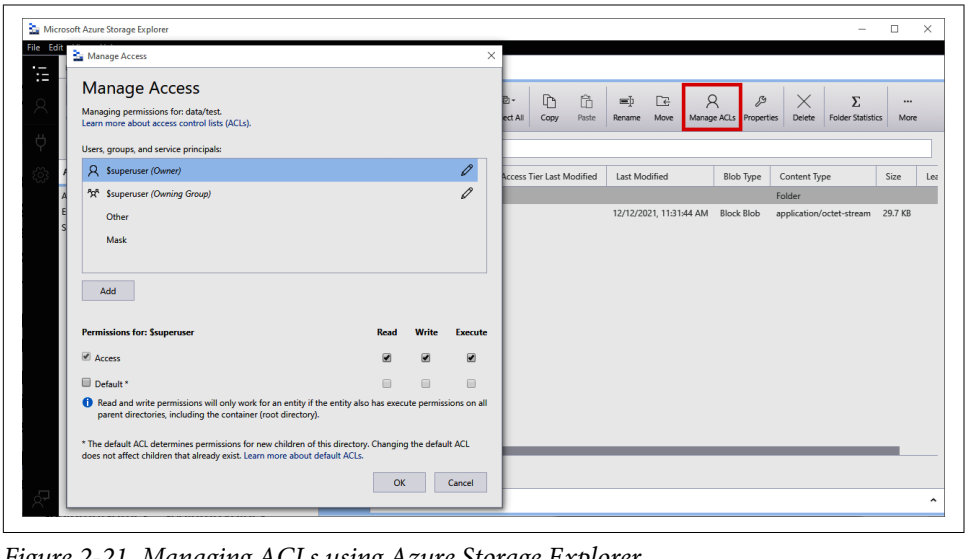


Figure 2-21. Managing ACLs using Azure Storage Explorer

From this screen, you'll be able to add users, groups, and/or service principals to the ACL, controlling if they can read, write, or execute at this location.

If you're wondering how ACLs and RBACs intersect, just remember that RBACs always take priority over ACLs. If your ACL grants access to a resource in a data lake that the RBAC disallows at a higher level, the user will not be able to access the object. Thus, RBACs grant general access to the data lake, which can be further locked down by ACLs (not unlocked).

This is an important concept as it applies to the logical way you might limit access. For example, you could give your entire department access to the data lake in general but then limit which directories or files individuals can access based on what research project they are on, their role on a project, etc.

Azure Open Datasets for Genomics

Azure hosts many popular public datasets for a variety of fields from transportation to economics. For genomics, they host mirrors of various public repositories of genome sequences, variants, and sample metadata in BAM, FASTA, VCF, and CSV formats.

Why? So you don't have to. If you use any of these references or sample files in your analyses, you don't have to host it in your data lake. This saves on the hassle of having to copy data from public repositories and saves money since Microsoft is footing the bill for storing these datasets. In other words, you can connect to these public data repository mirrors directly, which means you never have to copy and curate the data yourself. This is especially useful when you want to reference genomes or other standard reference data that would otherwise be a manual effort to have to copy and store in your own data lake.

Datasets	Description	Size	Account name	SAS Token
Illumina Platinum Genomes	Deep whole-genome sequence data and variants of 17 individuals in a three-generation pedigree.	2GB	datasetplatinumgenomes	sv=2019-02-02&se=2050-01-01T08%3A00%3A00Z&si=prod&sr=c&sig=FFfZ0QaDcnEPQmWsshtpoY0jbzd4jtwIWeK%2Fc4i9MqM%3D
Human Reference Genomes	Hg19 and Hg38 human-genome references assembled by the Genome Reference Consortium.	10GB	datasetreferencegenomes	sv=2019-02-02&se=2050-01-01T08%3A00%3A00Z&si=prod&sr=c&sig=JtQoPFqiC24GiEB7v9zHLi4RrA2Kd1r%2F3iFt2I9%2FIV8%3D

ClinVar Annotations	Clinical interpretations and reports of the relationships among human variations and phenotypes.	56GB	datasetclinvar	sv=2019-02-02&se=2050-01-01T08%3A00%3A00Z&si=prod&sr=c&sig=qFPPwPba1RmBvaffkzkLuzabYU5dZstSTgMwxuLNME8%3D
SnpEff	Annotation and functional effect prediction of genetic variants on genes and proteins.	2TB	datasetsnpeff	sv=2019-10-10&st=2020-09-01T00%3A00%3A00Z&se=2050-09-01T00%3A00%3A00Z&si=prod&sr=c&sig=isaf0a9tGnYBAvsXFUMDGMTbsG2z%2FShaihzp7JE5dHw%3D
gnomAD (Genome Aggregation Database)	Aggregated and harmonized exome and genome sequencing data from a wide variety of large-scale sequencing projects.	30TB	datasetgnomad	None
1000 Genomes	Human variation and genotype data for 2,504 individuals from 26 populations with 84 million identified variants.	815TB	dataset1000genomes	sv=2019-10-10&si=prod&sr=c&sig=9nzcxaQn0NprMPlSh4RhFQHcXedLQIcFgbERiooHEqM%3D
OpenCravat (Open Custom Ranked Analysis of Variants Toolkit)	Genomic variant interpretation, including variant impact, annotation, and scoring.	500GB	datasetopencravat	?sv=2020-04-08&st=2021-03-11T23%3A50%3A01Z&se=2025-07-26T22%3A50%3A00Z&sr=c&sp=rl&sig=J9J9wnJOXsmEy7TFMq9wjcxjXDE%2B7KhGpCUL4elsC14%3D

ENCODE (Encyclopedia of DNA Elements Consortium)	Comprehensive list of functional elements in the human genome, including elements that act at the protein and RNA levels, regulatory elements, and more.	756TB	datasetencode	sv=2019-10-10&si=prod&sr=c&sig=9qSQZo4ggrCNpybBExU8SypuUZV33igl11xw0P7rB3c%3D
GATK Best Practices	GATK workflow-specific plumbing, reference, and resources data.	542GB	datasetgatkbestpractices	sv=2020-04-08&si=prod&sr=c&sig=6SaDfKtXAIfdpO%2BkvNA%2FsTNmNij%2Byh%2F%2F%2Bf98WAUqs7I%3D
GATK Legacy Bundles	Broad public legacy b37 and hg19 reference and resource data.	61GB	datasetgatklegacybundles	sv=2020-04-08&si=prod&sr=c&sig=xBfxOPBqHKUCszzwbNCBYF0k9osTQjKnZbEjXCW7gU0%3D
GATK Test Data	Public test data focusing on smaller datasets such as whole genome BAM, FASTQ, gVCF, VCF, etc.	2TB	datasetgatktestdata	sv=2020-04-08&si=prod&sr=c&sig=fzLts1Q2vKjuvR7g50vE4HteEHBxTcJbNvf%2FZCeDMO4%3D
Broad Public Reference	Broad public legacy b37 and hg19 reference and resource data.	477GB	datasetpublicbroadref	sv=2020-04-08&si=prod&sr=c&sig=DQxmjB4D1IAfOW9AxIWbXwZx6ksbwjINkixw597JnvQ%3D
Broad Public	Public test data, often used to test workflows such as NA12878 CRAM, gVCF, and unmapped BAM files	3TB	datasetbroadpublic	sv=2020-04-08&si=prod&sr=c&sig=u%2Bg2Ab7WKZEGiAkwlj6nKiEeZ5wdoJb10Az7uUwis%2Fg%3D

You can retrieve data from any of these storage accounts using the `azure.storage.blob` Python library:

```
# %pip install azure-storage-blob
from azure.storage.blob import BlobServiceClient

## Platinum Genomes Example
### Note: Use info from Table 2.2
account_name = 'datasetplatinumgenomes'
container = 'dataset'
sas_token = 'sv=2019-02-02&se=2050-01-01T08...'

## Create clients to connect to Storage Account and container
account_url = f"https://{account_name}.blob.core.windows.net?{sas_token}"
blob_service_client = BlobServiceClient(account_url)
container_client = blob_service_client.get_container_client(container)

## List available datasets
blob_list = container_client.list_blobs()
for blob in blob_list:
    print(" " + blob.name)

## Create client for a particular blob
blob_path = '2017-1.0/hg38/small_variants/NA12877/NA12877.vcf.gz'
blob_client = blob_service_client
get_blob_client(container=container,  blob=blob_path)

## Download file
local_path = 'NA12877.vcf.gz'
with open(local_path, "wb") as download_file:
    download_file.write(blob_client.download_blob().readall())
```

In this chapter, we've covered some concepts about data lakes in Azure and also how to make a plan for housing your genomics data. The three keys to being successful with a data lake–centric architecture are organization, organization, organization. Given that data lakes integrate well with many other cloud services (both Azure and third party), it's important to have your data organized in a logical way that will help information flow between systems smoothly.

Data lakes, built in Azure Storage Accounts, are a flexible way to house both structured and unstructured data. They offer some key capabilities on top of traditional Blob storage, including the support for the hierarchical namespace. This gives us fine-grained controls over who can access parts of the data lake.

One recommended method for building a genomics data lake includes the concept of organizing by the data's level of utility and cleanliness (raw, staging, and curated). This is perfect for the various forms of genomics data. An example is how data changes its form and function when you convert from raw sequences (FASTQs) to aligned files (BAMs) to variant data (VCFs) to count information (CSVs). In addition to creating your own data lake, Azure Open Datasets hosts many useful resources for genomics (for free). You can connect to these various storage accounts to grab reference genomes, variant annotations, and test samples.

In the next chapter, we'll continue building your data architecture a step further by structuring data in a data warehouse.

CHAPTER 3

Querying Variant Data in SQL

Though the concept of "data warehouses" has existed since the 1980s, their popularity really started to grow in the 1990s, when two individuals, Bill Inmon and Ralph Kimball, separately published books on building and organizing data in data warehouses.

If you're familiar with traditional databases (like SQL Server or PostgreSQL), which we'll cover in the next section, you know that they contain tables of information that are related by IDs or keys. This allows you to join tables together and query the information as you wish. Databases are good for storage of data from a single system, and they support tons of concurrent connections. For example, think of a database behind a website that supports thousands of quick data reads and writes.

Data warehouses, on the other hand, have a different purpose. They are often used for the unification and aggregation of data from multiple source systems, joining together data for reporting and more. Instead of expecting thousands of data retrieval connections, they instead expect to perform a smaller number of complex queries to aggregate and join across a series of tables.

The difference with data warehouses is how the data in the tables is organized. We use the terms *facts* and *dimensions* as follows:

Fact
: This is a value, metric, or measure that varies between entities in a data warehouse. Thus, fact tables are often long (having many rows), with only columns to hold the values and key columns to relate these values to dimensions.

Dimension
: Dimensions are pieces of information that relate to an entity in the data warehouse. Dimension tables provide the reference information about the data in one or more fact tables.

This fact-dimension (often referred to as Kimball) style allows us to organize data so that humans can understand relationships between tables and extract data intuitively. It allows us to cut down on the redundancy of information in the warehouse while maintaining the cleanliness of the data through organization and standardization. We can standardize supporting information in dimension tables to provide consistency in our data (e.g., deciding on a standard from five different spellings of a country name: US, U.S., U.S.A., United States, United States of America).

There are two main types of schemas in data warehousing: *star* and *snowflake*. In a star schema, you have a single fact that is connected to multiple dimensions. Those dimensions are connected to only one or more facts and aren't connected to each other. Here's an example: *dimension1* > *fact1* < *dimension2* > *fact2*.

Comparatively, star schemas are faster to query and easier to understand, but they may not capture more complex relationships well. For example, star schemas do not support many-to-many relationships.

In a snowflake schema, you may have multiple separate fact tables and many dimensions that are more interrelated with each other. Given that these complex relationships are common in practice, snowflake schemas are very common. Thus, dimensions may have to relate to other dimensions to properly capture the relationship at hand. Here's an example: *dimension1* > *dimension2* > *fact1* < *dimension3* > *fact2*.

In more modern data warehousing services, it's common to see a blend of both, depending on the use case and complexity of data.

Why should we care about a data warehouse for genomics? Because it provides a perfect place to query large amounts of data. A data warehouse is a really useful asset to build as it allows us to answer questions from our bioinformatics data without having to manually search through our data lake each time. So, as we generate variant data from our bioinformatics pipelines, we can then load that data into tables in a data warehouse.

In a clinical trial or research setting, this may enable us to answer questions like these:

- What variants does sample *xyz* have in chromosome 1?
- What are the common indels that exist across our patient population?
- Are there any rare structural variants that exist across our samples? How many occurrences of a specific motif are there in the data?

Building a Genomics Data Warehouse

To get us comfortable with thinking like a data warehouse architecture, let's use a simple example of lab test results data.

Example: Lab Results

In Table 3-1, note how we have a couple patients' lab results for one or more rounds of tests. Note how there's quite a bit of redundancy in this table as we've repeated things like the patient's name, address, and lab test name multiple times. We can reduce this unnecessary repetition by breaking off these parts of the data into their own tables.

Table 3-1. Example raw lab results data in a flat table

First name	Last name	Birth date	Address	Test date	Test	Result
Rosalind	Franklin	7/25/1920	123 Helix St. Chelsea. London	9/3/1940	HDL Cholesterol	69 mg/dL
Rosalind	Franklin	7/25/1920	123 Helix St. Chelsea. London	9/3/1940	LDL Cholesterol	53 mg/dL
Rosalind	Franklin	7/25/1920	123 Helix St. Chelsea. London	9/3/1940	Blood Glucose	95 mg/dL
Rosalind	Franklin	7/25/1920	123 Helix St. Chelsea. London	9/3/1940	Blood Oxygen	95%
Rosalind	Franklin	7/25/1920	123 Helix St. Chelsea. London	10/15/1942	HDL Cholesterol	61 mg/dL
Rosalind	Franklin	7/25/1920	123 Helix St. Chelsea. London	10/15/1942	LDL Cholesterol	73 mg/dL
Rosalind	Franklin	7/25/1920	123 Helix St. Chelsea. London	10/15/1942	Blood Glucose	86 mg/dL
Rosalind	Franklin	7/25/1920	123 Helix St. Chelsea. London	10/15/1942	Blood Oxygen	98%
Kary	Mullis	12/28/1944	456 Science Way, Newport Beach, California	8/7/2015	HDL Cholesterol	69 mg/dL
Kary	Mullis	12/28/1944	456 Science Way, Newport Beach, California	8/7/2015	LDL Cholesterol	53 mg/dL
Kary	Mullis	12/28/1944	456 Science Way, Newport Beach, California	8/7/2015	Blood Glucose	95 mg/dL
Kary	Mullis	12/28/1944	456 Science Way, Newport Beach, California	8/7/2015	Blood Oxygen	95%

Take a look at the example schema shown in Figure 3-1, which shows a simple data warehouse for housing our lab test results. We could have a multitude of lab test results, but the number of distinct patients and tests is quite finite. So we can organize patient information and lab test information into dimensions, which will cut down on the redundant test descriptions and patient information. Then we can make a fact table of the lab test results that will simply house the value of each result. Over time, these dimensions may grow to have a few hundred to a few thousand rows, but the fact table could eventually contain hundreds of thousands of entries.

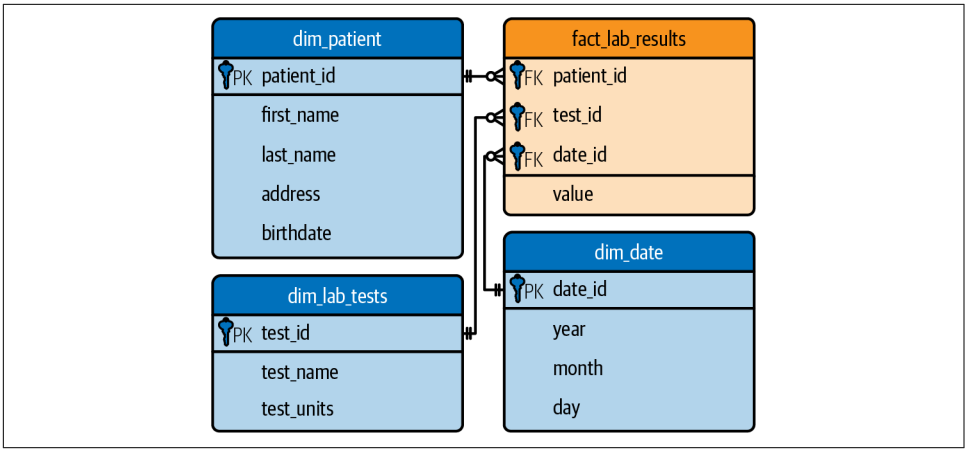


Figure 3-1. Example entity-relationship diagram of a fact table with dimensions

Data Warehouse Architecture for Genomics

When working with bioinformatics files, we often encounter semistructured file types, like FASTQ, that don't lend themselves well to being placed in queryable database tables. However, once we distill this information into, say, variant data in a VCF file, we are then able to read this data more readily. We can even use basic tools like Excel to open VCF files (assuming you have enough RAM on your machine). So this means that we can load data in this form into a data warehouse.

Why would you want to load data into a data warehouse instead of parsing it with Python or R? SQL is likely faster. Now, I'm not saying you'll never use R or Python again to work on your data, but harnessing the power of SQL with a relational data architecture is perfect for complex data tasks. With SQL, which is quite easy to read and understand, we can query and transform data with ease.

The benefit to doing this is that we can start to build out a multifunctional data warehouse for querying variant information and enriching those results with other ancillary information about the patient/subject, the study/clinical trial, etc.

In Azure, the data warehousing platform is called Synapse Analytics, which I'll cover more in a bit. For now, just know that Synapse is super scalable and provides some flexible ways to query your variant data either right from your data lake or by loading it into tables.

Variant data warehouse

In this example scenario, let's assume you work for a biotech company that makes pharmaceutical drug products and runs clinical trials to get them to market. You have an RNA-seq pipeline (as shown in Figure 3-2) that outputs an annotated VCF file of each sample.

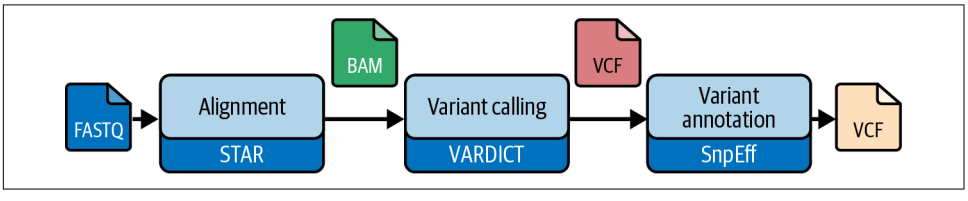


Figure 3-2. Example RNA-seq pipeline that outputs annotated VCF files from FASTQ inputs

This VCF file, once generated, will look like Table 3-2 and might live in your data lake at the following:

`datalake/research_data/product_A/trial_001/RNAseq/subject_123/sample_1/VCFs/annotated_variants.vcf`.

Table 3-2. Annotated VCF records from an RNA-seq pipeline (after skipping the header "##" lines)

CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT
chr1	16950	.	A	C	38	Q10	TYPE=SNV;DP=3;VD=2;AF=0.6667;...	GT:DP:VD:AD:AF:RD:ALD
chr1	185636	.	G	A	96	PASS	TYPE=SNV;DP=6;VD=6;AF=1...	GT:DP:VD:AD:AF:RD:ALD
chr1	188365	.	T	C	45	Q10	TYPE=SNV;DP=3;VD=2;AF=0.6667...	GT:DP:VD:AD:AF:RD:ALD

Now, let's look at an example data warehouse schema for our variant data warehouse. This schema is meant to serve only as a guide since your schema will likely differ based on your organization's needs.

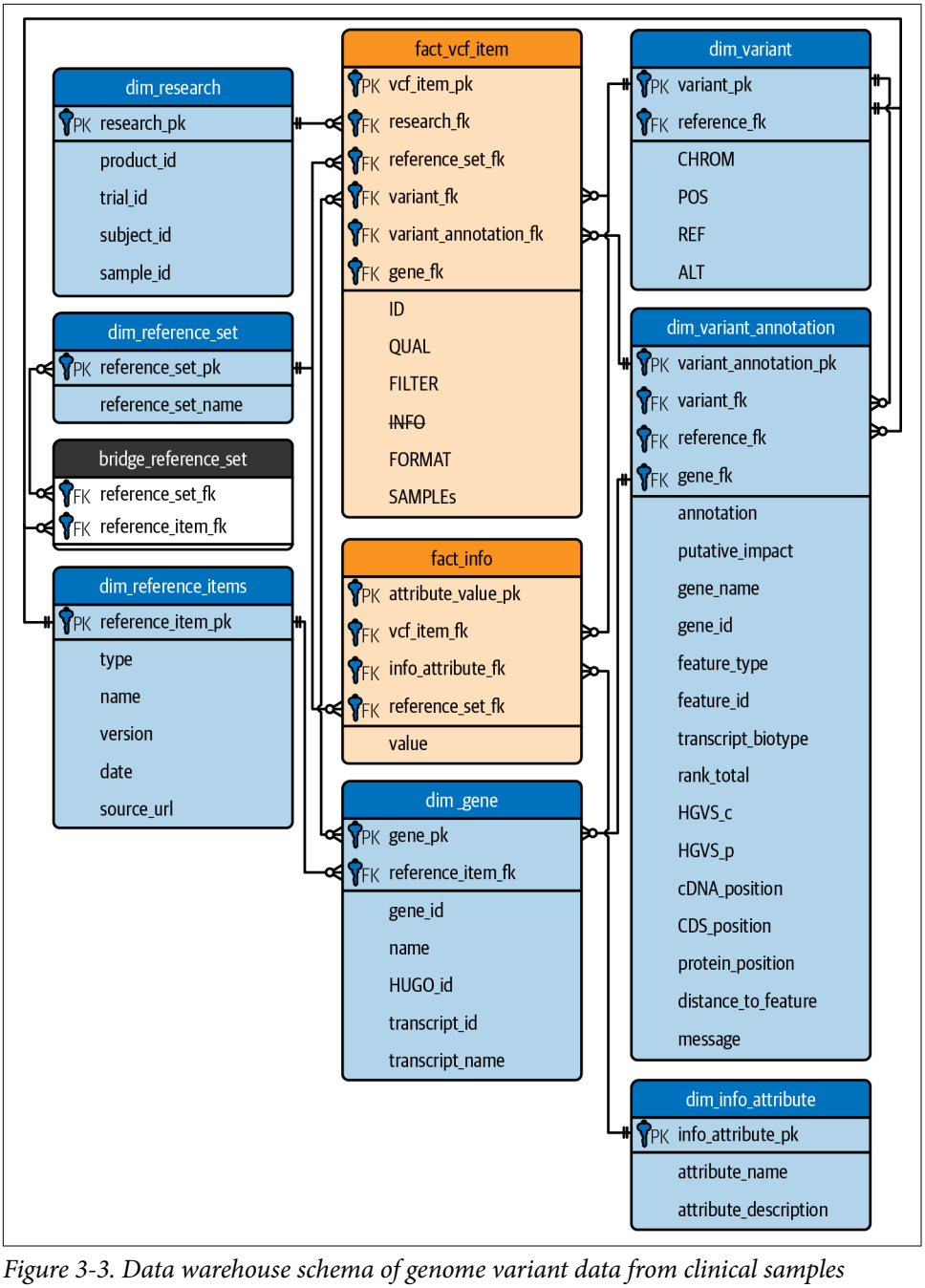


Figure 3-3. Data warehouse schema of genome variant data from clinical samples

After looking at this schema diagram for a second, it may bring to mind that quote, "Oh, what a tangled web we weave."[11] While this may look complex, I wanted to give a slightly more advanced example of how data goes from being in a series of static files that look like Table 3-2 into something that's easy to query to pull out the insights needed for your analyses.

Given that there may be a lot of repetition between samples (in this case, samples from various clinical trial participants), we may wish to break out some of the variants and variant annotation information as dimensions. Note how, in Figure 3-3, I have the VCF's fluid information, such as quality scores, etc., in the `fact_vcf_item` fact table while retaining the variant information (which may be the same between samples) in the `dim_variant` dimension. Similarly, I have broken out the data in the INFO field into paired dimension and fact tables, `dim_info_attribute` and `fact_info`, respectively.

Going back to the concept of standardization, data warehouses for genomics also provide us a place to store supporting data for our general variant information. For example, we could create a dimension of gene names (and their other IDs) and create another dimension for variant annotations and their clinical significance.

For the variant annotation information, this is data that is matched up with the variant-calling exercise from a standard database (in this case, SnpEff's database of annotations). Thus, we can assume it will not change between subjects or samples, and therefore we can store this data in a dimension (`dim_variant_annotation`) and simply reference the annotations in the `fact_vcf_item` fact table. The variant annotation information must be further parsed from the "ANN" attribute in the INFO field. Luckily, SnpEff's annotation provides a standardized set of 16 pipe-delimited ("|") values,[2] which we can turn into the columns in `dim_variant_annotation`.

There's a key difference in how we handle the general nonstandard INFO attributes versus the annotations. The relationship captured with `dim_info_attribute` and `fact_info` is a more like a key-value pair. (See Figure 3-4.) This is to be more flexible in that some VCF files that you load into the data warehouse may not have all of the same INFO attributes, and therefore you'd have a lot of missing data if you needed to capture all the possible attributes in different columns. In contrast, since we know what the output of the SnpEff annotation will always look like (the column names should stay the same), we can prepare a less flexible table but one that we can more easily query when needed.

1 Sir Walter Scott, *Marmion*, Canto XVI, Part XVII, Project Gutenberg, *https://oreil.ly/B9Rdv*.

2 For more on column definition, see *https://oreil.ly/C4iLX*.

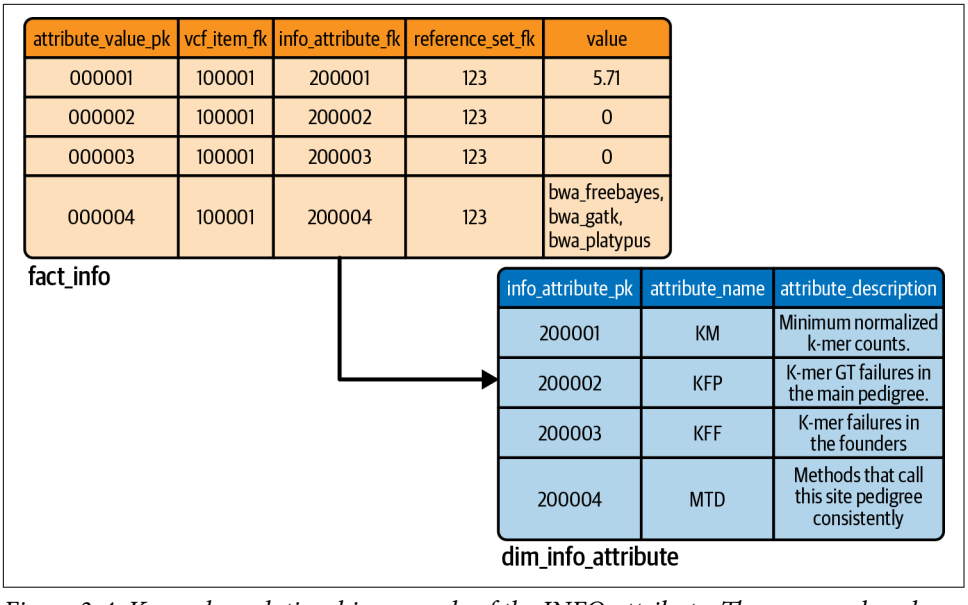


Figure 3-4. Key-value relationship example of the INFO attribute. These example values are from an Illumina Platinum Genome, which have nonstandard attributes in the INFO column.

This schema also captures the hierarchical sample information in `dim_research` by using the path information from the data lake. In this example, our data lake can be organized by `product/trial/subject/sample`. This table's information is easily derived when pulling the data in. Further, we also have dimensions to capture other ancillary information about the variants. For example, we have information on the references used to create the alignment (like which reference genome) and call the variants. And, as an example, we have a dimension of gene information (`dim_gene`) to help describe any genes or transcripts seen in the variant data.

Before we move on to creating your first data warehouse in Azure, I want to mention some caveats in this schema. For any of you reading this who, in a past life, used to be data warehouse engineers, you may notice that this architecture doesn't overly break out every possible dimension. Now that many cloud-based data warehouse products, including Azure Synapse, use column-based indexes for querying and massive parallel processing, there's not a huge need to break out small dimensional tables with a few columns into a series of tiny dimensions. For example, though the `dim_research` dimension *could* be split into individual dimensions for products, trials, and so on, it's not totally necessary here due to the way the Synapse engine will handle queries on this table. We'll revisit this concept again in Chapter 4, when we explore how to actually load data into tables.

Lastly, just to circle back on some initial concepts, this schema we've explored is a snowflake schema, as it has fact tables that are interrelated with dimensions that, in turn, are related to one another. Specifically, the fact that we've built a relationship between dimensions—such as the relationship between `dim_variant` and `dim_variant_annotation`—renders this a snowflake schema. Also, note that there is a bridge table (`bridge_reference_set`), which is type of table I didn't cover earlier, but it's a common way to handle many-to-many relationships. Specifically, this bridge table allows us to relate multiple reference items (like reference genomes or databases) to one or more reference sets. This allows us to have multiple reference sets that, for example, use the *GRCh38* human genome and multiple human genomes to be used in various reference sets.

Azure Synapse Analytics

Azure Synapse Analytics (Synapse), despite its nondescriptive name, is Azure's flagship data management and analytics solution. This single service allows us to manage our data lake, build data warehouses, create data orchestration pipelines, prepare data for Power BI reports, and more. Synapse combines a few data tools/capabilities together in a single platform, such as a SQL-based engine for data warehousing, which is an Apache Spark engine for even more scalable data analysis. Plus, Synapse includes a suite of management, monitoring, and security tools and integrates well with other systems such as Power BI and Azure Machine Learning.

It's common for organizations to have a single, giant enterprise data warehouse, which you'll have to plug into and create your own schema(s) for housing your own data. In the cloud space today, there are also a few third-party data warehouse options that you may run into—namely, Snowflake and Databricks. Both of these other options have similar capabilities and also include Apache Spark backends for distributed querying and analysis of data. In this book, we'll cover how to create a Synapse-based data warehouse for genomics, but many of the same principles will still apply even if your organization chooses another product for their enterprise data warehouse.

Creating an Azure Synapse Analytics Workspace

To get started creating your own Synapse workspace, click "Create a resource" from the main screen of the Azure Portal and search for "Synapse." Click the option that is called "Azure Synapse Analytics" and click "Create." See Figure 3-5.

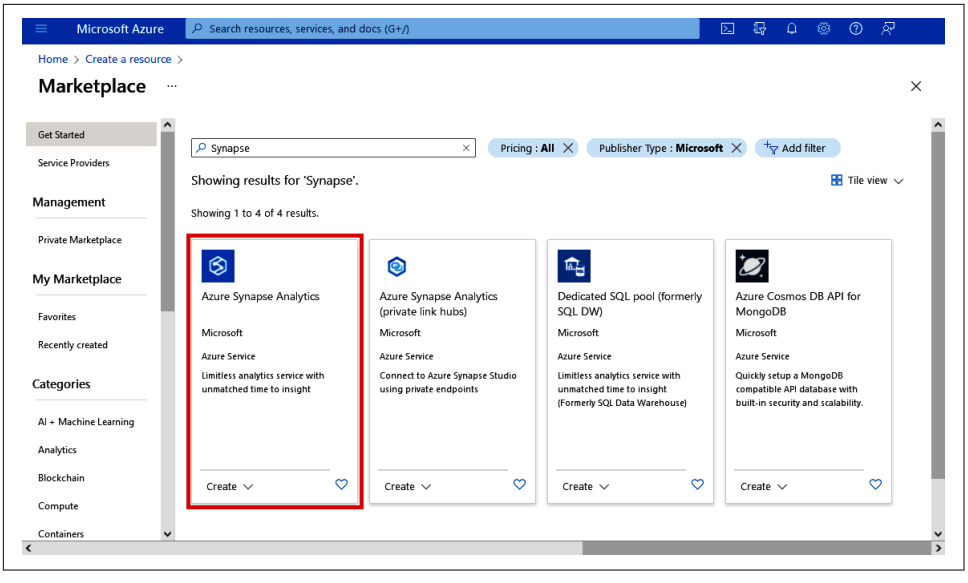


Figure 3-5. Finding the Synapse Analytics service in the Azure Marketplace

In the "Create Synapse workspace" blade that opens, shown in Figure 3-6, we'll need to define a few settings for the workspace. You can use the subscription and resource group that you created earlier in this book. Note that you'll have to specify a name for the "Managed resource group," which is a resource group that is managed by Synapse and will contain the compute services ("pools") and other supporting services that spin up for the data warehouse. I named mine *genomics-dev-asa-mrg*, where "asa" stands for "Azure Synapse Analytics" and "mrg" means "managed resource group," but the name of this doesn't matter all that much.

I specified that Synapse should use the data lake that we created in Chapter 2. Also, I told Azure to create a new filesystem (container) in the data lake called "datawarehouse," which is where Synapse will store temporary data and metadata.

Once you've filled out these fields, click the "Review + create" button, as shown in Figure 3-7. Azure will perform a quick check to validate that these settings are OK, after which you can click the "Create" button.

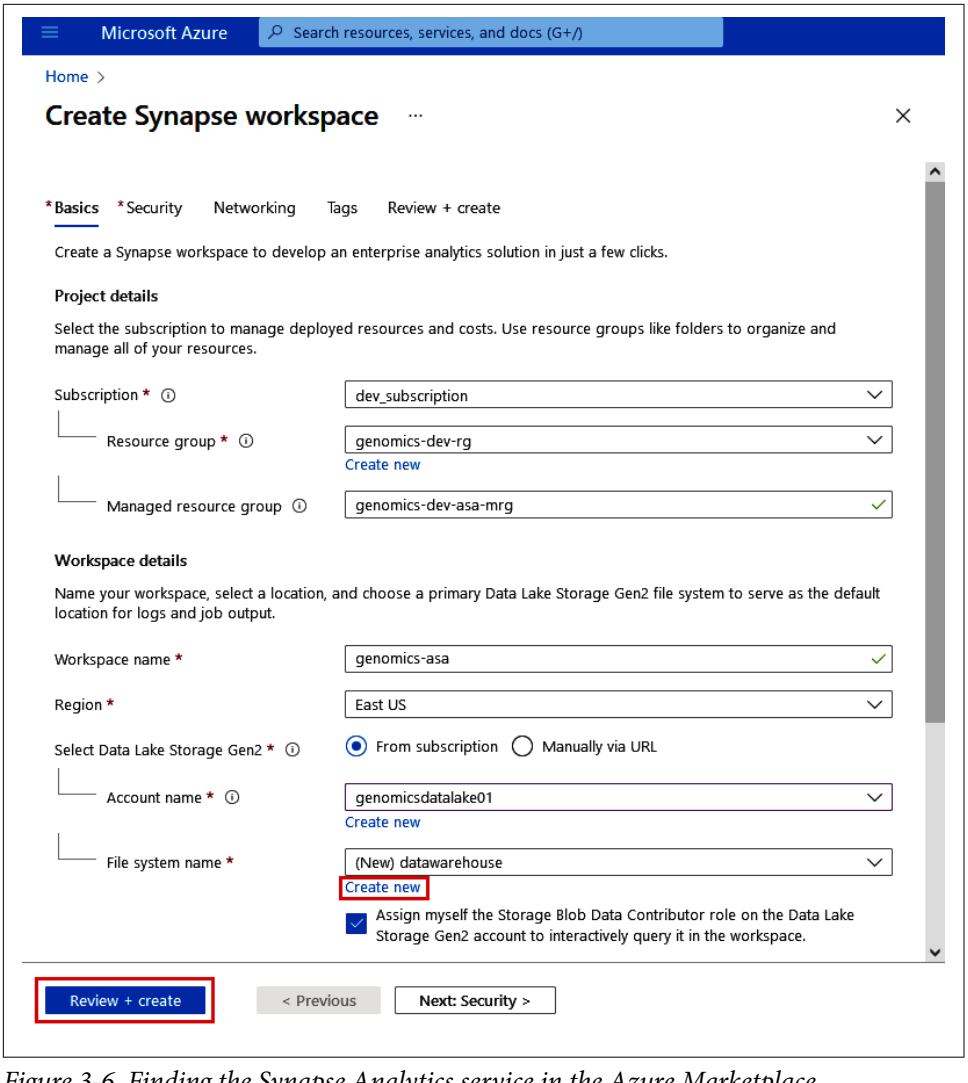


Figure 3-6. Finding the Synapse Analytics service in the Azure Marketplace

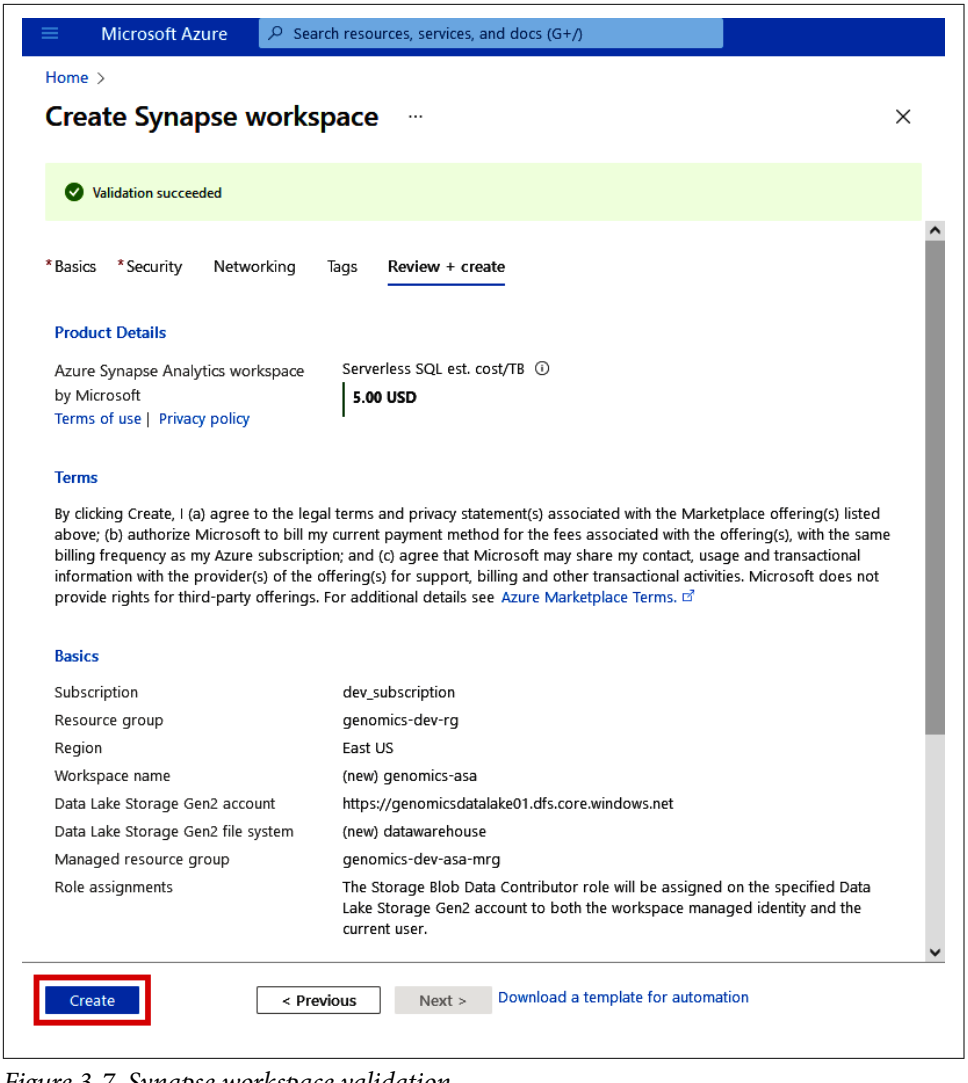


Figure 3-7. Synapse workspace validation

In just a few minutes, Azure will create the Synapse Analytics workspace, which you'll be able to find back on the main screen of the Azure Portal.

Registering Services in Subscriptions

If you happen to see the error message shown in Figure 3-8 on the "Create Synapse workspace" blade, you'll need to enable this service in your subscription. Certain services in Azure must be enabled at the subscription level before users can provision them. This is to prevent unapproved or pricier services from being accidentally created. But don't worry: it's a quick fix. (Feel free to skip this section if you didn't have any troubles provisioning your Synapse workspace.)

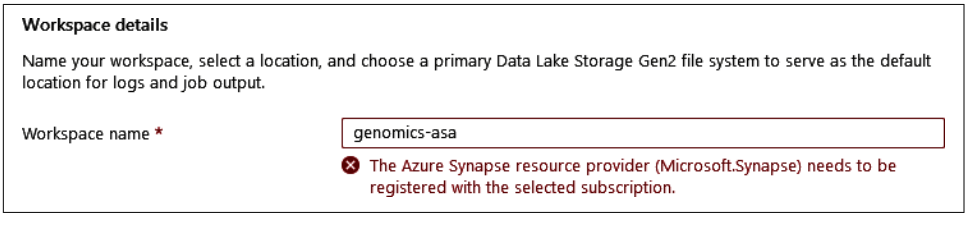


Figure 3-8. Error message when Synapse is not registered with the current subscription

From the main screen of the Azure Portal, click the "Subscriptions" icon, shown in Figure 3-9.

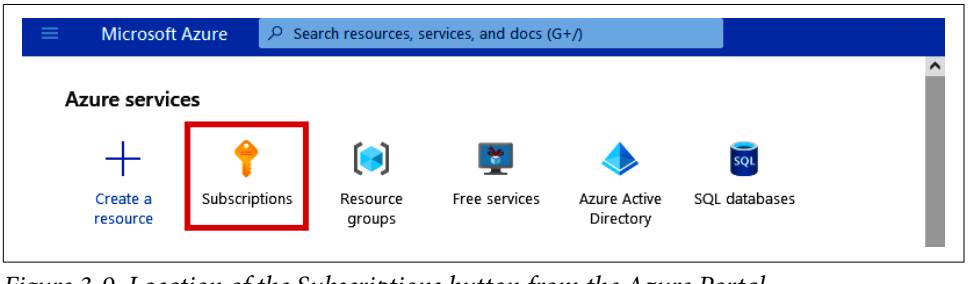


Figure 3-9. Location of the Subscriptions button from the Azure Portal

Then locate your subscription where you want to provision the Synapse workspace and click it, as shown in Figure 3-10.

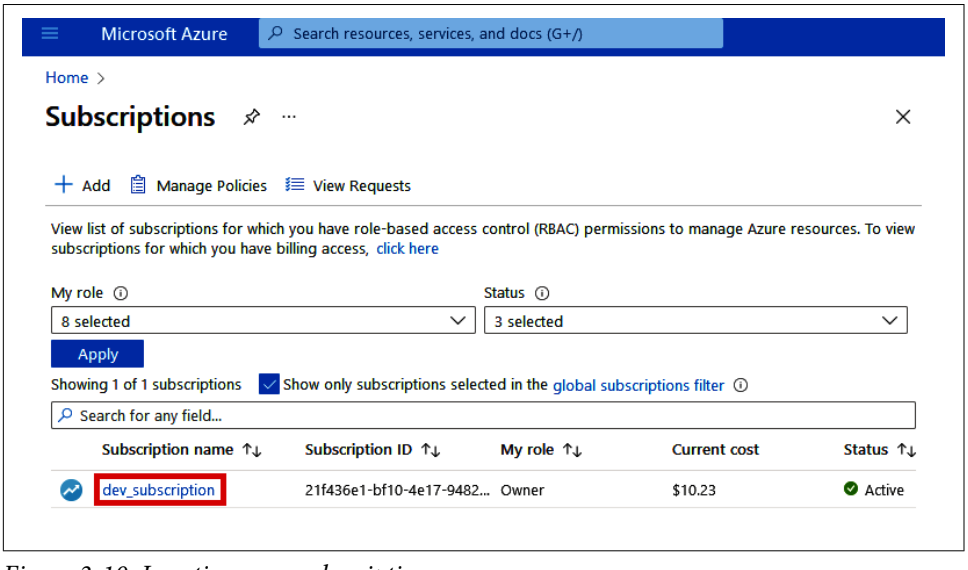


Figure 3-10. Locating your subscription

On the side menu, click "Resource providers," as shown in Figure 3-11.

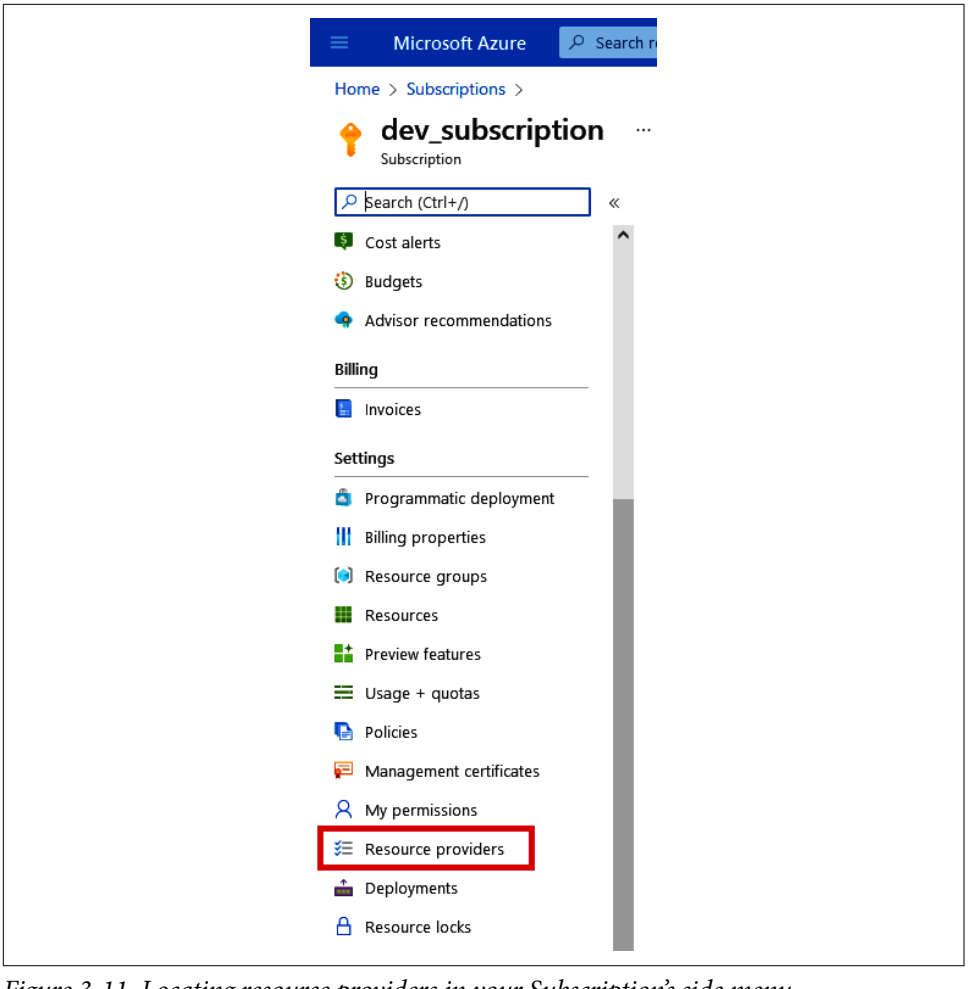


Figure 3-11. Locating resource providers in your Subscription's side menu

Finally, search for "Synapse" in the search box. If the *Microsoft.Synapse* provider is not registered, click the "Register" button at the top. (See Figure 3-12.) That's it! You can now return to the setup of your Synapse workspace.

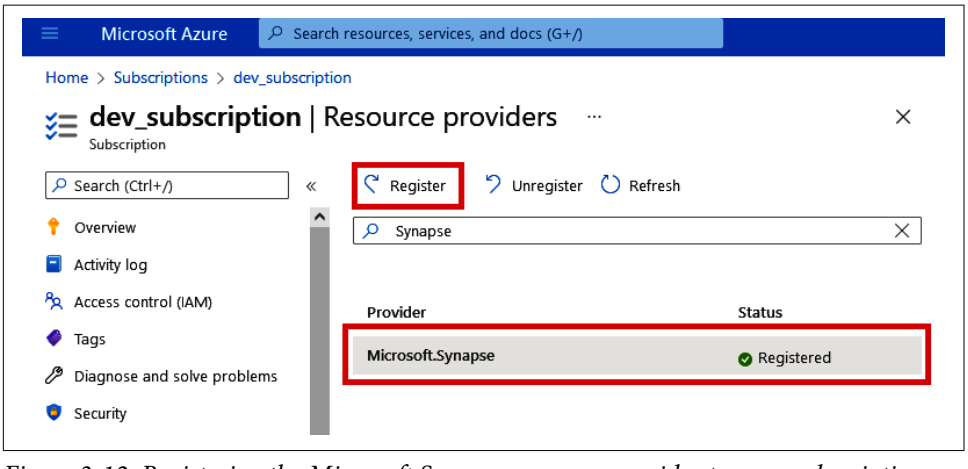


Figure 3-12. Registering the Microsoft.Synapse resource provider to your subscription

Getting to Work in the Synapse Workspace

After clicking the Synapse Analytics resource in the Azure Portal, you'll see the information about your workspace shown in Figure 3-13. To get into the actual workspace UI, locate the box titled "Open Synapse Studio" and click the "Open" link.

You'll then be taken to your actual workspace, where we can start exploring some data using Synapse. In the workspace UI, note that there are a couple of main items on the screen to "Ingest" data or "Visualize" data in Power BI. There are also links to documentation and tutorials. Feel free to explore these examples as they'll show you how to perform tasks that we may not cover in this book.

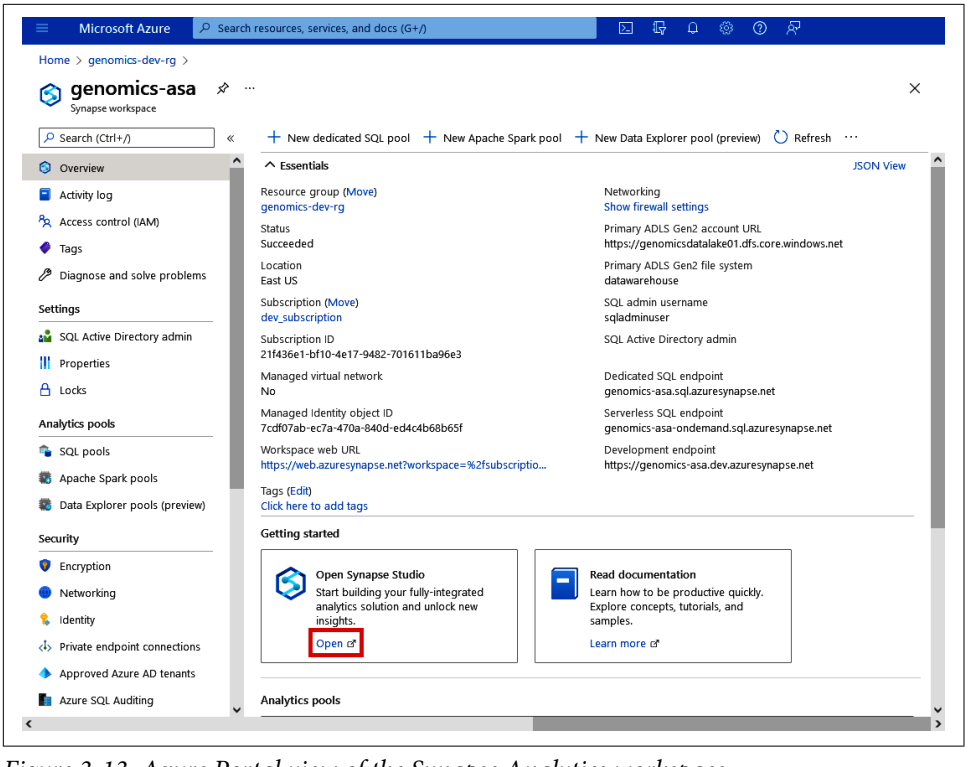


Figure 3-13. Azure Portal view of the Synapse Analytics workspace

Once you're ready to ingest your first dataset, click "Data" on the side menu, as shown in Figure 3-14.

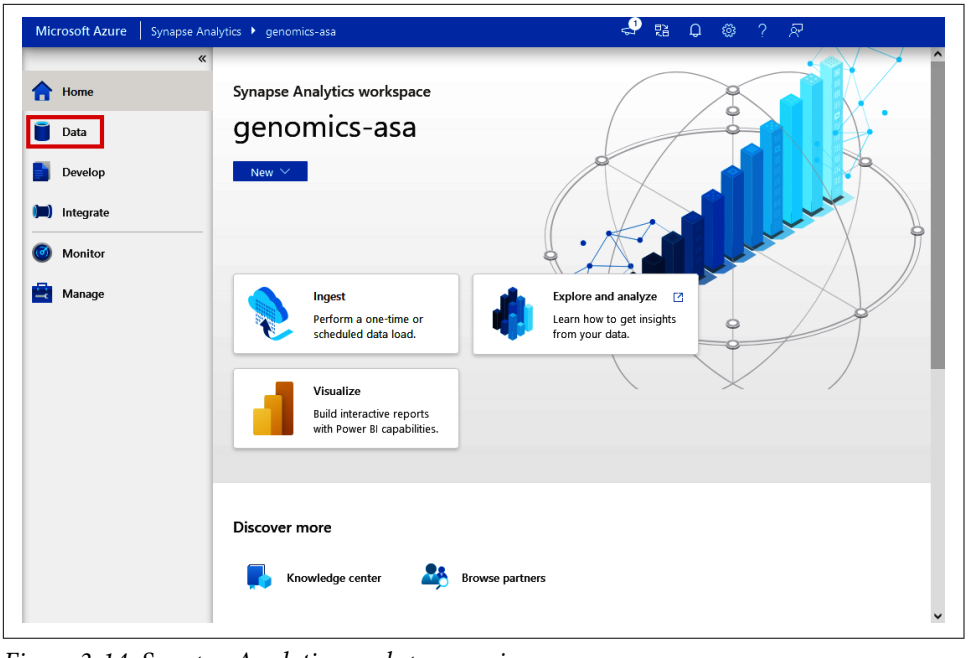


Figure 3-14. Synapse Analytics workspace main screen

You can begin exploring your data lake right from the Synapse UI. Let's start by navigating into the "Data" container that you created in Chapter 2. (If you named your container something different, that's fine.) Navigate into any folder you already have or create a new folder using the "+ New folder" button in the toolbar, as shown in Figure 3-15.

I have included a sample VCF file in the supplementary materials for this book under `03_data_warehouses/NA12877.vcf.gz`. Download this file to your local computer and unzip it. You can also try to use any other standard, single-sample VCF file for this tutorial.

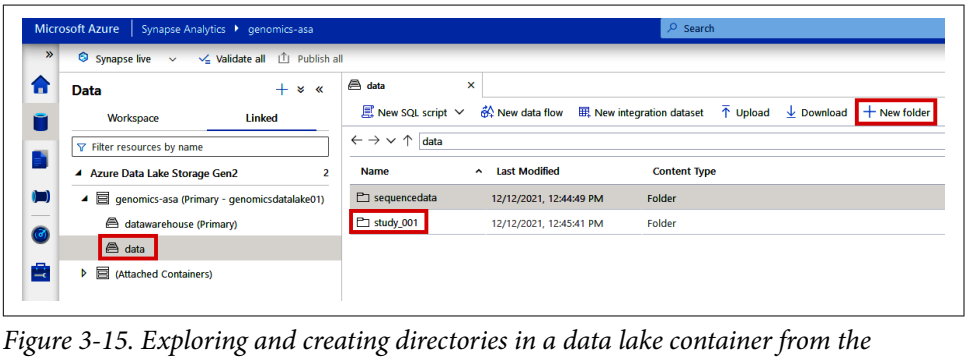


Figure 3-15. Exploring and creating directories in a data lake container from the Synapse UI

As shown in Figure 3-16, click the "Upload" button and upload the sample VCF file (`NA12877.vcf` or your own).

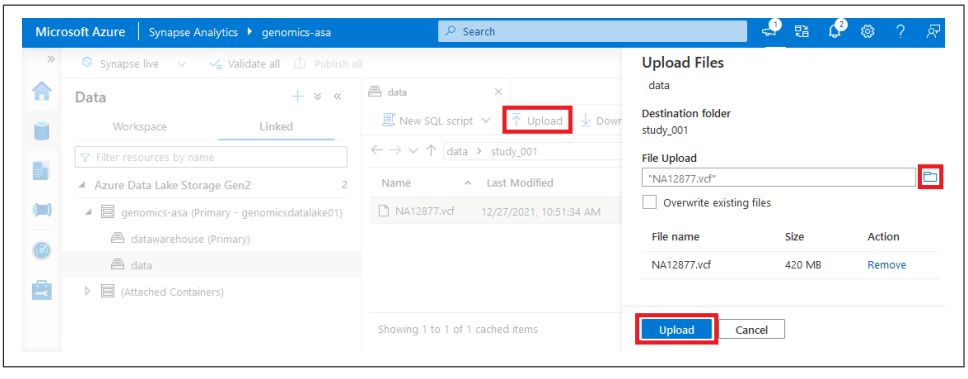


Figure 3-16. Uploading data into a data lake directory from the Synapse UI

Once this finishes uploading, note the path where your VCF file resides. You'll need that in just a second. In my case, my file is at `data/study_001/NA12877.vcf`.

Now that you have data in your data lake to play with, let's try to read that data using Synapse.

Using Open Row Sets

The first useful Synapse capability that we'll explore is called the *open row set*. This functionality allows you to read data from a linked service without having to create actual tables and load data into them. This is useful when you need to browse and query data that lives in your data lake but without having to formally load it into the data warehouse.

To start, click the "Develop" button on the left menu. Then, in the "Develop" pane, click the + icon and click "SQL script." (See Figure 3-17.) This will open a blank SQL script for you to write a new query.

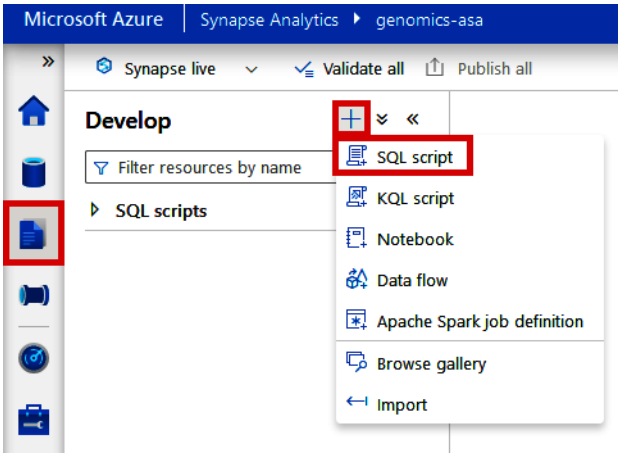


Figure 3-17. Locating the Develop menu in the Synapse UI

Copy the following code into the code editor. This code specifies a few settings on how to read the file, like the tab-delimiter and how many lines to skip (the "##" metadata lines). I also specified the column names in the `WITH` clause just to ensure that the data types don't get messed up.

Make sure to update the path to the VCF file in your data lake. Also, if you're using your own VCF file, you will need to change the `FIRSTROW` parameter to the line number of the line after the metadata section. You may also need to change the column names in the `WITH` clause if your VCF file contains different columns for some reason:

```
SELECT TOP 100 *
FROM
    OPENROWSET(
        -Update this path
        BULK 'https://<DL>.dfs.core.windows.net/<CON>/<DIR>/NA12877.vcf',
        FORMAT = 'CSV',
        FIELDTERMINATOR = ' ',
        FIELDQUOTE = '',
        FIRSTROW = 50,
        PARSER_VERSION = '1.0'
        ) WITH (
            [CHROM] nvarchar(50),
            [POS] bigint,
            [ID] nvarchar(1000),
            [REF] nvarchar(1000),
            [ALT] nvarchar(1000),
            [QUAL] float,
            [FILTER] nvarchar(1000),
            [INFO] nvarchar(4000),
            [FORMAT] nvarchar(1000),
```

```
        [SAMPLE] nvarchar(1000)
    ) AS [result]
```

Once you're ready, click "Run," as shown in Figure 3-18.

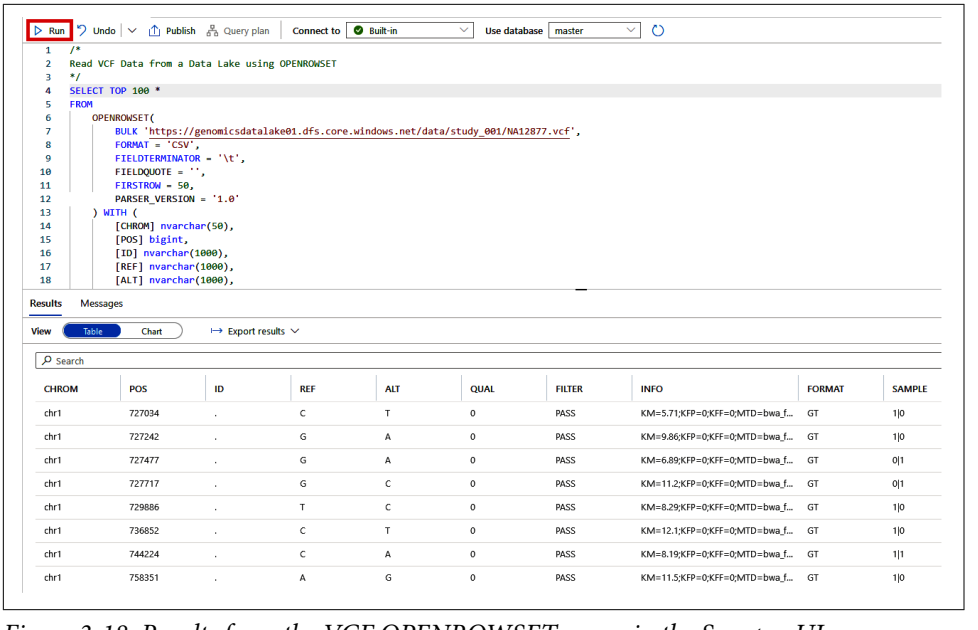


Figure 3-18. Results from the VCF OPENROWSET query in the Synapse UI

This will read from the file and display the variant entries in the results pane below in a few seconds. You can further edit the `SELECT` statement to answer your analysis questions like "How many variants were found by chromosome?" or "How many indels exist on the X chromosome?"

But what happens if we actually want to create a table in your data warehouse that we can repeatedly query? For that, we'll use external tables.

Creating External Tables

External tables are an interesting medium between directly querying files, as we did using an open row set in the previous section, and copying data directing into real (internal) tables. In contrast to traditional data warehouse tables, which actually house shaped copies of your data within them, external tables are more like pointers to datasets that live on other systems.

Synapse will create a link to a specified dataset and will dynamically read in the data at the time it's queried. This means that the data warehouse stores only the metadata about the dataset rather than the full data itself. I personally find this useful for browsing genomics data in a data lake without having to instantiate the data in a

formal data warehouse schema. It's perfect for creating a place where you can query your data lake–housed genomics data anytime you want without having to recreate the connection, schema, and query settings each time.

Before we can create an external table, we'll need to create a SQL database within Synapse. This database can be used to house both our external data and any real tables we may create later on. Navigate to the "Data" screen from the left menu. Click the + button and then click "SQL database," as shown in Figure 3-19.

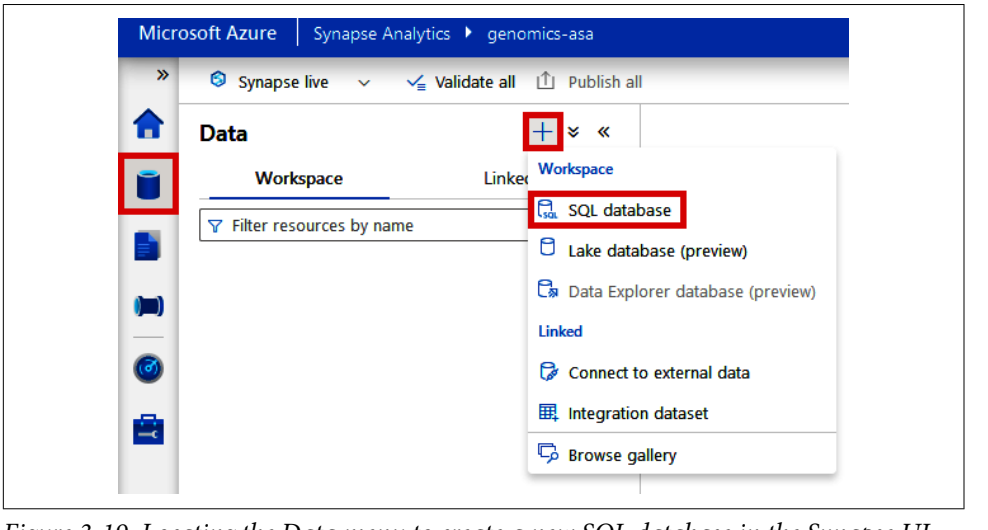


Figure 3-19. Locating the Data menu to create a new SQL database in the Synapse UI

This will open a "Create SQL database" blade on the right side of the screen. Leave the selected pool type as "Serverless," give your database a name (like `genomicsdw-01`), and click the "Create" button at the bottom, as shown in Figure 3-20. This will create a SQL database that we can use to house our external table in the next step. (We'll cover the pool types in just a bit.)

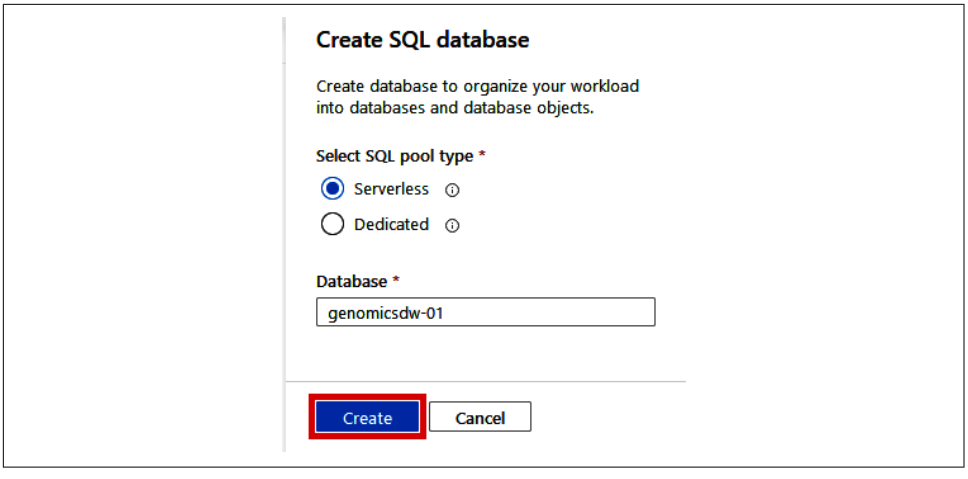


Figure 3-20. Creating a serverless pool-based database in the Synapse UI

To begin creating the external table, open a new SQL query window (as we did in Figure 3-17). Copy the following code, changing the path information to the location of your VCF file in your data lake.

Note that this time we specify some file format and connection information to the linked service. This syntax varies from the `OPENROWSET` syntax in the previous section, so be sure to look closely:

```
CREATE EXTERNAL FILE FORMAT [SynapseVCFFormat]
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS (
       FIELD_TERMINATOR = ' ',
       FIRST_ROW = 50
       )
    )
GO

CREATE EXTERNAL DATA SOURCE [<DL>_dfs_core_windows_net]
WITH (
    - Update this location path
    LOCATION = 'https://<DL>.dfs.core.windows.net/CON/'
)
GO

CREATE EXTERNAL TABLE platinum_smallvariants_NA12877 (
    [CHROM] nvarchar(50),
    [POS] bigint,
    [ID] nvarchar(1000),
    [REF] nvarchar(1000),
    [ALT] nvarchar(1000),
    [QUAL] float,
```

```
    [FILTER] nvarchar(1000),
    [INFO] nvarchar(4000),
    [FORMAT] nvarchar(1000),
    [SAMPLE] nvarchar(1000)
)
WITH (
    - Update this location path
    LOCATION = '<DIR>/NA12877.vcf',
    DATA_SOURCE = [<DL>_dfs_core_windows_net],
    FILE_FORMAT = [SynapseVCFFormat]
)
GO

SELECT TOP 100 * FROM platinum_smallvariants_NA12877
GO
```

In your query window, make sure to click the "Use database" drop-down menu and change the selected database from "master" to the database we just created. Once you're ready, click the "Run" button at the top. See Figure 3-21.

Similar to the open row set query we performed before, this will output the top 100 rows of this VCF file in the "Results" panel at the bottom of the screen.

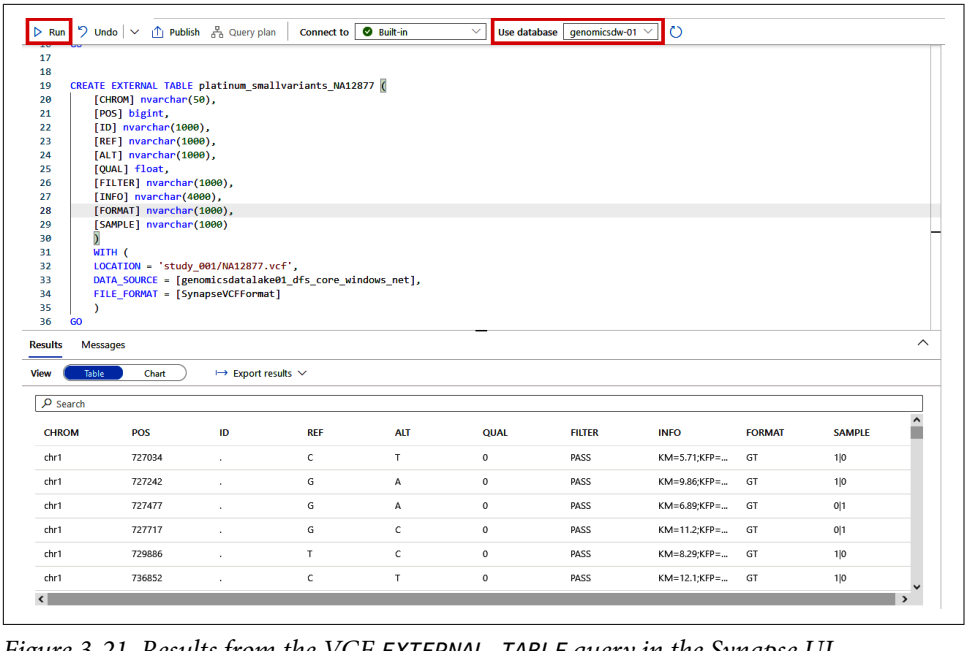


Figure 3-21. Results from the VCF `EXTERNAL TABLE` query in the Synapse UI

The difference with external tables is that this script will save the table to the database so that we can revisit and requery this data anytime we want. This method is quite useful when we want to write more complex queries for this VCF data. You simply

create the table once and then can hit it with various SQL queries to analyze or shape the data or join other data to it. You can see the external table's columns from the "Data" menu, as shown in Figure 3-22.

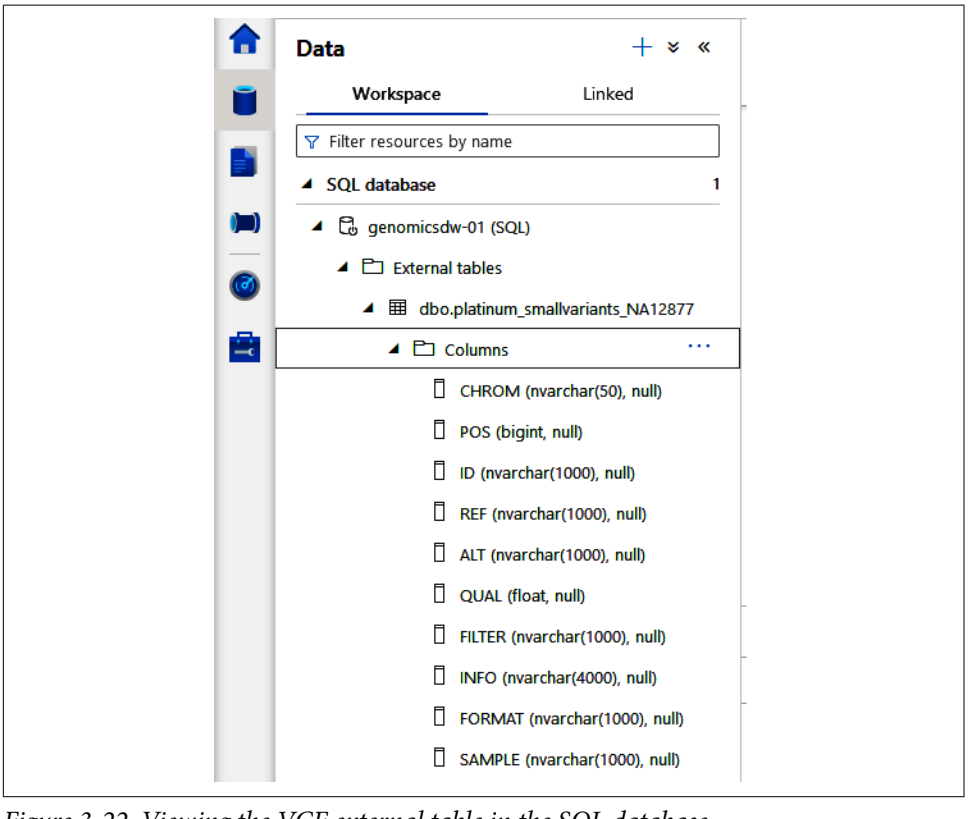


Figure 3-22. Viewing the VCF external table in the SQL database

Did Someone Say "Pool Party"?

You may have noticed that there were a couple pool type options (Serverless and Dedicated) when we were creating a SQL database earlier. These are Synapse's back-end system options for compute resources (i.e., the services that actually perform the querying tasks that you request). These are called *pools*, which are simply collections of compute resources. Each has its own strengths and cost structure, so it's important to pick the right pool for the job at hand.

You can view the various pools that have been created in your Synapse workspace (and create new pools) in the "Manage" window located at the bottom of the left menu. See Figure 3-23.

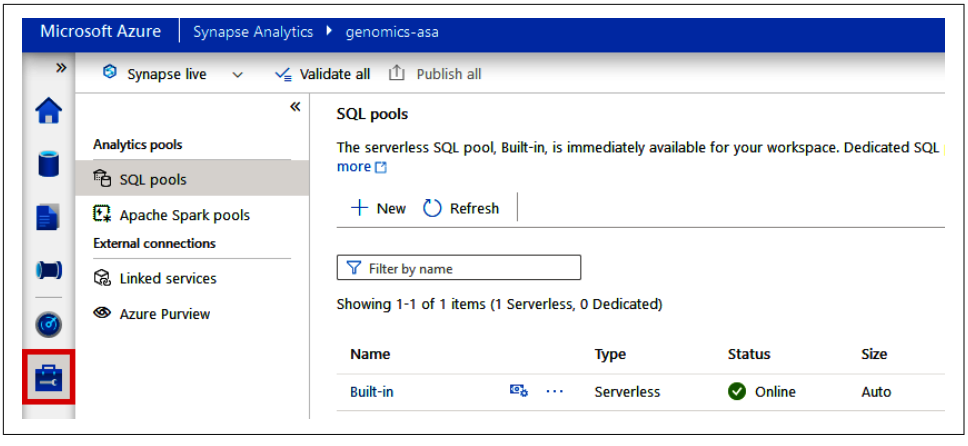


Figure 3-23. Viewing the Pools Synapse UI Manage window

Serverless SQL pools

The first and, in my opinion, most flexible pool type is the serverless SQL pool. This pool type allows you to use SQL without having to reserve a certain capacity of compute. You're charged based on how much data the pool processes rather than the number of physical nodes that are used. I would recommend this pool type for ad hoc querying of data, especially if you want to simply read and query data from your data lake, Microsoft Dataverse, or an Azure Cosmos DB.

This pool type is also useful for development purposes and hammering out any data transformation pipelines. Then, when you're done getting the logic right, you can quickly change to a dedicated SQL pool.

Dedicated SQL pools

In contrast to serverless SQL pools, dedicated SQL pools are provisioned at a specific size, and you pay for them while they are turned on, no matter how much data is being processed. This type of pool is ideal for larger, complex data querying or otherwise planned tasks. Why? Because you can turn dedicated pools off and avoid charges after you're done. With dedicated pools, you pick a specified performance level, which is measured in obscure units called "data warehouse units" or DWUs (as shown in Figure 3-24). The more DWUs, the more CPU cores, memory, and data I/O available to be utilized.

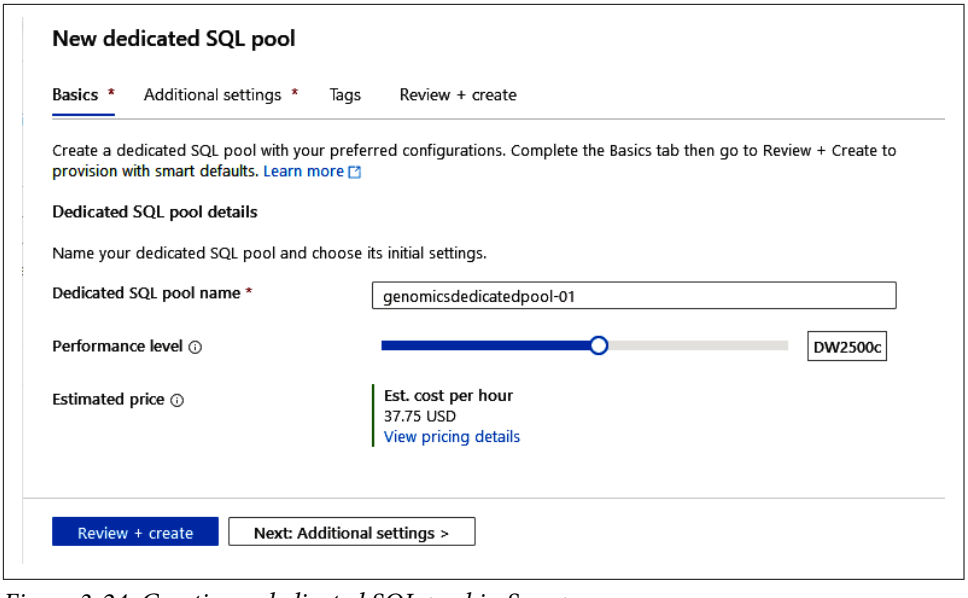


Figure 3-24. Creating a dedicated SQL pool in Synapse

Serverless Spark pools

The last type of pool is a serverless Spark pool, which creates a Spark session in the background. This allows users to utilize SparkSQL functionality inside of Synapse. The benefit? Highly distributed and scalable processing of data in a Spark cluster, which can speed up some complex data transformation tasks that might otherwise take longer in traditional SQL. This pool type is similar to the serverless SQL pool in that you are charged based on the amount of data that's processed. This also allows you to install Spark libraries on the pool, which is perfect if you have packages that make certain data processing tasks easier.

While Synapse is an excellent data warehousing platform with tons of data wrangling capabilities, the Spark engine isn't quite as flexible or directly configurable as other Spark-based systems like Azure Databricks, which we'll cover later in this book. I would mainly use Synapse Spark pools for larger data transformation tasks where you can employ the SparkSQL logic built into Scala or PySpark. I would not recommend using the Synapse version of Spark for running other analyses like machine learning or bioinformatics pipelines.

Pool cost considerations

Sometimes it's cheaper to buy a membership to your local community pool. Other times, you just want to be able to swim in your own pool whenever you like. The costs for the different Synapse pools vary quite a bit; see Table 3-3 for some examples.

Table 3-3. Cost examples for analyzing 100TB of data over five hours with various Synapse pool types

Pool type	Cost structure	Considerations	Options	Compute cost
Serverless SQL	per TB	Amount of data analyzed	None	$450
Dedicated SQL	per hour	DWU tier	DW1000c DW2500c	$76 $189
Serverless Spark	per hour	# Nodes, Node types	5 large, memory optimized nodes 5 large, hardware optimized nodes	$39 $60

Depending on how long it takes you to process your data, certain options may be more cost-effective than others. Also note that these cost examples do not include the costs for storing the data, which is about $23/TB.

Connecting to More Data Sources

In addition to the data lake that we've been working out of, you can also connect your Synapse workspace to a variety of other data sources. This is perfect for pulling in data from multiple places to enrich your data warehouse. For example, we could pull in annotation data, reference information, patient data, and more so that it's all accessible from our genomics data warehouse.

To connect to another external data source, click the plus (+) button at the top of the "Data" screen and click "Connect to external data," as shown in Figure 3-25.

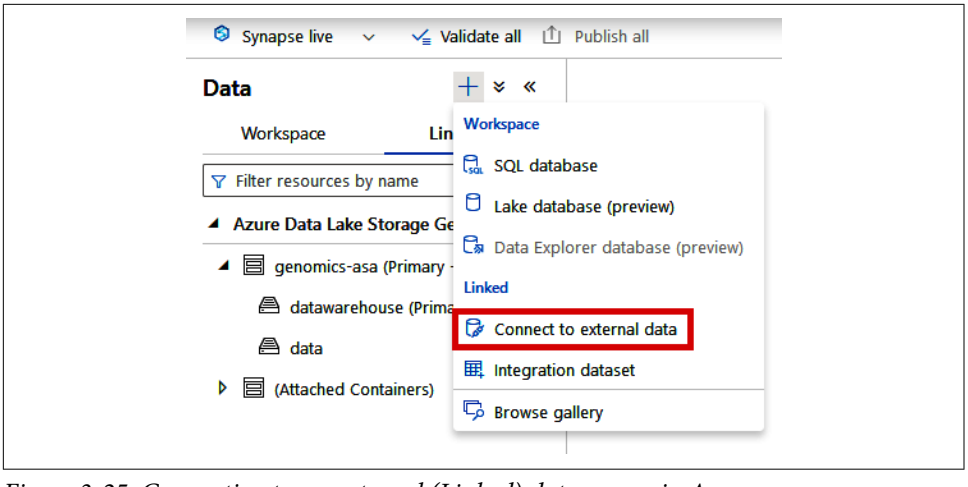


Figure 3-25. Connecting to an external (Linked) data source in Azure

From this list, shown in Figure 3-26, you can pull in other Azure-based data. Simply click the type of service you wish to connect to and fill out the service's information on the next screen. You'll then be able to pull data from these sources in the same way that we pulled from the genomics data lake.

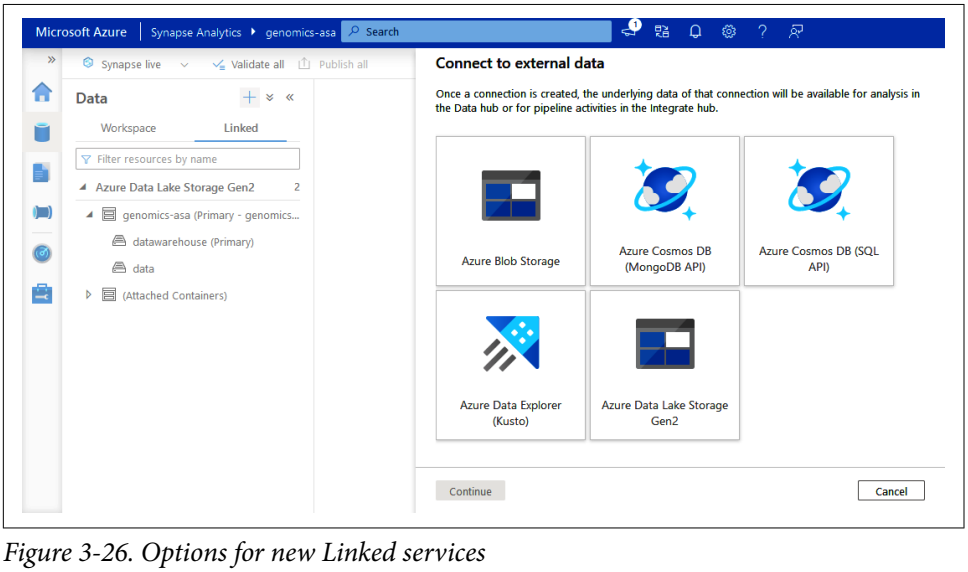


Figure 3-26. Options for new Linked services

If you have other data sources that are based outside of Azure, you can connect to those as well. As shown in Figure 3-27, click the plus (+) button at the top of the Data screen and click "Integration dataset."

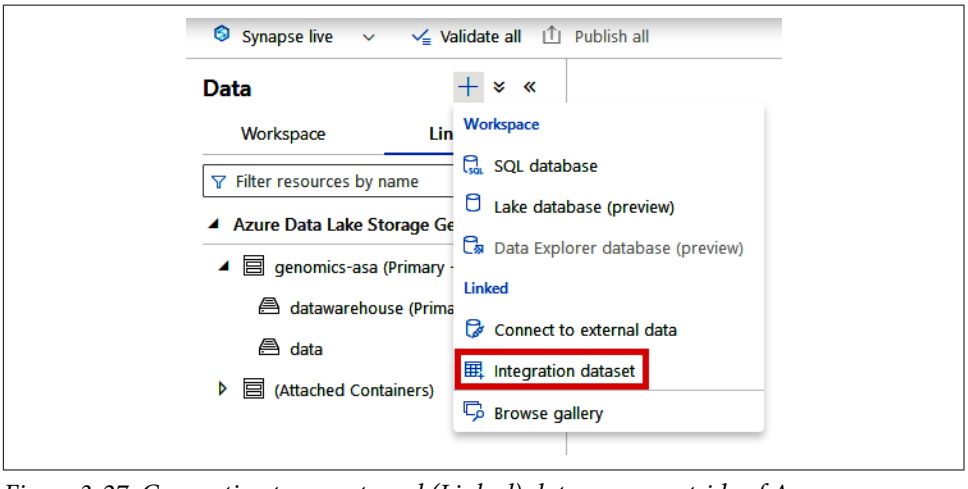


Figure 3-27. Connecting to an external (Linked) data source outside of Azure

From the "New integration dataset" screen, you'll find tons of options to use as data sources. You can find storage services from other clouds like AWS S3 and Google Cloud Storage or other database, data warehouse, and ERP services like Oracle, Snowflake, Salesforce, MongoDB, and MySQL. Finally, I recommend checking out the "File" and "Generic protocol" tabs, where you can find connectors for the FTP, HTTP, and REST protocols. These are especially useful when you want to pull genomics data from public sites. See Figure 3-28.

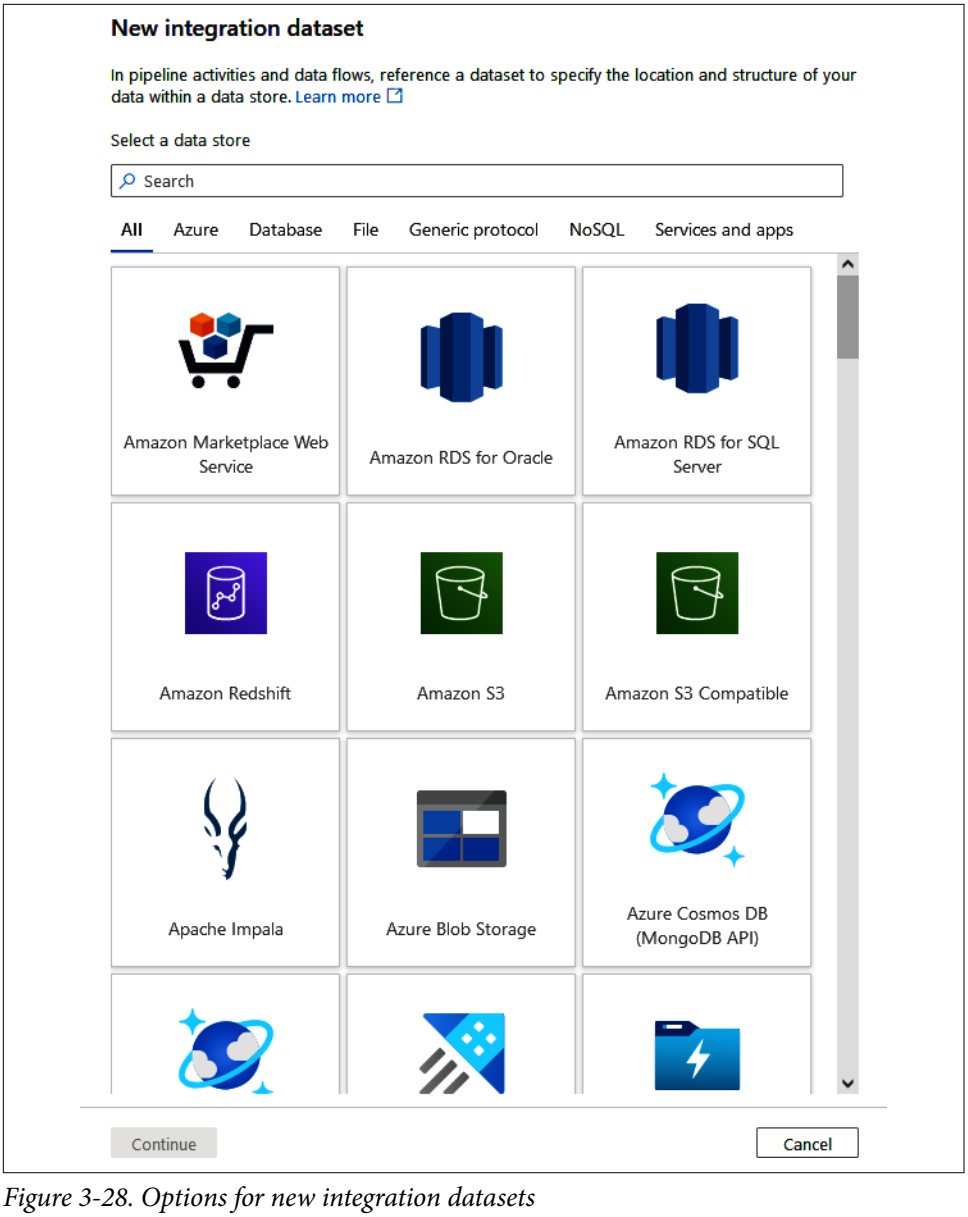


Figure 3-28. Options for new integration datasets

Whether you use the integration datasets or other Azure linked services, you may need to transform the data a bit before dropping it into data warehouse tables. We'll cover how to connect to other services and how to transform data in Chapter 4. In the meantime, let's also take a look at Azure SQL DB for smaller data-centric tasks in the cloud.

Azure SQL DB

While data warehouses are common in large enterprises that want to provide a standard source for historical data, the services they run on can sometimes be overkill if you don't have a ton of data. Specifically, while Azure Synapse is a super flexible and scalable platform for housing and analyzing data, it may be cost-prohibitive if you don't have much data to process.

Plus, there are plenty of use cases where basic SQL databases are useful. If you're creating an application that needs to read and write small amounts of data frequently to data tables, sometimes a data warehouse isn't the best option. Thus, I want to also talk about Azure SQL DB briefly, the main database service in Azure. (If you don't have the need for databases at the moment, feel free to skip this section for now.)

Creating a Database in Azure SQL DB

Back in the Azure Portal, you can often find the "SQL Database" service option in the "Create a resource" menu. If it doesn't show up for you, simply search for "SQL Database."

Once you locate it, click the "Create" button. See Figure 3-29.

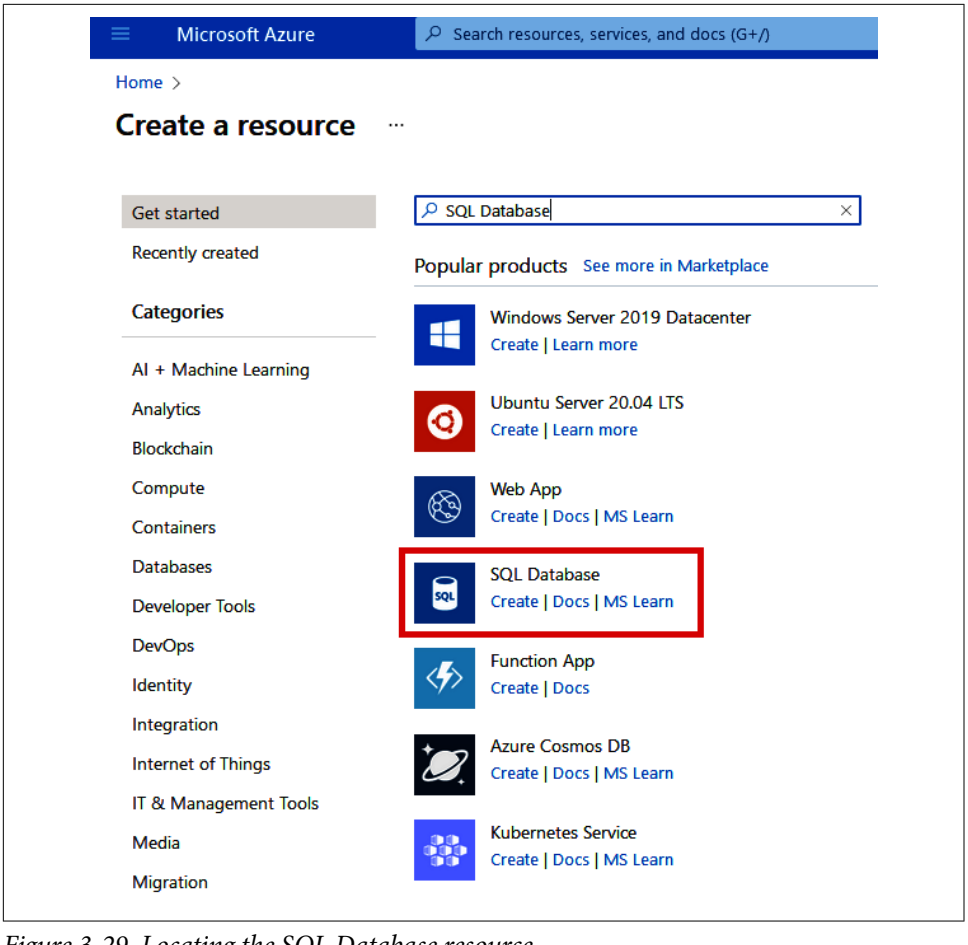


Figure 3-29. Locating the SQL Database resource

In the "Create SQL Database" blade, pick the subscription and resource group for your database and give the database a name. For the "Server" field, you'll need to create a new SQL database server, which is essentially the server backend that houses one or more SQL databases. Click the "Create new" link. See Figure 3-30.

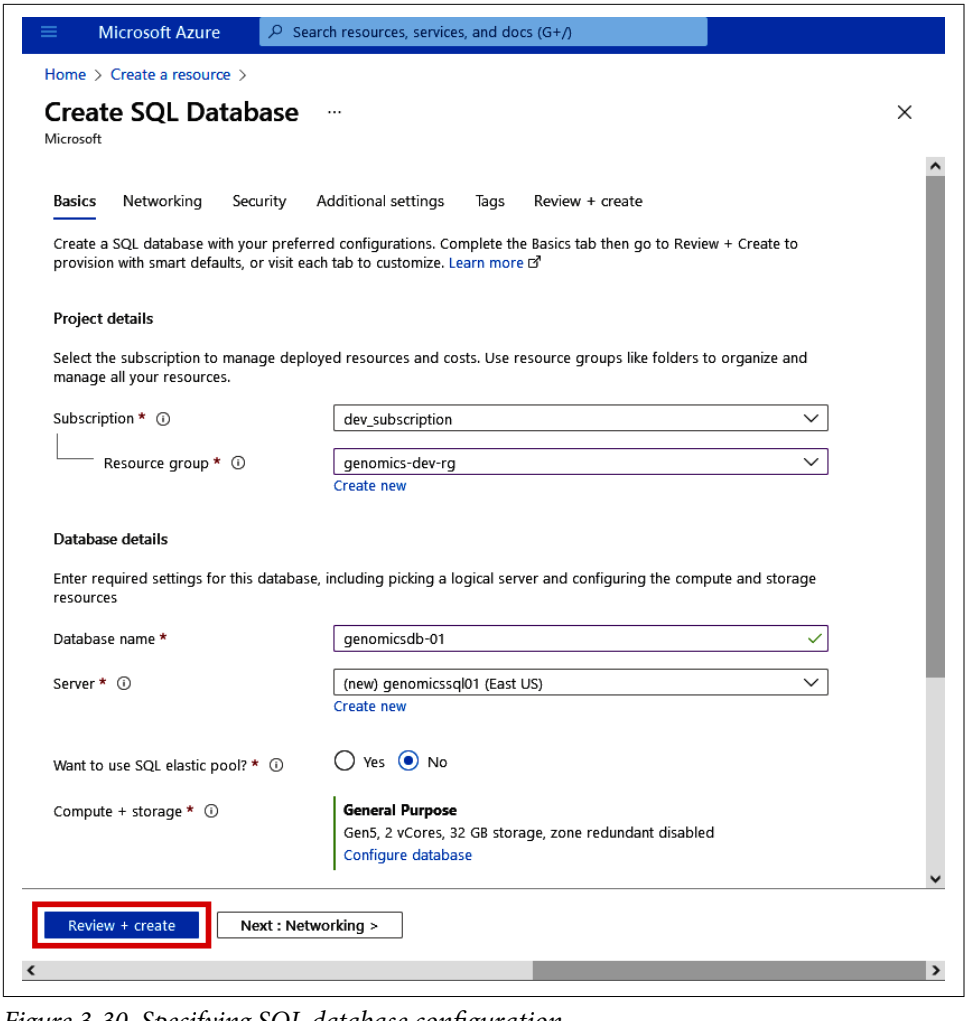


Figure 3-30. Specifying SQL database configuration

In the "Create SQL Database Service" blade, give your server a name. (This will need to be globally unique, so your server's name will have to be different than mine.) For the "Authentication method," I recommend using *both the SQL and Azure AD authentication* as this will allow you to connect to the database using your account while also having the admin credential set, which is useful when connecting applications to the database server. Once finished, click the "OK" button, which will take you back to the "Create SQL Database" blade. See Figure 3-31.

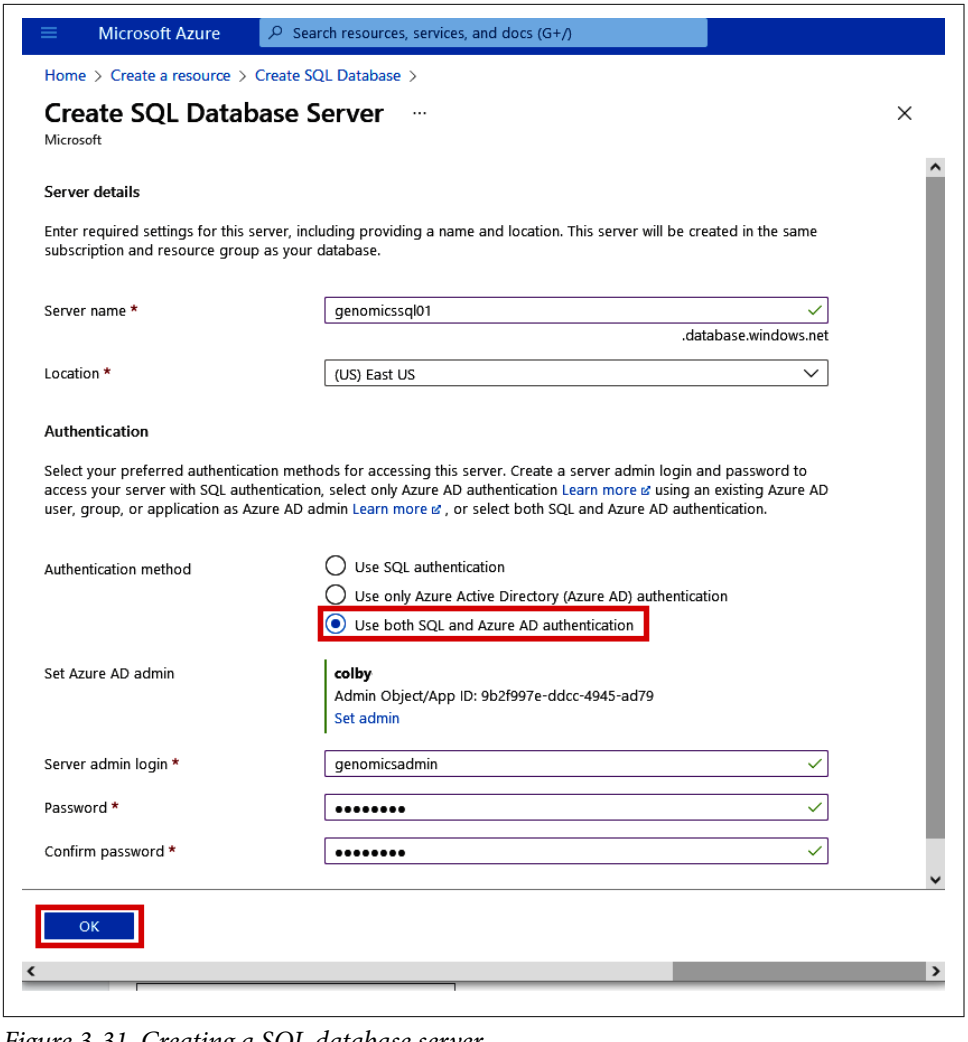


Figure 3-31. Creating a SQL database server

From a compute standpoint, Azure SQL Database has three main options (similar to Synapse): elastic pool and provisioned and serverless compute tiers.

Elastic pools

Elastic pools are useful when you have multiple databases with varying and unpredictable usage demands. An elastic pool will share the specified number of virtual cores (vCores) and memory across all of the databases in the pool. In the "Create SQL Database Service" blade, answer "Yes" to the "Want to use SQL elastic pool?" question and click "Create new" under the "Elastic pool" drop-down. See Figure 3-32.

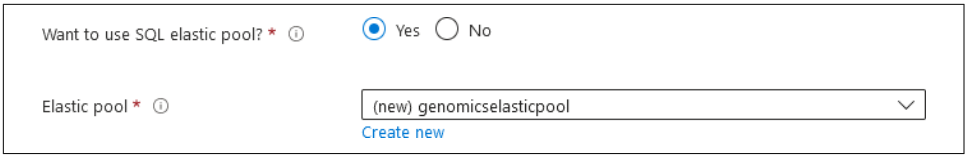


Figure 3-32. Choosing the elastic pool option during the SQL database creation

Finally, click "Configure elastic pool" beside "Compute + storage." Pick the number of vCores and amount of memory to be shared across your databases, as shown in Figure 3-33.

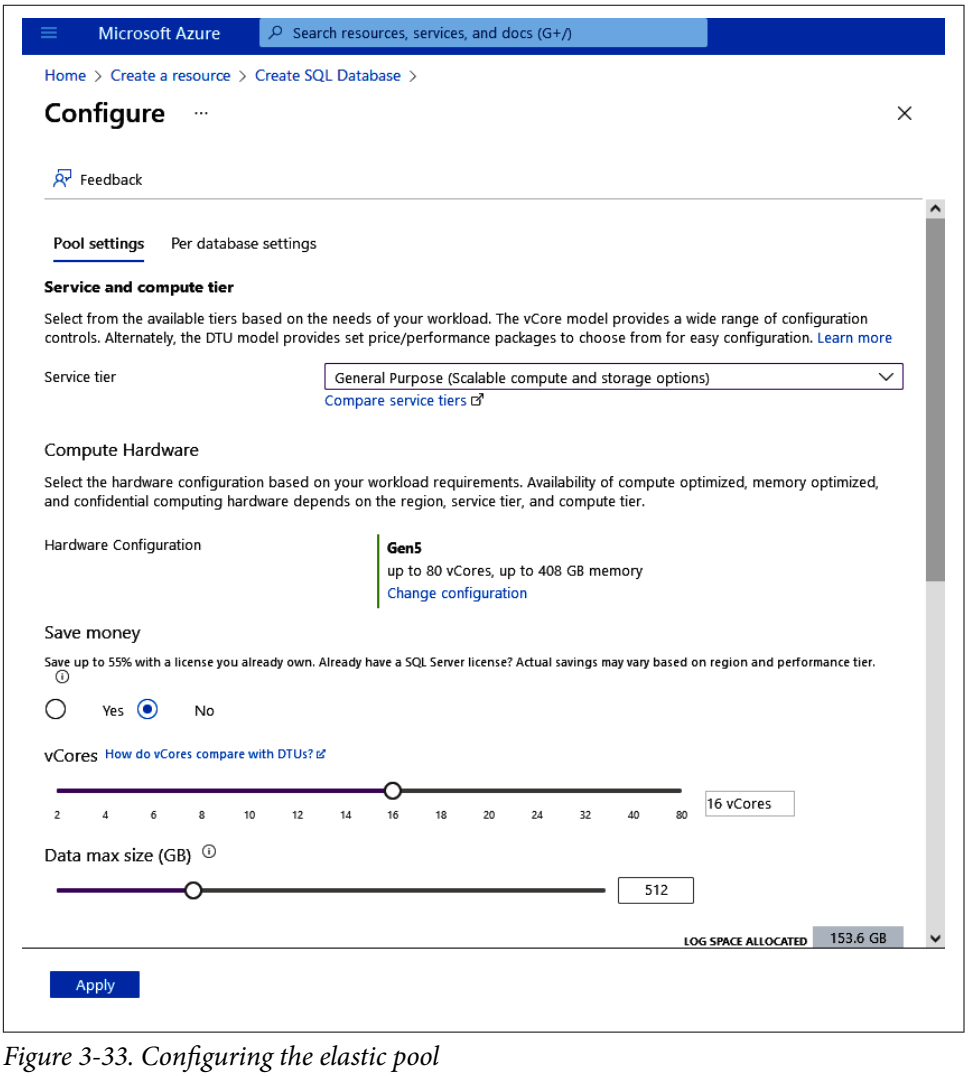


Figure 3-33. Configuring the elastic pool

Provisioned and Serverless compute

Provisioned and Serverless compute tiers are useful for specifying the resource limits for a single database. To pick one of these types, answer "No" to the "Want to use SQL elastic pool?" question in the "Create SQL Database Service" blade and click "Configure database" beside the "Compute + storage" section. See Figure 3-34.

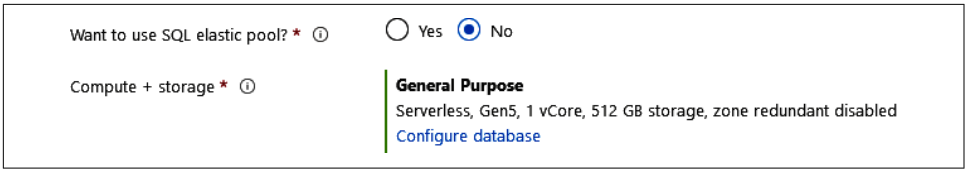


Figure 3-34. Choosing the Provisioned or Serverless compute tier option during the SQL database creation

In the Provisioned mode, compute resources are preprovisioned and are billed per hour. This means that the database won't share these resources, but it will always run at the specific size. This is useful for databases that have a consistent demand such as a database that supports a web app that queries multiple times per minute. See Figure 3-35.

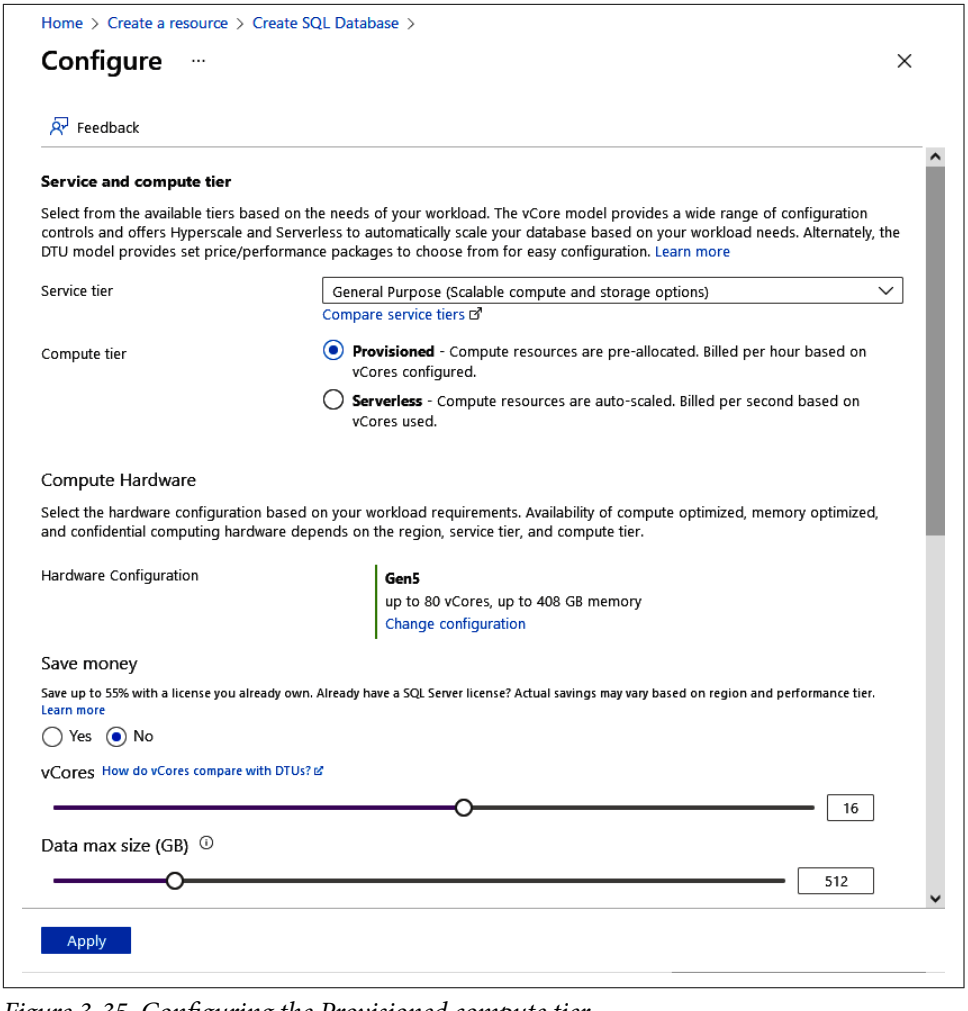


Figure 3-35. Configuring the Provisioned compute tier

In the Serverless mode, minimum and maximum compute resources (vCores) are specified and then autoscaled to meet demand, as shown in Figure 3-36. This tier is billed per second and can be autopaused if it is inactive for a certain amount of time. Serverless is useful for databases that are intermittently used, such as a research database that may be queried a couple times per week.

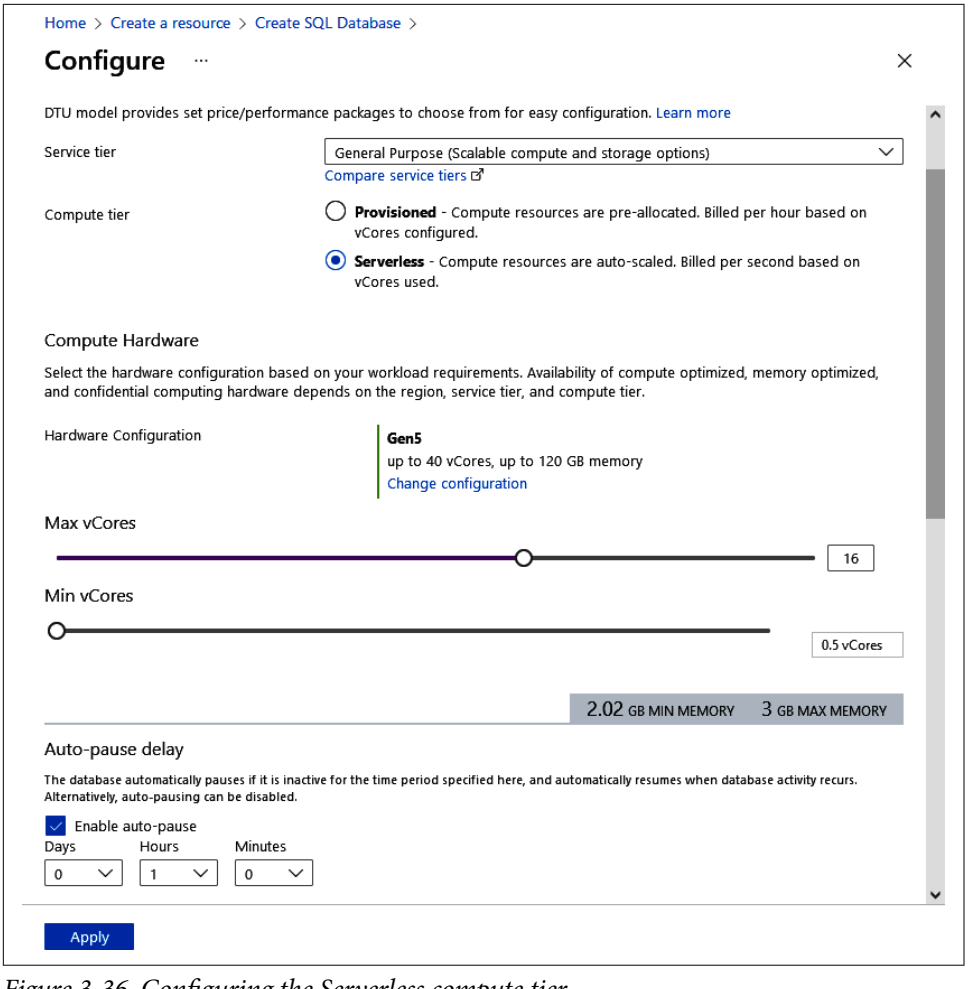


Figure 3-36. Configuring the Serverless compute tier

Alternatively, there are various other service tiers that are available, such as Business Critical, Hyperscale, and more. These other options may be useful to learn about for when very large, high-usage databases are needed in your organizations. You can find these options under the drop-down menu beside "Service tier." See Figure 3-37.

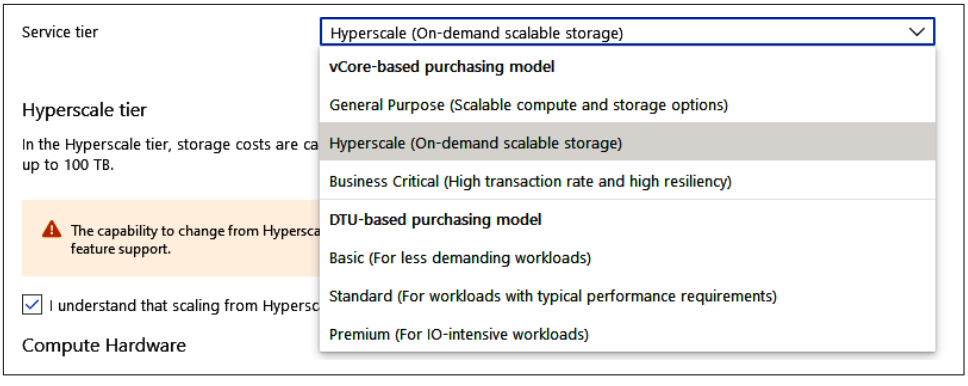


Figure 3-37. Alternative service tiers for SQL database

Once finished configuring any of these "Compute + storage" options, you can click the "Apply" button in the "Configure database" blade; then click "Review + create" and "Create" to create the database, database server, and elastic pool.

You've now created a SQL database, which can be used to house your genomics data, application data, or data for other business functions. Azure SQL Database provides an alternative option to Azure Synapse Analytics when more traditional database architecture is necessary or for when smaller amounts of data are being managed.

Relaxing at Your Genomics Data Lakehouse

If you've arrived at this point of the book and thought, "Wait, do I need a data lake or a data warehouse?" the answer might just be "Yes!"

So far, this chapter has built on Chapter 2 by showing how to create a genomics data warehouse that can connect to and pull from a genomics data lake. While Azure Storage and Azure Synapse are separate services, they often house different viewpoints of the same data. As such, we really should think about them as two parts of the central data repository for genomics.

For most organizations, a data lake serves as the central location for their data. It houses data from source systems in a landing zone, cleaned up data in another zone, and aggregate data in yet another zone. Using a data warehouse *with* your data lake is where the real magic happens. If we can query data from a data lake using a tool like Synapse, we can quickly begin answering questions without doing much data wrangling. Then, for more complex inquiries, we can instantiate our data into tables in a data warehouse and take advantage of the scalable querying capabilities and feed external tools (like Power BI) for reporting and visual analyses.

Databricks, the company behind the Azure Spark-based platform Azure Databricks (which we'll cover in Chapter 4), came up with the notion of a "data lakehouse"—a

blend between an expansive and flexible data lake governed and augmented with a more structured data warehouse. The lakehouse, while a having an odd name, brings a more modern architectural approach that helps to support many data-driven activities from business intelligence to machine learning. See the example architecture of a lakehouse in Figure 3-38.

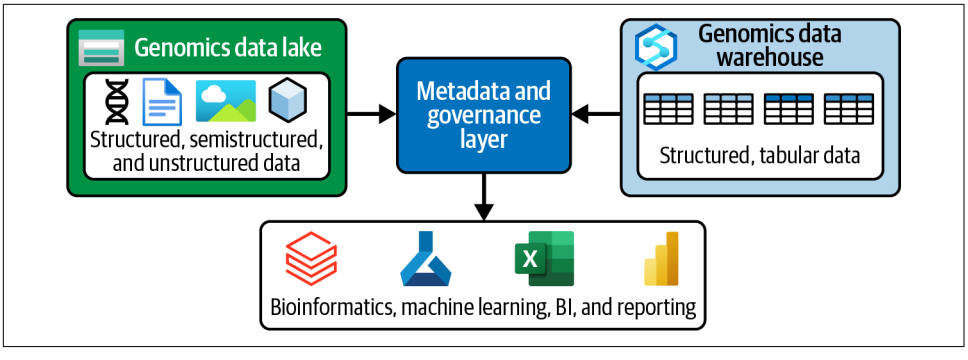


Figure 3-38. Conceptual data lakehouse architecture using Delta

For genomics, I think this concept fits really well with the bioinformatics lifecycle. If implemented correctly, we can balance the flexible housing of esoteric genomics data types with more structured and traceable paradigms to support data management (which will help keep IT happy).

Efficient File Formats

In modern cloud data architectures, there is a need to store large amounts of data in formats that are 1) supportive of columnar data with varying schemas and data types, 2) scalable and flexible to be read and written by different compute environments (single versus distributed computing contexts), and 3) able to be loaded into structured data stores (such as Synapse). Simply reading and writing giant CSV files isn't ideal for scalable data tasks. Thus, there have been some efficient data storage formats that have been created over the years.

Parquet to the floor

If you've ever used Apache Spark, Hadoop, or other distributed compute environments, you may have run across the Parquet file format before. This format is pretty much the default output format when you want to write data from Spark to a storage location (including data lakes). The benefit of this file type is that it is a columnar-based storage format that allows for partitioned data to be written and read efficiently. In distributed compute clusters (as is seen in Azure Databricks or a Synapse serverless Spark pool), when data is read into the compute cluster, it is partitioned into multiple pieces, each of which is stored on the nodes of the cluster. When the

data is written out in Parquet format, it can keep the partitioning scheme and write multiple mini-Parquet files to the storage location. The Parquet logo is shown in Figure 3-39.

Figure 3-39. Apache Parquet logo

For example, a 100GB CSV file could be written out into 10 Parquet files, each with a size of slightly less than 10GB (due to compression). This allows for faster reading of the data as parallel connections can be made to the storage location, each connection pulling a partitioned piece of the overall file. Also, columnar-based storage allows for faster querying as we do not need to read in entire rows if the query includes only a few columns.

ACIDity

The limitation of Parquet files (and all other nondistributed formats like CSV) stored in a data lake is that they lack a set of properties called "ACID" that guarantee data validity. ACIDity is common in database systems where all transactions in the system are traceable. ACID stands for:

Atomicity
: Atomicity refers to the unitary handling of transactions. Even if transactions contain multiple tasks or steps, they must be treated as a single unit and therefore must succeed together or fail together. In other words, atomic transactions cannot partially succeed.

Consistency
: Consistency means that any transaction must keep the data system in a valid state. For example, if data that doesn't meet the schema constraints of a table were to be written, this should cause an error given that the resulting state of the table would be invalid.

Isolation
: When multiple system transactions are performed concurrently, isolation ensures that the resulting state of the system is equivalent to what the state would have been had the transactions been carried out sequentially.

Durability

Durability ensures that once a transaction is complete, it will remain complete even in the event of a catastrophe, such as a server crash or a power outage.

Changing the tides with Delta

These ACID properties were lacking for data lakes until researchers at Databricks created a new standard called "Delta." Delta[3] is a newer file format that builds on what we love about Parquet while adding in ACID transaction logging. The Delta logo is shown in Figure 3-40.

Figure 3-40. Delta logo

Parquet is still a common file format used in data lakes today, but we now see the Delta format gaining popularity for its ACID capabilities. In fact, Synapse, Data Factory, and other Azure tools have native support for these formats.

We'll cover more about Delta in Chapter 4. For now, I wanted to get you thinking about data stability and how to ensure that your business-critical or research-critical genomics data stays valid and transactions on it stay traceable and reproducible.

In this chapter, we've discussed how to blend data warehousing techniques while changing things up to support genomics data. While a data lake serves as your central storage location for data, a data warehouse, like the one we provisioned in Azure Synapse Analytics, provides an interface for querying this vast amount of data at scale.

In summary, data warehouses are useful for storing large amounts of data from various sources and performing complex queries to answer questions at scale. This differs from a database where the usage is built around the storage and rapid retrieval of data for a given application. However, in Azure you may use Azure SQL DB for a small data warehouse if Synapse isn't needed (due to costs or complexity reasons). There are two main schema styles: star and snowflake. Star schemas are faster/easier to query since the dimensions don't relate to each other. Snowflake schemas, though, are quite common when complex relationships have to be captured, and thus you may encounter this schema style often in practice.

3 Michael Armbrust et al., "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores," *Proceedings of the VLDB Endowment* 13, no. 12 (August 2020): 3411–3424, doi: *https://doi.org/10.14778/3415478.3415560*.

Azure Synapse Analytics, Microsoft's cloud data warehousing offering, is a powerful tool for building a data warehouse connected to other Azure services (like your genomics data lake). By using external tables, we can employ Synapse to read from our data lake and query the data without having to actually load it all into real tables. Lastly, for variant data, we can set up a performant data warehousing schema to enable fast insights into our samples. This allows us to answer complex questions by querying this data, which is useful when we have millions of variants across tons of samples, say, from a clinical trial.

In the upcoming chapter, we'll cover how to orchestrate data movement into your data lake from various source systems and how to transform and load data into structured data stores, such as your genomics data warehouse.

CHAPTER 4

Orchestrating Data Movement and Transformation

When you work in a data-driven role (e.g., as a bioinformatician, data scientist, data analyst, etc.), having data available to you is paramount to being successful. With a cloud environment, a common hurdle that organizations have when they first start out is actually getting data into the cloud for everyone to use. Data orchestration encompasses the processes for getting data into and out of cloud resources and also managing certain data tasks. In this chapter, we'll learn how to orchestrate data movement in the cloud by connecting to sources (often outside our cloud environment) and copying data over to our data lake or other destination.

In Azure, the standard tool for data orchestration is called Data Factory. This tool blends the capabilities of a traditional extract-transform-load (ETL) tool, like SQL Server Integration Services, with orchestration capabilities for queuing up external data tasks. With Data Factory, we'll learn how to extract data from outside our Azure environment into our data lake and also learn how to transform and load that data into our data warehouse. See Figure 4-1.

Outside of Azure, there are certainly other third-party ETL tools available for you to purchase, but it's worth giving Data Factory a shot as it is very well-integrated with the other Azure services that we cover in this book. Plus, Data Factory supports more than 90 built-in connectors to common enterprise platforms like SAP, Snowflake, AWS S3 Buckets, and more. This allows you to start moving data into your Azure-based databases, data warehouses, and data lakes very quickly. In addition to other cloud sources, there are connectors for standard protocols like SFTP and HTTP, which makes pulling data from public -omics repositories straightforward. So, pulling large reference genomes or extra public samples to use in your research can now be

performed in the cloud without you having to download and re-upload the data from your local workstation.

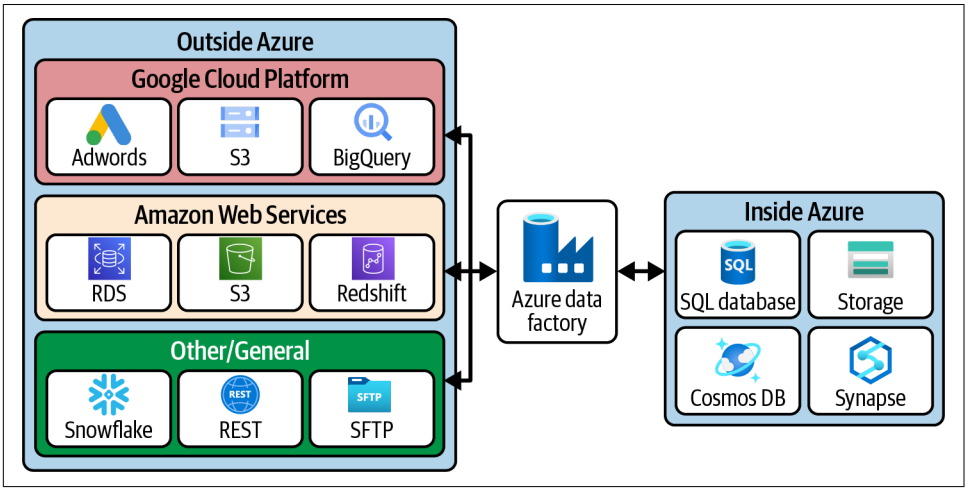


Figure 4-1. Some example connectors included in Data Factory

While there are bioinformatics pipeline tools like NextFlow and Snakemake that perform analytical tasks, this chapter will focus on data-centric tasks such as copying data up to the cloud, moving it around, and performing some basic data transformations in Data Factory.

Coming up, we'll provision our Data Factory service and start building a pipeline to move data around.

Creating Your Data Factory

If you created a Synapse Analytics Workspace in Chapter 3, you can skip to the section "Getting Started with Data Movement," as Synapse comes with a Data Factory instance already available from its UI. Otherwise, we can create a standalone Data Factory service in just a few steps from the Azure Portal. The Synapse-housed Data Factory and the standalone version work almost identically, so the choice is yours. I find the Synapse-housed version a little easier to start with as it comes preconfigured with the credentials to connect to the data that's in the data warehouse. (In the standalone version, there's just an extra step to set up this access.)

Navigate to the main page of the Azure Portal and click "+ Create a resource." Search for "Data Factory," which should automatically bring up the page to create the Data Factory resource. See Figure 4-2.

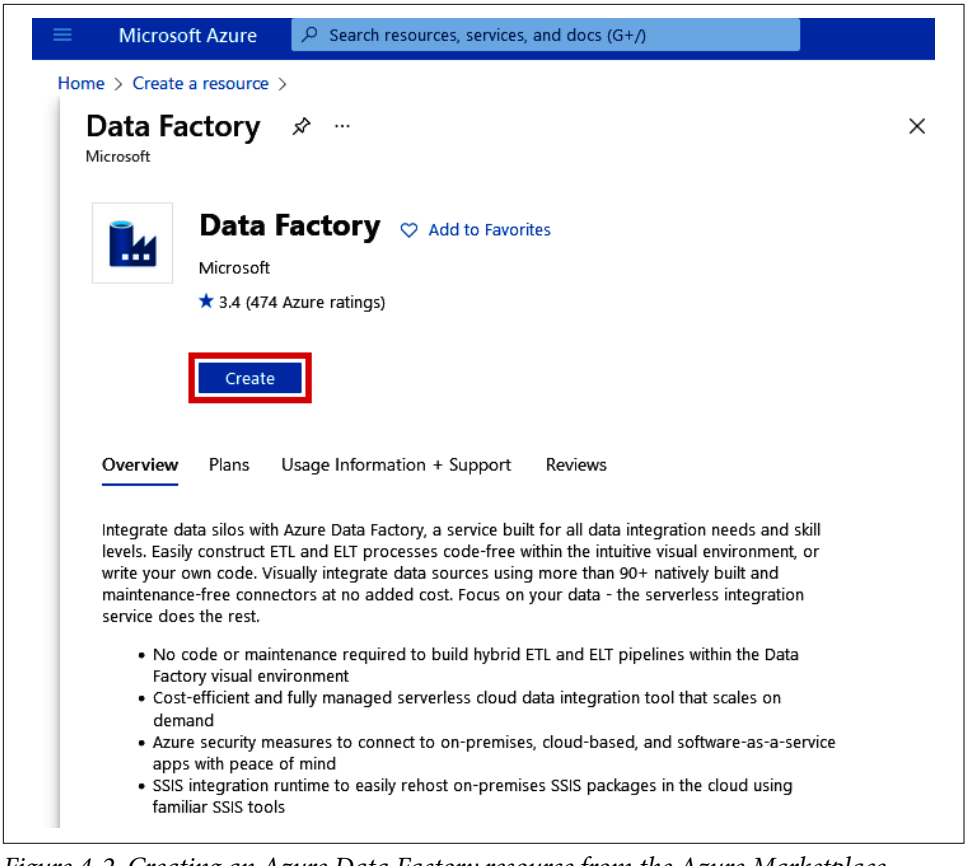


Figure 4-2. Creating an Azure Data Factory resource from the Azure Marketplace

Click the "Create" button. This will take you to a Create Data Factory form, which should look similar to the other setup processes we've gone through.

For Data Factory, there isn't much to configure. Pick the resource group that you've been using (e.g., *genomics-dev-rg*), pick the region that houses all your other services, and then give your Data Factory a name (such as *genomics-df*). But wait: don't click "Review + create" yet. We have to go to the "Git configuration" tab next, as shown in Figure 4-3.

Data Factory integrates well with Azure DevOps and GitHub Git repositories. When you write SQL scripts to create tables or query data, these can be automatically committed to your Git repository when you click the "Publish" button (which you'll see in a bit). This is also helpful for saving any data pipelines that you create inside Data Factory. Since this configuration can be updated later, let's click the checkbox beside "Configure Git later" (as shown in Figure 4-4) and just proceed without it for now.

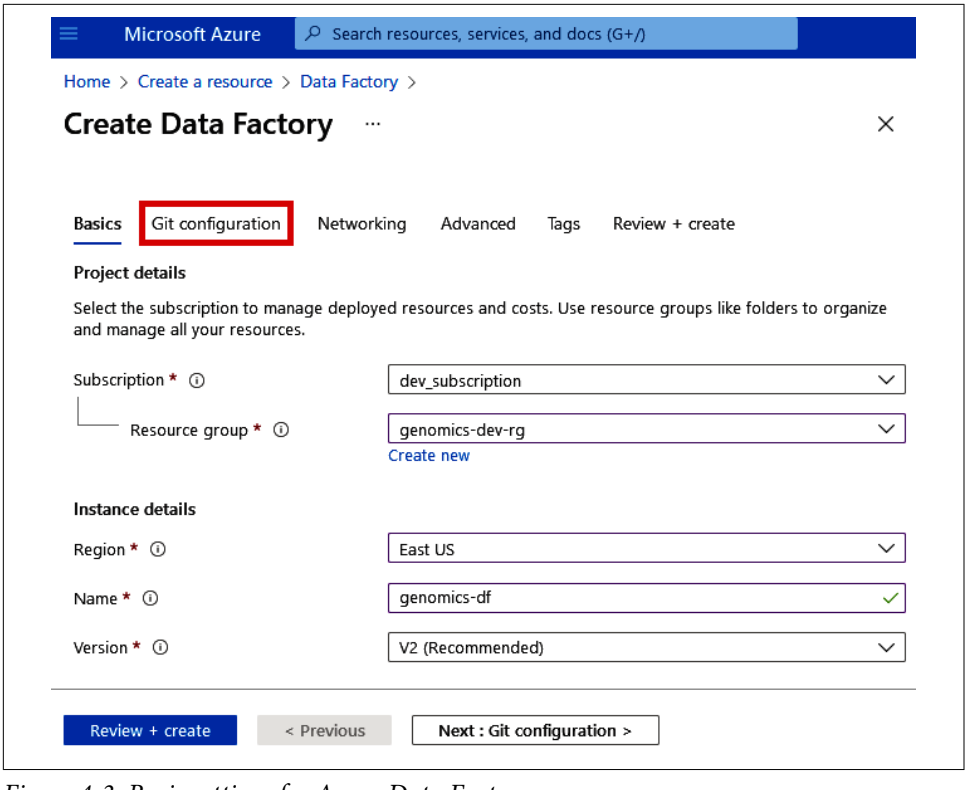


Figure 4-3. Basic settings for Azure Data Factory

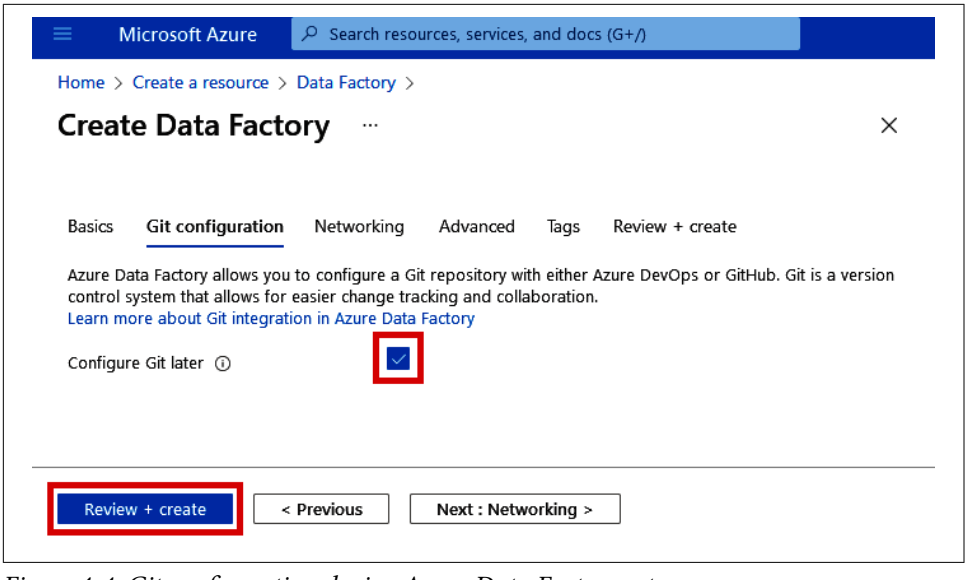


Figure 4-4. Git configuration during Azure Data Factory setup

Now you can click the "Review + create" button. Once Azure performs its validation check, you can then click "Create" (as shown in Figure 4-5), which will provision your Data Factory in just a few seconds.

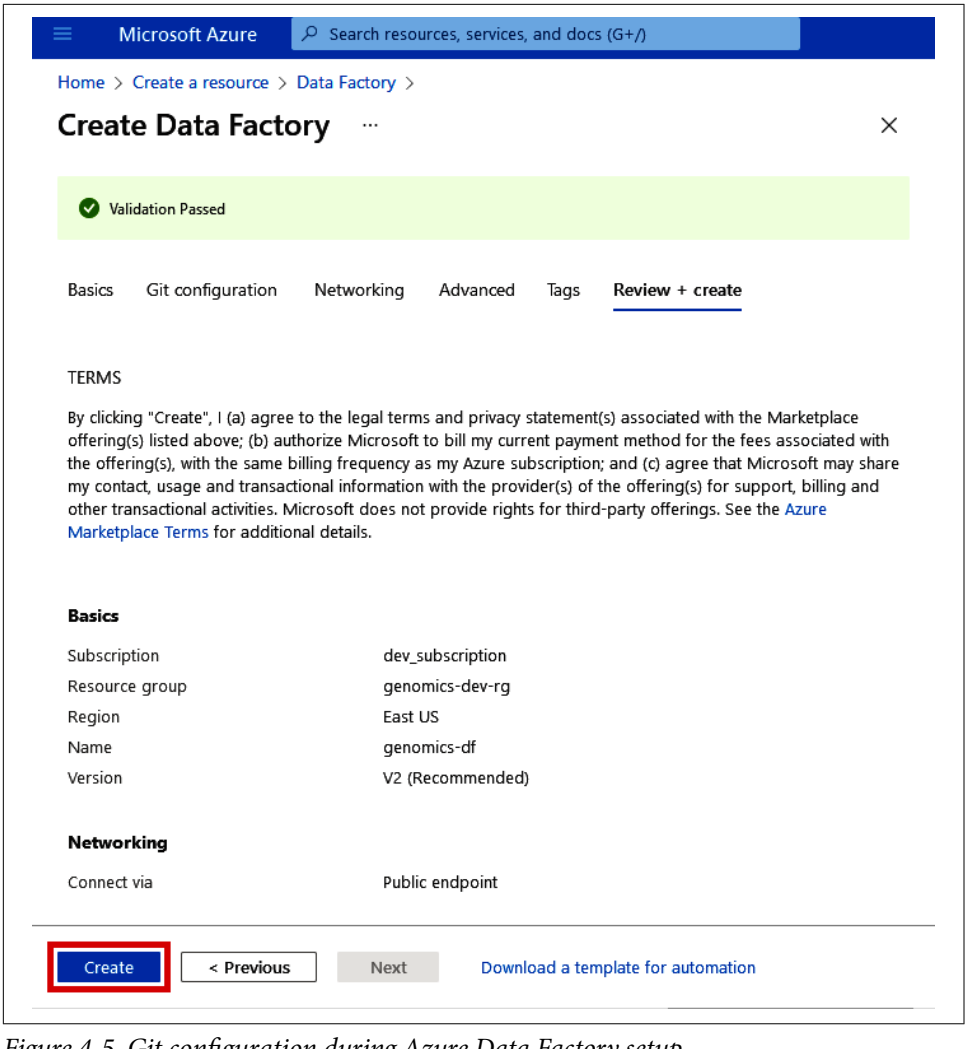


Figure 4-5. Git configuration during Azure Data Factory setup

Now that we have Data Factory up and running, let's create our first data orchestration task by copying data into our cloud environment.

Getting Started with Data Movement

In the following sections, we'll be walking through how data orchestration works within Data Factory to copy data, move it around, and even transform it. We'll also be able to tell other services when to perform actions on our data.

Getting Data into Your Data Lake Using the Copy Data Tool

The first step to doing anything in the cloud is moving some data up to Azure. One option is to take the manual route, which is to download data to your local computer and then upload it to our data lake. This may be fine for a small number of tiny files, but what happens if we have a lot of data we want to copy over?

Downloading and uploading gigabytes (or even terabytes) of data manually is a slow and futile exercise. Azure Data Factory has the Copy Data tool, which I find really useful for this exact task.

From the standalone Data Factory, you can use the Copy Data tool right from the main screen. Click the "Ingest" button, as shown in Figure 4-6.

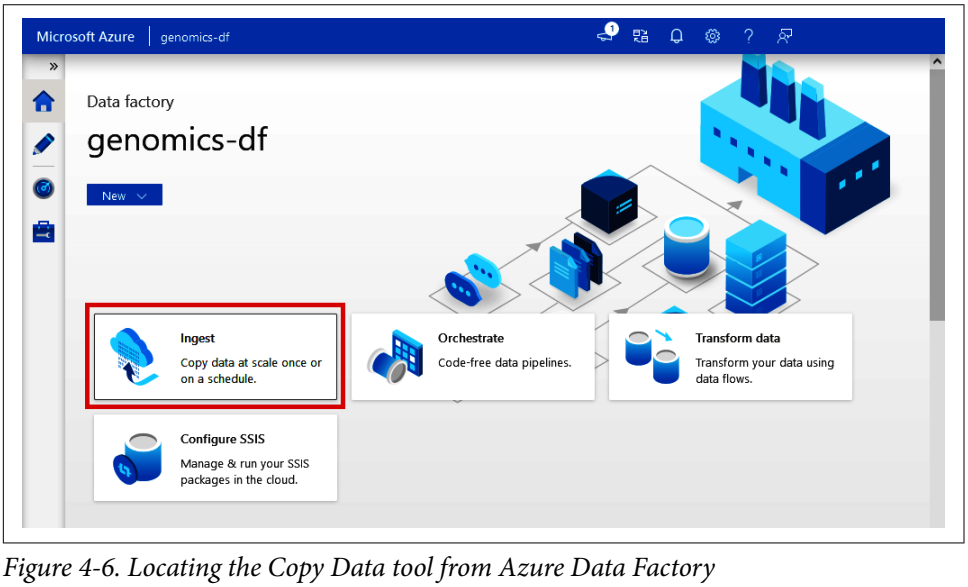


Figure 4-6. Locating the Copy Data tool from Azure Data Factory

If you're using Synapse, you'll find the Copy Data tool under the "Integrate" tab. As shown in Figure 4-7, click the "+" at the top and select "Copy Data tool."

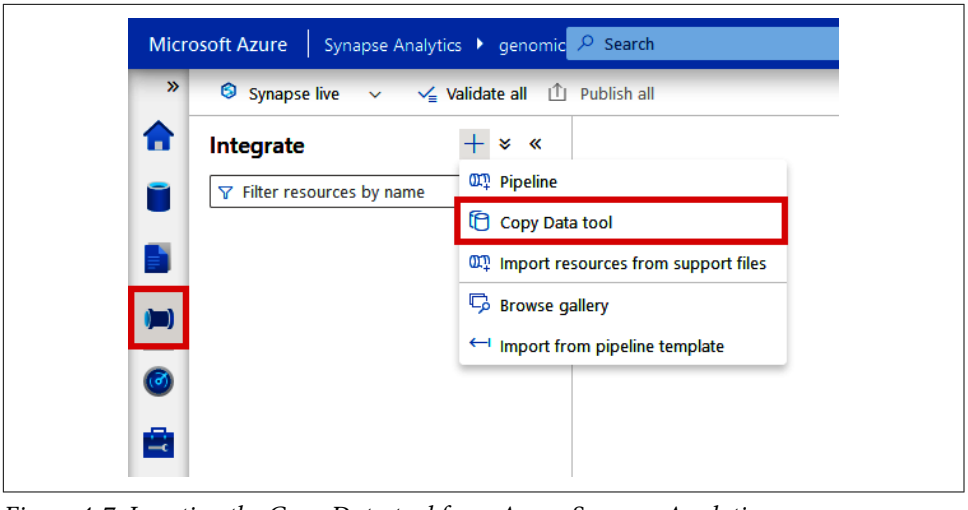


Figure 4-7. Locating the Copy Data tool from Azure Synapse Analytics

The next screen is the same whether you're using the standalone version of Data Factory or the version built into Synapse. Click "Built-in copy task" and then click the "Next" button. See Figure 4-8.

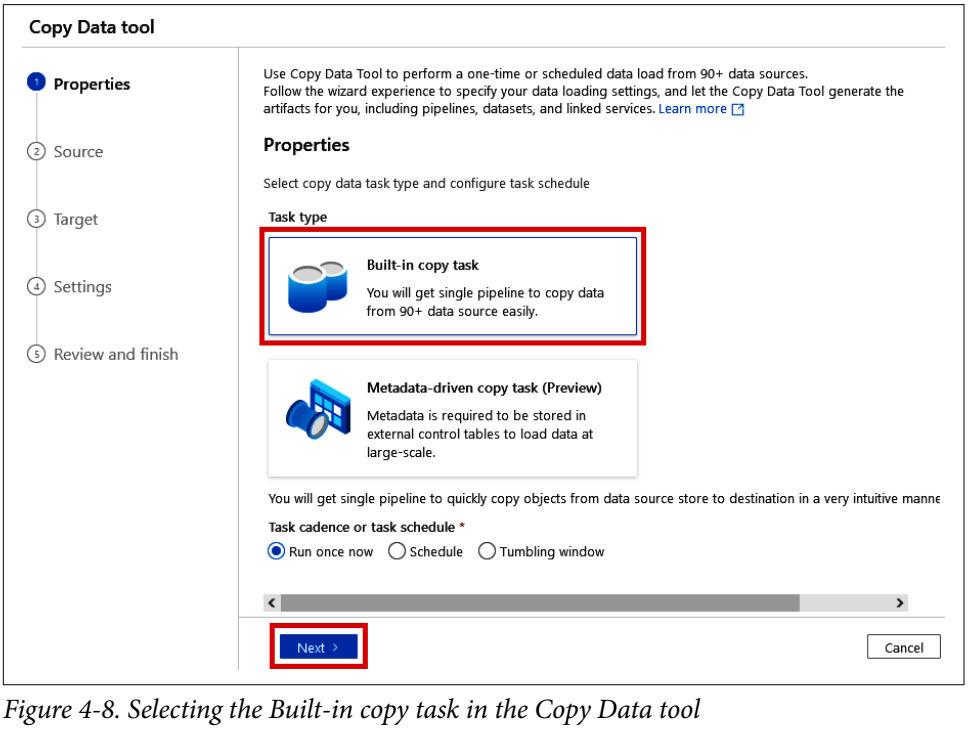


Figure 4-8. Selecting the Built-in copy task in the Copy Data tool

Next, let's grab some data from a public database.

Linking to NCBI's FTP Server

In research, we often rely on public data repositories to provide reference genomes, public sequences and samples, additional metadata or ancillary information, and more. Thus, it's useful to have the ability to pull this data into our Azure environment in an automated and scalable way.

Many public data repositories like National Center for Biotechnology Information (NCBI) provide access to their data through FTP (file transfer protocol). In Data Factory, we can directly connect to FTP sites and copy data right from Data Factory to our Data Lake. In this example, we'll grab a sample VCF file from NCBI's FTP server and copy it to our data lake, which we'll use later.

On the next step of the Copy Data tool ("2. Source"), set the "Source type" to "FTP" (note how many other options there are in this drop-down list) and click "+ New connection" beside the "Connection" drop-down. See Figure 4-9.

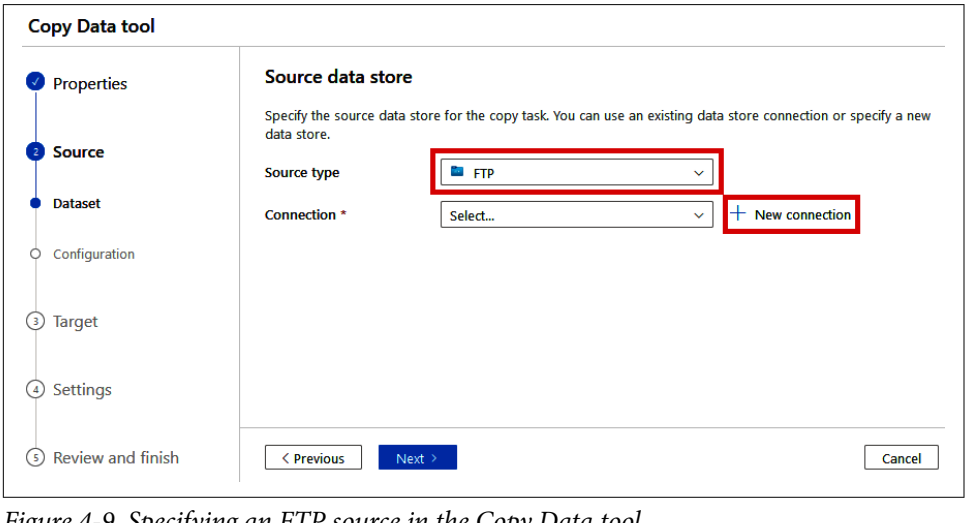


Figure 4-9. Specifying an FTP source in the Copy Data tool

In the "New connection" blade that pops up, give this connection a name (such as "NCBI_FTP"). In the "Host" box, paste in the following URL:

```
ftp://ftp.ncbi.nlm.nih.gov
```

As shown in Figure 4-10, disable SSL under "Secure Transmission" and set the "Authentication type" to "Anonymous." (Since this is a public site, we don't have to log in.)

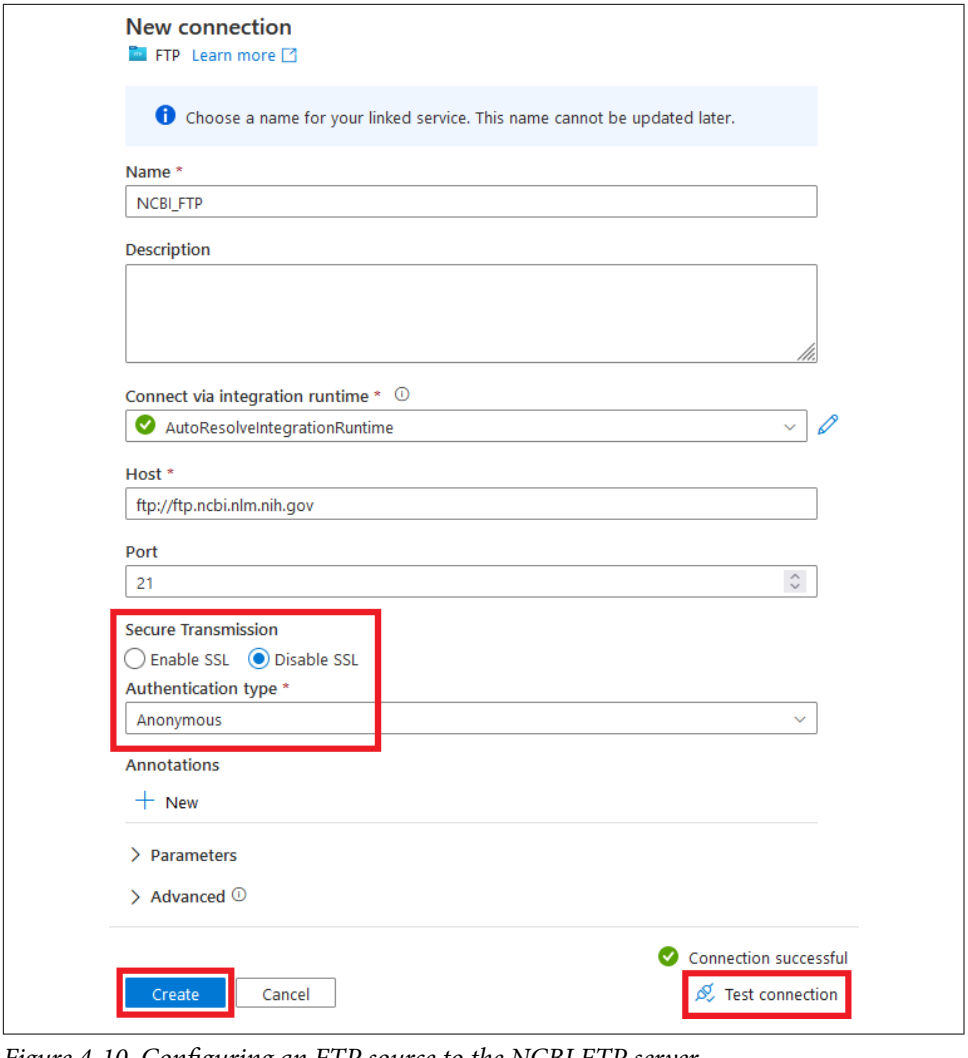


Figure 4-10. Configuring an FTP source to the NCBI FTP server

Once you have these properties filled out, click the "Test connection" button at the bottom of the screen. If you get a green check mark, click the "Create" button to create the connection to the NCBI FTP server.

On the next screen, we'll provide the information about the specific file that we want to copy. In the "File or folder" box, you can click the "Browse" button to explore the FTP site and find the file(s) that you're wanting to copy. For now, simply copy the following path into the box:

```
toolbox/gbench/samples/vcf/GCA_000001215.4_current_ids.vcf.gz
```

Since this is a compressed VCF file (about 60MB), we can use the binary copy option to avoid having Data Factory check the schema of the file being copied. See Figure 4-11 for the other recommended selections for the "Options" section.

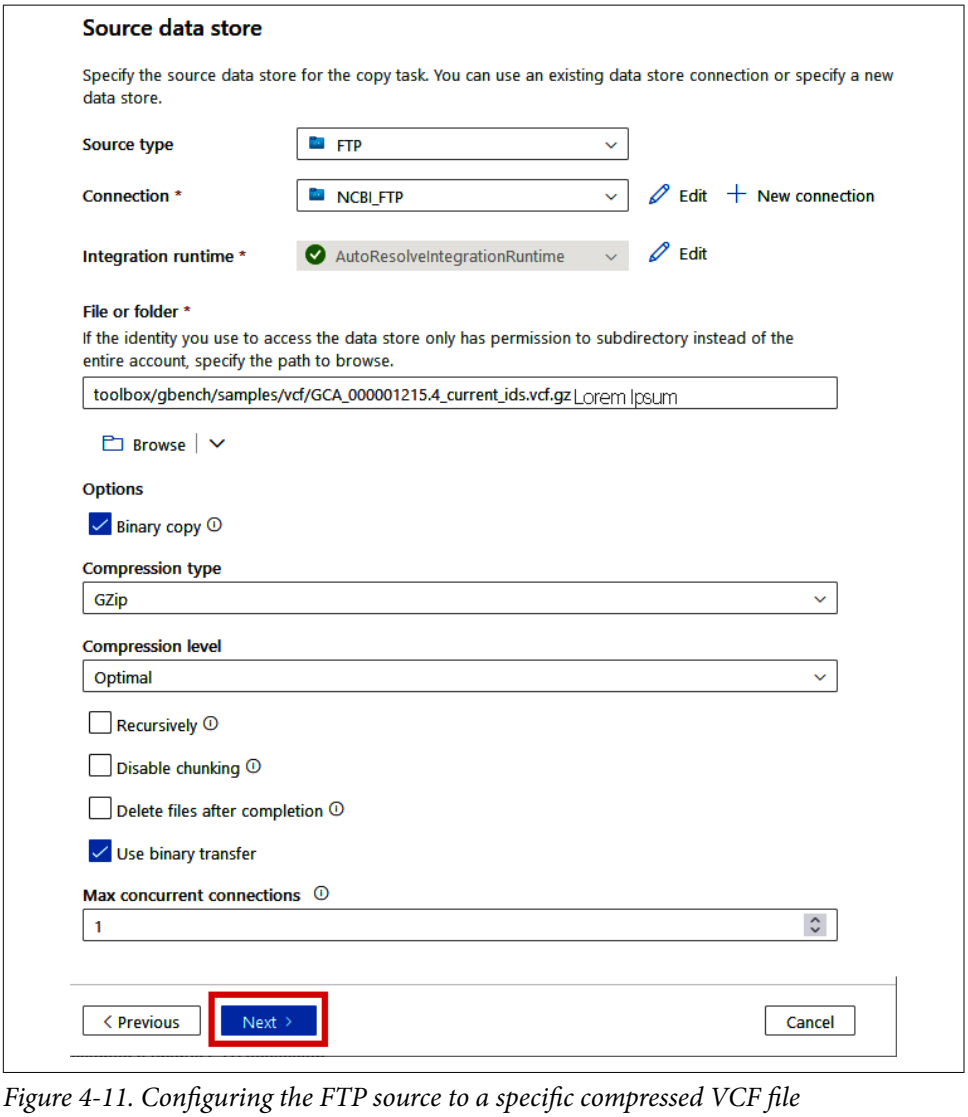


Figure 4-11. Configuring the FTP source to a specific compressed VCF file

Once you've filled in this information, click the "Next >" button.

On the "Destination data store" screen ("3. Target"), you'll now specify the location where this VCF file should be copied. We'll be copying this file to our data lake that we created in Chapter 2, so select "Azure Data Lake Storage Gen2" for the "Target type." If you're using the Copy Data tool from Synapse, you'll find your data lake in the "Connection" drop-down (from where we specified this connection during Synapse's setup process). If you're using the Copy Data tool in the standalone version of Data Factory, we haven't yet made this connection, so you'll need to click "+ New Connection," as shown in Figure 4-12.

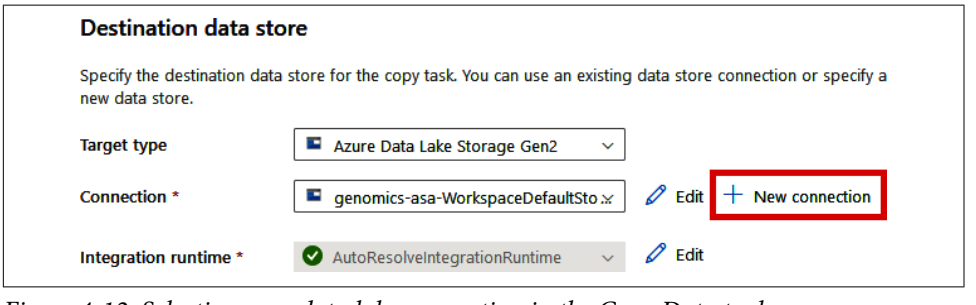


Figure 4-12. Selecting your data lake connection in the Copy Data tool

Skip this step if you're using Synapse or if you already see your data lake in the "Connection" drop-down.

Linking up your data lake with Data Factory is simple since both resources exist in the same Azure tenant. Simply give your connection a name (such as "GenomicsDataLake") and then find your data lake under the "Storage account name" drop-down menu (after selecting the right subscription), as shown in Figure 4-13.

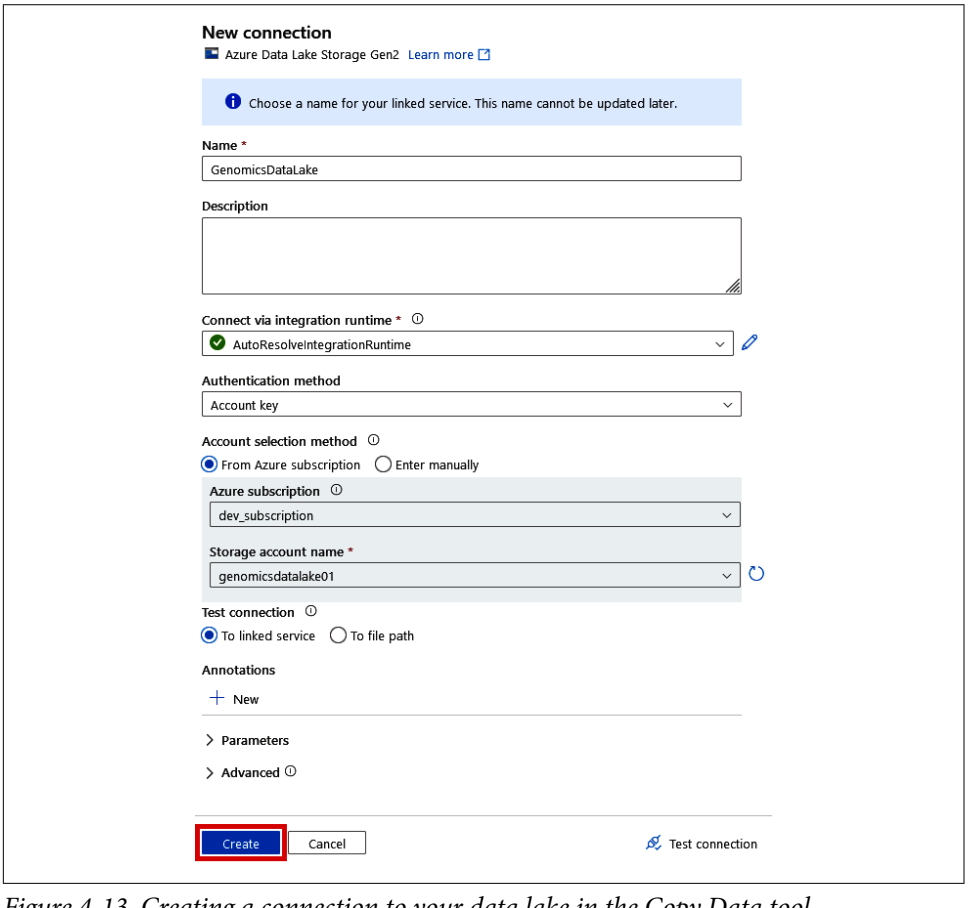


Figure 4-13. Creating a connection to your data lake in the Copy Data tool

This will automatically load the access keys for you so that you don't have to go find them in the data lake. Click "Create" once it finds your genomics data lake.

Once you have your data lake selected in the Connection drop-down, proceed to fill out the remaining boxes in the "Destination data store" blade. For the "Folder path," we can provide a path to a directory that doesn't yet exist (such as "data/ncbidata"), and Data Factory will create it during the copying process. See Figure 4-14 for the rest of the recommended options. Once you've filled everything out, click the "Next" button.

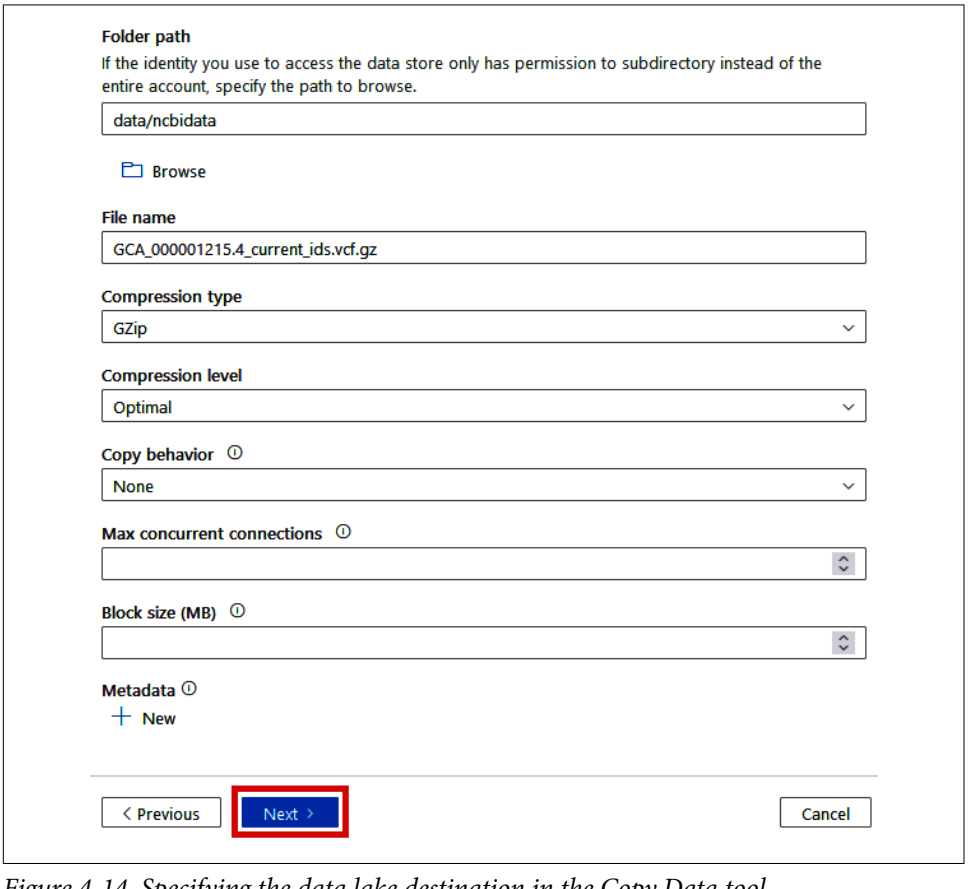


Figure 4-14. Specifying the data lake destination in the Copy Data tool

Then, on the “4. Settings” step, give your pipeline a name (such as “CopyNCBIVCF-Pipeline”) and click the “Next” button, as shown in Figure 4-15.

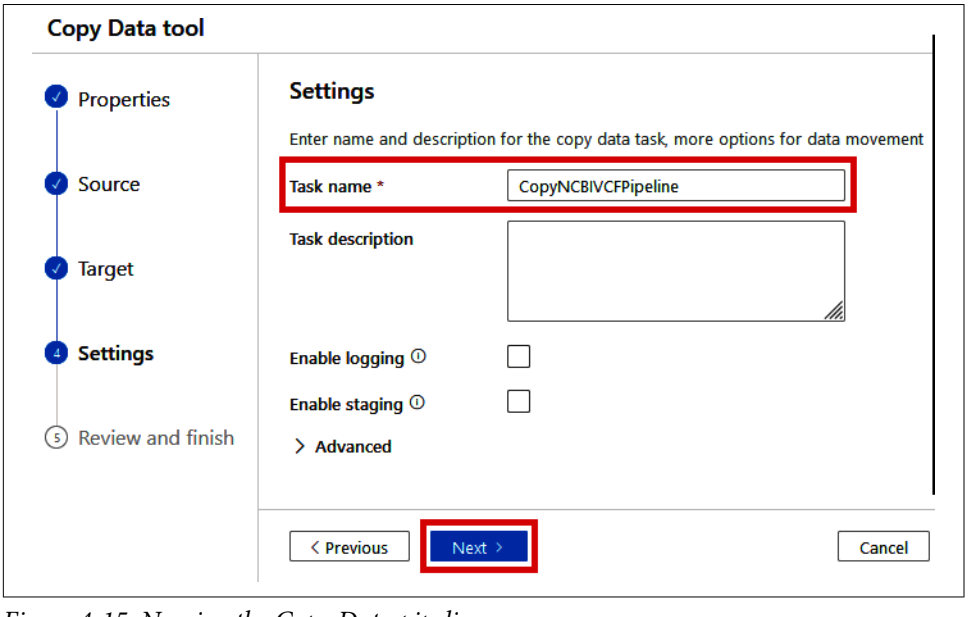


Figure 4-15. Naming the Copy Data pipeline

On the final "5. Review and finish" step, the Summary page will give you some information about the Copy Data pipeline that is to be executed. See Figure 4-16.

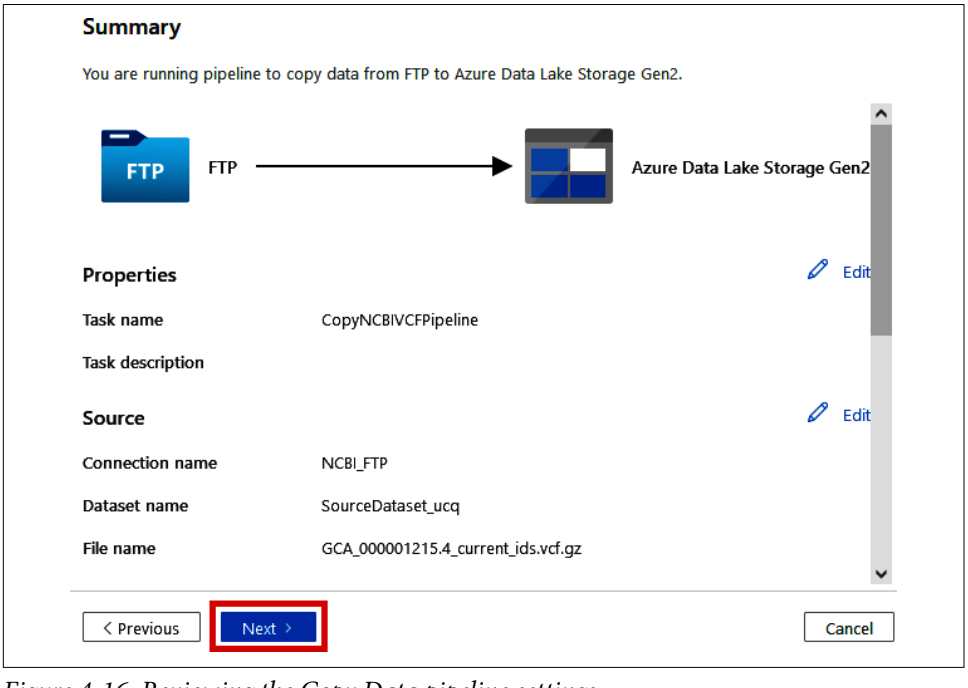


Figure 4-16. Reviewing the Copy Data pipeline settings

After reviewing the information, you can click the "Next" button. This will immediately kick off the pipeline and start copying this file from NCBI's FTP server to your data lake. Once you see the green check marks, which indicate that the steps have succeeded, your VCF file is now in your data lake. See Figure 4-17.

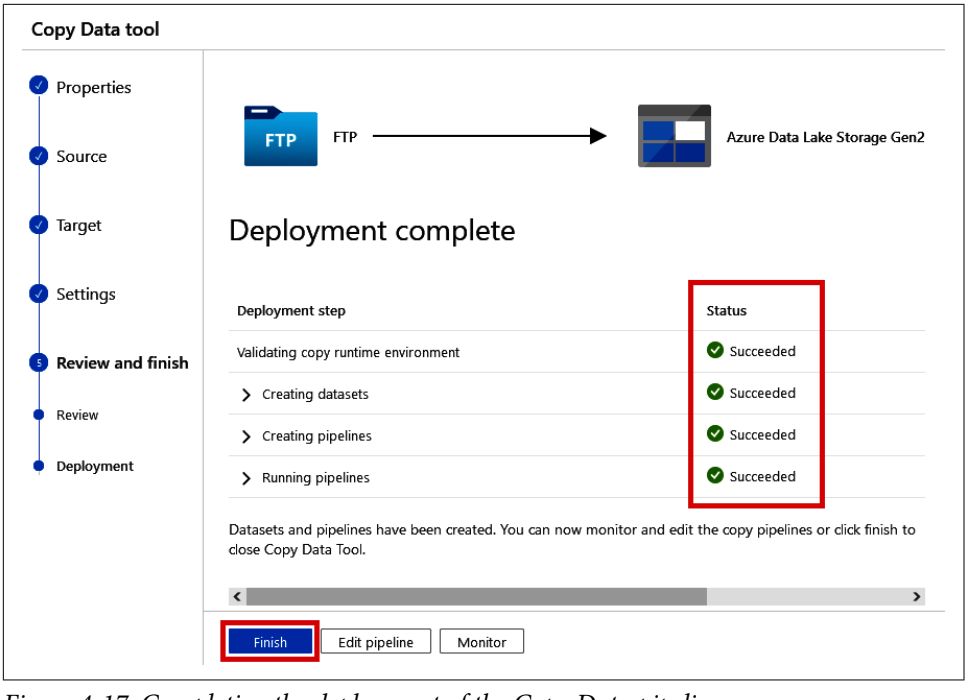


Figure 4-17. Completing the deployment of the Copy Data pipeline

Click the "Finish" button, as shown in Figure 4-17. Now you can navigate to your data lake to see if the VCF file is there.

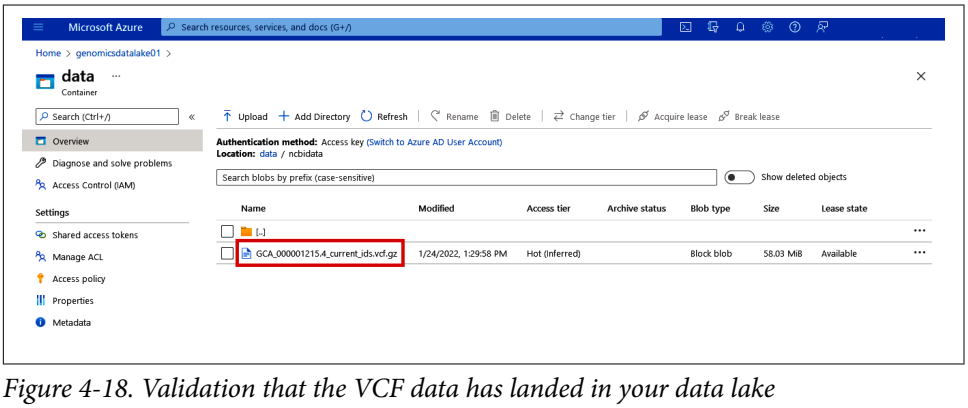


Figure 4-18. Validation that the VCF data has landed in your data lake

Now your data lake is no longer empty! See Figure 4-18. Repeat this process if you have more files in other systems that you want to copy over. In the next example, we'll walk through the creation of more complex pipelines to actually transform data.

Transforming Data Using Data Flows

Sometimes, we may want to make some basic transformations on our data before loading it into our destination system. We can do this in Data Factory with Data Flows. While pipelines can move or copy data from place to place, we can use a Data Flow as a step in a pipeline to perform more complex transformations.

Parsing a VCF file with a data flow

Since we now have a VCF file in our data lake, let's parse it using a Data Flow and then convert it to a more universally friendly format for cloud services like Synapse or Databricks.

If you're using the Data Factory included in Synapse, navigate to the "Integrate" tab and click the + button at the top. Then, click "Pipeline." See Figure 4-19.

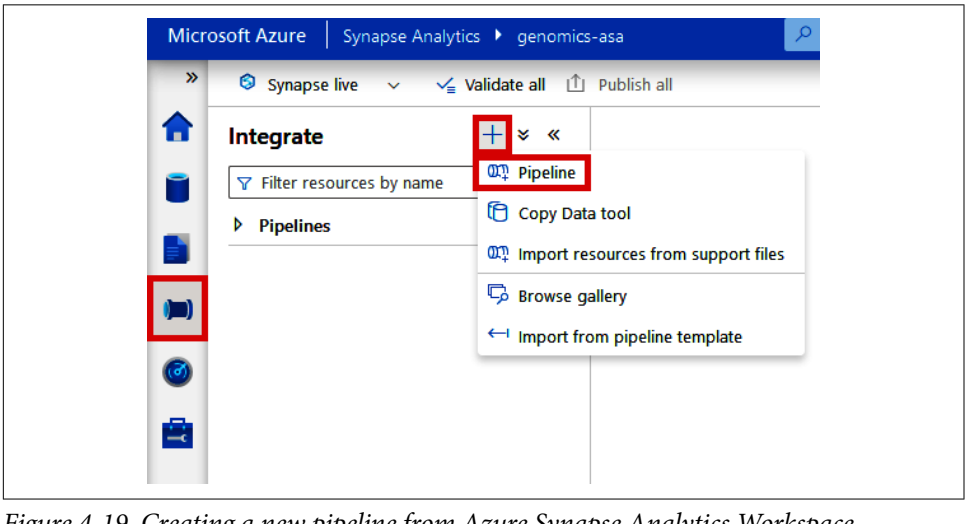


Figure 4-19. Creating a new pipeline from Azure Synapse Analytics Workspace

If you're using a standalone version of Data Factory, navigate to the Author tab (pencil icon) and click the + button at the top. Then click "Pipeline" and "Pipeline." See Figure 4-20.

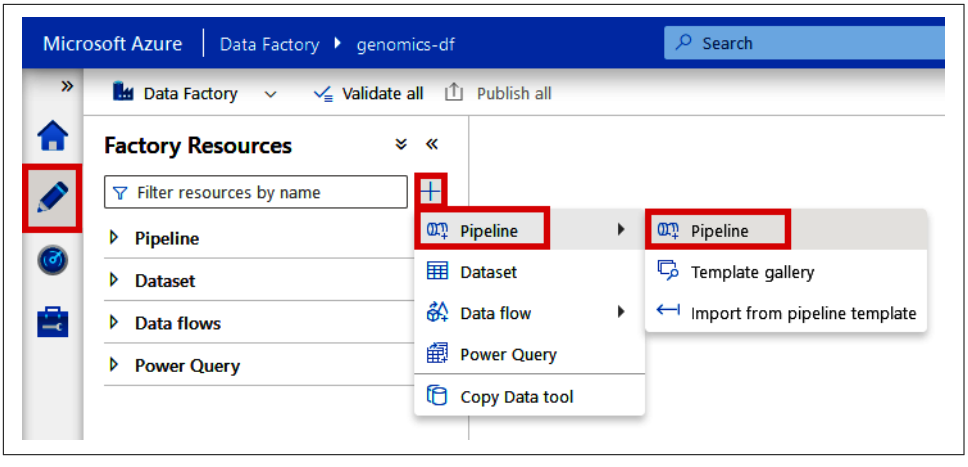


Figure 4-20. Creating a new pipeline from Azure Data Factory

This will open a new Pipeline canvas, where we can use the drag-and-drop interface to build out the steps of our pipeline. Under "Move & transform," locate the "Data flow" module and drag it onto the canvas. Then click the "Data flow" module that is now on your canvas and locate the "Settings" tab at the bottom of the screen. Click the "+ New" button beside the "Data flow" drop-down menu, as shown in Figure 4-21.

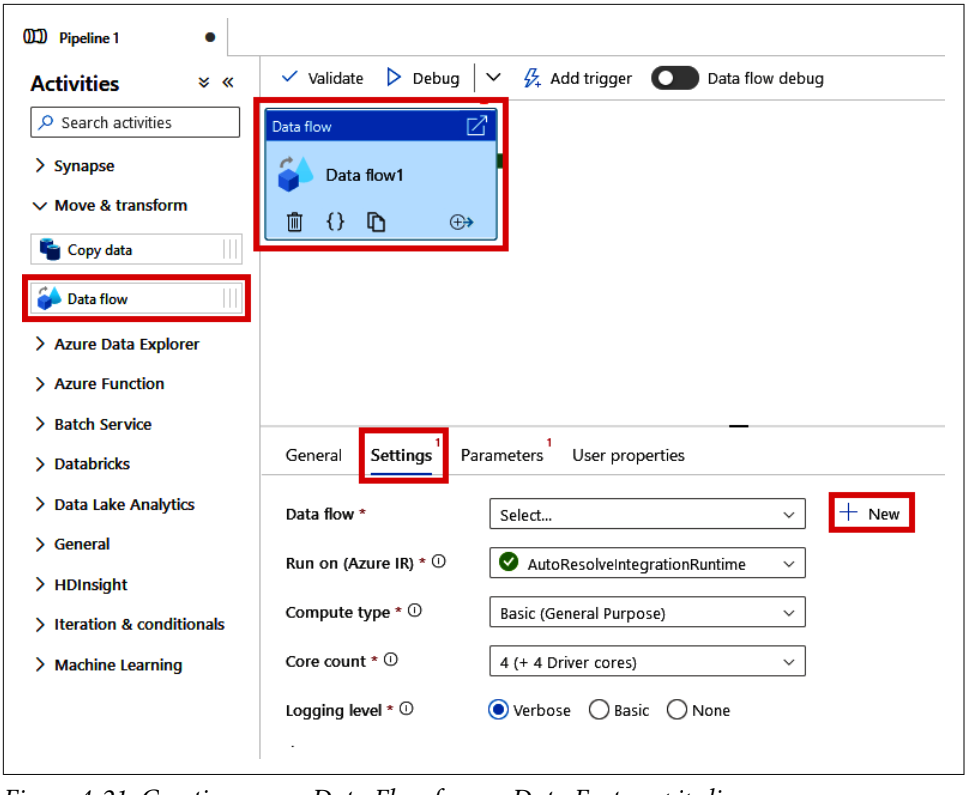


Figure 4-21. Creating a new Data Flow from a Data Factory pipeline

This will open a new canvas tab for the Data Flow, where we can define our data source, the transformation steps, and the "sink" or the destination where we want the transformed data to land.

To start building the Data Flow, click the "Add Source" box (shown in Figure 4-22). This will create a generic source step in the flow.

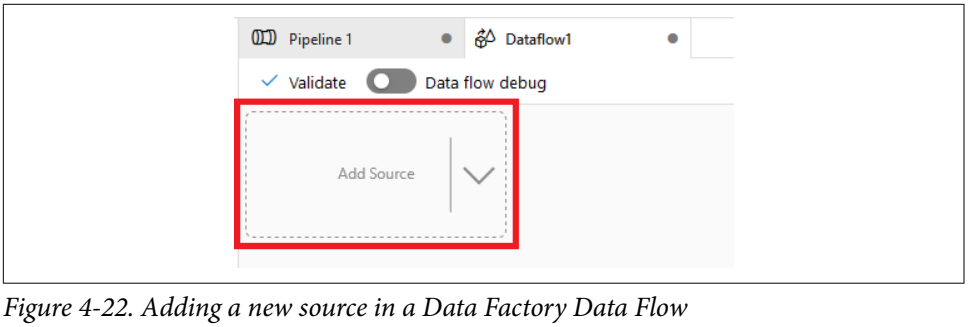


Figure 4-22. Adding a new source in a Data Factory Data Flow

Rename the "Output stream name" to "VCFSource" and click the "+ New" button next to the Dataset drop-down menu. See Figure 4-23.

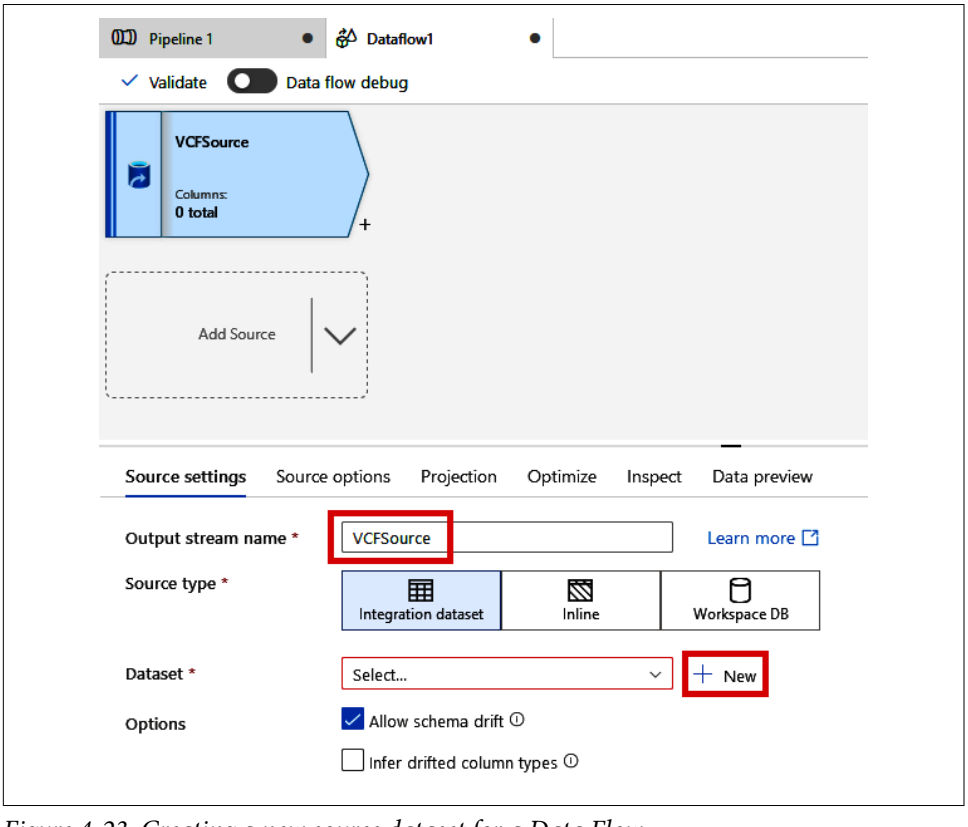


Figure 4-23. Creating a new source dataset for a Data Flow

In the "New integration dataset" windows that pop up, as shown in Figure 4-24, click "Azure Data Lake Storage Gen2" and click the "Continue" button.

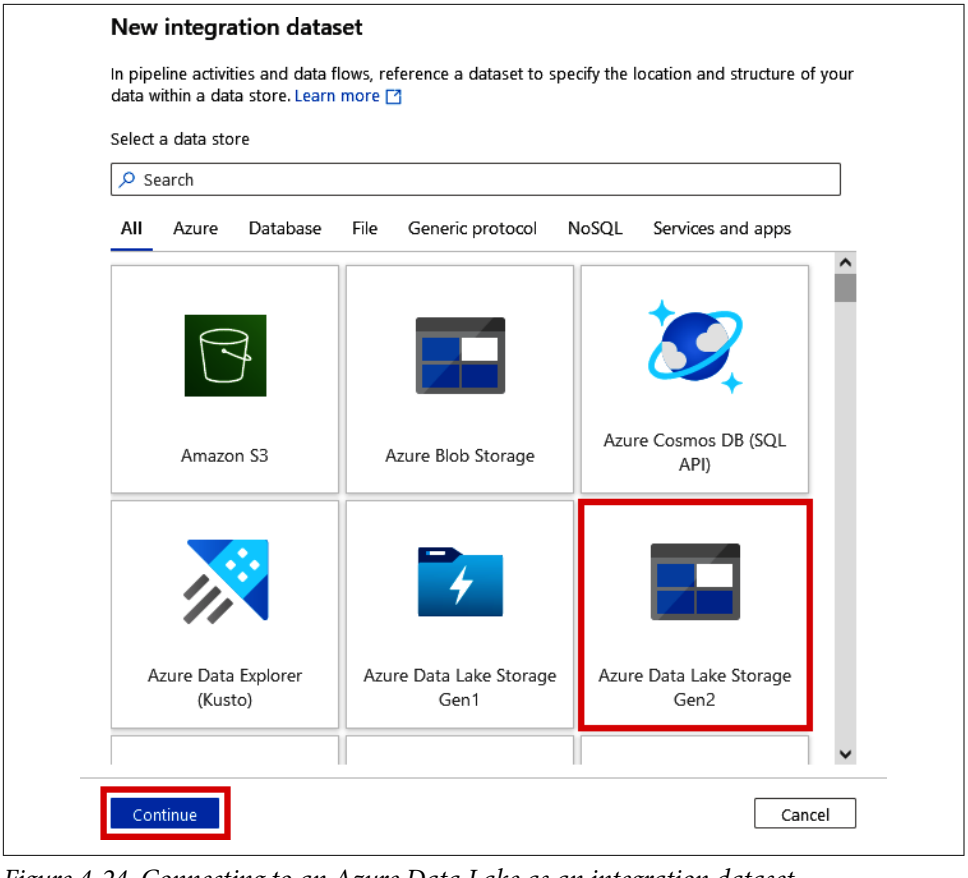


Figure 4-24. Connecting to an Azure Data Lake as an integration dataset

Next, specify that the type of file we want to interact with is a "DelimitedText" format and click the "Continue" button. (There isn't a VCF format type in Data Factory, so we'll need to make do with the more general delimited text format.) See Figure 4-25.

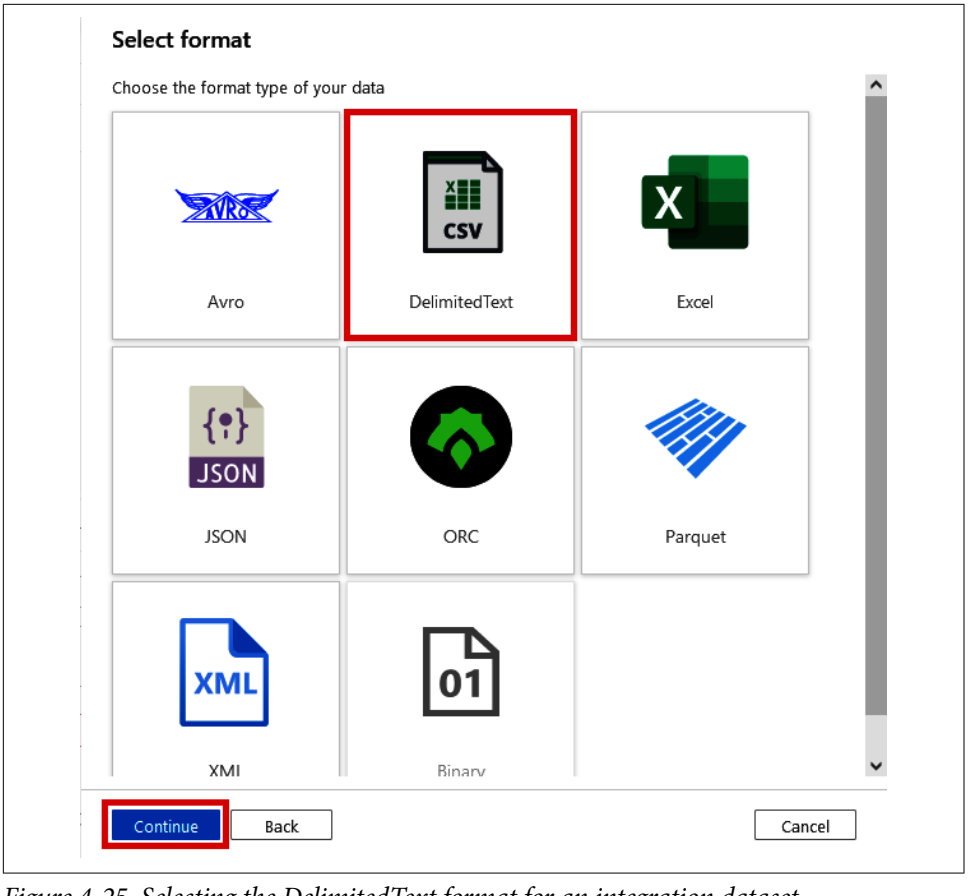


Figure 4-25. Selecting the DelimitedText format for an integration dataset

On the "Set properties" screen, shown in Figure 4-26, name this dataset "NCBIVCF-File" (or whatever you like) and select your data lake from the "Linked service" drop-down menu. This linked service will be either the "GenomicsDataLake" that we set up in Figure 4-13 (if you're using a standalone Data Factory) or the workspace default storage "genomics-asa-WorkspaceDefaultStorage" (if you're using a Data Factory in Synapse).

Also, specify the file path either by typing in the path to your VCF file in the format *<container>/<folder>/<filename>* or by clicking the folder button to browse the data lake and selecting the file. Lastly, click the "OK" button.

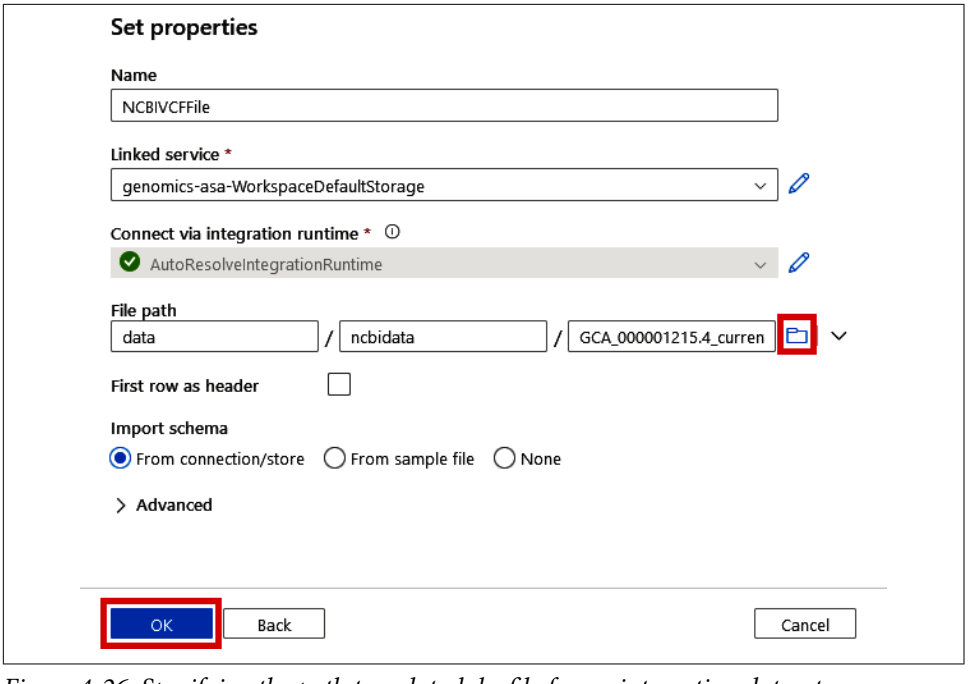


Figure 4-26. Specifying the path to a data lake file for an integration dataset

This will create the link to the VCF file. Now we need to check to see if Data Factory can actually read the file.

Back on the main Data Flow canvas, shown in Figure 4-27, turn on the "Data flow debug" toggle and click the "OK" button on the "Turn on data flow debug" window. This will give us a temporary compute context where we can test our Data Flow. While that's spinning up, we need to specify a few things about our VCF file parsing.

Change the "Skip line count" to 17 (this is determined by looking at the VCF file to see how many ## header lines there are). Next, click the "Open" button beside "Dataset."

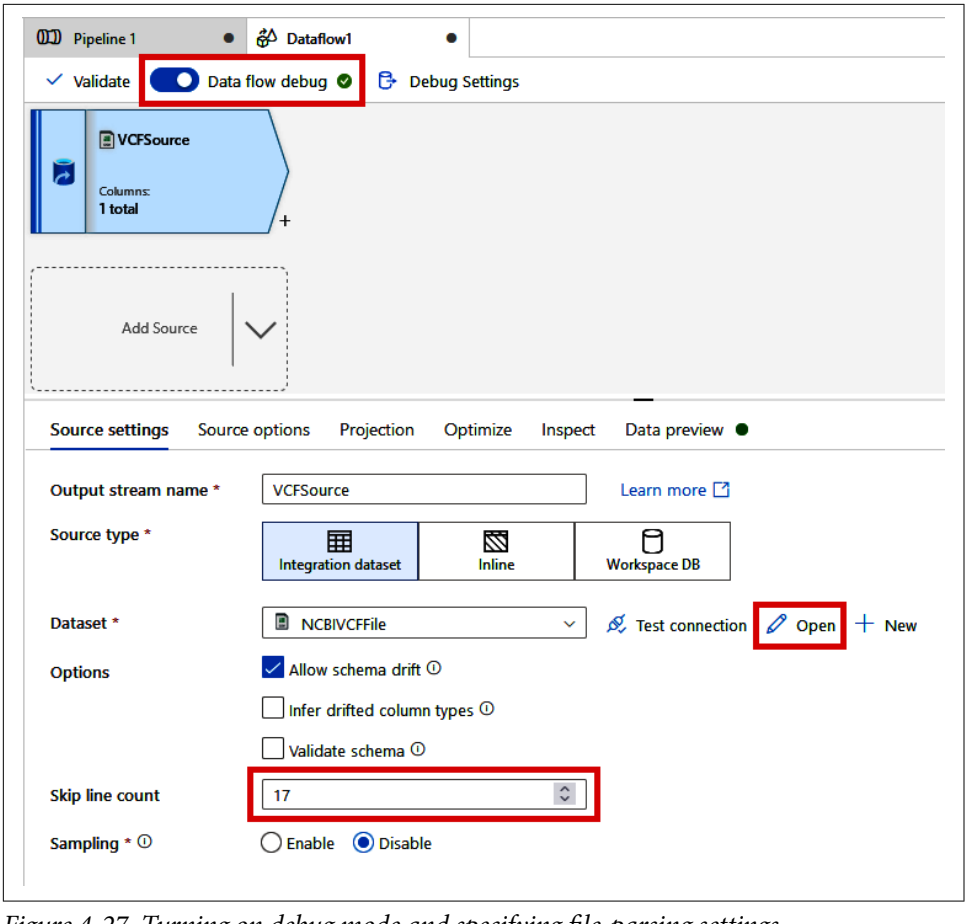


Figure 4-27. Turning on debug mode and specifying file-parsing settings

This will open a new canvas tab to allow you to further configure the connection settings to this VCF file.

On the Connection tab, shown in Figure 4-28, change "Compression type" to "gzip (.gz)," "Compression level" to "Fastest," and "Column delimiter" to "Tab (\t)."

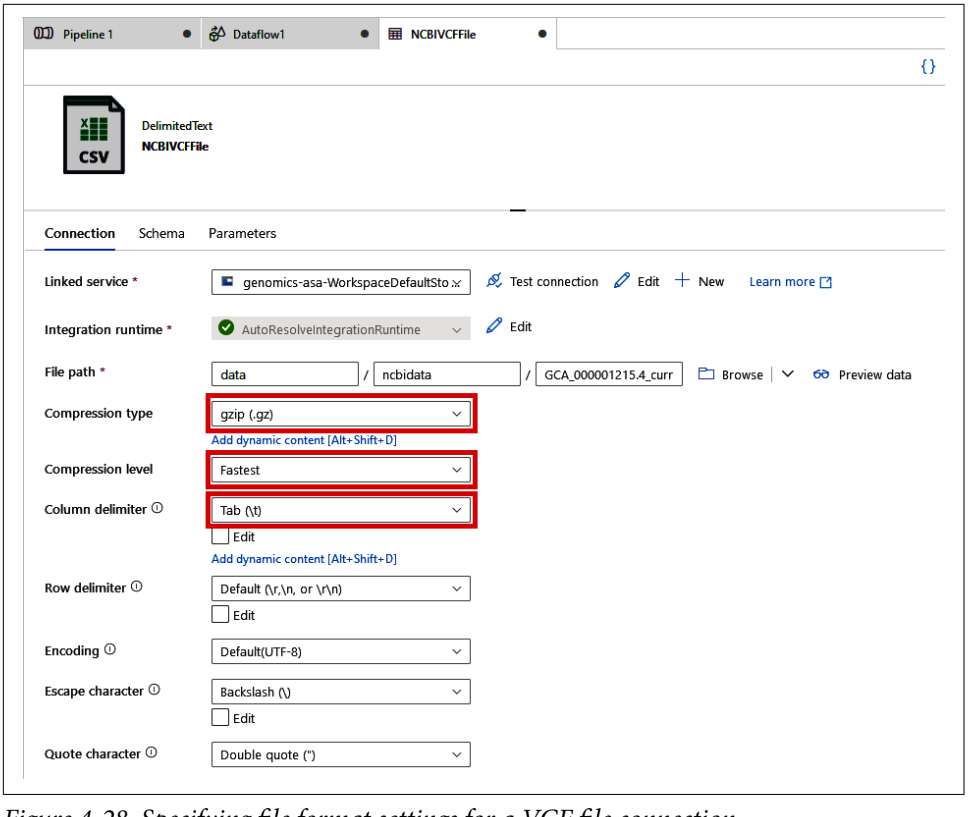


Figure 4-28. Specifying file format settings for a VCF file connection

On the "Schema" tab, shown in Figure 4-29, click "Import Schema > From sample file" and upload the *vcf_schema.tsv.gz* file from the */04_data_orchestration* folder in the supplementary materials. If you do not have access to this file (or want to customize the column names), simply create a tab-separated file with the VCF headers (as shown in the following code block) and gzip it. This will define the column names as the VCF file is read:

```
CHROM POS ID REF ALT QUAL FILTER INFO
```

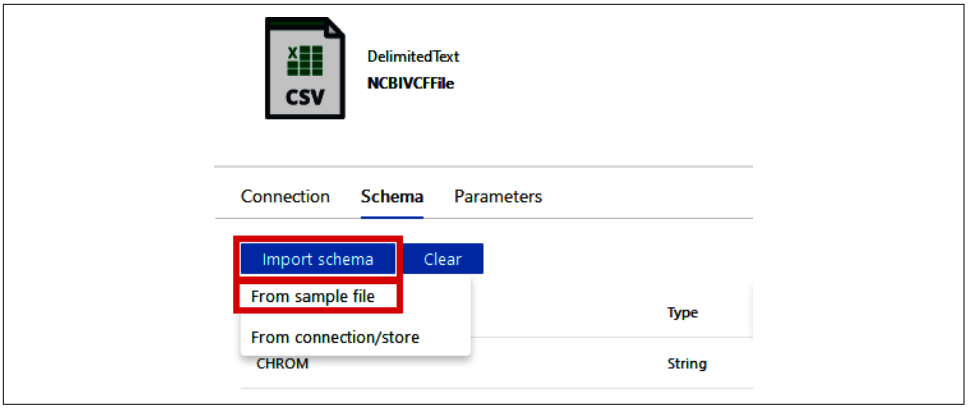


Figure 4-29. Specifying VCF file schema

Back over in the Data Flow tab, shown in Figure 4-30, click the "VCFSource" module and click the "Data preview" tab in the bottom part of the screen. Click the "Refresh" button. Once your debug session is ready, Data Factory will read in a sample of the VCF file and parse it according to your specifications. The "Data preview" tabs are very useful in understanding the state of the data at each step of the Data Flow.

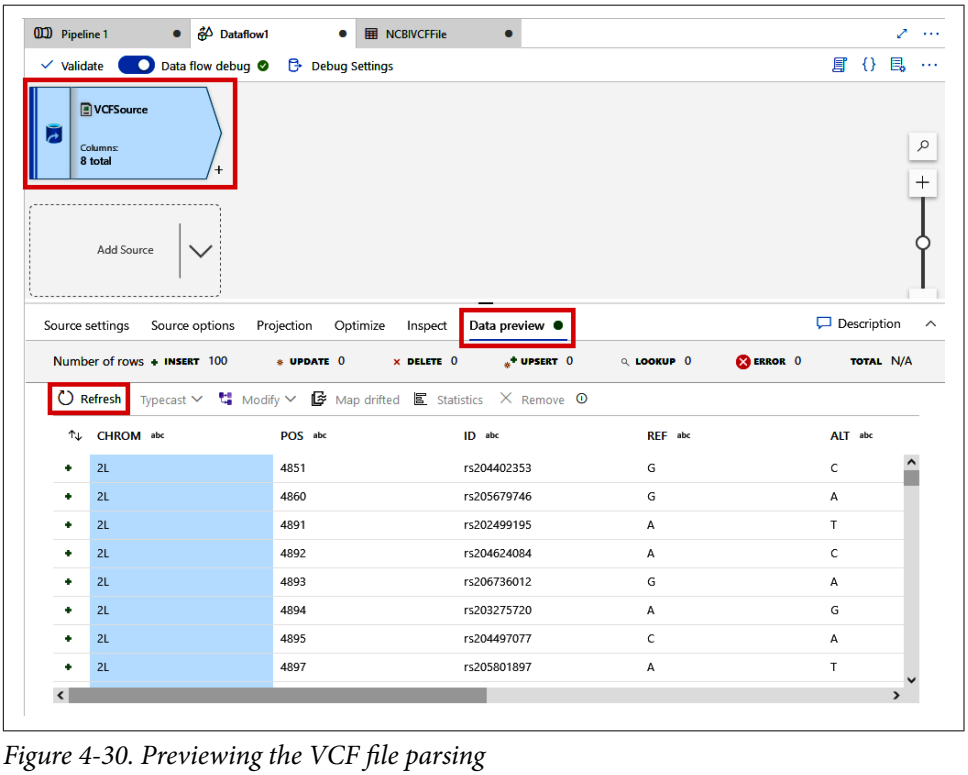


Figure 4-30. Previewing the VCF file parsing

Click the small + sign by the VCFSource module. Note that there are many different modules that we can use to join datasets together, change a dataset's shape (Pivot, Unpivot, etc.), or change the contents of the data (Aggregate, Filter, etc.).

For this next step, we will parse the `INFO` column of the VCF data to create an array of values, which will make this flexible column of information more useful later. From the + menu, click "Derived Column," as shown in Figure 4-31.

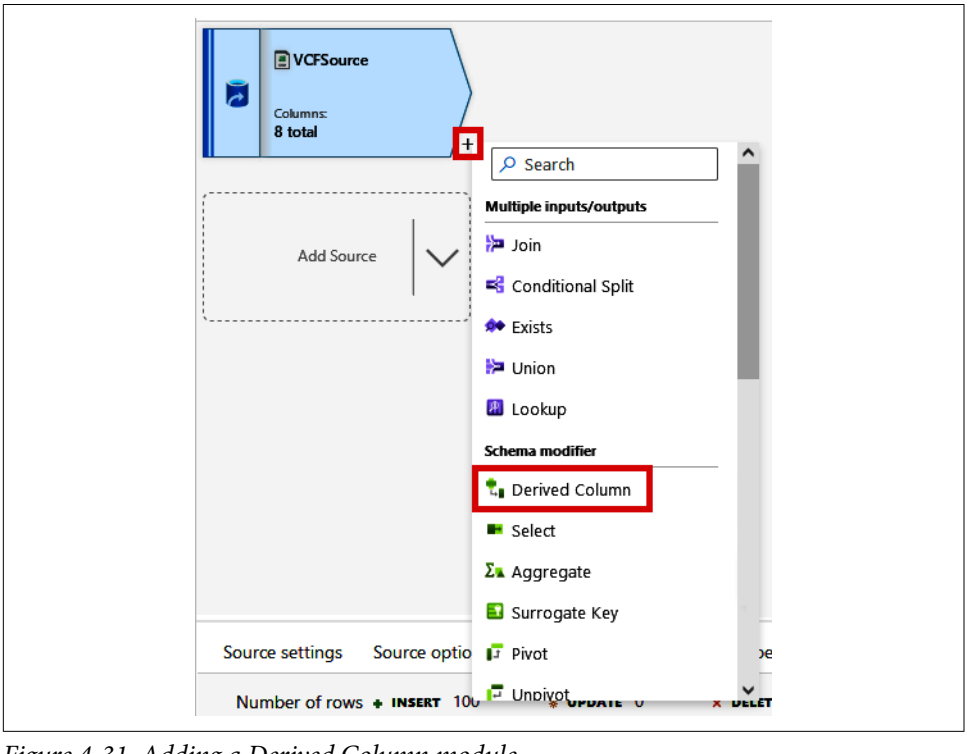


Figure 4-31. Adding a Derived Column module

As shown in Figure 4-32, click the new Derived Column module that gets created and change the Output stream name to "SplitINFOColumn." Click the "+ Add" button and select the INFO column from the drop-down box. For the expression, you can use the expression builder, which will help you see all of the functions and their documentation. For now, you can simply paste in `split(INFO, ";")`.

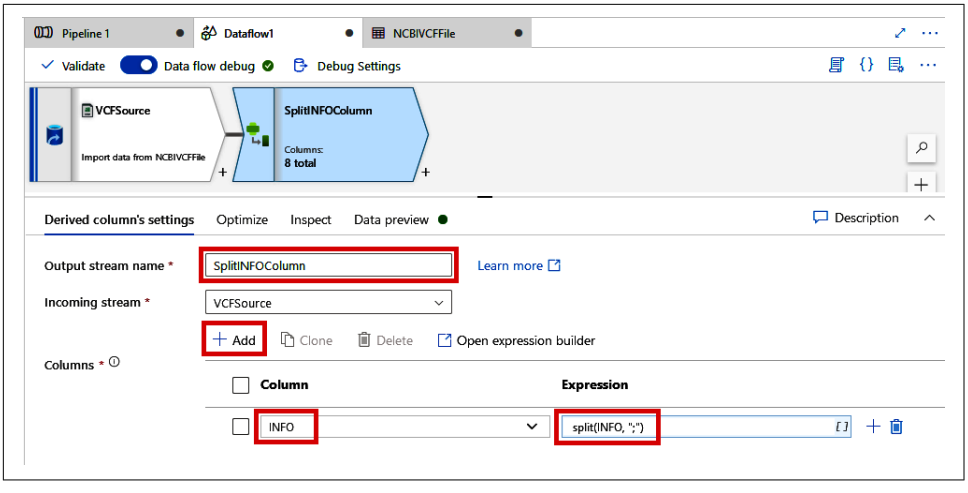


Figure 4-32. Splitting the INFO column of a VCF file using the Derived Column module

Once you have specified how the INFO column should be split, you can view the output of this process under the "Data preview" tab, shown in Figure 4-33. Note how the INFO column is now a column of arrays rather than long strings. Later, this will be useful for when we need to pull specific information out of this column.

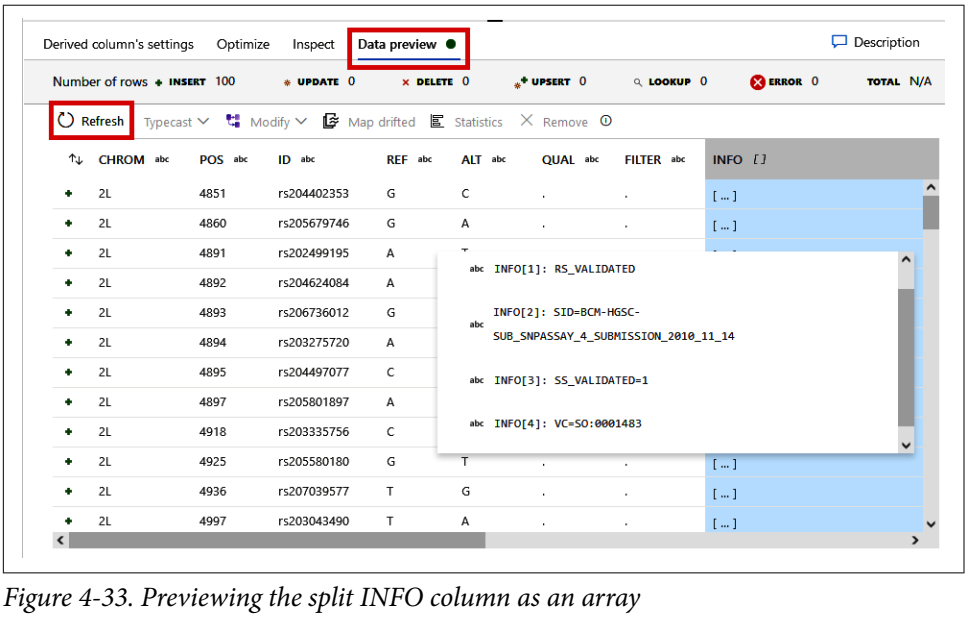


Figure 4-33. Previewing the split INFO column as an array

Finally, we need to write this data back to our data lake but as a different file format. Click the small + button beside your SplitINFOColumn module and select the "Sink"

option. See Figure 4-34. This will add a Sink module, which basically allows us to write out data to a destination.

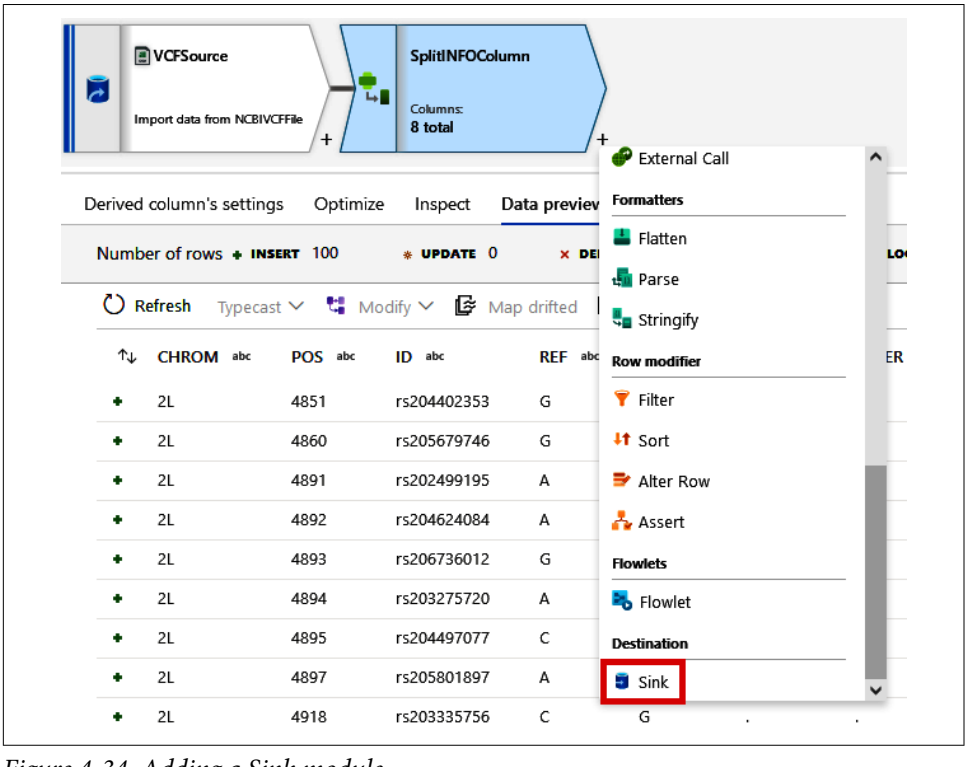


Figure 4-34. Adding a Sink module

In the Sink module that gets created, change the "Output stream name" to "ParquetOutput." We have not yet defined our destination, so click the "+ New" button next to "Dataset." Click the "Azure Data Lake Storage Gen2" option and click the "Continue" button (the same as in Figure 4-24). This time, though, select the Parquet option on the "Select format" screen and click the "Continue" button. See Figure 4-35.

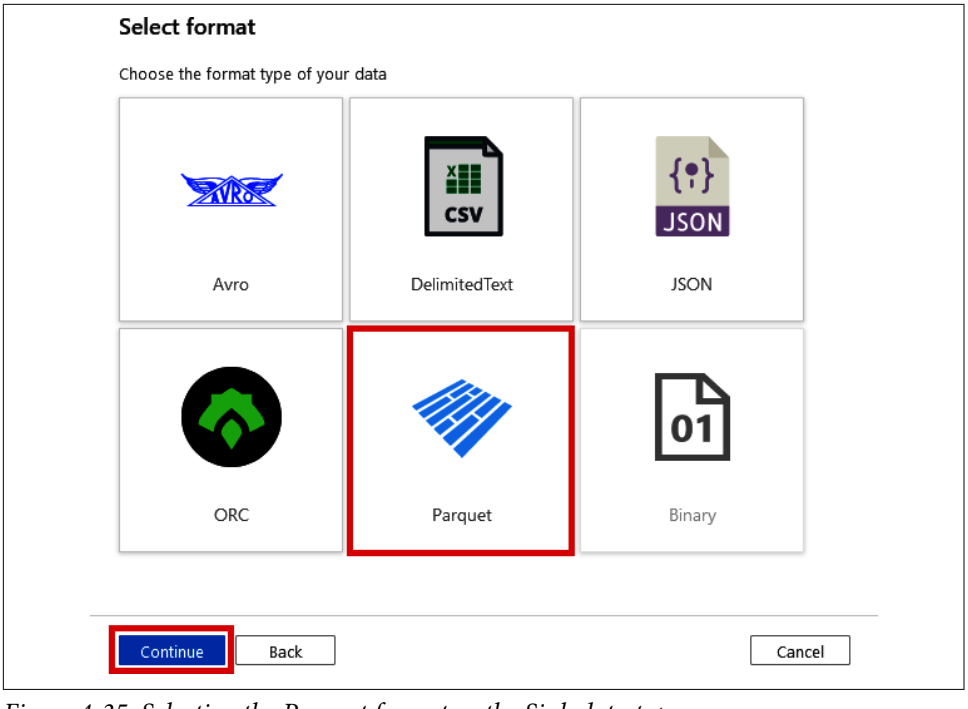


Figure 4-35. Selecting the Parquet format as the Sink data type

On the Set properties screen, shown in Figure 4-36, change the "Name" to "NCBIParquetFile" and use the same "Linked service" you used in Figure 4-26. Use the same file path from before as well, except leave the specific filename blank (as shown in Figure 4-36) and click the "OK" button.

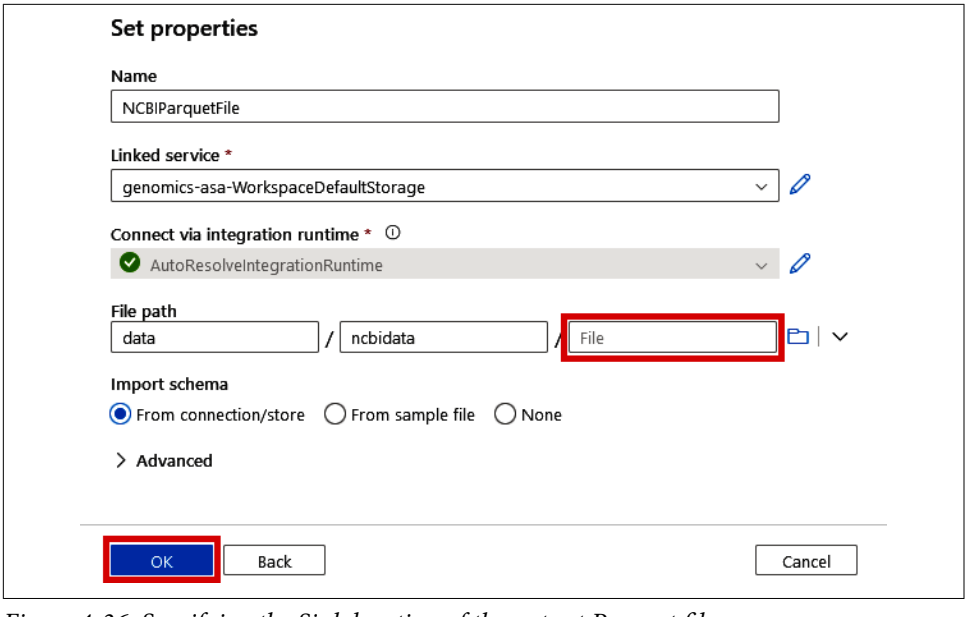


Figure 4-36. Specifying the Sink location of the output Parquet file

At this point, you should have a simple Data Flow of three steps: 1) read data from the VCF file; 2) split the INFO column into an array; and 3) write the data out as a Parquet file, similar to what is shown in Figure 4-37. Click the "Properties" button on the top right and rename your Data Flow to "VCF Parsing Data flow." After this, navigate back to your main Pipeline tab (likely called "Pipeline 1").

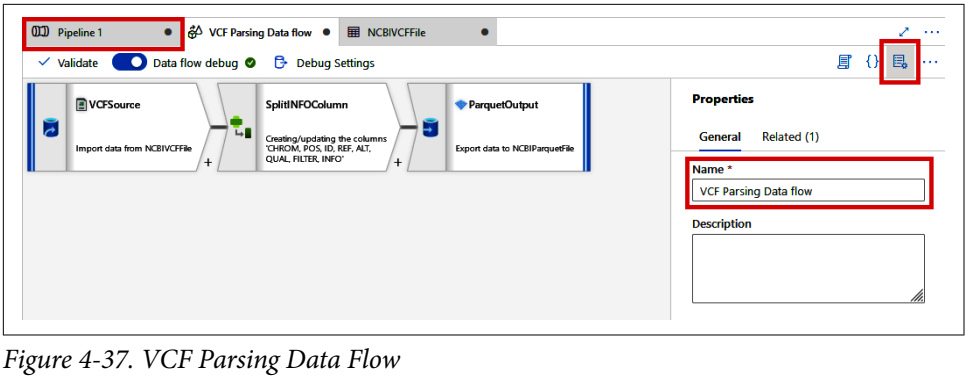


Figure 4-37. VCF Parsing Data Flow

Click the "Properties" button for the pipeline and rename it to "VCF Parsing Pipeline." In addition, ensure the Data Flow selected under the Settings tab is the "VCF Parsing Data flow" we just created. See Figure 4-38.

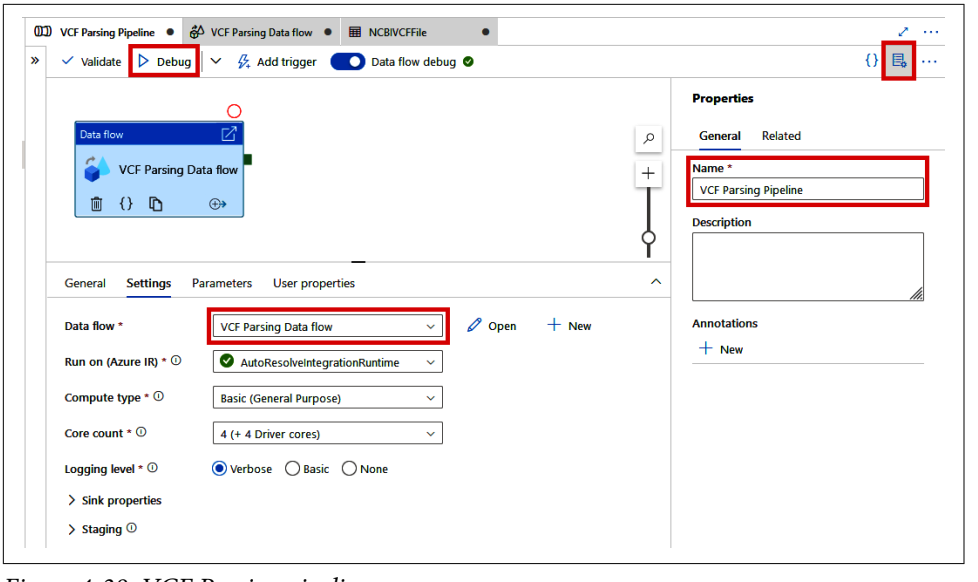


Figure 4-38. VCF Parsing pipeline

Ready to give it a spin? Click the "Debug" button at the top of the screen, which will run the pipeline (which will execute the Data Flow) and create a Parquet file of the NCBI VCF data in your data lake. If you see the green check mark (shown in Figure 4-39) on the Data flow module and in the Output tab, success!

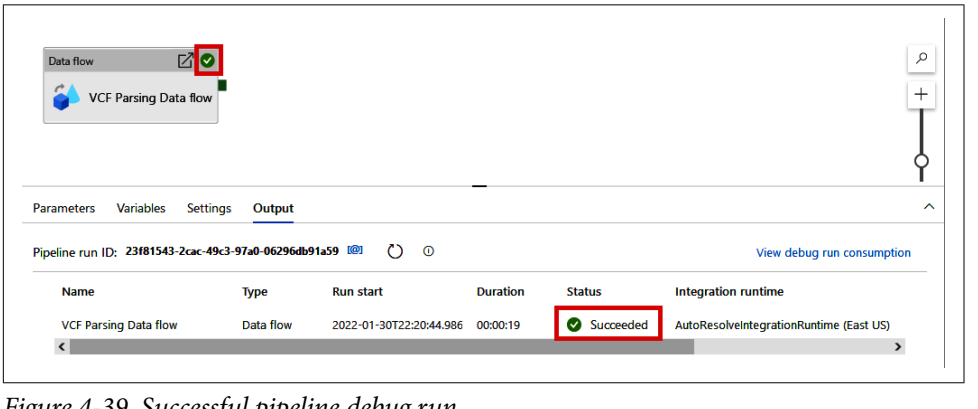


Figure 4-39. Successful pipeline debug run

Now you can go back to the Azure Portal and check in your data lake to see the Parquet file that was just created. See Figure 4-40.

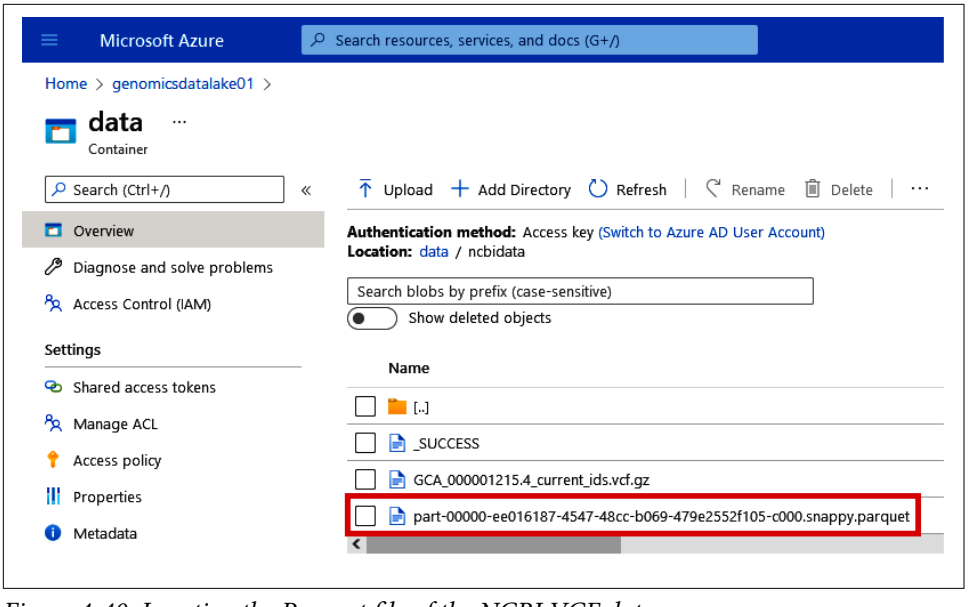


Figure 4-40. Locating the Parquet file of the NCBI VCF data

One final, very important thing. Before you close out of Data Factory, make sure to always click the "Publish all" button at the top of the screen, shown in Figure 4-41. This will save any datasets, pipelines, or Data Flows that you have created.

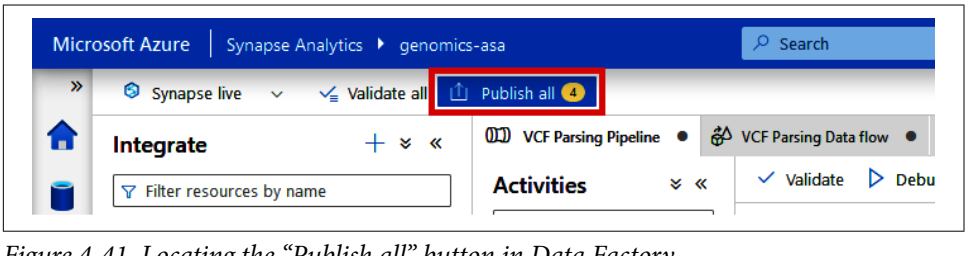


Figure 4-41. Locating the "Publish all" button in Data Factory

In this section, we've shown how to use the power of Data Flows to perform complex parsing on nonstandard file formats like VCF. This functionality is really important when we need to get our esoteric bioinformatics file formats to play well with enterprise data tools. Note that there is a ton of other Data Flow capabilities that we haven't covered in this book, but I encourage you to explore Data Flows with other human-readable file types such as SAM, GFF, and more to see if you can convert those files into other formats as we did here with a VCF file.

Building and Triggering Pipelines for Automation

As powerful and useful as a Data Flow in Data Factory is at transforming data, it does have its limits. Other services in Azure may be better suited for larger or more specialized transformation tasks. For such tasks, we can simply use Data Factory to farm out this processing to other services such as Databricks.

In addition to orchestrating data operations on external services (such as pulling sequence data from an NGS vendor's FTP site), we can trigger such actions to kick off Pipelines based on certain conditions. For example, we can schedule a pipeline to run every day at 8 a.m., or we can kick off the pipeline anytime new data with a certain file extension is added to a particular directory.

In fact, we'll be creating a Databricks-based VCF conversion pipeline in the section "Orchestrating a Databricks Notebook from Data Factory" in Chapter 5.

In this chapter, we've covered the basics of cloud-based data orchestration using Data Factory. As you can see, Data Factory is a fairly versatile tool for copying or moving data into Azure from external services (using Pipelines) or orchestrating the transformation of data, by using either Data Flows or other Azure-based services like Databricks. In summary, data orchestration is important in that it enables us to get data into the cloud and interact with other cloud-based data services. Azure Data Factory provides an interactive interface to create pipelines and copy data from source to destination (sink) and call other services.

Within Azure Data Factory, Data Flows are useful for transforming data in Pipelines, providing a single place to both orchestrate tasks and manipulate data. Plus, with triggers, any pipeline can be scheduled to run on a standard interval or triggered based on an event. Event-based triggers are useful for kicking off pipelines when new data becomes available.

In the next chapter, we'll be covering compute services that allow you to scale all sorts of data processing from bioinformatics pipelines to machine learning and beyond. Many of the services we cover in the next chapter can also be called from Data Factory, which will help you further automate and scale your data processing needs.

CHAPTER 5

Azure Databricks (and Apache Spark)

In genomics, we often have to perform computationally-intensive tasks such as sequence alignment, variant calling, and machine learning. Traditionally, we can perform these tasks on our local workstations or, if we're really fancy, on a physical computing cluster. The struggle with these options is in how they limit our scale or access when we need it most.

Physical workstations are quite convenient, but they obviously won't scale to handle workloads that are larger than the memory or physical disk space on the machine. Plus, they usually have only a small number of cores, limiting the amount of parallelization that can be achieved when running code.

High-performance computing (HPC) clusters are often a pain to use because you usually need to submit your job into a queue. This means that you can't really run your code interactively, making debugging very difficult. Plus, job queue purgatory is awful when you're racing against a deadline and really need your results fast.

Luckily, the Azure cloud has quite a few options for flexibly running various types of tasks at scale. In the next few chapters, I'll be covering a few different computational services and how they may help make your life easier when trying to analyze some data. In this chapter, we'll walk through how Spark-based cluster computing can elevate our data analysis capabilities and how Azure Databricks in the cloud alleviates most of our struggles that we've previously experienced when using HPC clusters on premises.

Introduction to Apache Spark and Databricks

Spark was created as a distributed computing platform in 2009 by Matei Zaharia at the University of California Berkeley's AMP Lab. It's now an open source project supported by the Apache Software Foundation with more than 1,700 contributors.

Apache Spark is touted as the "Unified engine for large-scale data analytics" that includes multilanguage support for using the Spark engine. The Spark logo is shown in Figure 5-1.

Figure 5-1. Apache Spark logo

Spark includes the following components:

Spark Core
: This is the central engine that supports the distribution of tasks across a computing cluster. This includes task dispatching and scheduling, handling of the network I/O between machines, and distributed processing functionality. While the underlying functionality is written in Java, most bioinformatics/data science users interact with the Spark engine using either the PySpark (Python), SparkR (R), or Scala APIs. These APIs are used so that anyone can utilize the power of Spark using a language with which they are comfortable.

Spark SQL
: This is the SQL engine that runs on top of the Spark Core to support DataFrames, which are distributed data abstractions for spreading data across clusters for distributed processing. The SQL engine provides data transformation capabilities and support for queries using a familiar dialect of SQL. Also, the SQL engine gives us support for importing and exporting data between the Spark environment and other data sources.

MLlib
: This is the machine learning library for Spark. It provides distributed versions of many traditional machine learning algorithms. This library is very useful for scaling the training of models when you have a ton of data or want to explore a large search space of hyperparameters and/or algorithms.

Spark Streaming

For streaming data use cases, the Spark Streaming component provides scalable stream processing that integrates well with the rest of the Spark environment. If you have a streaming data source from, say, Apache Kafka or Flume, Azure Event Hubs, or Amazon Kinesis, we can process data as it's "in flight" and land the data in various destinations.

GraphX

This is a newer (underused) component of Spark that provides support for distributed graph computation. This includes distributed versions of various graph algorithms and graph builders for scaling graph analytics tasks.

Spark clusters start with one or more worker nodes and a single driver node, which orchestrates all of the activities to the workers. The worker nodes are the nodes that actually perform the tasks. Note that a *node* here means a physical server or, in the cloud, a virtual machine. See Figure 5-2.

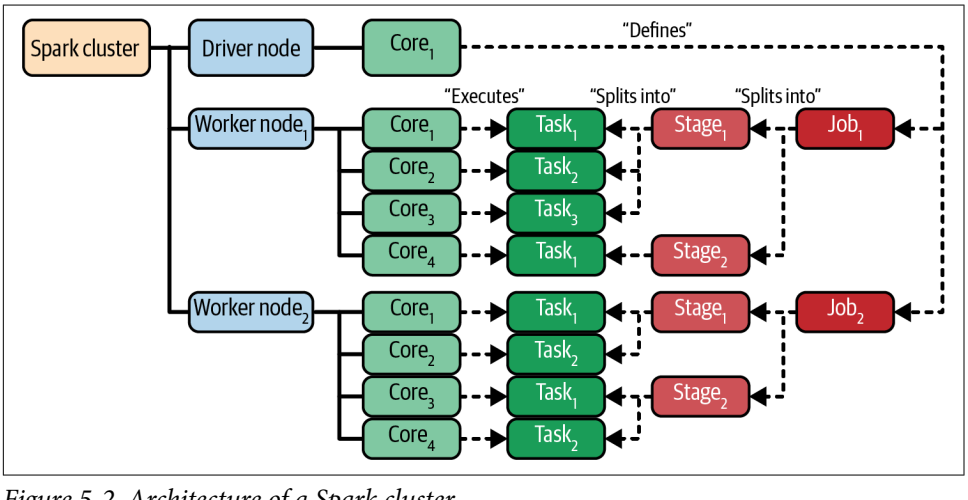


Figure 5-2. Architecture of a Spark cluster

Spark is deemed "embarrassingly parallel" because the engine distributes data and tasks across nodes of the cluster. This means that, once you load in some data, it comes through the driver node of the cluster where it is then split so that each node of the cluster holds a similarly sized subset of the data. When you execute actions on that data, the driver node then takes the action and divides it into jobs, stages, and tasks. Normally, a single core of a machine's processor works on a single task at a time.

Given this level of distribution of tasks and data, Spark is an excellent environment for performing complex tasks on large amounts of data. Well, in genomics, we're all too familiar with complex tasks and large amounts of data, making Spark a good option for many bioinformatics workloads.

So how does Databricks play into all of this? Once Spark was open sourced to the Apache Software Foundation, the research team that started the Spark research project at UC Berkeley founded Databricks in 2013. Databricks is a cloud-based PaaS for data and AI in the cloud, with Spark at its core. The Databricks logo is shown in Figure 5-3.

Figure 5-3. Databricks logo

As of now, Databricks is available on Azure, AWS, GCP, and Alibaba. In the next section, we'll walk through how to set up your Azure Databricks workspace, but the rest of the examples in this chapter are transferable if you happen to use Databricks on another cloud.

Setting Up an Azure Databricks Workspace

From the Azure Portal, click the "+ Create a resource" button at the top and search for "Databricks." You'll want to select the "Azure Databricks" option, as shown in Figure 5-4.

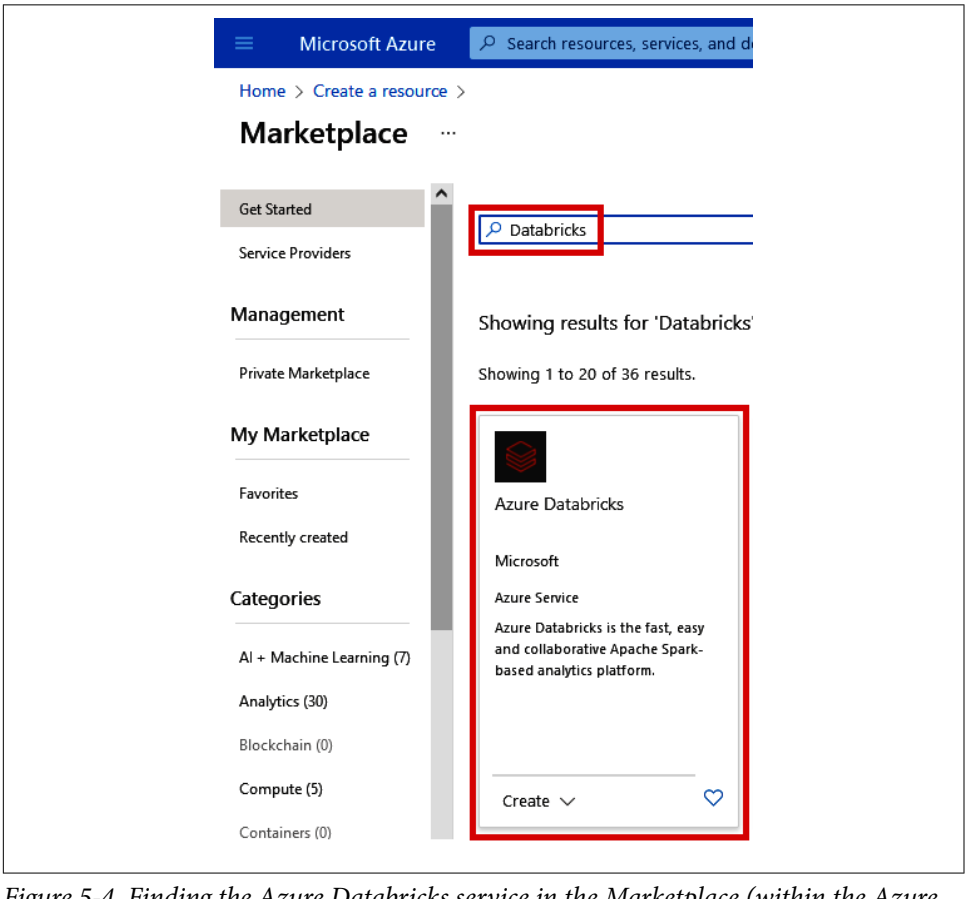


Figure 5-4. Finding the Azure Databricks service in the Marketplace (within the Azure Portal)

Click the "Create" button on the next screen, as shown in Figure 5-5.

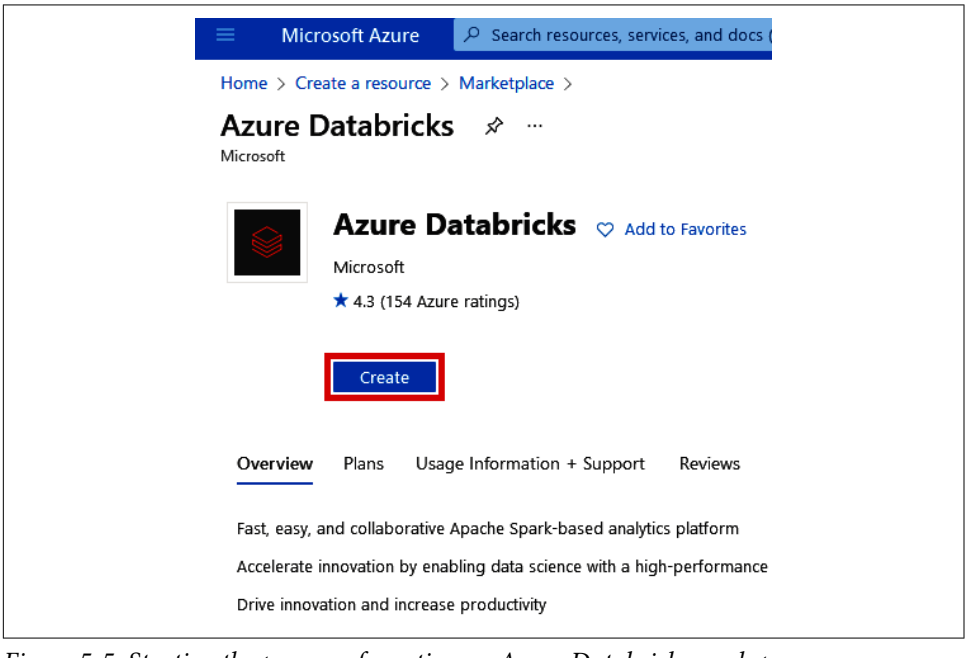


Figure 5-5. Starting the process of creating an Azure Databricks workspace

In the "Create an Azure Databricks workspace" blade, shown in Figure 5-6, select the subscription, resource group, and region that you've been using. Give your workspace a name (such as "genomicsdbx") and make sure to select the "Premium (+ Role-based access controls)" option for "Pricing Tier." Click the "Review + create" button and then click the "Create" button once Azure has performed the validation check on your selections.

In this setup process, we're choosing the "Premium" tier. This allows us to use Active Directory credentials (role-based access controls) in our Spark clusters, which is ideal in an enterprise setting. There is a "Trial" option where you won't have to pay for the Databricks Units (DBUs) for the first 14 days. Feel free to select the free trial if you're just testing out Databricks as part of this book.

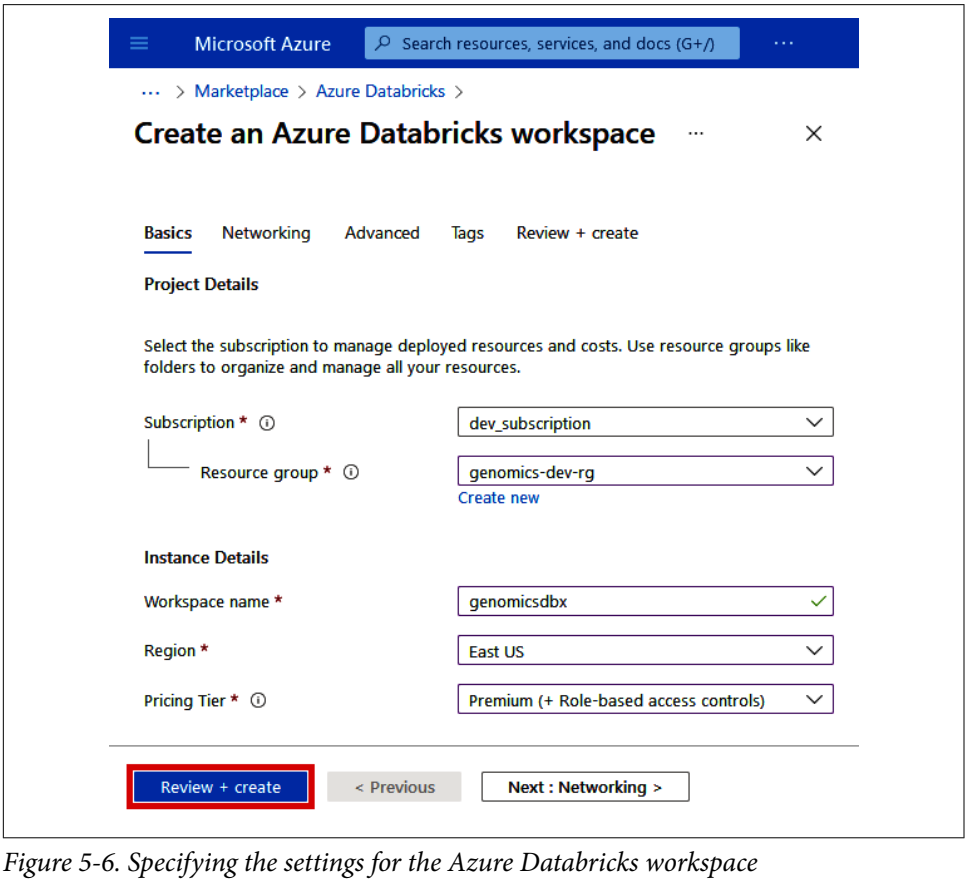


Figure 5-6. Specifying the settings for the Azure Databricks workspace

This will take just a few minutes to provision, but you'll soon have an Azure Databricks workspace show up in your resource group, as shown in Figure 5-7.

Locate the Databricks workspace in your resource group and click it.

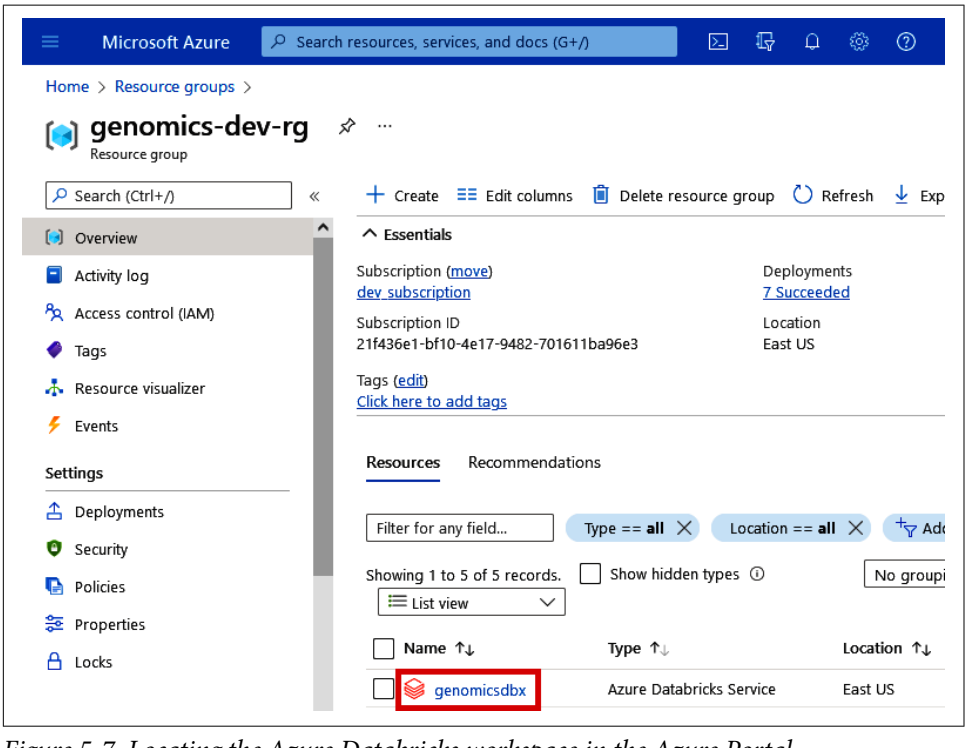


Figure 5-7. Locating the Azure Databricks workspace in the Azure Portal

Click the "Launch Workspace" button, shown in Figure 5-8, which will take you to the Databricks UI.

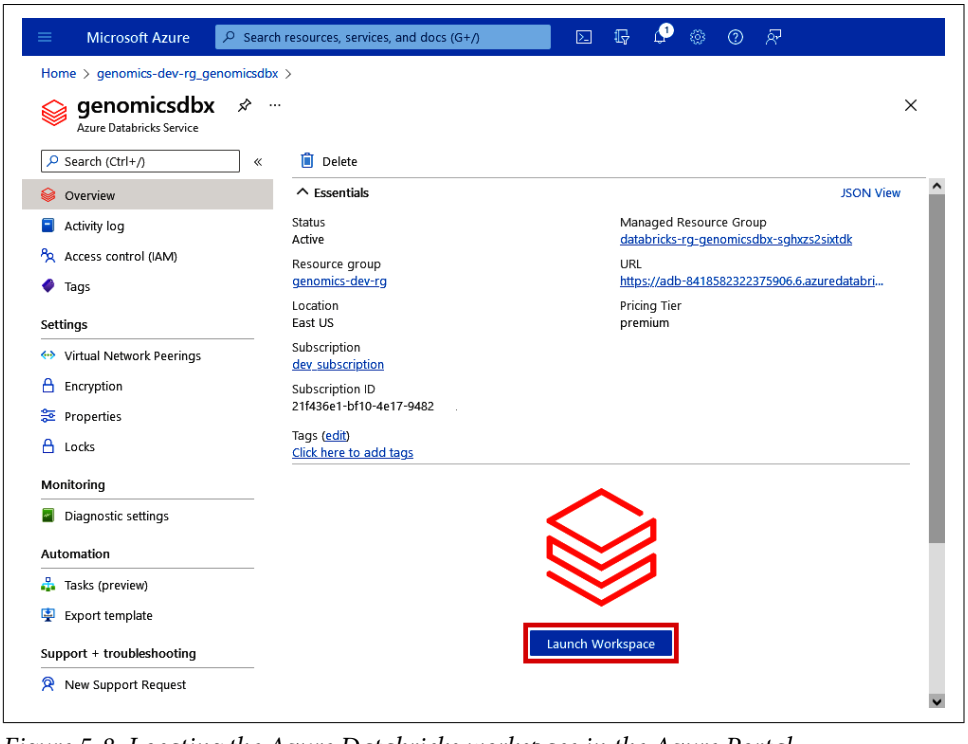


Figure 5-8. Locating the Azure Databricks workspace in the Azure Portal

Once inside the Databricks workspace, hover over the side menu. Enable Databricks Container Services by navigating to "Settings > Admin Console" and click the "Workspace Settings" tab at the top. Then, scroll down to the "Cluster" section and ensure that the "Container Services" setting is enabled. See Figure 5-9. Refresh the page and Container Services will be enabled.

This allows us to use custom Docker containers that have all of the preinstalled software we need for our bioinformatics tasks. Alternatively, you could spin up a normal cluster and then spend time installing everything manually. This is the easier choice for running Spark-enabled packages that are challenging to install on their own.

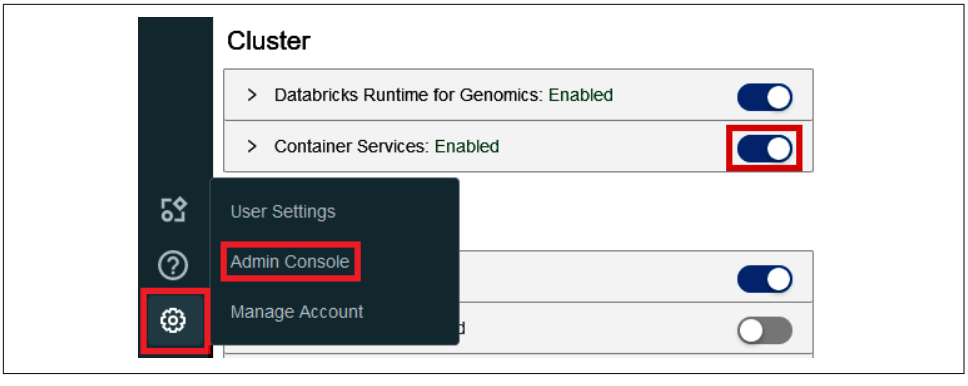


Figure 5-9. Enabling Container Services in Azure Databricks

Next, we need to create a compute cluster before we can run any code. So, from the side menu, click "Compute," as shown in Figure 5-10.

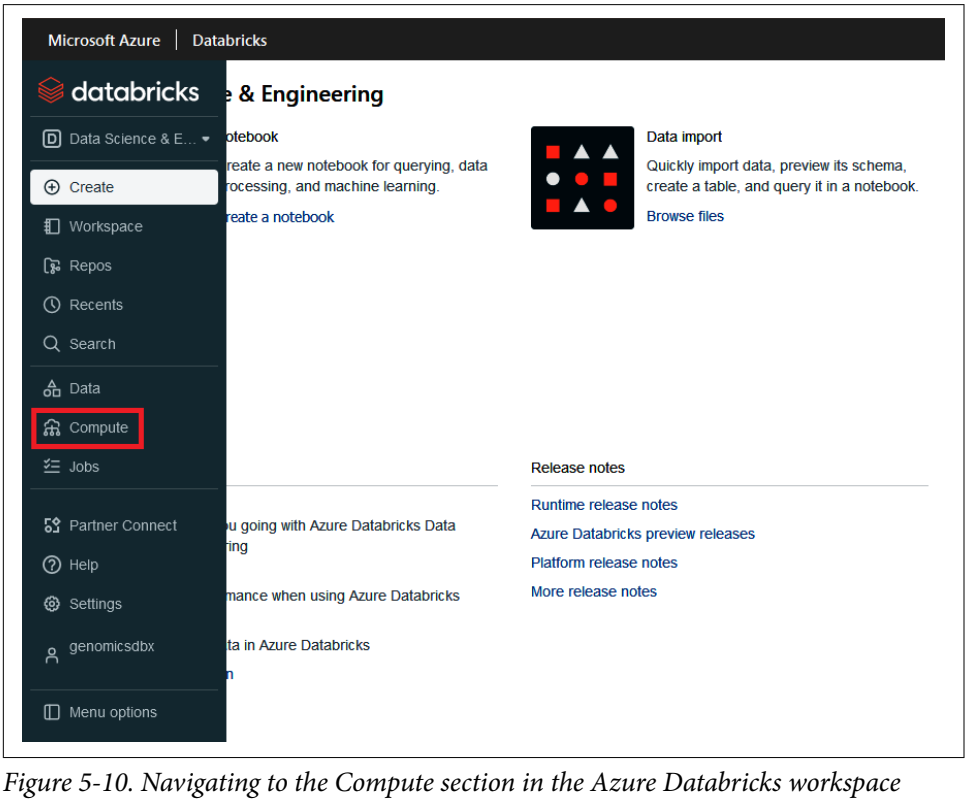


Figure 5-10. Navigating to the Compute section in the Azure Databricks workspace

On the "Compute" screen, shown in Figure 5-11, click "Create Cluster."

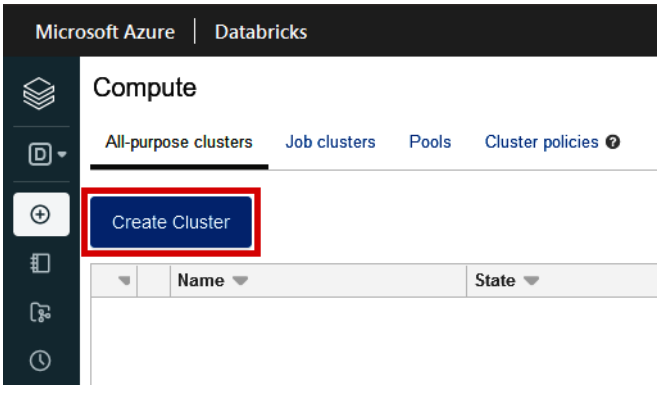


Figure 5-11. Creating a cluster in Azure Databricks

There are quite a few options when creating a Databricks cluster. These clusters are all Spark-based with one or more worker nodes led by a driver node (that handles all of the parallelization and distribution across the workers). When we create a cluster, we'll specify the quantity (or range) of workers that we want along with the sizes of the virtual machines for the driver and workers.

There are three different cluster modes:

High concurrency
: For top-tier performance when running concurrent workloads (for example, with multiple users sharing a cluster).

Standard
: Perfect for single users on a cluster who want to run their own workloads.

Single node
: For small jobs. This cluster type has no workers, meaning everything is performed on the driver node.

Since Databricks is an optimized version of Apache Spark, it has special runtimes that come preconfigured for certain purposes. Specifically, it has a Standard runtime for general data tasks as well as an ML (machine learning) runtime that comes with tools for training, deploying, and managing machine learning models at scale. Plus, Databricks has a special ML runtime that adds GPU support for deep learning tasks.

One big advantage of the Databricks platform is autoscaling and autotermination. Autoscaling allows for us to pick a minimum and maximum number of worker nodes in our cluster. When the cluster spins up, it only provisions the minimum number of nodes. Then, once we give the cluster additional work, it will elastically scale to the maximum number of nodes. This is beneficial for ad hoc usage, where the workload on the cluster is inconsistent. (Plus, it can save us money.)

Autotermination is really useful as it shuts off the cluster if no tasks are running after a specified period of time. This is key to cost savings as it eliminates the worry that someone will accidentally leave a cluster on over the weekend. See Figure 5-12.

Figure 5-12. Autoscaling and autotermination in Azure Databricks

For our upcoming variant data activity, we'll be learning how to read VCF files in Databricks using a package called Glow in a small cluster. This package was created by the genomics team at Databricks and Regeneron to scale bioinformatics tasks using Spark. Before we get to coding, we need to get the Glow package using a pre-built Docker container.

Let's choose the following options for our cluster (also shown in Figure 5-13):

- Cluster name: "glow_cluster_01"
- Cluster mode: Single Node
- Databricks runtime version: 10.4 LTS
- Use your own Docker container: Check this box
- Docker Image URL: "projectglow/databricks-glow:10.4"
- Authentication: "Default"
- Autopilot options: Set to autoterminate after 120 minutes
- Note type: Standard_DS3_v2

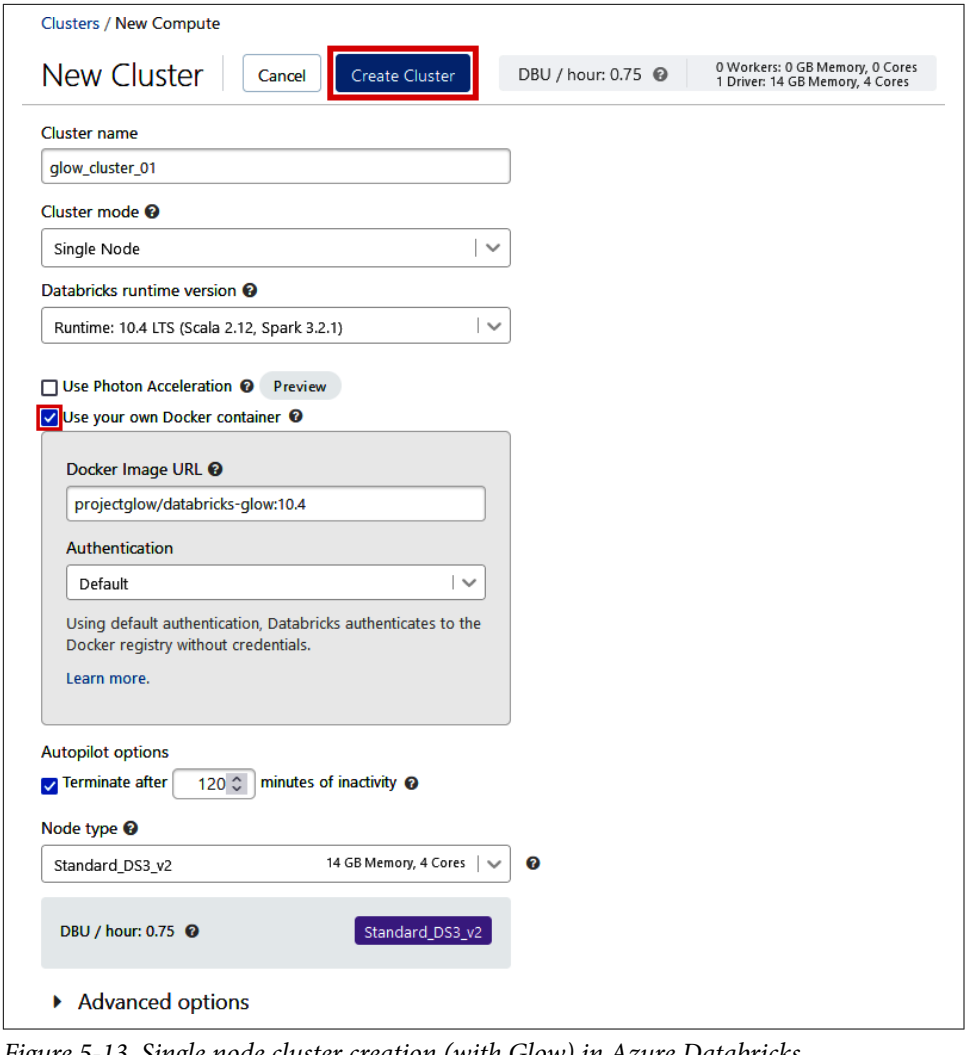


Figure 5-13. Single node cluster creation (with Glow) in Azure Databricks

Once you've configured these settings, click the "Create Cluster" button at the top. This will create a single-node cluster using a containerized version of Glow from Docker Hub and install it on a cluster in a few minutes. This image contains all of the necessary dependencies to run Glow on Spark, specifically in Databricks, which we'll use in a bit. This is much easier than us having to install everything on a blank cluster. This will come in handy in the section "Processing Variant Data with the Glow Package" on page 165.

You'll know your cluster has been successfully created and turned on when you see the green check mark beside the cluster's name.

Next, we'll be connecting our Databricks workspace to our data lake so that we can grab our data easily right from our Spark environment.

Connecting Databricks to Your Data Lake

It's common to use Databricks with a data lake as the central location from which to read and write data. (How many "times can I say "data" in the same sentence?) To do so, we can easily mount our data lake that we created in Chapter 2. Mounting a storage location will allow for Databricks to interact with our data lake as if it is part of the filesystem, making it super easy to read and write to the data lake.

Jump back to your Azure Portal and navigate to your data lake. On the side menu of the Azure Storage account, shown in Figure 5-14, click "Access keys."

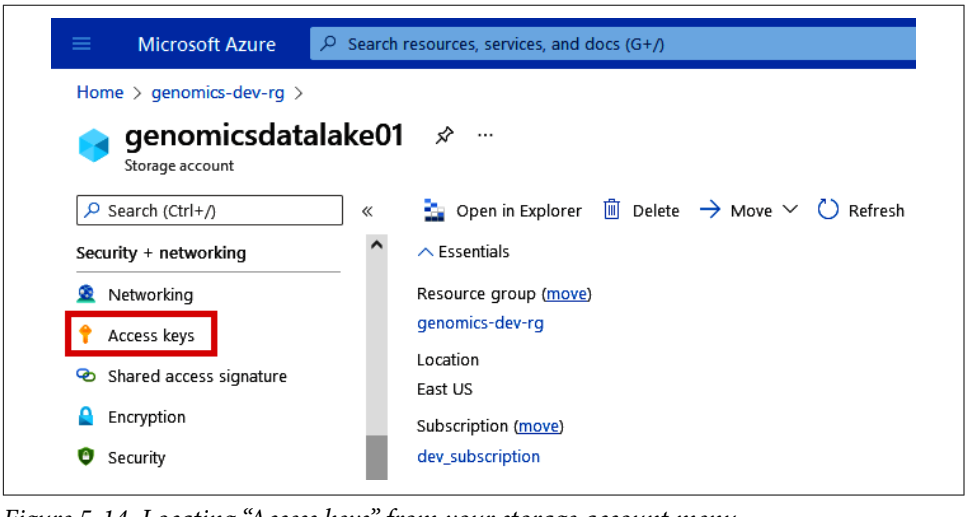


Figure 5-14. Locating "Access keys" from your storage account menu

In the "Access keys" screen, shown in Figure 5-15, copy your "Storage account name" and "Key" (key1) into a text editor as you'll need them in the next step.

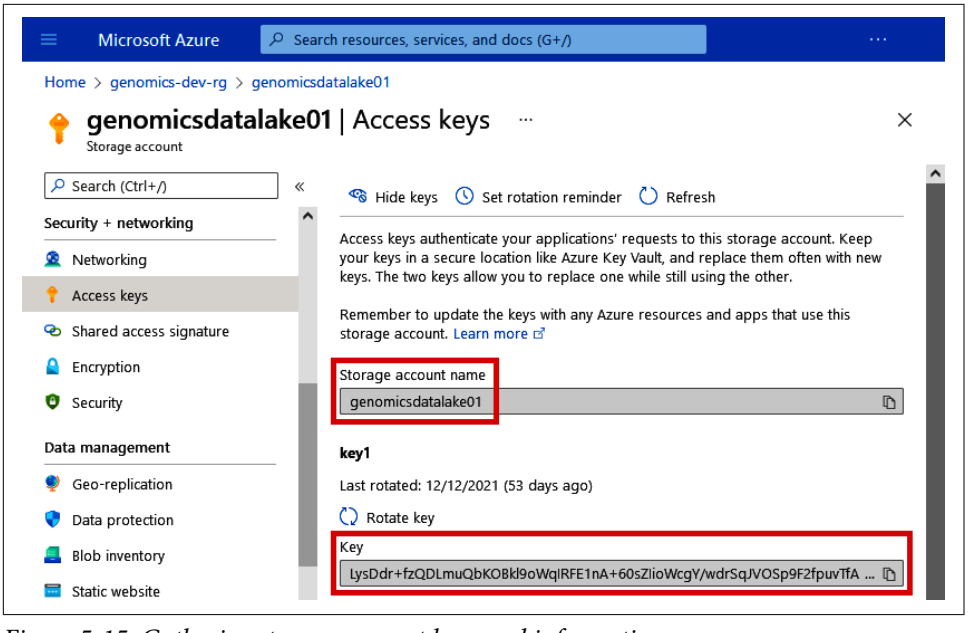


Figure 5-15. Gathering storage account keys and information

Navigate back to your Databricks workspace. From the main screen, click the "Workspace" tab. Anywhere in the whitespace in your user folder, right-click "Create > Notebook." See Figure 5-16.

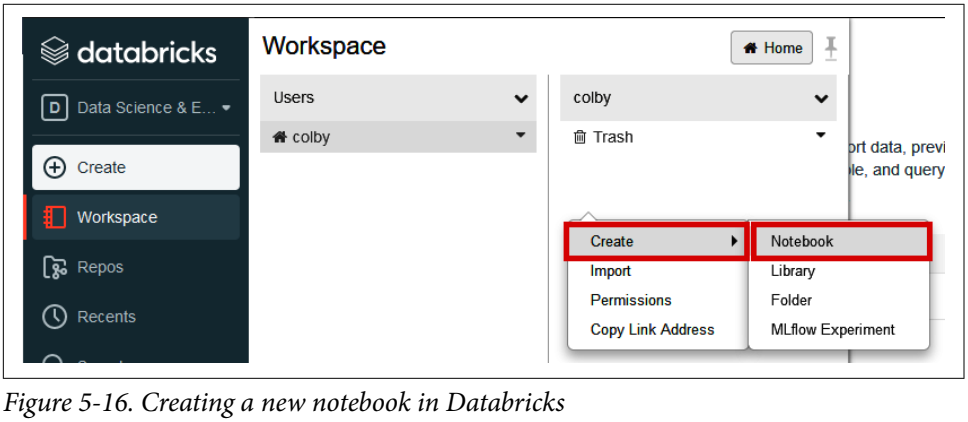


Figure 5-16. Creating a new notebook in Databricks

In the "Create Notebook" window, shown in Figure 5-17, give the new notebook a name (such as "mount-data-lake"). Make sure you're creating a Python-based notebook, and select the cluster you created in the last step. Click the "Create" button.

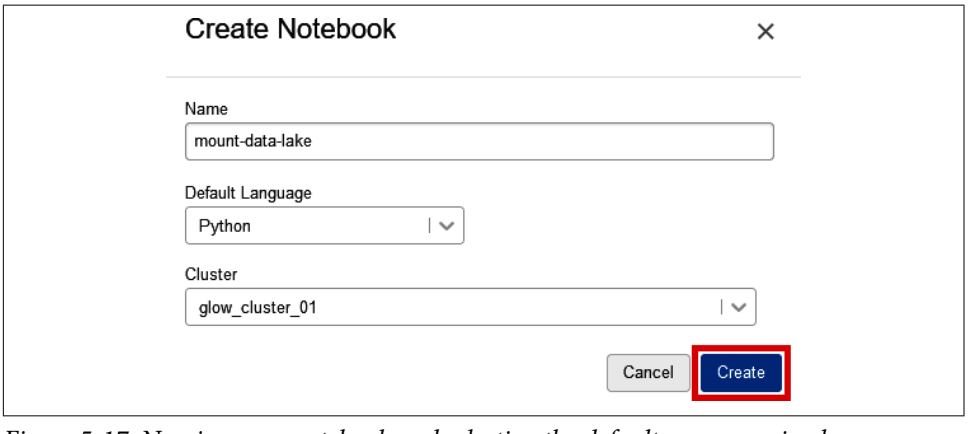


Figure 5-17. Naming your notebook and selecting the default programming language and cluster

In the new notebook that gets created, copy and paste the following code into a new cell (but don't run it yet):

```
dbutils.fs.mount(
    source = "wasbs://<CONTAINERNAME>@<STORAGEACCOUNT>.blob.core.windows.net",
    mount_point = "/mnt/genomicsdatalake/",
    extra_configs = {"fs.azure.account.key.<STORAGEACCOUNT>.
        blob.core.windows.net":"<KEYGOESHERE>"})
```

Now, fill the values in the "<>" brackets with your storage account information as follows:

- For "<CONTAINERNAME>", fill in "data" (or the name of your container if it's named something else).
- In place of "<STORAGEACCOUNT>" fill in the name of your storage account (e.g., "genomicsdatalake01").
- In the "<KEYGOESHERE>" section, fill in the key for your storage account (e.g., "LysDdr+fzQDLmuQ...").

Once you fill in the information, run the cell in the Databricks notebook. If the mounting process works, you'll see a "True" output from the cell. This means that we can now read and write from the data container in our data lake. See Figure 5-18.

```
Cmd 3

1  dbutils.fs.mount(
2    source = "wasbs://data@genomicsdatalake01.blob.core.windows.net",
3    mount_point = "/mnt/genomicsdatalake/",
4    extra_configs = {"fs.azure.account.key.genomicsdatalake01.blob.core.windows.net":"LysDdr+fzQDLmuQ

Out[14]: True

Command took 21.24 seconds -- by colby on genomics_cluster_01
```

Figure 5-18. Mounting a storage account container in Databricks

This container in your genomics data lake will now be available from the Spark environment at */mnt/genomicsdatalake*. You can use the `display()` function in Databricks to view 'the data lake location of the NCBI VCF data, as shown in Figure 5-19. We'll be using this data in the next step.

```
Cmd 4

1  display(dbutils.fs.ls("/mnt/genomicsdatalake/ncbidata"))

▸ (2) Spark Jobs

Table   Data Profile

| | path |
|---|---|
| 1 | dbfs:/mnt/genomicsdatalake/ncbidata/GCA_000001215.4_current_ids.vcf.gz |
| 2 | dbfs:/mnt/genomicsdatalake/ncbidata/_SUCCESS |
| 3 | dbfs:/mnt/genomicsdatalake/ncbidata/part-00000-ee016187-4547-48cc-b069-479e2552f105-c000.snappy.parquet |

Showing all 3 rows.

Command took 0.46 seconds
```

Figure 5-19. Locating the NCBI VCF data from Databricks

Processing Variant Data with the Glow Package

Now that we have our data lake mounted inside our Databricks cluster, we can start experimenting with PySpark to explore our VCF file. We'll also learn how the Glow package can help us read and process other bioinformatics data.

Create another new Python notebook just as you did in Figure 5-16. This time, name the notebook "vcf-exploration" (and connect it to your cluster).

We'll be using Glow to help us read in our VCF data as a traditional Spark DataFrame. Spark DataFrames feel quite familiar if you're used to Pandas DataFrames in Python or DataFrames in R. Once our VCF data is in a DataFrame, we then simply use any standard SparkSQL logic to explore and analyze its contents.

To start, import Glow and some useful PySpark functions and register Glow to the Spark environment using the following code:

```
import glow
from pyspark.sql.functions import *
glow.register(spark)
```

To run a cell, you can simply click the ▷ button on the top right of the cell or press Shift+Enter while active in a cell. See Figure 5-20.

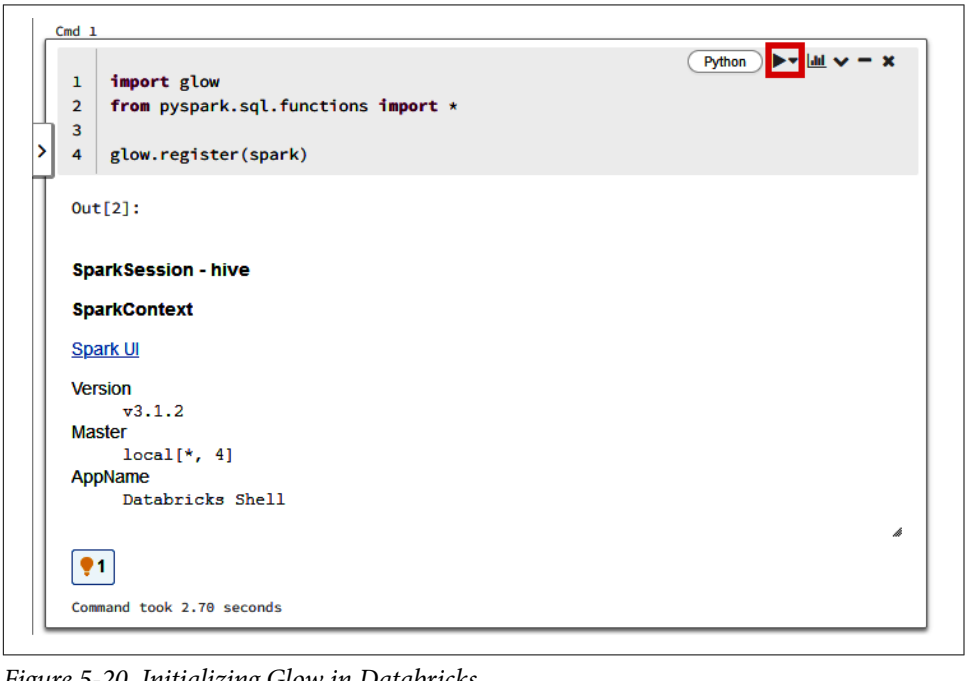


Figure 5-20. Initializing Glow in Databricks

Next, we need to look into our mounted data lake location to get the path to our sample VCF file. Click the + icon just under the first cell to create a new one. In this cell, type the following (changing the path if your VCF data is stored somewhere else):

```
display(dbutils.fs.ls("/mnt/genomicsdatalake/ncbidata/"))
```

The `display()` function is similar to the `.show()` method in Pandas, except this provides a nice interactive table in the output. `dbutils` is a set of Databricks-specific functions that are useful for executing filesystem commands, storing secrets, and more. Here, we use the `dbutils.fs.ls()` function to list the files in a specified directory.

When you execute this cell, the notebook will list the files that it finds at that location. This is useful for locating different files that you want to read. Note that my VCF file is located at */mnt/genomicsdatalake/ncbidata/GCA_000001215.4_current_ids.vcf.gz*, as shown in Figure 5-21.

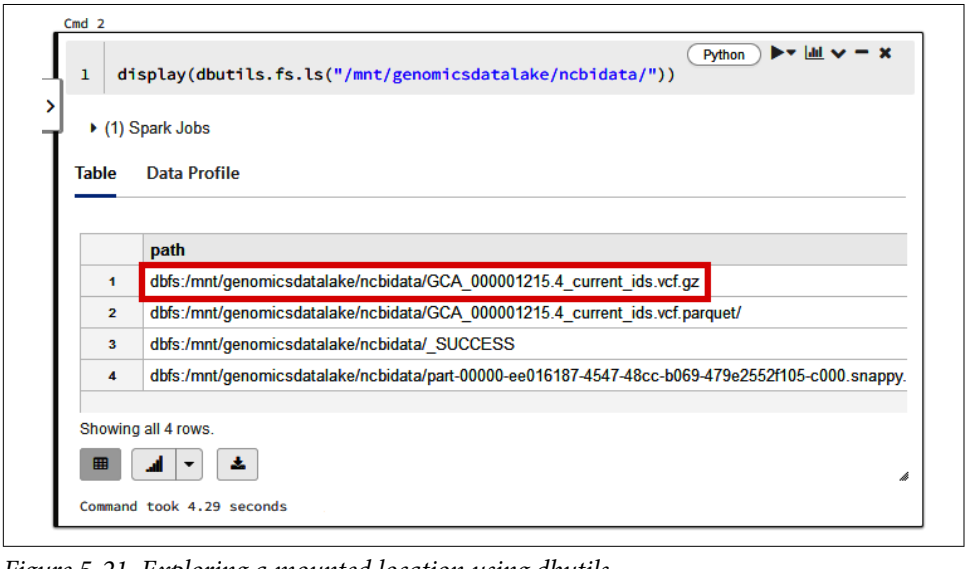


Figure 5-21. Exploring a mounted location using dbutils

Create a new cell below, and use this path information to create a new "vcf_path" variable. We'll then pass this path information into the VCF reader function as follows:

```
vcf_path = '/mnt/genomicsdatalake/ncbidata/GCA_000001215.4_current_ids.vcf.gz'
vcf_df = spark.read.format('vcf').load(vcf_path)

display(vcf_df)
```

Spark natively has the ability to read CSVs, Parquet files, JSON files, and more. Glow extends this functionality and adds support for reading VCF, BGEN, PLINK, and GFF files.

Once you execute this cell, you should see a preview of the data in the output (due to the previous `display()` line). Take a second to explore the columns in this VCF DataFrame output. See Figure 5-22.

Cmd 3

```
vcf_path = "/mnt/genomicsdatalake/ncbidata/GCA_000001215.4_current_ids.vcf.gz"
vcf_df = spark.read.format("vcf").load(vcf_path)

display(vcf_df)
```

(2) Spark Jobs

vcf_df: pyspark.sql.dataframe.DataFrame = [contigName: string, start: long ... 15 more fields]

Table | Data Profile

	contigName	start	end	names	referenceAllele	alternateAlleles	qual
1	2L	4850	4851	["rs204402353"]	G	["C"]	null
2	2L	4859	4860	["rs205679746"]	G	["A"]	null
3	2L	4890	4891	["rs202499195"]	A	["T"]	null

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 3.09 seconds

Figure 5-22. Reading a VCF into a Spark DataFrame

Note how the columns differ from the standard VCF specification. This is due to the fact that the flexibility of VCF files often makes for a nightmare when attempting to repeat work between VCFs that were made using different tools. Glow attempts to standardize columns regardless of the VCF source, therefore adding stability into our data processes.

Also note that Glow can break out all of the attributes contained in the INFO column. This is useful for when we want to use this information in our exploration. This and other options can be turned off like so:

```
vcf_df = spark.read.format("vcf") \
    .option("includeSampleIds", False) \
    .option("flattenInfoFields", False) \
    .load(vcf_path)
```

Exploring DataFrames

For our final activity of this section, let's start exploring this DataFrame that we've read into Spark. I'll provide some queries relevant to bioinformatics, but I encourage you to continue playing with this dataset and others to get comfortable with the PySpark syntax.

Filtering to Chromosome Coordinates

Often, we may be interested only in mutations that occur at certain locations across the genome. To do this in Spark, we can simply filter the contig name and position columns (start and end) to the region of interest. We can also use the `.withColumn()` method to derive other features in the querying process. For example, find the length of the alternative allele to see if it's an indel:

```
display(vcf_df.filter((col("contigName") == "2L") &
    (col("start") >= 5000) &
    (col("end") <= 5100)) \
    .withColumn("altSize", size(col("alternateAlleles"))) \
    .select("names", "contigName",
    "start", "end", "referenceAllele", "alternateAlleles", "altSize"))
```

This should display a result output similar to Figure 5-23.

	names	contigName	start	end	referenceAllele	alternateAlleles	altSize
1	["rs203929846"]	2L	5001	5002	G	["T"]	1
2	["rs205742710"]	2L	5008	5009	A	["G"]	1
3	["rs206382444"]	2L	5020	5021	T	["A"]	1
4	["rs204018914"]	2L	5037	5038	T	["A"]	1
5	["rs206239448"]	2L	5038	5039	C	["A"]	1
6	["rs202911109"]	2L	5039	5040	G	["A"]	1
7	["rs203695525"]	2L	5041	5042	C	["A"]	1
8	["rs205363191"]	2L	5045	5046	G	["A"]	1
9	["rs202649288"]	2L	5063	5064	A	["C"]	1

Figure 5-23. Filtering variants to genome location

Count of variants by chromosome

PySpark has a convenient `.count()` method that works well once data has been grouped by some feature. So let's group our data by the "contigName" column (which is the chromosome) using the following code:

```
display(vcf_df.groupBy('contigName').count())
```

This will output a small table, similar to Figure 5-24, that shows the few chromosomes that our data contains along with how many variants there are for each chromosome.

	contigName	count
1	2L	1265412
2	2R	1044044
3	3L	1263577
4	3R	1245855
5	mitochondrion_genome	2
6	X	811807
7	4	3174

Showing all 7 rows.

Figure 5-24. Counting of variants by chromosome

Feel free to continue exploring this basic VCF file and practice your PySpark skills. Even if you're a Python and Pandas pro, there's a lot to learn with the PySpark dialect and how Spark handles different activities. For example, I would suggest you practice querying and filtering the data as this is a skill that can help you answer questions about the presence/prevalence of various variants in your data.

A great resource to look up functions is the Apache Spark Getting Started documentation.

In the next section, we'll be taking this process a step further by automating this VCF file parsing.

Automating Variant Data Processing

Next, we will be using a notebook that processes VCF files (using the Glow package) and saves them back out in Parquet format. This notebook will be called by Data Factory.

From the side menu shown in Figure 5-25, click "Workspace" and, under your username, right-click anywhere in the whitespace. Click "Import."

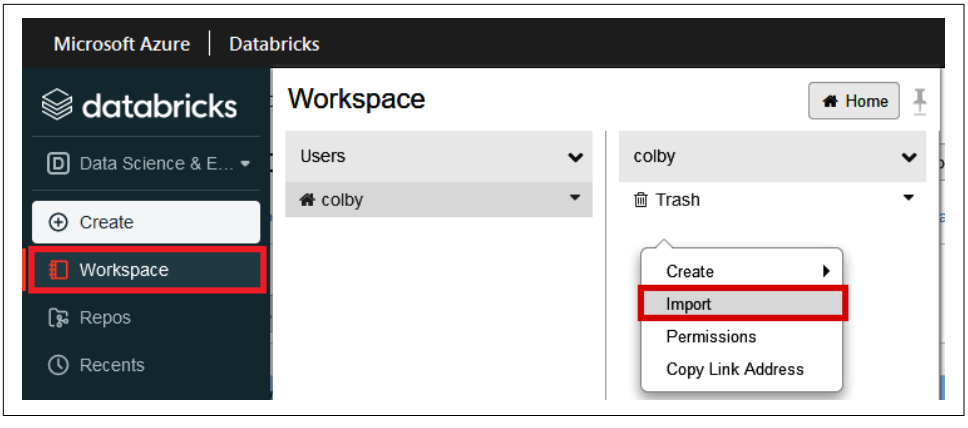


Figure 5-25. Creating or importing notebooks

Upload the `05_computation/databricks-vcf-to-parquet.dbc` notebook from the supplementary materials and click the "Import" button, as shown in Figure 5-26. (Alternatively, you can right-click and create a new Python notebook and write the code from scratch.)

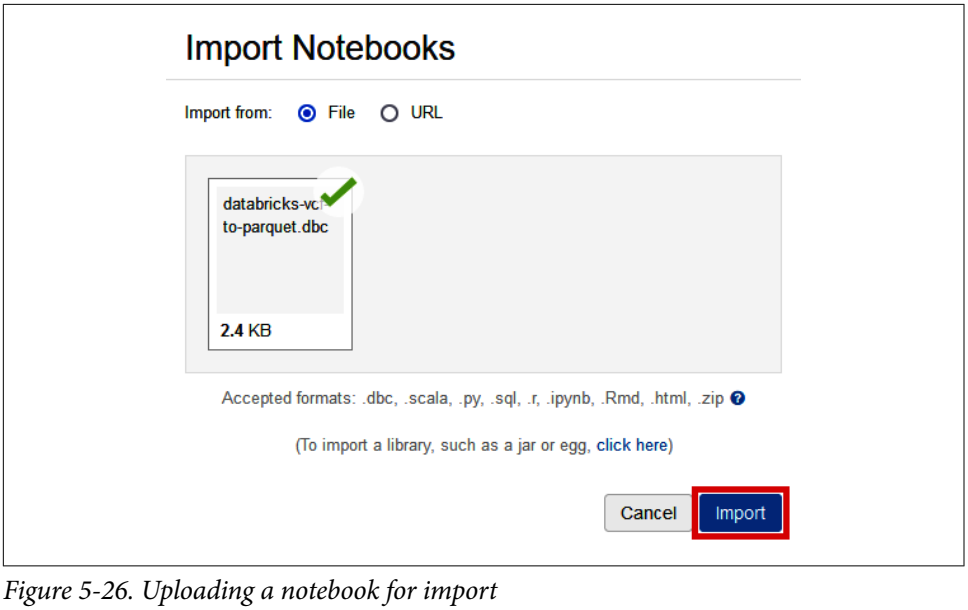


Figure 5-26. Uploading a notebook for import

Locate the notebook under the "Workspace" tab, shown in Figure 5-27, and click it.

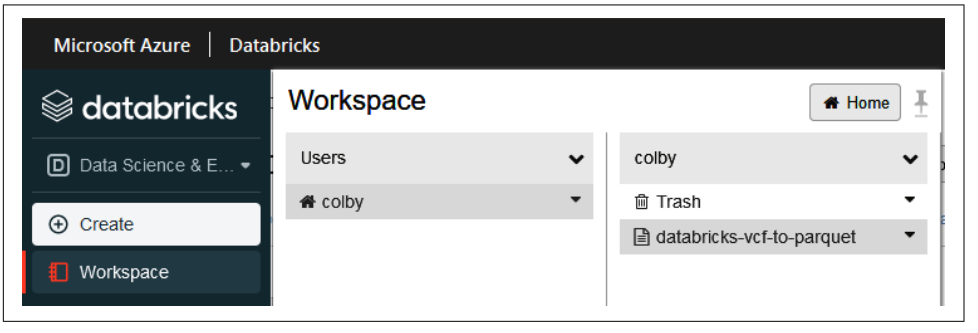


Figure 5-27. Locating the imported notebook

The uploaded notebook contains the following steps:

1. Importing and Configuring Glow:

```
import glow import os
from pyspark.sql.functions import *
glow.register(spark)
```

2. Configure Input VCF Path Widget:

```
dbutils.widgets \
    .text("input_vcf_path", "/mnt/genomicsdatalake/ncbidata/...")
```

3. Set Paths from Widget:

```
input_vcf_path = '/mnt/genomicsdatalake/' +
    dbutils.widgets.get('input_vcf_path')
output_parquet_path = os.path.splitext(input_vcf_path)[0] + '.parquet'
```

4. Read VCF File into DataFrame:

```
vcf_df = spark.read.format("vcf") \
    .option("includeSampleIds", False) \
    .option("flattenInfoFields", False) \
    .load(input_vcf_path) \
    .drop(col("genotypes"))
```

5. Write DataFrame to Parquet:

```
vcf_df.write.format("parquet") \
    .mode("overwrite") \
    .save(output_parquet_path)
```

In the previous steps, you may have asked yourself what a widget is and what it is doing for us. Widgets are like parameters that we can set and access from external tools. We can then grab their values from within our code to perform more dynamic

tasks. In this case, we're setting up a widget for the path where the notebook will grab data. This widget's value will get updated via a Data Factory pipeline in a bit.

To test the notebook, fill in the path to your NCBI VCF file in terms of the mounted data lake. For example, mine is located at */mnt/genomicsdatalake/ncbidata/GCA_000001215.4_current_ids.vcf.gz*. Then, run each of the cells (except the cell where we write out the Parquet file).

After the "Read VCF File into DataFrame" cell, your VCF data will be read into a Spark DataFrame. This means that any DataFrame functionality that you want to run (filtering, querying, aggregation, etc.) will now work on this data. This unlocks a ton of analytical capabilities for large amounts of VCF data by reading them into Spark as standard DataFrames.

You should see the first few rows of your VCF data as shown in Table 5-1. Note that Glow performs some schema modifications to the VCF format to handle variations between VCFs generated by different tools or files that have different types of data (for example, multiallelic entries).

Table 5-1. Example output of the VCF data loaded by Glow

contigName	start	end	names	reference Allele	alternate Alleles	qual	filters	splitFrom MultiAllelic
2L	4850	4851	["rs204402353"]	G	["C"]	null	null	FALSE
2L	4859	4860	["rs205679746"]	G	["A"]	null	null	FALSE
2L	4890	4891	["rs202499195"]	A	["T"]	null	null	FALSE

The last cell of this notebook converts the DataFrame into Parquet format as the data is being written back out to the data lake. Next, let's call this notebook from Azure Data Factory to automate this conversion activity.

Orchestrating a Databricks Notebook from Data Factory

Once we have our VCF converter notebook ready to go, we can operationalize this logic using Data Factory.

As you can imagine, data may come in sporadically during a research study. So scheduling a process to run—say, every Tuesday at 8 a.m.—doesn't necessarily make sense. Thus, we can kick off a processing pipeline upon the arrival of data, regardless of when it comes in. This means that we can configure Data Factory to trigger a pipeline by watching for the presence of a new VCF file and then execute this notebook on that data.

Before we switch back to Data Factory, let's generate a token in Databricks for Data Factory to use in a bit.

Access tokens

To allow external services to connect to your Databricks workspace, you'll likely need an access token, which you can create by locating "Settings > User settings" in the side menu, as shown in Figure 5-28.

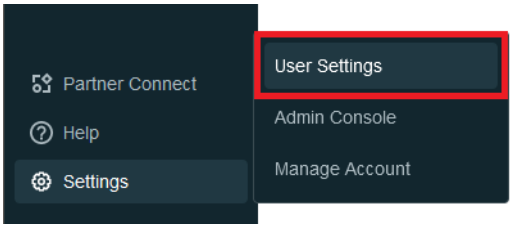


Figure 5-28. Databricks user settings

In the "User Settings" screen, shown in Figure 5-29, click the "Generate New Token" button.

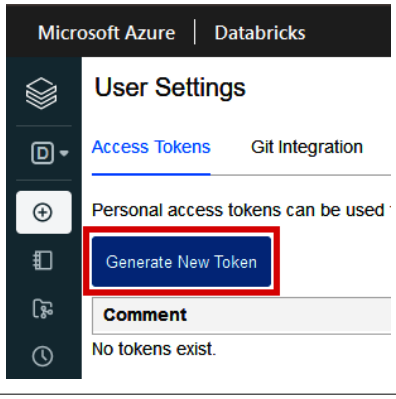


Figure 5-29. Generating an access token

In the "Comment" box, specify what the token will be used for. For example, enter "Data Factory connection" if you're completing the exercise from the section "Building and Triggering Pipelines for Automation" from Chapter 4. See Figure 5-30.

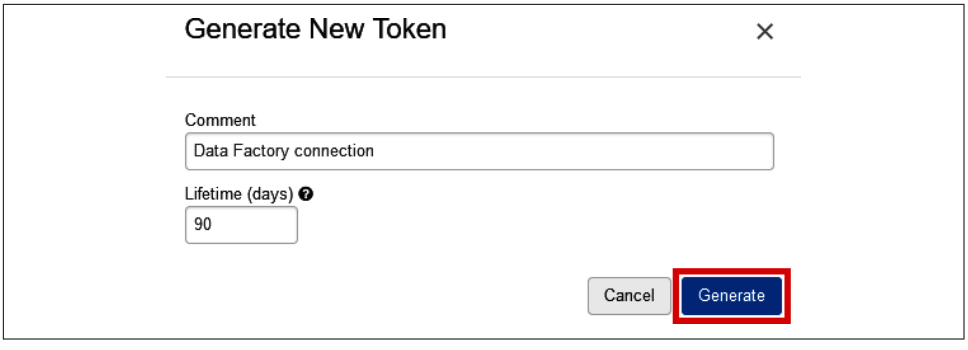


Figure 5-30. Adding a comment and duration to an access token

One the new screen, Databricks will show you your new token, as shown in Figure 5-31. Be sure to copy it somewhere since you won't be able to see it again. (You can always create another one if you forget to save this one.)

Azure Key Vault is a great tool for storing passwords, keys, tokens, and other secrets in the cloud. It interfaces well with many Azure services and prevents you from storing sensitive information as plain text in your code. For more information about Azure Key Vault, visit *https://oreil.ly/a5Ww6*.

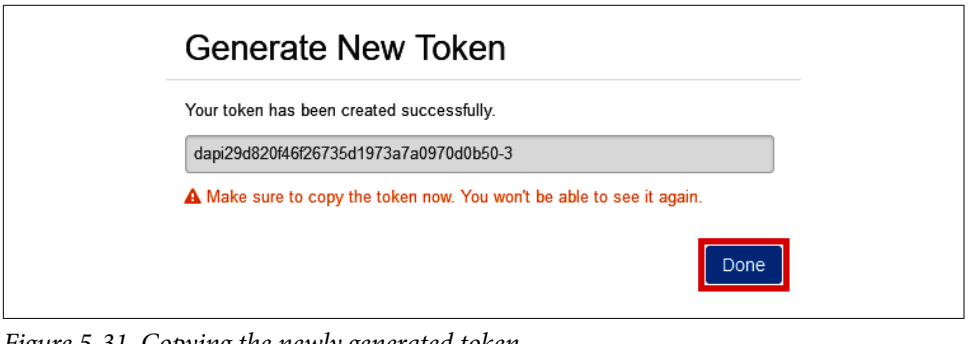


Figure 5-31. Copying the newly generated token

Once you click the "Done" button, you can now see the list of active tokens for your Databricks workspace. You can delete tokens at any time by clicking the "X" button to the right of each token. See Figure 5-32.

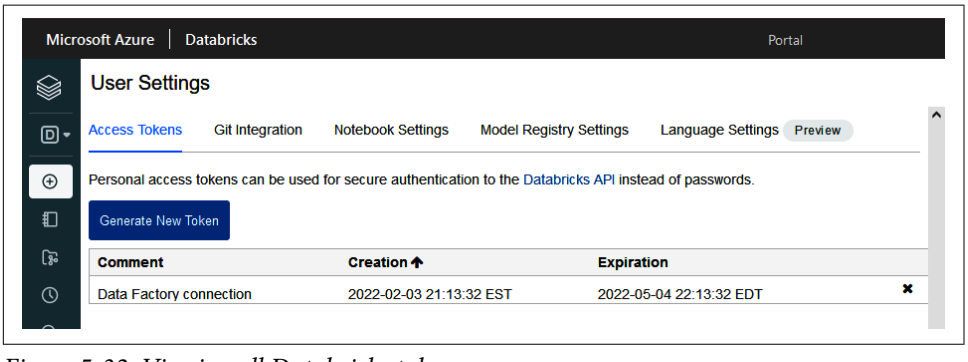


Figure 5-32. Viewing all Databricks tokens

Creating the pipeline

Navigate to your Data Factory that we created in Chapter 4 (either a standalone instance or in Synapse). Create a new pipeline called "Databricks VCF Parsing Pipeline." In the list of "Activities," look for the "Notebook" module under the "Databricks" section. Drag the module over onto the canvas and name it "VCF-to-Parquet Conversion." See Figure 5-33.

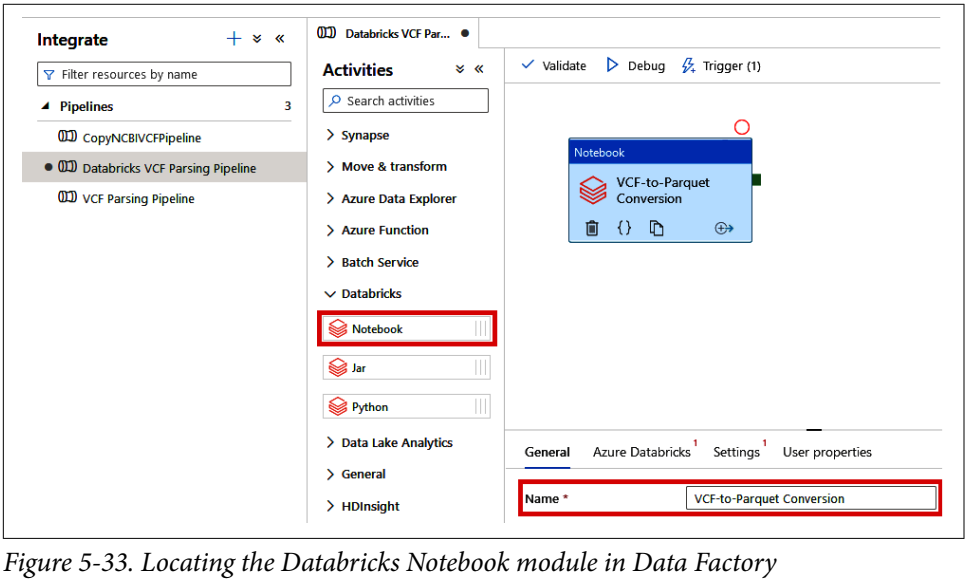


Figure 5-33. Locating the Databricks Notebook module in Data Factory

Under the "Azure Databricks" tab for the module, shown in Figure 5-34, click the "+ New" button to create a new linked service to Databricks.

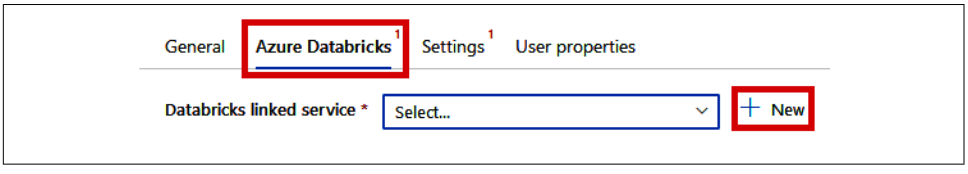


Figure 5-34. Create a new Azure Databricks linked service

In the "New linked service" panel that opens, fill in the following values:

- Name: "GenomicsDatabricks"
- Azure subscription: The subscription you've been using
- Databricks workspace: "genomicsdbx"
- Select cluster: Existing interactive cluster
- Authentication type: Access token
- Access token: The token you created in the last step
- Existing cluster ID: Locate the Glow cluster you're using ("glow_cluster_01")

And click the "Create" button.

Next, on the "Settings" tab, shown in Figure 5-35, click the "Browse" button for the Notebook path. Find your "databricks-vcf-to-parquet" notebook and click it. This will point this module to the conversion notebook when the pipeline runs. Also, create a new Base parameter called *input_vcf_path* with a value of *@concat(pipeline().parameters.input_vcf_folder,'/',pipeline().parameters.input_vcf_file)*. This will concatenate the pipeline parameters that we'll set in a few steps and will pass in this parameter to the widget we created in the Databricks notebook.

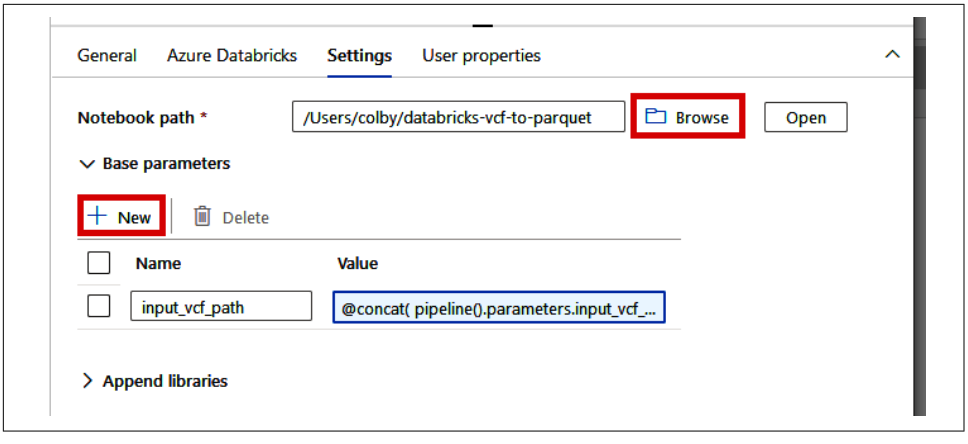


Figure 5-35. Selecting a Databricks notebook to run

For the final set of steps in configuring this pipeline, we need to define some parameters that get passed into the Databricks notebook when it's called from Data Factory. Click off of the "Notebook" module (anywhere in the white background of the canvas) and click the Parameters tab.

As shown in Figure 5-36, create two parameters using the "+ New" button: 1) *input_vcf_folder* and 2) *input_vcf_file*. We can set the default values (to *ncbidata* and *GCA_000001215.4_current_ids.vcf.gz*, respectively) for convenience when testing.

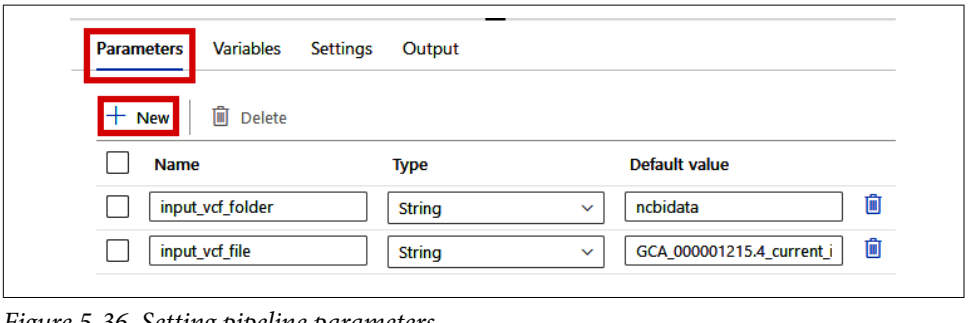


Figure 5-36. Setting pipeline parameters

Near the top of the canvas, locate the "Trigger" button and click "New/Edit," as shown in Figure 5-37. We'll be creating a trigger that kicks off the pipeline when Data Factory detects a new VCF has been loaded into the data lake.

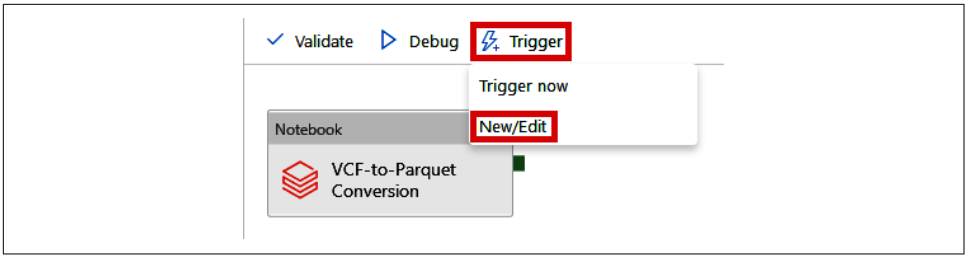


Figure 5-37. Locating the Trigger menu

In the "Add triggers" panel, shown in Figure 5-38, click the "Choose trigger…" drop-down menu and click "+ New."

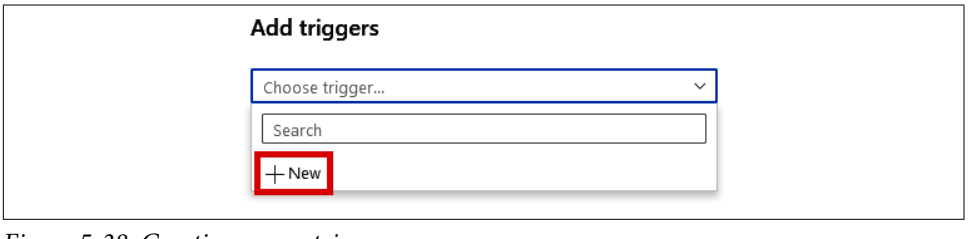


Figure 5-38. Creating a new trigger

On the New trigger panel, fill in the following values (also shown in Figure 5-39):

- Name: "VCF Trigger"
- Type: Storage Events
- Account selection method: From Azure subscription
- Azure subscription: Your Azure subscription.
- Storage account name: The name of the storage account you've been using ("genomicsdatalake01").
- Container name: The name of the container where your VCF file is held ("data").
- Blob path ends with: ".vcf.gz"
- Event: Blob created
- Ignore empty blobs: Yes

And click the "Continue" button at the bottom.

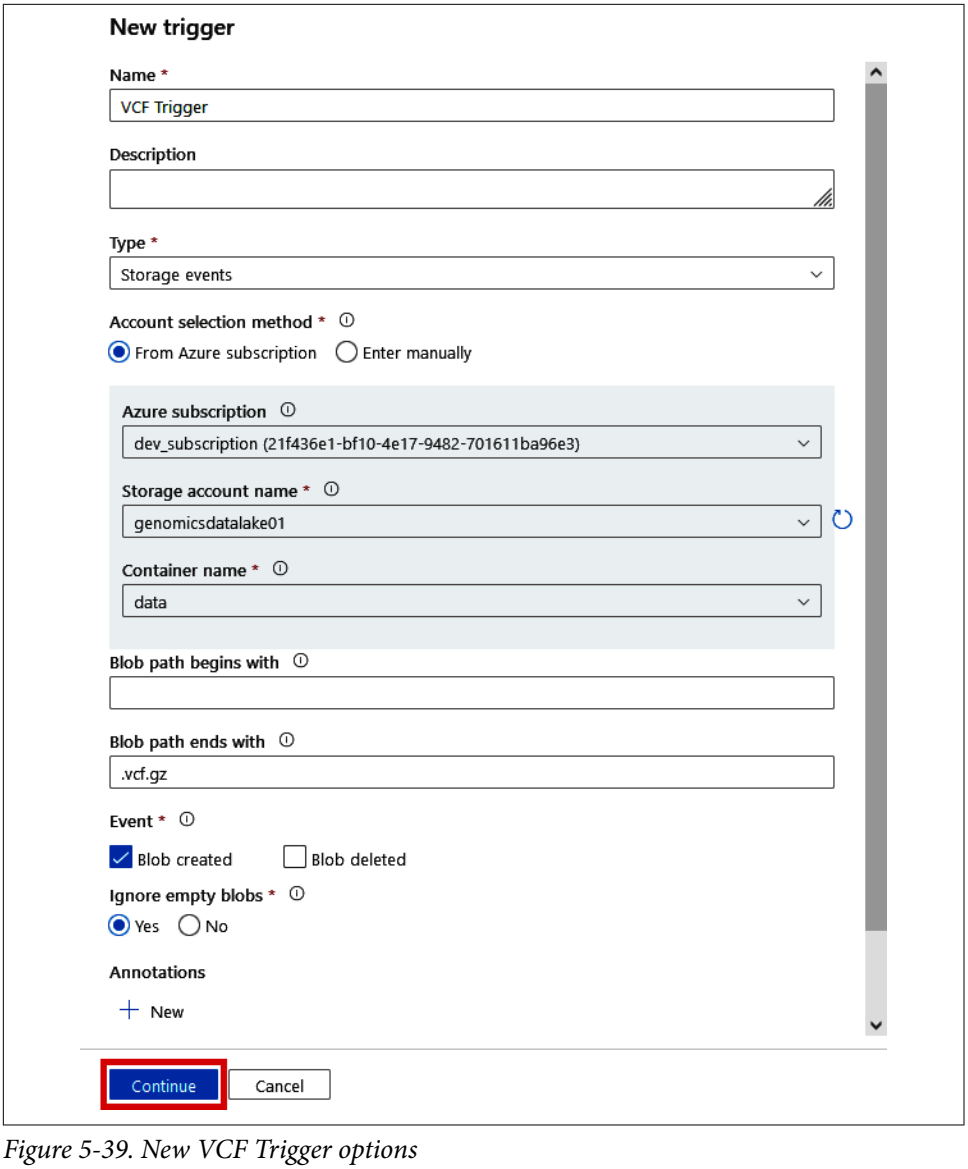


Figure 5-39. New VCF Trigger options

Data Factory will then look in your data lake container to see if it finds any blobs (files) that match the pattern we specified (files ending with *.vcf.gz*). It should find the NCBI VCF file we've been working with. Click the "Continue" button, as shown in Figure 5-40.

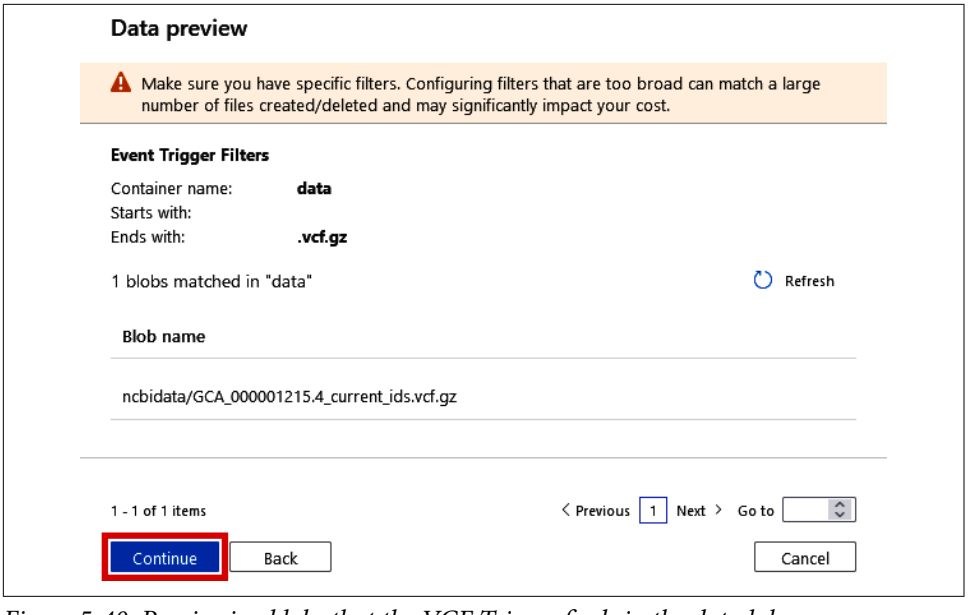


Figure 5-40. Previewing blobs that the VCF Trigger finds in the data lake

On the last panel, shown in Figure 5-41, input the following values for the Trigger Run Parameters:

- input_vcf_folder: "@triggerBody().folderPath"
- input_vcf_file: "@triggerBody().fileName"

And click the "OK" button. This will create the trigger for the pipeline.

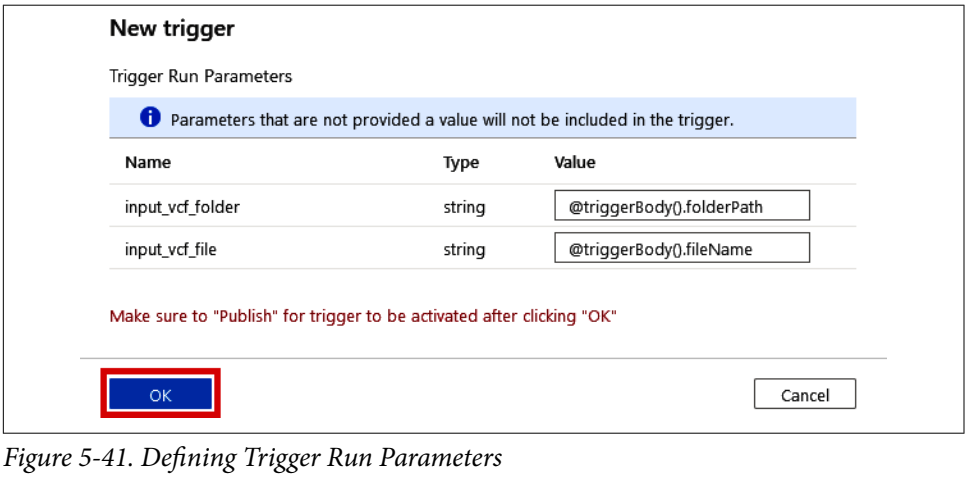


Figure 5-41. Defining Trigger Run Parameters

Click the "Publish all" button at the top of the screen and then click the "Publish" button on the "Publish all" panel to save all of these changes we've just made. See Figure 5-42.

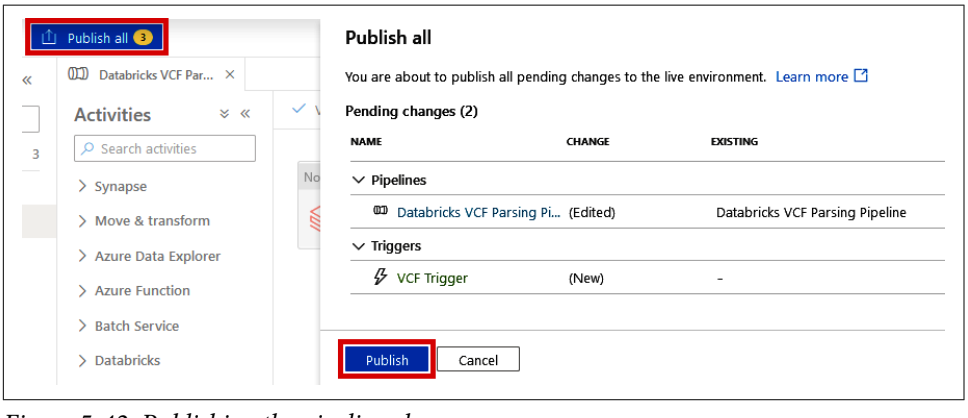


Figure 5-42. Publishing the pipeline changes

Ready to test our pipeline? Click the "Trigger (1)" button at the top of the screen and click "Trigger now," as shown in Figure 5-43.

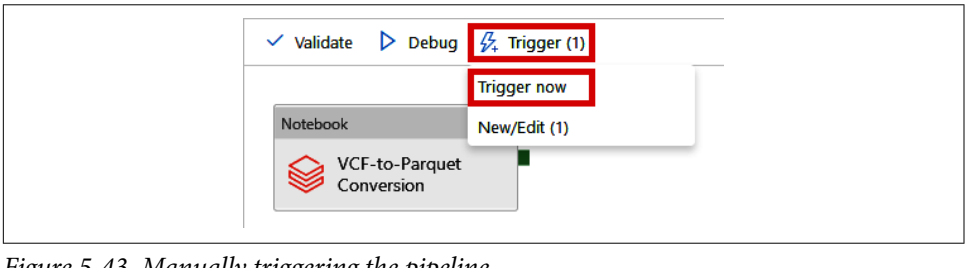


Figure 5-43. Manually triggering the pipeline

In the Pipeline run panel, shown in Figure 5-44, you'll notice that it will have filled in the parameters for us based on our pipeline parameter defaults. Click the "OK" button to trigger the pipeline. Remember that these two parameters will get concatenated as a path and will be passed into the "input_vcf_path" widget in the Databricks notebook.

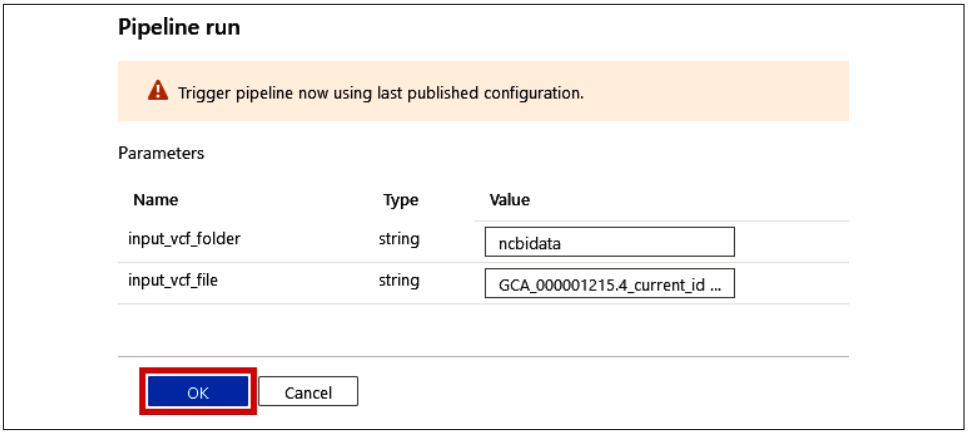


Figure 5-44. Checking the parameters of the manual pipeline trigger

You'll see a small notification pop up in the top-right corner of the screen, as shown in Figure 5-45. Click the "View pipeline run" link to see the triggered run in Data Factory.

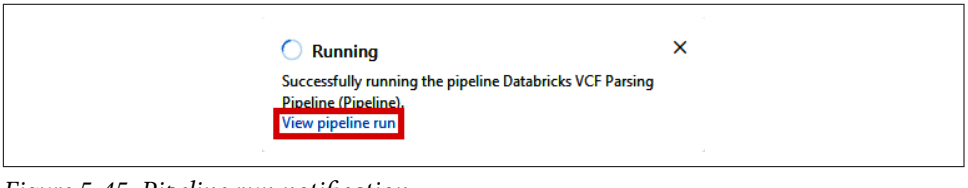


Figure 5-45. Pipeline run notification

This will bring you to the activity screen for this triggered run, which should look something like Figure 5-46. You can sit and wait for the notebook to execute from here, or you can click the button with the glasses icon to get the link for the corresponding Databricks job run.

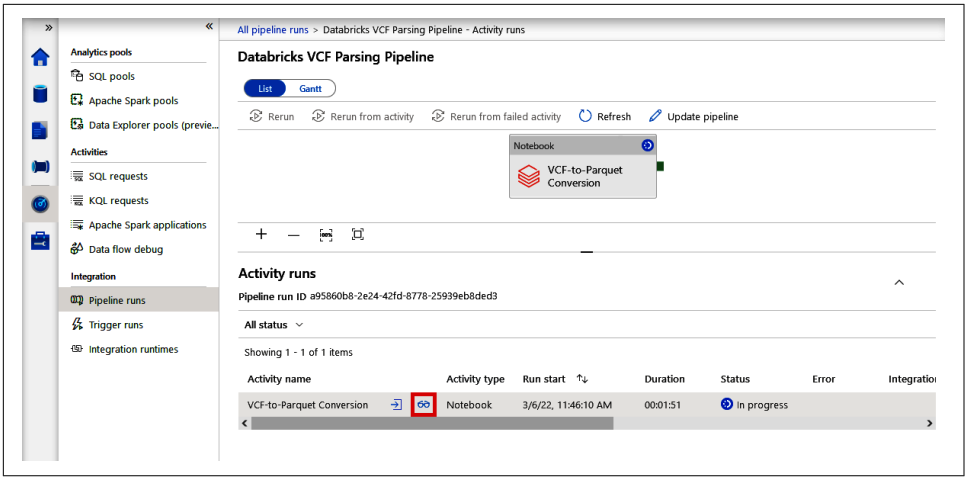


Figure 5-46. Pipeline run activity screen

On the Databricks Run screen, shown in Figure 5-47, you can see a more detailed view of what's being run. This also shows the parameter(s) passed in, the cluster information, and more.

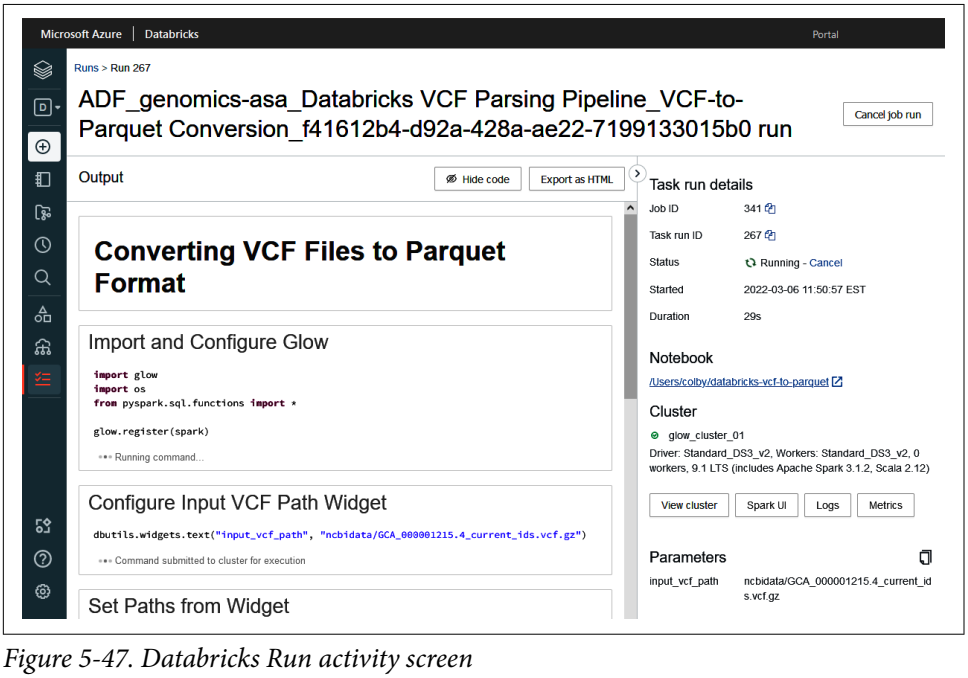


Figure 5-47. Databricks Run activity screen

Once the run succeeds, you see "Status Succeeded" with a green checkmark on both the Databricks side and in Data Factory. You can also check to see if the Parquet file was written successfully in your data lake back in the Azure Portal. See Figure 5-48. (Note that Parquet files are partitioned based on the level of distribution in Databricks. So you may see multiple files that begin with *part-*.)

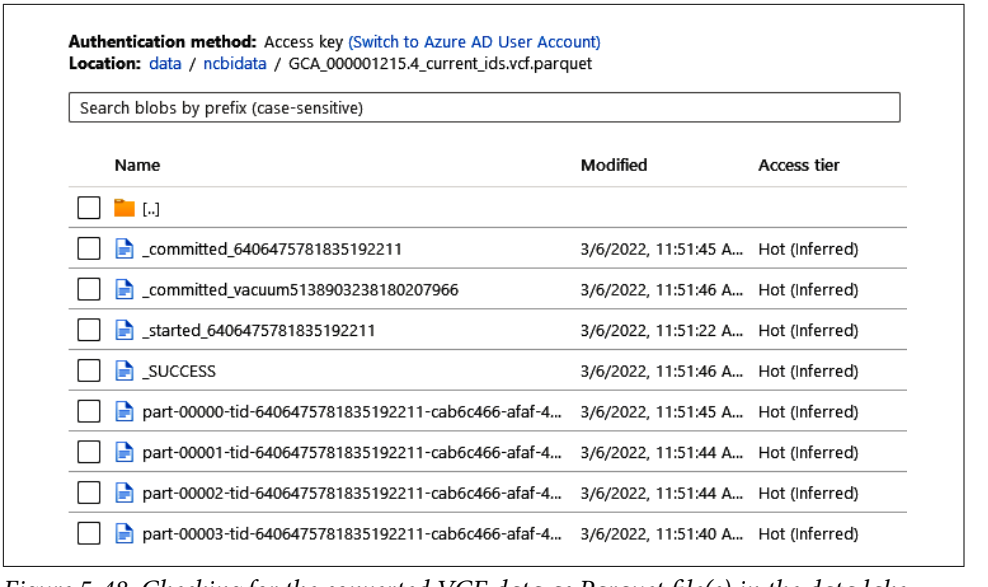


Figure 5-48. Checking for the converted VCF data as Parquet file(s) in the data lake

That's it! You've now created your first connected pipeline that orchestrates activities in Databricks from Data Factory. Our pipeline gets triggered by the presence of new VCF files in the data lake and executes a Databricks notebook automatically. This provides a scalable and automatable way of processing genomics data through various bioinformatics tasks in Databricks together with Data Factory.

A big goal of this chapter is to showcase how we can use various cloud tools both interactively and in an automated way. As a bioinformatician, I thrive at developing new pipelines and analyzing results to gain scientific insights, but my time is better spent not having to manually click a "run" button over and over again.

A Brief Interlude About Distributed File Formats

In the previous two sections, I've completely glossed over the fact that we're reading in human-readable text files (like CSVs for VCF files) and writing them out in a potentially unfamiliar, nontext format.

Since Spark is a massively parallelized system, it is often beneficial to use file formats that enable the scalability when reading and writing data.

Parquet

The Parquet format is commonly used in Spark workloads as it is natively supported. Parquet was developed in 2013 by a joint effort between Twitter and Cloudera.

Parquet is a column-oriented data format, meaning data is stored in columns. This allows for faster retrieval of data in queries where the entire row isn't always needed.

This format was designed for efficient data storage and retrieval, which is exactly what we need when reading and writing large amounts of data from a data lake. Parquet files are often written in multiple, smaller physical files based on the level of parallelism from the compute system doing the writing. This enables us to write data faster and also read the data in more efficiently later.

Finally, the Parquet format provides efficient data compression automatically and also includes the schema of the data it contains. So when you go to read in a Parquet file, you won't have to define the column names and types as you're probably used to doing when reading CSVs or other formats.

Delta

The other format I haven't yet mentioned is one that is gaining popularity within the Databricks world: Delta.

The Delta format was developed by Databricks but is now open source. The goal of this format is to keep many of the best features of Parquet and also add in additional useful capabilities. The main feature that Delta gives us is ACIDity in our data. ACID is a set of properties from the database world that attempt to ensure data integrity and validity even in the event of an error.

The letters stand for the following four features:

Atomicity
: This specifies that parts of a transaction (a data operation) are split into units of work. Each "unit" must succeed or fail completely, and if any of the units fail, the entire transaction is considered to have failed. This helps to avoid corrupt or partially changed data in the event of an error or even something external like a power outage.

Consistency
: Sometimes called "Correctness," this feature ensures that each transaction can only take the data system from a valid state to a valid state. This helps to avoid corruption due to illegal transactions.

Isolation
: Given that many data operations could be executed at the same time, Isolation ensures that transactions that execute concurrently will leave the data system in

the same state as if each transaction were to have been executed sequentially. This feature is to ensure that the order of operations does not change the result.

Durability
: This final feature handles the state of the data once a transaction has been completed and ensures that the changes will remain intact even if an error occurs afterward (such as a server crash or a sudden power outage). Physically, this often means ensuring that the information is written to the permanent data storage (hard drives, etc.) rather than being left in volatile memory (RAM).

Previously, ACID was mainly possible only in database systems. However, with the gaining popularity and utility of data lakes, Delta allows us to ensure these features are there as well.

In addition to ACIDity, Delta gives us the ability to "time travel." That is, we can access or revert to earlier versions of data. Delta also allows for schema evolution and scalable metadata to help you grow with your data over time.

Delta is easy to use and is natively supported in Databricks. In fact, if you want to try Delta on the notebook from the section "Automating Variant Data Processing," simply change the last cell where we wrote out in Parquet format:

```
vcf_df.write.format("delta") \
    .mode("overwrite") \
    .save(output_parquet_path)
```

Once you write data out in the Delta format, you can also read it in a familiar way. Plus, you can go back in time and grab an earlier version of the file if it has changed:

```
vcf_df = spark.read.format("delta") \
    .option("timestampAsOf", "2022-01-01") \
    .load(output_parquet_path)
```

As you can see, this feature could be incredibly important in the genomics research when experimental reproducibility is paramount. Whether it's VCF data or maybe some ever-changing clinical trial data, you need your data to be stable enough so that you can perform analyses and repeat those analyses if asked. With Delta, even if data changes, your previous versions will still exist, allowing you to reproduce virtually any analysis as needed. So now your code isn't the only thing version-controlled… your data can be, too!

You can learn more about Delta at *https://delta.io*.

Using Other Tools in Databricks

In addition to using Glow in Databricks for reading bioinformatics data and performing some tertiary analyses, you can use some additional features in Databricks that are useful for bioinformatics.

Single-Node Bioinformatics Tools

If you have traditional bioinformatics packages that you use in Python, R, or some Java jars, we don't have to say goodbye to them just because we're in Databricks. We can install virtually any package from PyPI, CRAN, or Maven or upload *.jar* files on our cluster. This makes them available right from your notebook.

Within the "Workspace" panel in Databricks, shown in Figure 5-49, right-click anywhere and select "Create > Library."

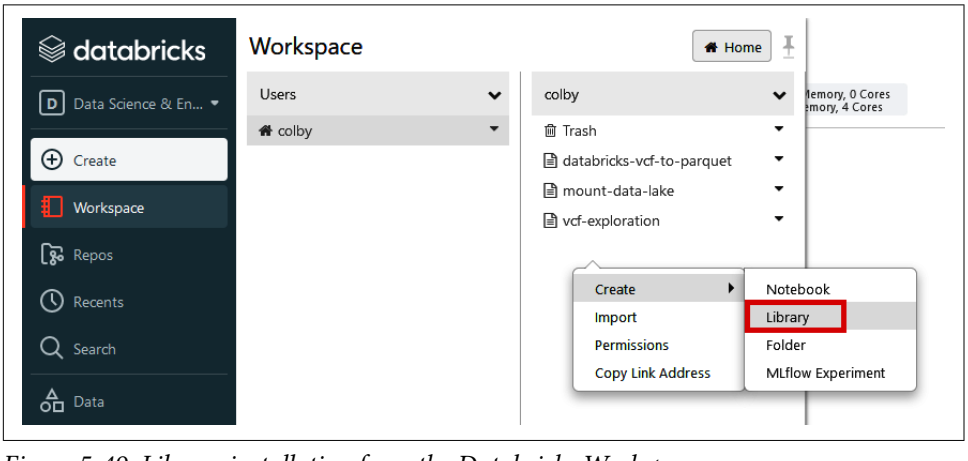


Figure 5-49. Library installation from the Databricks Workspace

This will open a "Create Library" page, shown in Figure 5-50, where you can upload a package or select a package to install from various package repositories.

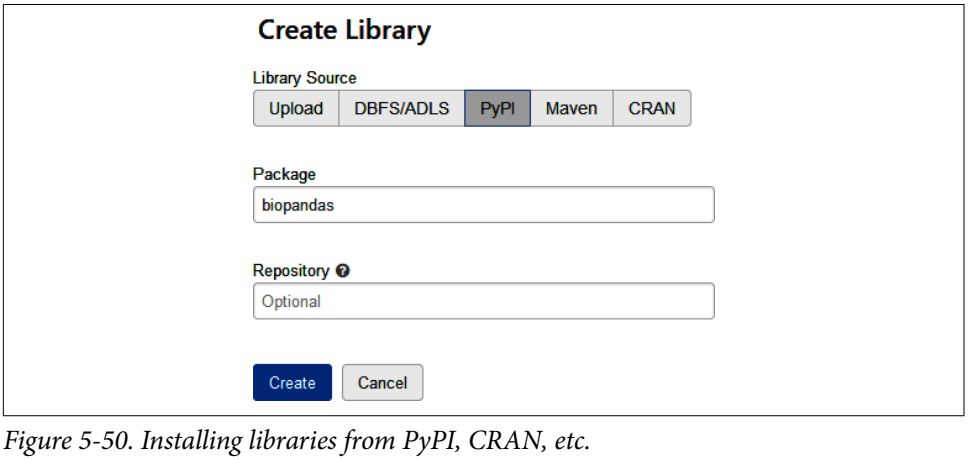


Figure 5-50. Installing libraries from PyPI, CRAN, etc.

The benefit of using the libraries inside Databricks isn't necessarily for scale but for data access. Since Databricks allows for us to mount our genomics data lake, we can use that data directly from our notebook without having to download any data. This is perfect for working with large files. We can simply point to the mounted location as if the file was on our filesystem and use our favorite libraries on this data as if we were working on our local laptop.

Koalas

While you're learning how to use PySpark in Databricks (or Spark in general), you'll likely notice there's a lot to learn when it comes to the core set of DataFrame functions. Even if you are a Python pro, the syntax in PySpark is a bit different and can sometimes be frustrating.

Luckily, there's now an open source project from Databricks called Koalas that attempts to bridge the gap between the Pandas DataFrame (which is single-node) and the Spark DataFrame (which is distributed across a cluster). The Koalas logo is shown in Figure 5-51.

Figure 5-51. Koalas logo

This package allows you to use familiar Pandas-like syntax but perform distributed operations on a Spark backend. Specifically, Koala implements the Pandas DataFrame API on top of Spark and provides nearly identical syntax to Pandas. By reducing this learning curve, Pandas pros can use Databricks to scale all sorts of data transformation tasks, making Koalas a great tool for bioinformaticians looking to use Spark to scale their work. Once you convert your DataFrame to Koalas, you get the best of both worlds: the syntax of Pandas with the performance and scale of Spark!

```
## Load Data
### Load data using Pandas
pandas_df = pd.DataFrame(data, index=index)

### Load data using Koalas
```

```
koalas_df = ks.DataFrame(data, index=index)

### Load data using Spark
spark_df = spark.createDataFrame(data)

## Convert to Koalas
### Convert a Pandas DataFrame with Koalas koalas_df_from_pandas =
ks.from_pandas(pandas_df)

### Convert a Spark DataFrame with Koalas koalas_df_from_spark =
spark_df.to_koalas()
```

You can learn more about Koalas by visiting *https://koalas.readthedocs.io*.

Pandas on Spark

The Koalas library was created for Python pros who were familiar with Pandas-based functionality but were having to learn a whole new syntax in PySpark to perform very similar tasks. That being said, I feel that the Koalas library is the temporary solution to bridging this gap between Pandas and PySpark.

Over time, more and more of the Pandas API will be integrated natively into PySpark. In fact, beginning with version 3.2, Spark comes with a new way of scaling existing Pandas-based Python logic by simply changing a single line of code. This works by replacing the backend of Pandas with Spark SQL. So you can write normal Pandas code, but the Spark SQL engine is doing the work at scale.

We can now import the PySpark flavor of Pandas like so:

```
from pyspark.pandas import read_csv
df = read_csv("mydata.csv")
```

This currently works on data reading tasks, some math calculations (such as computing the standard deviation of a column), joins, and handling streaming data. The Pandas API on Spark outperforms basic Pandas even on single-node environments due to the way the Spark SQL logic is optimized behind the scenes.

This functionality will become more seamless in the coming year such that your Pandas code can be executed with a distributed Spark backend without changing very much in your code at all.

Hail

The last tool I want to mention is similar to Glow in that it provides a more scalable way to work with genomics data. Hail is a package maintained by the Neale lab at Harvard Medical School that was created to simplify the analysis of genomic data using a familiar DataFrame-like format. Hail is an open source Python-based tool that provides additional data types and methods to scale multidimensional structured data. This is especially useful when preparing large amounts of genomic data for tasks

such as a genome-wide association study (GWAS). The Hail logo is shown in Figure 5-52.

Figure 5-52. Hail logo

Hail provides a Python-based API that, under the hood, uses primitives for distributed queries and linear algebra implemented in Scala, Spark, and C++. Instead of traditional Spark DataFrames, Hail uses its own powerful and distributed data structure that combines features of matrices and DataFrames called MatrixTable:

```
import hail as hl
## Load in some VCF data
mt = hl.import_vcf('/mnt/genomicsdatalake/my_vcf.vcf')

## Filter to rare, loss-of-function variants
mt = mt.filter_rows(mt.variant_qc.AF[1] < 0.005)
mt = mt.filter_rows(mt.csq ==
'LOF')

## Run variant QC
mt = hl.variant_qc(mt)

## Filter for variants that are >95% call rate and >1% frequency
mt = mt.filter_rows(mt.variant_qc.call_rate > 0.95)
mt = mt.filter_rows(mt.variant_qc_.AF[1] > 0.01)
```

While the Hail package works differently than Glow in terms of reading and filtering data, the overall purpose is similar: to scale the analytics of large amounts of genomics data.

Similar to the Glow cluster creation in Figure 5-13, we can create a Hail cluster by referencing a different Docker image that enables Hail to run on Databricks. See Figure 5-53.

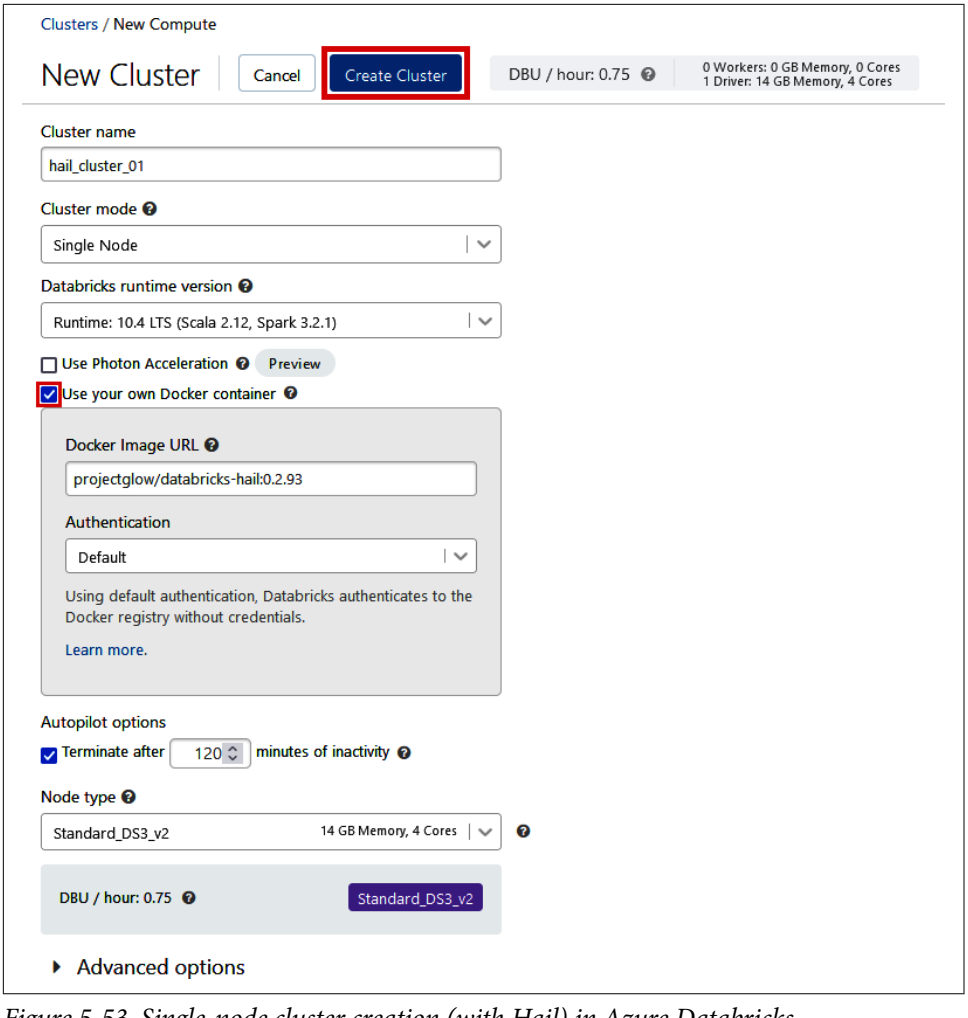


Figure 5-53. Single-node cluster creation (with Hail) in Azure Databricks

Check out both Glow and Hail to see which you like best and which best suits your needs. Then you can create larger clusters with either package and scale the reading and processing of -omics data in Spark with ease!

To learn more about using Hail on Spark, visit *https://hail.is*.

Databricks and Spark provide a plethora of opportunities to scale up various data processing tasks from basic data cleansing to machine learning. Plus, using libraries such as Glow, we can even perform some bioinformatics-specific tasks at scale as well.

In this chapter, we've covered the basics of Spark and Azure Databricks and have shown how Glow unlocks the capabilities of the Spark engine to scalably analyze our

genomics data. In the next chapter, we will be looking at Azure Machine Learning, a powerful platform for managing machine learning models (that secretly also works well for bioinformatics tasks).

In summary, Apache Spark provides a powerful distributed data processing engine for everything from basic data analysis to machine learning. Using Azure Databricks, we can quickly create clusters and utilize the cloud-based notebook environment. With Glow, we can extend the capabilities of Spark to support bioinformatics workloads.

We can now read in various genomics-specific file formats into Spark DataFrames, which unlock our capabilities to explore and transform the data at scale. These tools provide a more usable replacement for traditional HPC environments, alleviating the need for queue-based job submission and giving users the ability to interactively code in the cluster environment. Azure Databricks seamlessly integrates with other platforms in Azure, such as Azure Data Factory. This provides a way to automate large workloads simply by operationalizing notebooks of code. With Data Factory and Databricks, we can trigger the Spark jobs when new variant data becomes available in our data lake.

CHAPTER 6

Azure Machine Learning

Machine learning (ML) is likely top of mind these days given the onset of game-changing artificial intelligence, from autonomous driving to protein structure prediction.

In bioinformatics, we can use ML to predict drug sensitivity, classify an organism's resistance to a drug, or model mutations in a gene.

Any of these use cases often requires a considerable amount of data. Thus, the cloud is a perfect place for ML as we can get access to ample data from our genomics data lake and scale our modeling tasks with cloud compute services.

In this chapter, we'll speak more broadly about machine learning, focusing on how to use Azure Machine Learning to scalably train and track your modeling tasks. Plus, despite its name, the Azure Machine Learning system is great for non-ML activities as well, including bioinformatics tasks.

How to Scale Machine Learning Tasks

When we start training ML models on our local workstation, we experience slow model training due to resource limits. Also, while performing a larger-scale search for a great predictive model, the process could be executing in an inefficient way.

During the process of performing k-fold cross-validation or tuning an algorithm's hyperparameters, the code may be executing iterations of the process in serial—that is, one after another, waiting on the previous step to complete before running the next iteration. This is unnecessary as we don't need the results from a previous step before proceeding to the next.

Luckily, there are R and Python packages that help to locally parallelize this work. However, we're still limited by the resources on our local machine.

For example, let's say you have 10GB training dataset on a laptop with 16GB of RAM and you're wanting to test various scaling methods for your dependent features. Your local machine will run out of memory once the system creates a scaled version of the input training dataset. So even if we could parallelize the training, we will hit a hard limit in terms of memory. See Figure 6-1.

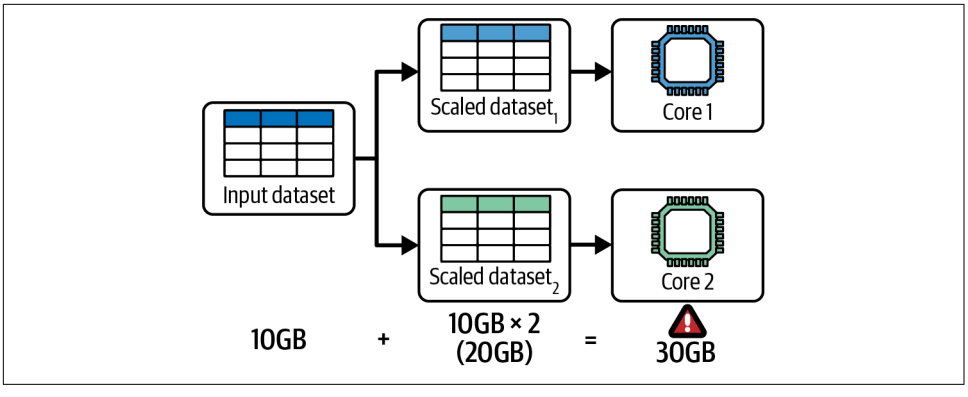


Figure 6-1. Example of the RAM limitations of local parallelization

How we can tackle this is by using multiple machines in a cluster and distributing the tasks to each node. This means that a single node will have enough resources to perform its task without having to share with other tasks.

In Azure Machine Learning, the distribution of model training tasks is an innate feature. We can simply specify the size of a training cluster and provide the options we want the system to try (hyperparameters, algorithms, scaling methods, etc.). Then the platform will handle the logic to distribute the tasks and train individual models using our specifications. See Figure 6-2.

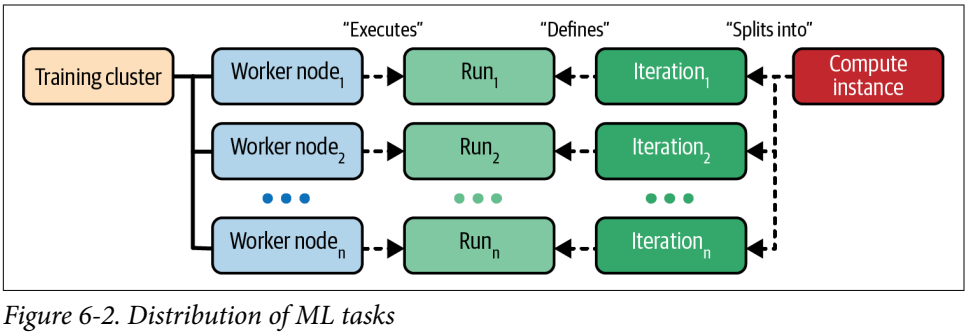


Figure 6-2. Distribution of ML tasks

Cluster-based distribution alleviates the resource limitations of a single computer while continuing to scale individual iterations of an ML training process to test various scaling methods, hyperparameters, algorithms, and more.

Next, let's create your Azure Machine Learning workspace and start training some ML models.

Creating an Azure Machine Learning Workspace

Creating an Azure Machine Learning workspace works just like any of the other services we've spun up thus far. Navigate to the Azure Portal and click "+ Create a resource." On the "Marketplace" page, shown in Figure 6-3, search for "machine learning" and locate the Azure service from Microsoft. Click the "Create" button.

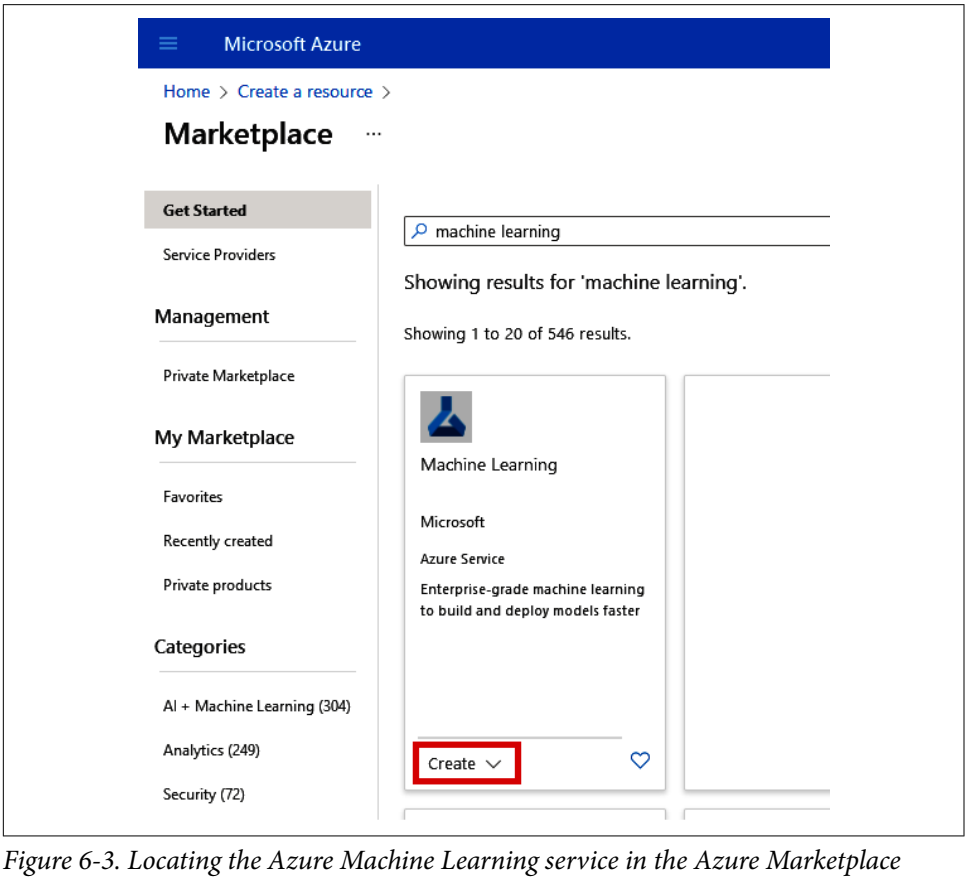


Figure 6-3. Locating the Azure Machine Learning service in the Azure Marketplace

On the next blade, shown in Figure 6-4, you'll need to fill in some information about the ML workspace. Select the "Subscription," "Resource group," and "Region" that you've been using for the other services in this book. For the "Workspace name," give the service a descriptive name like "genomics_ml." This will autofill the "Storage account," "Key vault," and "Application insights" fields. For the "Container registry" field, if you don't already have one, you'll likely need to click the "Create new" link

below the drop-down box and create a new container registry. (I named mine "genomicsacr" and chose the Standard SKU. We'll use this container registry later when we start creating custom Docker images.)

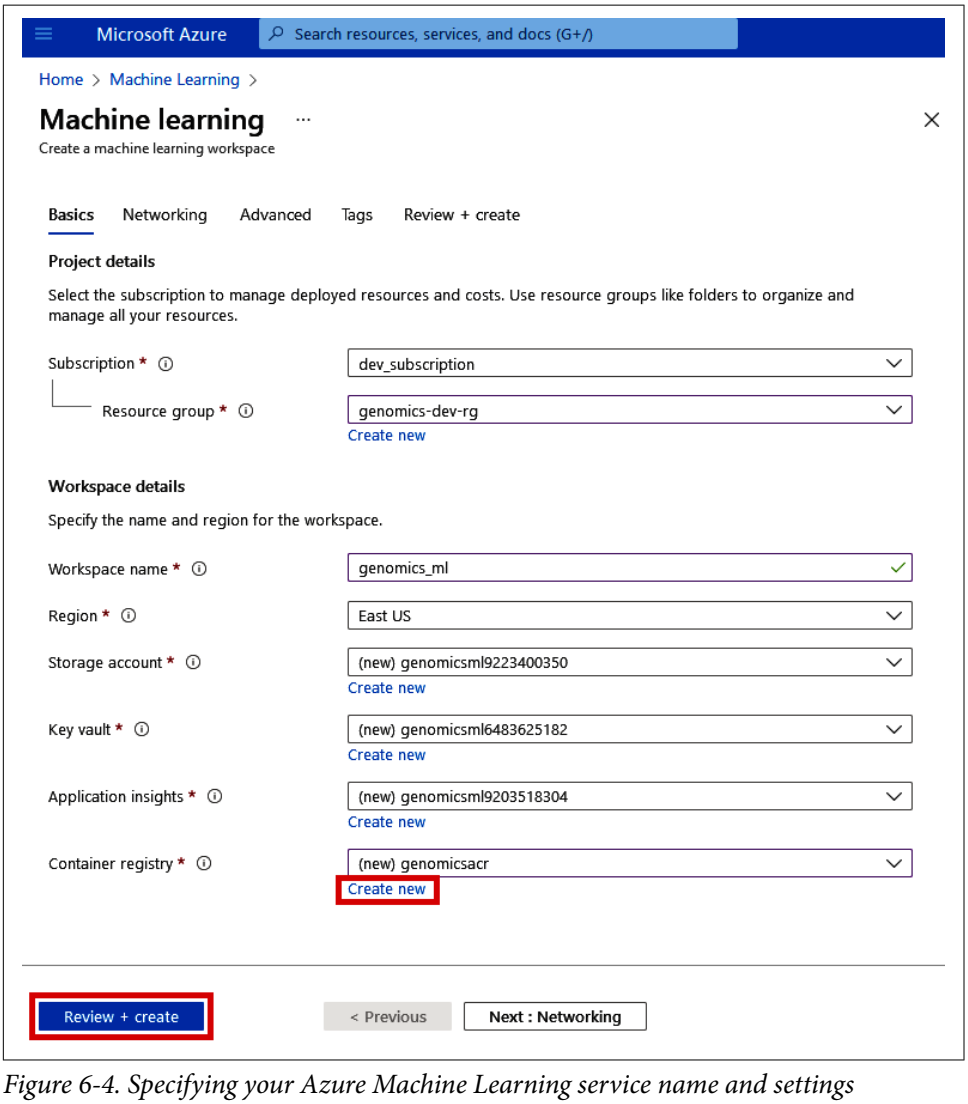


Figure 6-4. Specifying your Azure Machine Learning service name and settings

Once you have all of these fields filled out, you can click the "Review + create" button at the bottom of the screen. This will run the validation check to make sure you can provision the ML workspace with these settings. If it passes the validation, click the "Create" button at the bottom of the "Review + create" tab.

Easy peasy, right? In just a few minutes, Azure will have your workspace ready. Once the deployment finishes, locate the ML service (mine's called "genomics_ml") in your resource group in the Azure Portal. Click the "Launch studio" button, as shown in Figure 6-5.

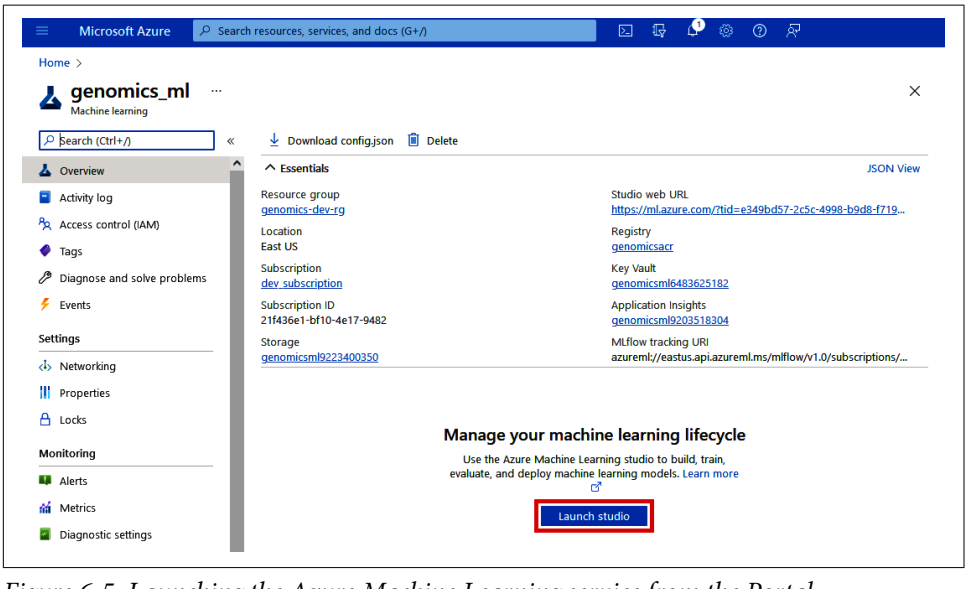


Figure 6-5. Launching the Azure Machine Learning service from the Portal

(Alternatively, you can navigate your web browser to *https://ml.azure.com*, which will take you to the same place without having to go through the Azure Portal.)

The Azure Machine Learning service UI (called the Studio) will open, and as you can see in Figure 6-6, there are lots of features of this service.

At its core, Azure Machine Learning is a powerful tool to train machine learning models, track their performance, and also manage the other parts of the machine learning lifecycle. This includes managing input training datasets and deploying models as APIs. We'll begin exploring some of these features in the next few sections.

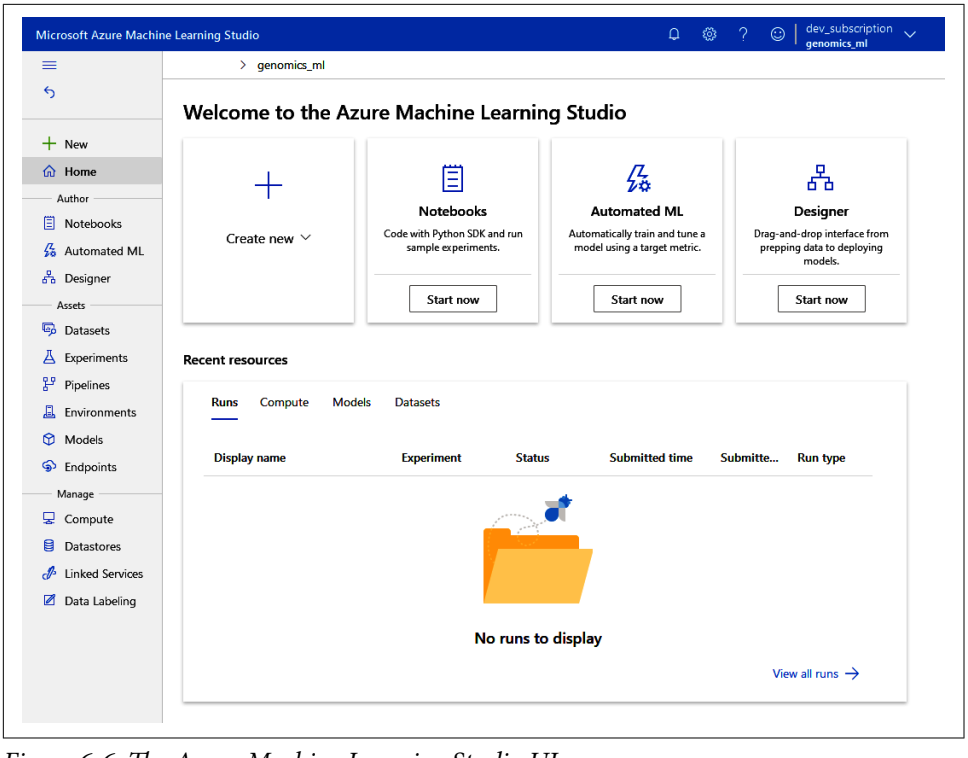


Figure 6-6. The Azure Machine Learning Studio UI

Training a Drug Sensitivity Model

For our next activity, we'll be training a drug sensitivity model using some publicly available datasets. We'll be creating the model within Azure Machine Learning so that you can see how the platform assists with experiment management and model tracking.

Since this is a genomics book, I thought we'd train an oncology drug sensitivity model using some gene expression data. Not only will we create an ML model, but we'll also employ explainable ML techniques to derive some biological insights out of our model.

Creating a Compute Instance in Azure Machine Learning Studio

Inside of Azure Machine Learning, the main way to begin coding is within a compute instance, which is a fancy term for a virtual machine that comes preinstalled with lots of data science tools. If you're familiar with writing code in a Jupyter notebook or RStudio, the compute instance provides these tools via the web.

To create your compute instance, click "Compute" from the side menu, as shown in Figure 6-7. Under the "Compute Instances" tab, click the "+ New" button.

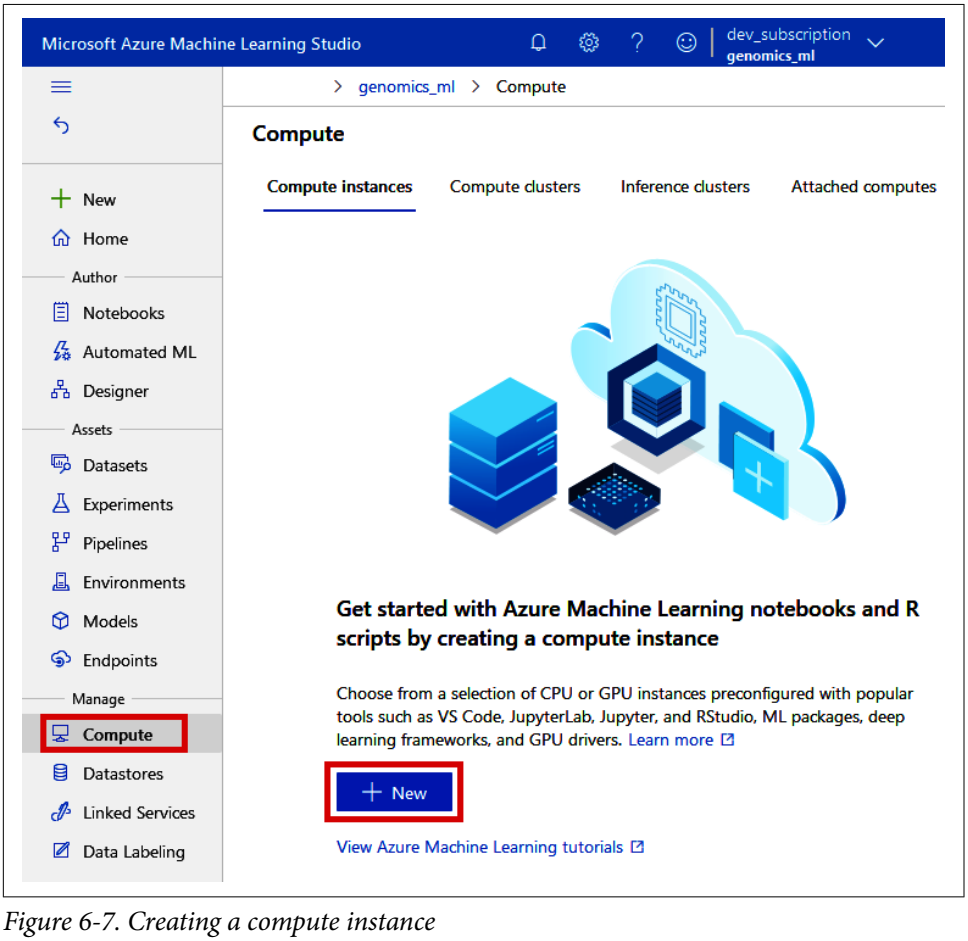


Figure 6-7. Creating a compute instance

Next, give your compute instance a name, such as your name plus a number ("cford0" in my case). We'll just be using CPU-based compute instances, but as you can see, you can also create GPU-based ones as well. Under the "Virtual machine size," pick a smallish size such as "Standard_DS3_v2." (Since you'll usually do your machine learning training on a separate compute cluster, which we'll create in a bit, there's no need to have a giant compute instance unless you plan on doing some heavy lifting directly in your notebook.) Once you're ready, click the "Create" button at the bottom. See Figure 6-8.

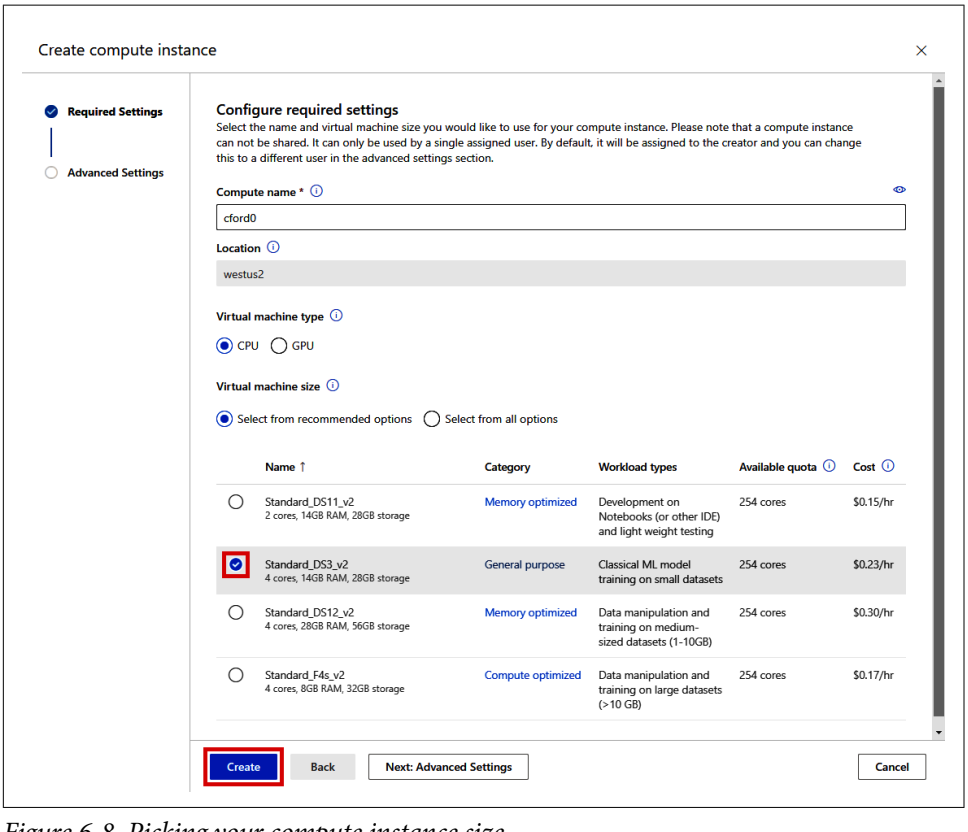


Figure 6-8. Picking your compute instance size

This will take a few minutes to spin up. While we're waiting, let's get our training data ready.

Datastores and Datasets

Azure Machine Learning is a perfect tool for research and development in highly regulated environments such as healthcare as it enables full traceability to your machine learning workflow. This all starts with connecting your workspace to any data sources that you will be using as datastores. Datastores hold the connection information, credentials, etc., so that you can access data in these locations as necessary.

From the workspace UI, shown in Figure 6-9, click the "Datastores" link on the side menu and click the "+ New datastore" button at the top of the screen.

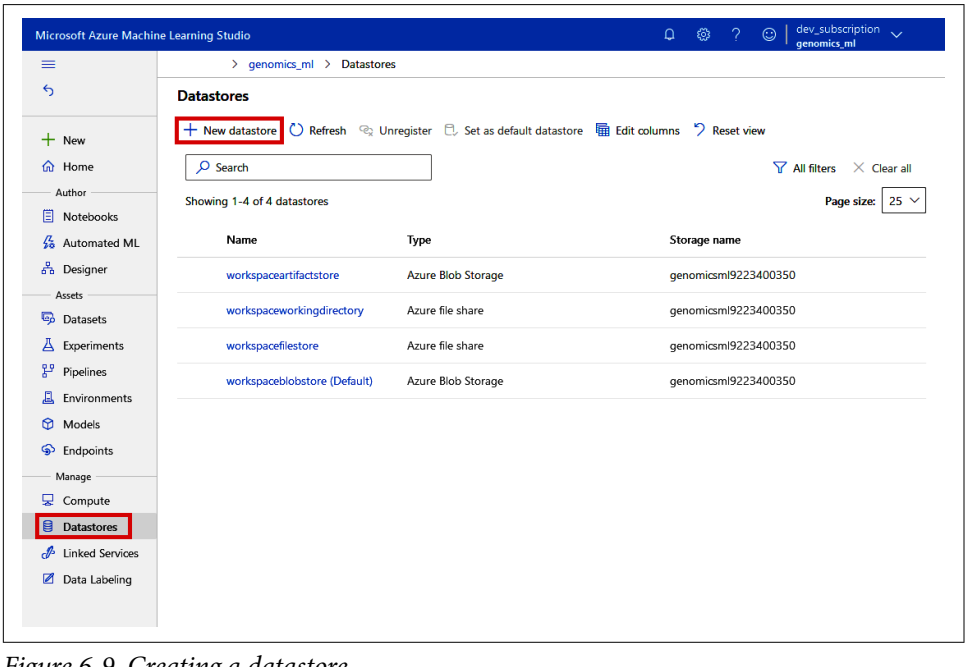


Figure 6-9. Creating a datastore

On the "New datastore" blade that opens, name your datastore "genomics_datalake" and select "Azure Data Lake Storage Gen2" for the "Datastore" type. Note the other options I've selected in Figure 6-10.

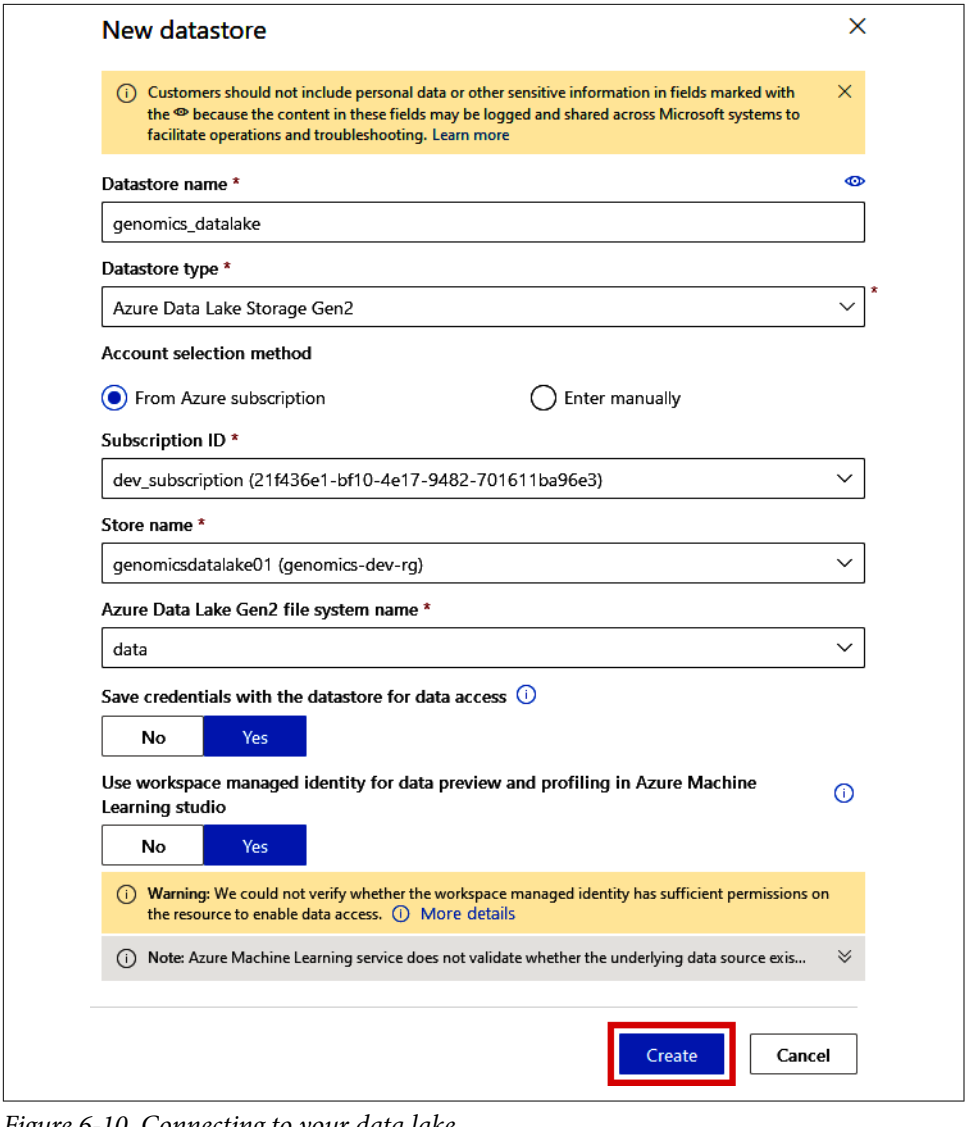


Figure 6-10. Connecting to your data lake

Once we have a datastore, we can then register a particular file or series of files from that datastore as a dataset. A dataset is like a versioned reference to a data source location along with a copy of its metadata. The data isn't actually copied anywhere (so you don't need to worry about costs), but datasets aid in your ML workflow by serving up data as needed for model training.

For this activity, we'll be training a model using some data that I've prepared, which is available in the supplementary code repository under */06_amls/docetaxel_sensitivity.zip*. (Make sure to unzip it.)

About the data

I have created a dataset that we'll use for the model training in a bit. This data includes a series of gene expression values and a column that tells the sensitivity to the drug docetaxel.

Each row represents a cancer cell line that was used to determine the IC50, the "half maximal inhibitory concentration" of the drug, which is basically a measure of the sample's sensitivity to docetaxel. This data came from the Genomics of Drug Sensitivity in Cancer (GDSC) database.[1]

Docetaxel is a microtubulin inhibitor commonly used as a chemotherapy agent against multiple types of cancer, including breast, head and neck, prostate, and non-small-cell lung cancer.

Each column of this dataset includes gene expression values, determined by RNA-seq, for 19,177 genes. Gene expression is represented here as Log2(TPM + 1) values.[2] This expression data is from the Cancer Cell Line Encyclopedia (CCLE).[3]

A sample of the data is shown in Table 6-1. Note that the L10_IC_50 column is the Log10(IC50) sensitivity value, and all the other columns are genes.

Table 6-1. Docetaxel sensitivity with gene expression

L10_IC_50	TSPAN6	TNMD	DPM1	SCYL3	…	ABCB8	H3C12	SND1
−1.66437	2.643856	0	6.219556	3.41819	…	5.578637	0.214125	7.035184
−2.2658	2.9855	0	6.778734	4.130931	…	4.857483	0.526069	6.941811
−2.19477	4.574707	0	6.632414	1.937344	…	4.179511	0	6.633141
…	…	…	…	…	…	…	…	…
−2.66481	4.863938	0	5.977967	2.553361	…	5.441616	0	7.417431
−3.03437	5.914086	0	6.749668	2.809414	…	5.276869	0.495695	7.052894
0.299115	4.130931	0	5.817879	3.891419	…	4.414136	0.464668	5.510646

Download this file to your local computer and then upload it to your genomics data lake (see the section "Getting Started with Data Movement" in Chapter 4). Then, back in your Azure Machine Learning workspace, click the "Datasets" tab on the side

1 See the GDSC Data Source at *https://oreil.ly/N2ev8*.

2 TPM = transcripts per million reads.

3 See the CCLE Data Source at *https://oreil.ly/rCuSY*.

menu. Under the "Registered datasets" tab, click "+ Create dataset > From datastore," as shown in Figure 6-11.

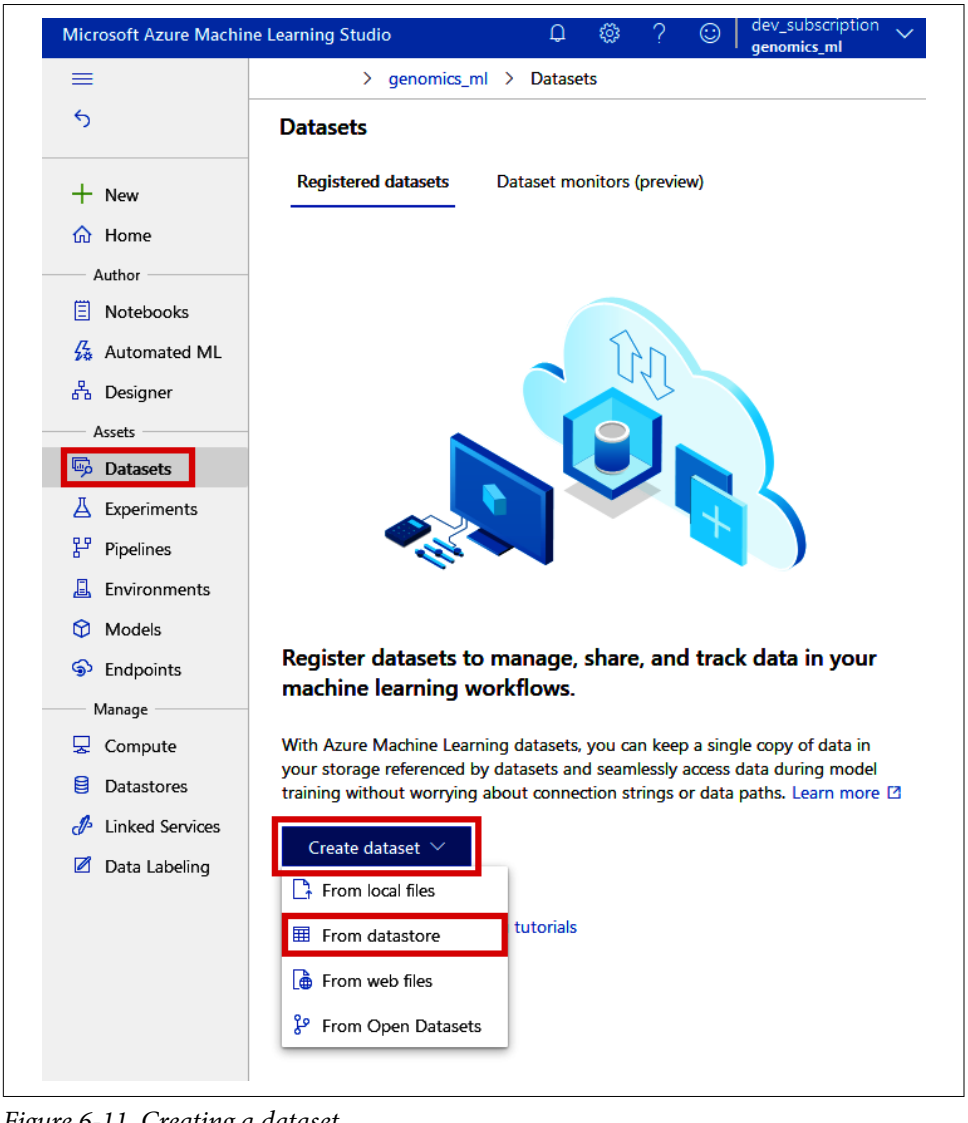


Figure 6-11. Creating a dataset

In the "Create dataset from datastore" blade that opens, shown in Figure 6-12, name the dataset "docetaxel_sensitivity" and click the "Next" button at the bottom.

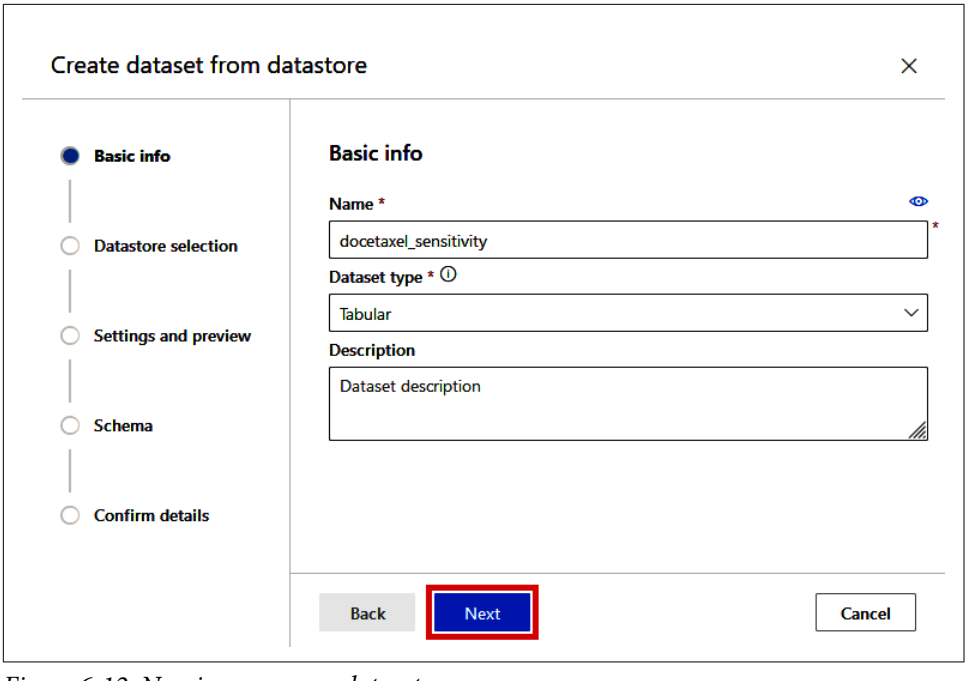


Figure 6-12. Naming your new dataset

On the "Datastore selection" screen, shown in Figure 6-13, select the datastore that you just recently created. Then, under the "Path" section, click the "Browse" button and locate the file in your data lake. Also, click the checkbox beside "Skip data validation." Once finished, click the "Next" button at the bottom.

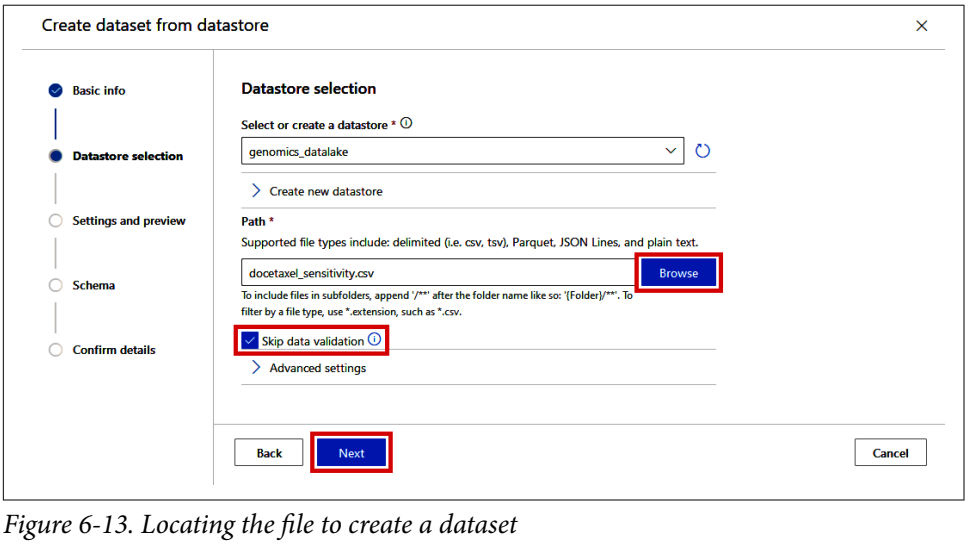


Figure 6-13. Locating the file to create a dataset

On the "Settings and preview" screen, this is where you would see a preview of the data (if the workspace is told to perform data validation). We shouldn't need to change any of the settings for this file (as long as the settings say the "File format" is "Delimited" and that for "Column headers," it says "Only the first file has headers"). The same goes for the "Schema" screen. You shouldn't need to change any column types in the schema for this file (as long as Azure Machine Learning detects that all the columns are of "Decimal" type). Click the "Next" button through both of these screens. On the "Confirm details" screen, shown in Figure 6-14, you can now see the information about the dataset that's about to be created. Once you've verified your selections, click the "Create" button at the bottom. This will take just a few moments for the dataset to become available for use.

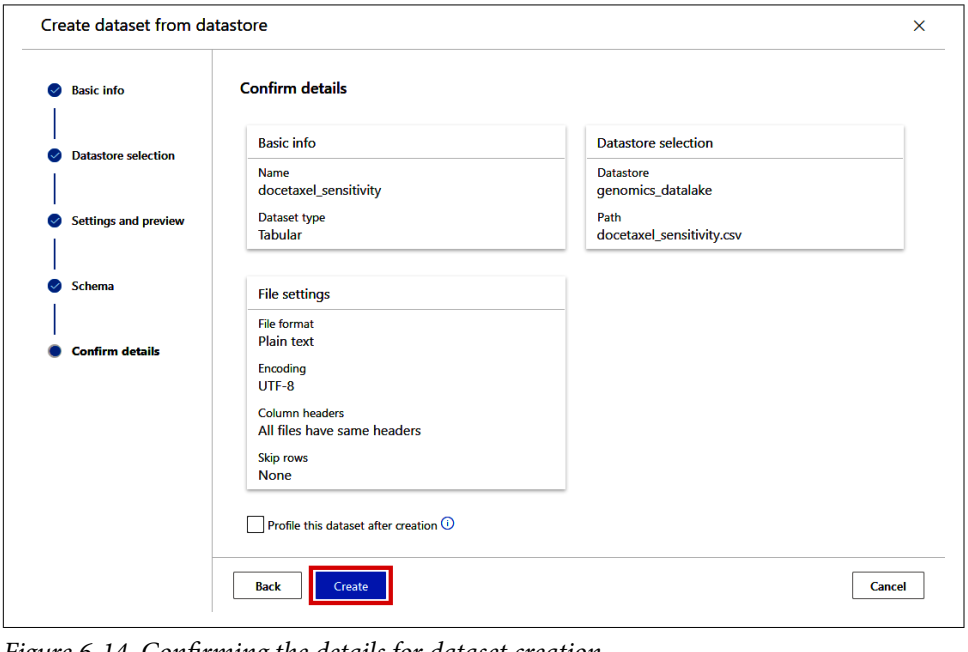


Figure 6-14. Confirming the details for dataset creation

This will take just a second to create the dataset, which will then be shown on the "Datasets" screen in the workspace. See Figure 6-15.

You can also create datasets from the Python SDK. This is useful since the code can be tracked under Git version control, which adds visibility into the dataset creation for traceability purposes.

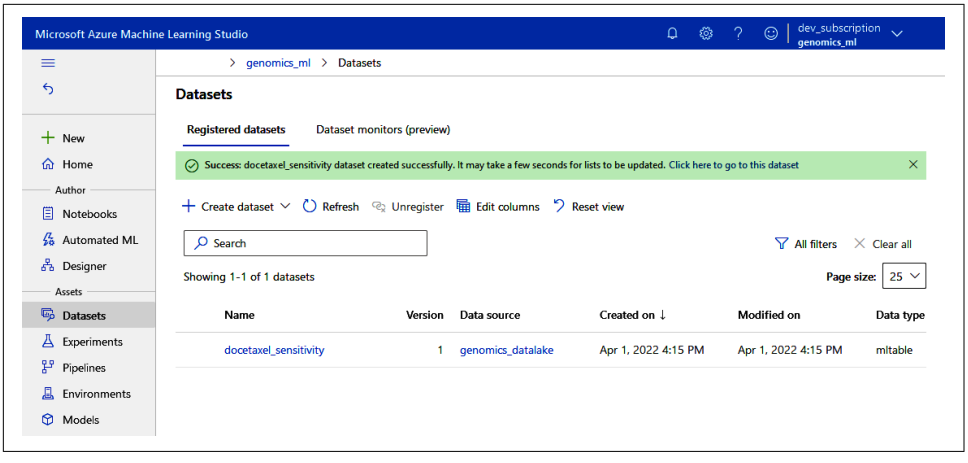


Figure 6-15. Locating the docetaxel sensitivity dataset

Experimenting with Cluster-Based Training

Now that we have our training data registered as a dataset (and hopefully our compute instance is up and running), navigate back to the "Compute" screen and click the "JupyterLab" link next to your compute instance, as shown in Figure 6-16. Note that there are multiple development applications (IDEs) that are available via the web, including RStudio Service, VS Code, and even a browser-based terminal.

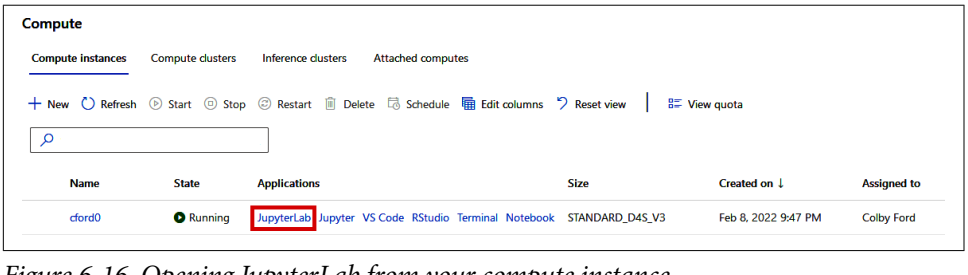


Figure 6-16. Opening JupyterLab from your compute instance

This will open up a new JupyterLab tab in your browser. From the supplementary materials for this book, upload the */06_amls/Docetaxel_ML.ipynb* notebook file. See Figure 6-17 to locate the upload button.

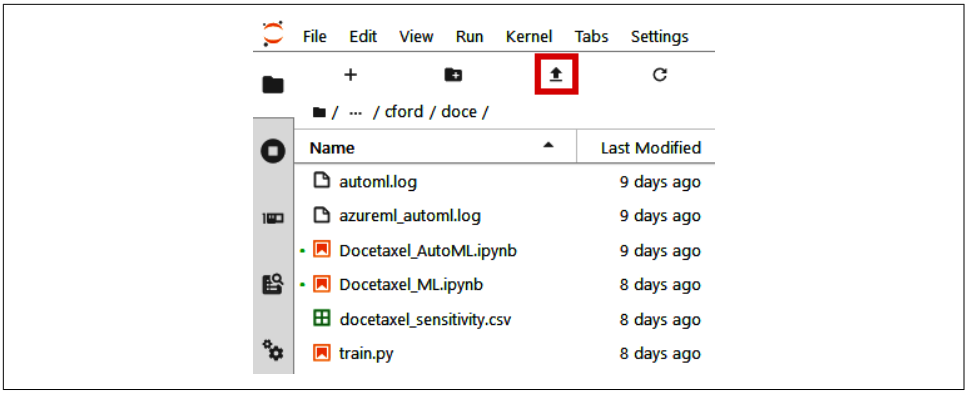


Figure 6-17. Uploading notebooks into JupyterLab

Have a look through the *Docetaxel_ML.ipynb* notebook. In this notebook, we have the following steps:

1. Load in some libraries and connect to the Azure Machine Learning workspace backend.
2. Grab the docetaxel sensitivity dataset and convert it to CSV.
3. Create the training script that gets run on the cluster.
4. Create the compute cluster target.
5. Configure HyperDrive.
6. Run the experiment.

Azure Machine Learning's HyperDrive system provides an easy way for us to distribute the training of ML models across various algorithms and hyperparameters. In this notebook, I have configured the HyperDrive run to train multiple XGBoost models using random sets of parameters. In the following code, note that I specify the parameter options (in the "params" object) and how many total combinations I want to try (using "max_total_runs"). The HyperDrive system also attempts to pick the best model based on a single primary metric. In this case, I have chosen to maximize the R^2 value, but you could also minimize error, maximize AUC for classification problems, etc.:

```
## Sample a range of parameter values
# params = GridParameterSampling( params =
    RandomParameterSampling(

        '-n_estimators': choice(1, 5, 100),
        '-max_depth': uniform(100, 5000),
        '-booster': choice('gbtree', 'gblinear', 'dart')

    )
```

```
## Configure hyperdrive settings
hyperdrive = HyperDriveConfig(run_config = script_config,
                              hyperparameter_sampling = params,
                              policy = None,
                              primary_metric_name = 'r2',
                              primary_metric_goal = PrimaryMetricGoal.MAXIMIZE,
                              max_total_runs = 10,
                              max_concurrent_runs = 4)
```

I have also written a training script that will get run on each of the nodes of the cluster, taking in a set of parameters from the previous HyperDrive configuration. Here is a sample of this script:

```
## Set the input parameters
...

n_estimators = args.n_estimators
max_depth = int(args.max_depth)
booster = args.booster
...

## Get the experiment run context
run = Run.get_context()
ws = run.experiment.workspace
...

## Initialize Algorithm
xgb = XGBRegressor(objective ='reg:linear',
                   n_estimators = n_estimators,
                   max_depth = max_depth,
                   booster = booster,
                   seed = 1337)

## Train Model w/ Cross-Validation (5-fold)
scores = cross_validate(xgb,
                        X_train, y_train,
                        cv=5, scoring=('r2'),
                        return_estimator=True)

...
## Log model performance
run.log('r2', r2_score(y_train, y_train_hat))
...
```

Note that this training script should look quite familiar if you've trained ML models with scikit-learn or XGBoost before. The only difference here is that we are taking in arguments at the beginning of the script from the HyperDrive engine and passing those parameter selections on to the model training process. Toward the end of the script, we're also logging the model performance for each run so that the engine can help pick the best model.

Once you submit this HyperDrive run, you can see the run back in your Azure Machine Learning workspace under "Experiments." See Figure 6-18. If you locate this experiment, you'll then be able to see all of the child runs that get created (one for each of the 10 parameter combinations we configured).

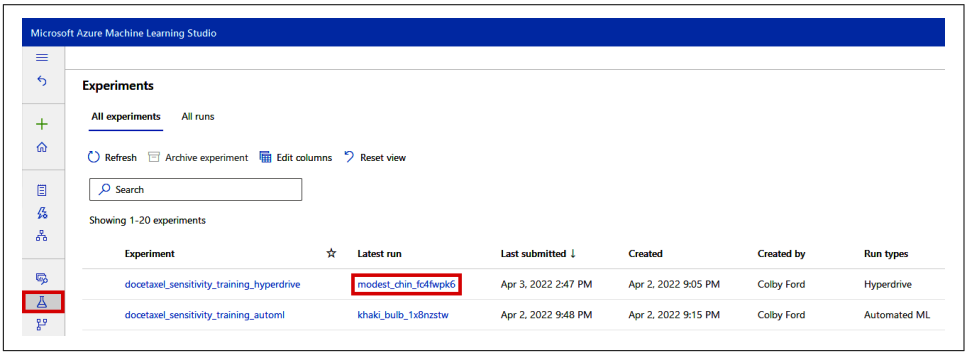
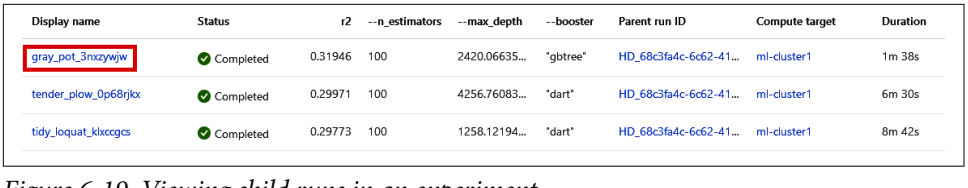


Figure 6-18. Locating Experiments in Azure Machine Learning

You'll see a list of the child runs for the main HyperDrive run, each with their own parameter combination and performance. (In my case, shown in Figure 6-19, I haven't yet found any exceptional models…yet.)



Display name	Status	r2	--n_estimators	--max_depth	--booster	Parent run ID	Compute target	Duration
gray_pot_3nxzywjw	Completed	0.31946	100	2420.06635...	"gbtree"	HD_68c3fa4c-6c62-41...	ml-cluster1	1m 38s
tender_plow_0p68rjkx	Completed	0.29971	100	4256.76083...	"dart"	HD_68c3fa4c-6c62-41...	ml-cluster1	6m 30s
tidy_loquat_klxccgcs	Completed	0.29773	100	1258.12194...	"dart"	HD_68c3fa4c-6c62-41...	ml-cluster1	8m 42s

Figure 6-19. Viewing child runs in an experiment

If you click the name of the best-performing model and locate the "Outputs + logs" tab, as shown in Figure 6-20, you can find the model's pickle file, which you can use for future deployment and scoring of new data.

While this was a very quick walk-through of distributed model training in Azure Machine Learning, I hope it highlights that using this system shouldn't be scary. If you know how to write scikit-learn code on your laptop, we can quickly scale that logic using the Python SDK for Azure Machine Learning. With the SDK, we can do a better job of experiment tracking, version control, etc. With the HyperDrive engine, a few extra lines of code gives us incredible scale for testing various algorithms or parameters.

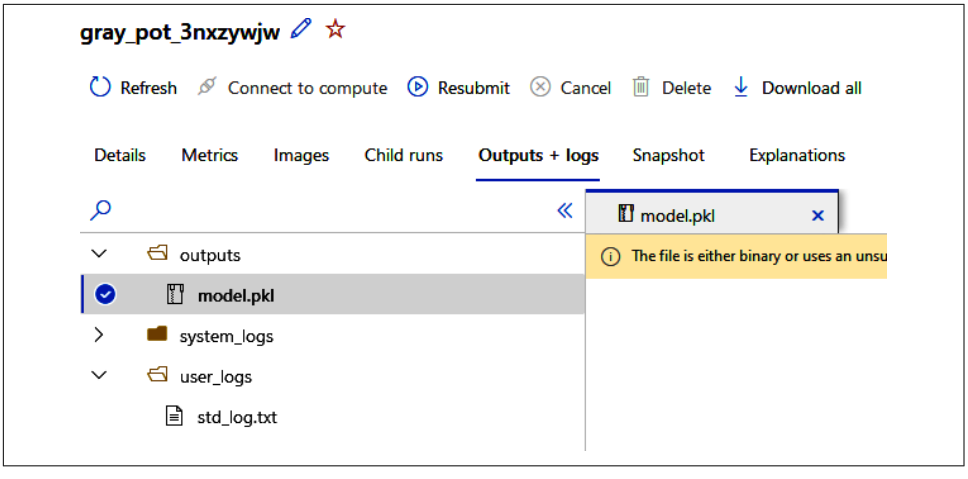


Figure 6-20. Locating the model file in an experiment run

In the next section, we'll be automating this ML training process even further with AutoML.

Automating Model Training with AutoML

Are you ready for some déjá vu? In this section, we're going to perform a similar model training as the previous section. This time, though, we'll be letting the Azure Machine Learning AutoML system do most of the thinking for us.

AutoML (or "automated machine learning") is a newer tool that, given a specific ML task, will train a series of models for us. AutoML tools include R's `caret` and `parsnip` libraries and the Python `PyCaret`, `AutoKeras`, and `Scikit-Optimize` packages. Plus, cloud-based services have AutoML capabilities that include Databricks,[4] Amazon SageMaker Autopilot, and more.

In Azure's AutoML system,[5] we can specify the ML task to be regression, classification, or forecasting. Then, we specify a primary metric (like accuracy, AUC, or RMSE) for the system to try to optimize. Once we have everything set up, the AutoML system will train multiple models using different algorithms, parameter sets, and scaling methods, and will help us pick the best one in the end. AutoML doesn't replace machine learning expertise or your company's team of data scientists. Instead, it allows us to train tons of models, which will still require some machine learning chops to sift through and find the best one for enterprise use.

4 For more on Databricks AutoML, see *https://oreil.ly/GSke1*.

5 For more on Azure ML's AutoML concept, see *https://oreil.ly/Fgs7z*.

Azure Machine Learning's AutoML system has two ways of submitting jobs: the SDK (which we used in the last step) and the Studio's UI. While I would encourage you to check out the UI-based submission system, it can be a bit sluggish when you have datasets with tons of features. So, we'll stick with the SDK method for now.

From the supplementary materials for this book, upload the */06_amls/Docetaxel_AutoML.ipynb* notebook file. In this notebook, we have the following steps:

1. Load in some libraries and connect to the Azure Machine Learning workspace backend.
2. Grab the docetaxel sensitivity dataset.
3. Create the compute cluster target.
4. Configure AutoML.
5. Run the experiment.

You'll notice that this notebook is quite similar to the HyperDrive notebook that we used before. This time, though, we don't need to write our own training script. We simply need to tell the AutoML engine about the task at hand. AutoML can handle regression, classification, and forecasting (timeseries) tasks, and, like HyperDrive, it uses a primary metric to help pick the best model.

In the `AutoMLConfig()`, we specify the number of iterations and cross-validation folds and point to the training dataset. We can also specify which algorithms feature scaling methods we'd like the AutoML system to try:

```
from azureml.train.automl import AutoMLConfig

automl_config = AutoMLConfig(name='docetaxel_sensitivity_automl_config',
                             task='regression',
                             compute_target = training_cluster,
                             training_data = doce_data,
                             label_column_name = 'L10_IC_50',
                             primary_metric = 'r2_score',
                             n_cross_validations = 5,
                             iterations = 25,
                             max_concurrent_iterations = 8,
                             experiment_timeout_minutes = 240,
                             featurization = 'off',
                             enable_voting_ensemble = True,
                             enable_stack_ensemble = True,
                             model_explainability = True)
```

Once you submit this AutoML Experiment, you can navigate back to the "Experiments" section of the Azure Machine Learning workspace and locate your AutoML run, as shown in Figure 6-21.

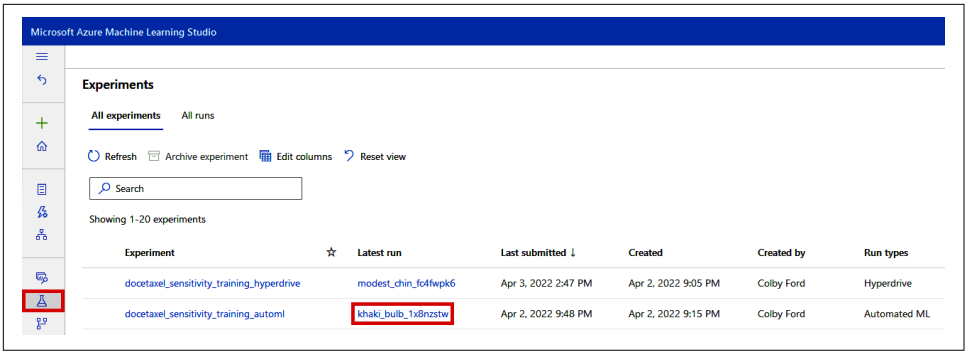


Figure 6-21. Locating Experiments in Azure Machine Learning

You'll notice that the AutoML run has numerous child runs, each using a different algorithm, scaling method, and set of parameters. It will automatically sort the models by the primary metric you chose (R^2, in our case). You'll also notice that AutoML will perform two ensemble methods on your models: VotingEnsemble and StackEnsemble. See Figure 6-22.

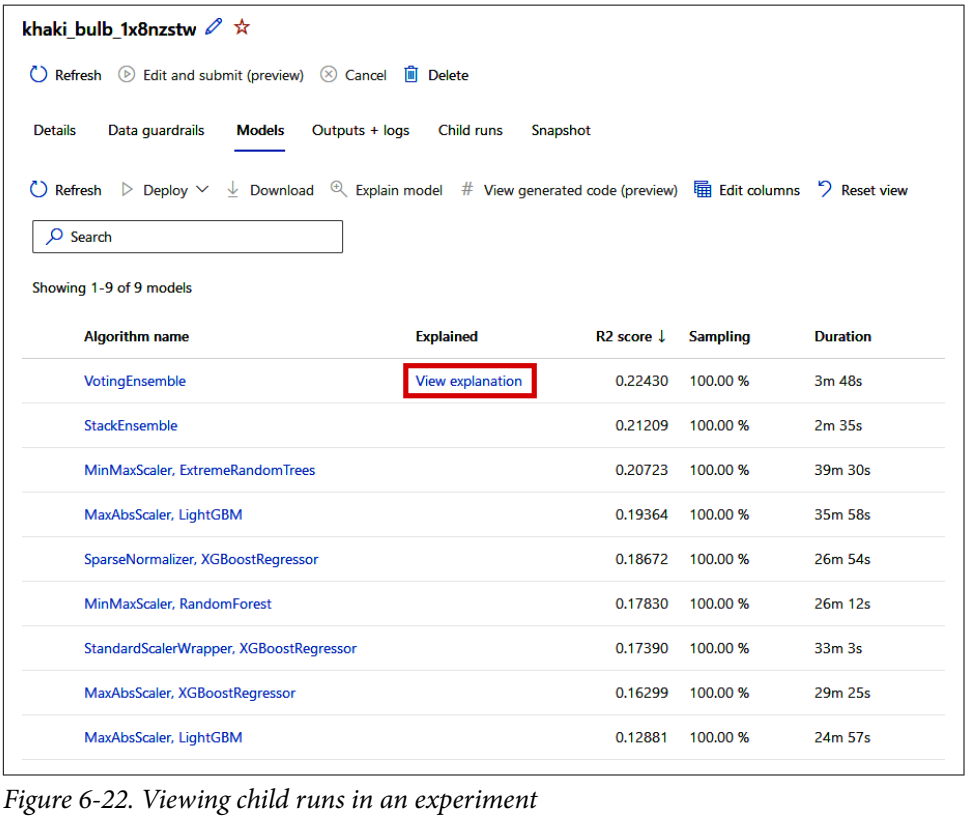


Figure 6-22. Viewing child runs in an experiment

Ensemble models combine the results of multiple individual models to create a composite prediction. In my experience, these ensemble models often outperform any of the singular models. Thus, they are useful to give you an extra lift in model performance. AutoML supports two ensemble methods:

Voting
: Calculates the weighted average of predicted class probabilities (for classification tasks) or predicted values (for regression tasks) and uses this value as the final prediction.

Stacking
: Combines multiple heterogeneous models and trains a metamodel based on the results of the individual models. (This usually trains a logistic regression metamodel for classification tasks or an elastic net metamodel for regression and forecasting tasks.)

As you can see in Figure 6-22, my best model (albeit still terrible) is the VotingEnsemble model. Since we have the "model_explainability = True" flag turned on in the `AutoMLConfig()`, the AutoML system has performed a model explainability run on this model to help us understand how its predictions are made. Let's take a closer look at the inner workings in a bit.

Explainable Machine Learning

A common misconception about ML is that it's all about the predictions. Au contraire! Due to advancements in explainable ML, we can derive feature importance information to tell us how the model arrived at its prediction. This can provide us with the "why" surrounding the relationship between features and the thing we're trying to predict.

Azure Machine Learning has a feature explainability tool built-in, and in the case of AutoML models, explaining your model can be performed automatically.

In the case of your docetaxel sensitivity model, clicking the "View Explanation" link (shown in Figure 6-22) shows that TMEM139, FAM72B, BCL2L1, and CLDN3 are the top four genes, respectively, that aid in the model's prediction. Clicking any of the genes will show you the individual feature importance by record. See Figure 6-23.

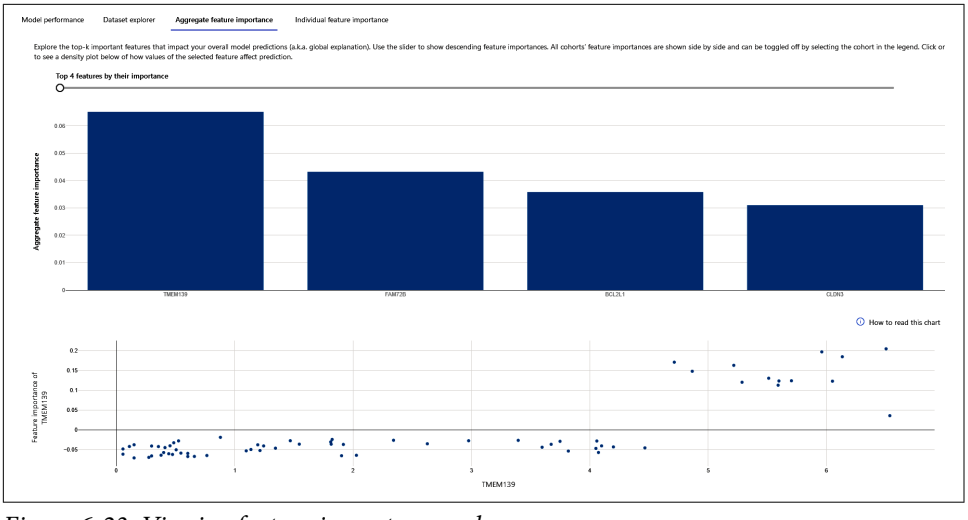


Figure 6-23. Viewing feature importance values

These numbers are SHAP values (SHapley Additive exPlanations). SHAP uses game theory to understand the importance of features in a model.[6]

If we do a little literature search, we can find that TMEM139 is a known biomarker for paclitaxel sensitivity.[7] So it may make sense that docetaxel sensitivity may be similar. Similarly, FAM27B is a biomarker for docetaxel resistance in lung adenocarcinoma[8] and prostate cancer.[9]

As you can see, model explainability in genomics can be extremely useful in the development of companion diagnostics and assay design and for understanding more about the underlying biology of a given condition. It can also help you semivalidate if your model is making valid predictions by telling you what is important, which we can then cross-reference with published information.

6 Learn more about SHAP here at *https://github.com/slundberg/shap*.

7 Peter M. Haverty et al., "Reproducible Pharmacogenomic Profiling of Cancer Cell Line Panels," *Nature* 533 (2016): 333–337, *https://doi.org/10.1038/nature17987*.

8 Yilin Yu, Zhiping Wang, Qunhao Zheng, and Jiancheng Li, "FAM72 Serves as a Biomarker of Poor Prognosis in Human Lung Adenocarcinoma," *Aging* 13, no. 6 (2021): 8155—8176, *https://doi.org/10.18632/aging.202625*.

9 Thiago S. Lima et al., "Molecular Profiling of Docetaxel-Resistant Prostate Cancer Cells Identifies Multiple Mechanisms of Therapeutic Resistance," *Cancers* 13, no. 6 (2021), *https://doi.org/10.3390/cancers13061290*.

I've personally used these methods for understanding drug resistance and in designing new rapid diagnostic tests in my malaria research. So I'd suggest you use this handy feature in any of your upcoming research that uses ML. You never know what you might learn!

Using Azure Machine Learning Not for Machine Learning

As I mentioned at the beginning of the chapter, Azure Machine Learning is not just an excellent tool for ML. In addition, we can perform lots of other compute-heavy tasks. To make Azure Machine Learning work for bioinformatics, we can simply install biorelated packages right in our notebook environment and run our code. This means that packages from Bioconductor or Biopython work just as they would on your local machine.

Performing Alignment in a Notebook

As a simple example, let's see how we can install and use biorelated packages in our notebook environment. Perhaps we'd like to perform sequence alignment using functionality in Biopython. (For this example, you can find the notebook of the commands in the supplementary materials under */06_amls/Notebook-Based_Alignment.ipynb*.)

We can install Biopython from a Jupyter notebook cell using pip:

```
pip install biopython
```

Then, we can also install other system dependencies, such as MUSCLE from BioConda:

```
conda install -c bioconda muscle
```

Finally, we can align a FASTA file using the `MuscleCommandline()` module (this FASTA file can be found in the supplementary materials at */06_amls/grain_28s_rRNA.fasta)*:

```
from Bio.Align.Applications import MuscleCommandline
muscle_cmd = MuscleCommandline(input="grain_28s_rRNA.fasta",
                               out="grain_28s_rRNA_aligned.fasta",
                               diags = True,
                               maxiters = 1,
                               log="muscle_log.txt")
muscle_cmd()
```

This process outputs an aligned FASTA file of 26s ribosomal RNA sequences of various grains, as shown in Figure 6-24. We could then use this data to build a phylogenetic tree or perform some haplotyping. All you need to do is continue installing whatever packages you need, and the notebook environment should be able to perform these biorelated tasks as easy as you do them on your local workstation.

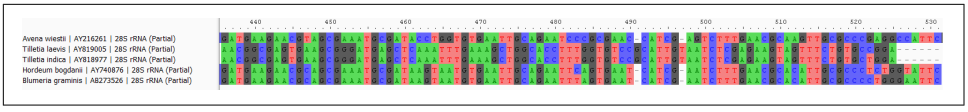


Figure 6-24. Molecular sequence alignment of grain 26S rRNA sequences output from MUSCLE

You may be wondering why you'd want to use Azure Machine Learning's Jupyter notebook or RStudio server instead of the equivalent tool on your local workstation. Here are a few reasons:

Data
: Given that we connected our workspace to our genomics data lake, this makes it quite easy to pull data from our data lake into the notebook environment. This is a bit harder to do on a local machine, and it may be time-consuming to move all that data around. In Azure, the services are well-connected, making access and data transfers quick and easy.

Sharing
: Since everyone who shares an Azure Machine Learning workspace has access to the same workspace storage, it's easier to share and collaborate on code. You could get feedback on your notebook or provide help to a colleague simply by navigating to their folder and opening a file.

Scale
: Compute instances can be provisioned with a considerable amount of RAM and cores. So if you're running into resource constraints on your laptop, a cloud-based notebook may help overcome that barrier to finishing your work faster.

Besides scaling our actual compute instance, we can also use the training clusters for tasks other than ML training. Let's do that next.

Custom Docker Images for Bioinformatics

When we go to train ML models using the HyperDrive or AutoML systems, what's actually being run on a cluster is a Docker container. These containers come with all the prerequisite software already installed to perform our model training. We can actually extend this capability to run other bioinformatics or non-ML tasks on our Azure Machine Learning clusters.

Let's do some protein structure prediction using AlphaFold2 (with ColabFold) in Azure Machine Learning. Now, I understand that AlphaFold2 is technically a deep learning model, but running it for your own structural predictions isn't an ML training task. Rather, it's a molecular sequence alignment, structural database comparison, and model scoring task. (We'll be using ColabFold, which is a more accessible implementation of the AlphaFold2 system.) The sample notebook, Dockerfile, and

sequence data for this example can be found at *https://github.com/colbyford/azureml-alphafold2*.

To start, I've prepared a Docker container that uses an AzureML base image. This includes the packages and logic needed to run in the HyperDrive cluster environment plus neural network libraries, like `cuDNN`, that we'll use to interact with GPUs. Next, I've added all of the dependencies to run ColabFold itself:

```
FROM  mcr.microsoft.com/azureml/openmpi3.1.2-cuda10.2-cudnn7-ubuntu18.04


ENV DEBIAN_FRONTEND noninteractive
USER root

## Set a working directory for the ML resources
WORKDIR /alphafold2

## Install System Libraries and Azure ML Dependencies
RUN apt-get update && \
    apt-get -y install g++ gcc git

RUN conda install -c r -y \
    conda \
    openssl=1.1.1c && \
    conda clean -ay

RUN pip install –no-cache-dir \
    azureml-defaults \
    azureml-dataprep \
    azureml-core \
    azure-ml-api-sdk \
    azureml-train-automl-runtime

## Install Torch
RUN conda install -y -c pytorch \
    cudatoolkit \
    pytorch \
    torchvision && \
    conda clean -ya

## Install ColabFold
RUN pip install –no-warn-conflicts -q \
    'colabfold[alphafold] @ git+https://github.com/sokrypton/ColabFold'

## Download ColabFold data
RUN python -m colabfold.download

## Install Bio-Dependencies
RUN conda install -y -q \
    -c conda-forge \
    -c bioconda \
    kalign3=3.2.2 \
```

```
hhsuite=3.3.0 \
openmm=7.5.1 \
pdbfixer
```

This Docker image is then pushed to a Docker repository. In my case, this image is available at *https://hub.docker.com/r/cford38/alphafold2_aml*, but you could also house private images on Azure's Container Registry service in a similar way.

Once we have the image ready in a container repository, we can use it in a new HyperDrive run in Azure Machine Learning. The goal here will be to pass in the ID of the sequence for which we'd like to predict the structure.

For this example, we have a FASTA file of four variants of the SARS-CoV-2 spike protein receptor binding domain (RBD), shown in Figure 6-25.

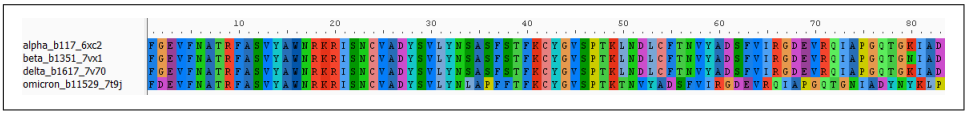


Figure 6-25. SARS-CoV-2 spike receptor binding domain sequences

Next, let's get the HyperDrive submission ready. This structural prediction process has the following steps:

1. Load in some libraries and connect to the Azure Machine Learning workspace backend.
2. Grab the SARS-CoV-2 RBD sequence data.
3. Create the compute cluster target.
4. Configure HyperDrive.
5. Run the experiment.

Back in our Jupyter notebook, we will set up a GPU-based training cluster. You will notice that the "vm_size" parameter is set up as a VM SKU that starts with an "NC." In Azure, the NC-series VMs are optimized for CUDA-based workloads and are equipped with NVIDIA Tesla V100 GPUs. These are data center–grade GPUs that have 5,120 CUDA cores, 640 Tensor cores, and 32GB of GPU memory. In other words, it is quite an impressive chip to be running on:

```
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

cluster_name = "alphafold2-ic"
compute_config = AmlCompute.provisioning_configuration(vm_size = 'STANDARD_NC6',
                                                        max_nodes = 4)
training_cluster = ComputeTarget.create(ws, cluster_name, compute_config)
```

Next, we must define the HyperDrive configuration. You'll notice that we provide the location of the Docker image that I previously built. When a custom environment is defined in the configuration, the nodes of the cluster will have the container running on them, and the system will install any additional dependencies that you define. This is the key to making pretty much anything run in Azure Machine Learning…even if it's not ML.

Also note that the parameters that we pass in are not ML algorithm settings but rather just the sequence ID of the structure to predict:

```
from azureml.core import Experiment, ScriptRunConfig, Environment
from azureml.core.conda_dependencies import CondaDependencies
from azureml.train.hyperdrive import
           HyperDriveConfig, PrimaryMetricGoal, choice
from azureml.widgets import RunDetails

## Create a Python environment for the experiment
alphafold2_env = Environment('alphafold2 ')
alphafold2_env.docker.base_image = 'cford38/alphafold2_aml:latest'
alphafold2_env.python.user_managed_dependencies = True

## Create a script config
script_config = ScriptRunConfig(source_directory = '.',
                                script='predict.py',
                                arguments = ['-msa_mode',
                                             'MMseqs2 (UniRef+Environmental)",
                                             '-num_models', 1,
                                             '-num_recycles', 3,
                                             '-stop_at_score', 90],
                                environment = alphafold2_env,
                                compute_target = training_cluster)

## Sample a range of parameter values
params = GridParameterSampling( '-sequence_id': choice('alpha_b117_6xc2',
    'beta_b1351_7vx1', 'delta_b1617_7v70', 'omicron_b11529_7t9j') )

## Configure hyperdrive settings
hyperdrive = HyperDriveConfig(run_config = script_config,
                              hyperparameter_sampling = params,
                              policy = None,
                              primary_metric_name = 'complete',
                              primary_metric_goal = PrimaryMetricGoal.MAXIMIZE,
                              max_total_runs = 4,
                              max_concurrent_runs = 4)
```

The training script here looks a bit different than your traditional scikit-learn stuff. This time, I have some logic to select the sequence of interest and pass that to the `ColabFold foldrun()` function:

```
...
## load the sequence(s)
```

```
for record in SeqIO.parse("sequences.fasta", "fasta"):
    if sequence_id == record.description:
       ## Write out the specific sequence fasta file for this node
       SeqIO.write(record, open("run_sequence.fasta", "w"), "fasta")

## Output and Input directories
result_dir = 'outputs/predicted_structures/'
input_dir = 'run_sequence.fasta'
...
queries, is_complex = get_queries(input_dir)
...

## Run Fold Prediction
foldrun(
    queries=queries,
    result_dir=result_dir,
    use_templates=use_templates,
    use_amber=use_amber,
    msa_mode=msa_mode,
    model_type="auto",
    num_models=num_models,
    num_recycles=num_recycles,
    model_order=[1],
    is_complex=is_complex,
    data_dir=default_data_dir,
    keep_existing_results=do_not_overwrite_results,
    rank_by="auto",
    pair_mode="unpaired+paired",
    stop_at_score=stop_at_score,
    zip_results=zip_results,
)
...
```

Anything that gets written out to the */outputs* directory of the node will then make it into the outputs of the run in the workspace. So we can look locate this experiment in the workspace and see each of the child runs, one for each sequence. See Figure 6-26.

Display name	Status	complete	--sequence_id	Parent run ID	Submitted time
icy_truck_07g4pp20	Completed	1.00000	"alpha_b117_6xc2"	HD_80b9b8f0-d5a3-4...	Mar 14, 2022 4:55 PM
jolly_king_3sqqv279	Completed	1.00000	"beta_b1351_7vx1"	HD_80b9b8f0-d5a3-4...	Mar 14, 2022 4:55 PM
lucid_bulb_b5lsq628	Completed	1.00000	"omicron_b11529_7t9j"	HD_80b9b8f0-d5a3-4...	Mar 14, 2022 4:55 PM
yellow_pear_03h3pl9z	Completed	1.00000	"delta_b1617_7v70"	HD_80b9b8f0-d5a3-4...	Mar 14, 2022 4:55 PM

Figure 6-26. Viewing the runs in the Azure Machine Learning Studio

Clicking any of these child runs and navigating to the "Output + logs" tab, as shown in Figure 6-27, we can see that the *predicted_structures* folder contains the output PDB file of the structure as well as the logs and figures of the predicted performance.

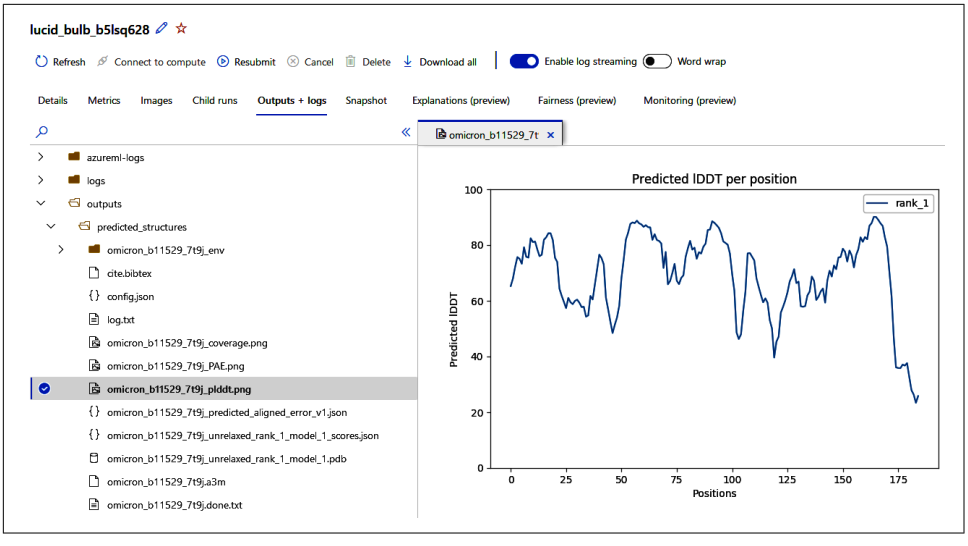


Figure 6-27. Viewing a run's outputs

If we were to run a single sequence through this structural prediction, it may take ~30 minutes (or longer) to complete. This is fine if I have only a couple sequences I want to predict, but what happens if I have hundreds? This is where using Azure Machine Learning for tasks that are computationally intensive or time-consuming can come in handy, especially for ad hoc analyses.

Here are some other ad hoc bioinformatics tasks that could be distributed in Azure Machine Learning:

- Molecular sequence alignment
- Phylogenetic tree generation
- Small molecule docking
- Quality-control tasks
- Biostatistics calculations
- Figure generation (#publishorperish)

In this chapter, we've walked through many of the powerful features of Azure Machine Learning. Not only can this platform be used for scaling your model training, but it helps you to track model performance and deploy any winning models. Plus, the system just so happens to be great for other computationally intensive workflows, such as some tasks in bioinformatics.

In summary, machine learning scalability comes from not only running our model training on a machine with more CPU cores and memory but also distributing tasks from a hyperparameter search or cross-validation to different compute contexts. Plus, certain workloads, such as deep neural network training tasks, benefit greatly from GPU-enabled machines.

Azure Machine Learning provides a lot of capabilities for scaling machine learning modeling tasks with ease and tracking the performance of various models. Plus, model deployment in Azure Machine Learning is simple and provides a flexible way of operationalizing your model as an API or batch scoring service.

AutoML is a new tool for automating the machine learning training process. This will train many models, testing various algorithms, scaling methods, and hyperparameters, and will help you pick the best one. (But beware! AutoML has the tendency to overfit models, so make sure to consult your data science pro on your team.) Despite its name, Azure Machine Learning also doubles as a general compute service for many different tasks, including bioinformatics-related workloads.

In the next chapter, we will begin diving into other compute-based platforms in Azure. While Azure Databricks and Azure Machine Learning are the most common, the next few services that we'll cover certainly deserve consideration for bioinformatics-specific workloads.

CHAPTER 7

High-Performance Computing and Other Compute Services

Show of hands…how many of us used our university's high-performance computing (HPC) environment and thought it was amazingly powerful, but despised having to submit jobs to the scheduling system and fight over resources with your department? In the previous chapters, we've talked a lot about using Azure Machine Learning or Databricks with Apache Spark for lots of data-intensive jobs as well as some bioinformatics tasks. However, not every piece of bioinformatics software that's meant for an HPC cluster will natively scale in these services.

In Azure, there are plenty of other scalable compute options that allow us to replicate our familiar on-prem HPC environments without the hassle of fighting for resources or fighting against the dreaded scheduler. This allows us to use the plethora of distributed bioinformatics software in the cloud without having to reinvent the wheel. Plus, instead of investing millions of dollars on HPC equipment, we can turn our compute services on or off and pay only for what we use.

In this chapter, I'll cover a few additional scalable compute options in Azure and begin talking about cloudifying your existing bioinformatics pipelines using these services.

While we'll cover quite a few options, you won't likely use more than one at a time in practice. Each of the services that we'll cover have slightly different features that may fit your particular use case better than others.

Bring Your Own Pipeline (BYOP)

If you've already created a bioinformatics analysis pipeline that fits your specific needs, there's really not a lot you'll have to change to get it working in the cloud. You can almost always port your existing work to a service in the cloud. The trick? Picking the right service for your specific use case.

The key to being successful with any bioinformatics pipeline is to have your set of steps be fully automated. That is, use a tool to orchestrate the inputs and outputs of each step such that an entire pipeline can be run without your intervention. Once this is achieved, we can run these pipelines inside of various cloud tools.

Often, bioinformaticians use a workflow management/pipeline tool such as WDL or CWL (with Cromwell), Snakemake, Nextflow, or others (as pictured in Figure 7-1). These are useful as they allow us to define the steps of a pipeline and then, at each step, execute the software or scripts that need to be run. These tools also keep track of the required inputs, parameters, and outputs for each step and provide logging to help see the status or troubleshoot any issues.

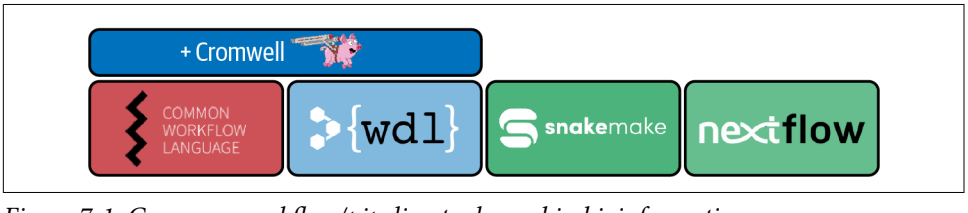


Figure 7-1. Common workflow/pipeline tools used in bioinformatics

Another pro of using one of these pipeline tools is that many template pipelines have already been created. Many Cromwell-focused WDL pipelines exist from the Broad Institute, and there are plenty of Snakemake and Nextflow pipelines available online. So, if you're looking to make an RNA-seq pipeline, for example, there are plenty of open source templates to use as a starting point. After selecting your pipeline tool, you can then get it working on a larger HPC environment, thus scaling and automating the processing of your pipelines.

Why Azure for HPC?

Beyond the financial differences of running cloud HPC infrastructure versus on-premises HPC environments, there are two other facets to consider: connectivity and access. In traditional HPC environments, we often have to connect to a specific node and copy our data over to the cluster. Then, after the task has completed, we have to copy the results back down to our workstation for analysis and interpretation.

In Azure, having our data centralized in a data lake allows us to connect our cloud-based HPC environment to the storage and read and write directly. Plus, the networking between resources in Azure is lightning fast. So no more waiting hours to copy terabytes of data from a hard drive to a cluster. See Figure 7-2.

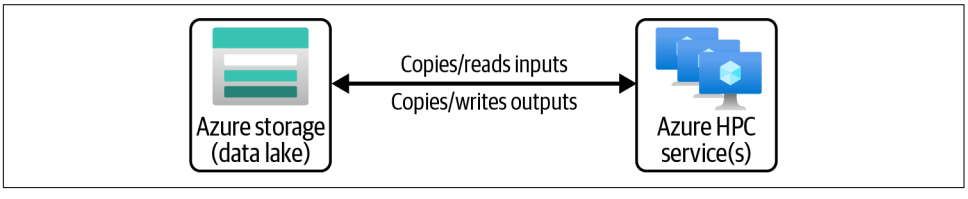


Figure 7-2. Simplified HPC environment–to–data lake communication

In addition, Azure HPC services have flexible options for various VM types, each with their own strength. Some VM types that are important for various bioinformatics and data science use cases include:

D-Series
: "General purpose." Good for many enterprise applications, training CPU-based machine learning models, and workloads that require a balance of CPU cores, hard drive space, and memory.

E-Series
: "Memory optimized." Great for memory-intensive workloads such as R or Python-based libraries that run analyses in-memory. These VMs have a higher memory-to-CPU ratio (specifically, 8GB of RAM per CPU core).

F-Series
: "Compute optimized." Great for compute-intensive workloads. These VMs feature a higher CPU-to-memory ratio (specifically, 2GB RAM and 16GB of SSD storage per CPU core). Use this type when you have applications that are locally parallelized and can take advantage of more CPU cores without needing a ton of memory.

H-Series
: "HPC machines." Recommended (but not required) for HPC applications. These VMs use optimized CPU cores without hyperthreading, better memory bandwidth, and better SSD performance.

N-Series
: "GPU enabled." Great for deep learning and simulation activities, these VMs offer NVIDIA GPUs such as the Tesla V100, P40, and M60 chips.

To learn more about Azure Virtual Machine types, including many types not discussed here, visit *https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series*.

Now that we've gone over the basics of HPC and various VM types, let's create our first HPC environment in Azure using Azure Batch.

Azure Batch

For our first foray into scalable computing in Azure, let's talk about Azure Batch. Azure Batch is an HPC service that allows us to run large jobs efficiently. Batch will provision pools of compute nodes and installs the appropriate software on them. Then it schedules the jobs to run on these nodes. Rather than directly configuring the cluster or dealing with a scheduler directly, you use Batch's API or other tools, command-line scripts, or the Azure Portal to submit and monitor jobs. In other words, Azure Batch provides a scheduler as a service under which you can run virtually any analysis tools you wish.

Batch works well with workloads that are parallelized locally. In other words, workloads with tasks that can be run independently on separate machines fit well in Batch. Each task can be completed separately, using all the cores of its respective compute node and then relaying the results back to storage without needing inputs or outputs from other nodes (as shown in Figure 7-3). These types of workloads can be performed at a very large scale, limited only by the number of machines and compute resources available.

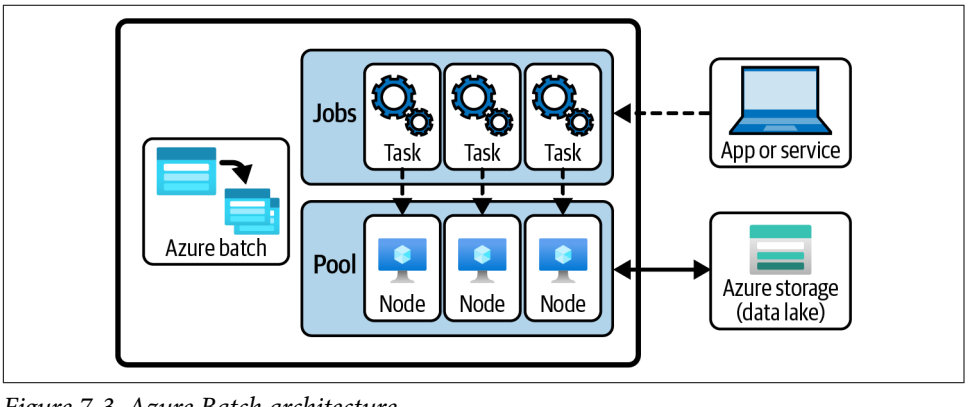


Figure 7-3. Azure Batch architecture

A benefit of Azure Batch is that there is no additional charge for using it.[1] You pay only for the underlying services that are provisioned. So only normal VM, networking, and storage costs apply.

1 For more information on setting budgets in Azure Batch, see *https://oreil.ly/IhD5r*.

In Azure Batch, we could run a variety of things, from distributed machine learning to data processing. For genomics, we can use this tool to automate bioinformatics workloads. Each time a new set of samples comes in, we can spin up a new Batch job and process the samples individually but at the same time as other samples are being processed. This could alleviate processing bottlenecks that we see in on-prem architectures with limited resources and inefficient task delegation.

Scaling Workloads with Cromwell

If you work in human genomics, you may be familiar with the Broad Institute's Cromwell. Cromwell is an open source workflow management system that was designed for bioinformatics workflows. Cromwell is deployable on Azure Batch thanks to some engineering by the Genomics team at Microsoft Research.[2]

Let's get started deploying Cromwell on Azure. (Even if you're not interested in running Cromwell specifically, this tutorial will help you see how other tools can be run in Azure Batch.) Before we begin, you'll need your Subscription ID, which you can find in the Azure Portal by clicking your subscription. Copy and paste this ID somewhere safe (like Notepad) as we'll need this in just a second. See Figure 7-4.

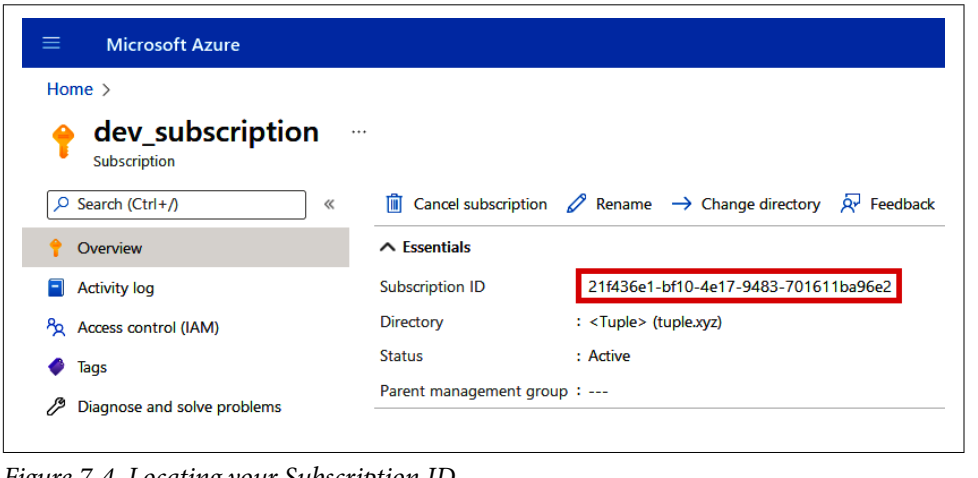


Figure 7-4. Locating your Subscription ID

Next, we're going to deploy Cromwell from the Azure CLI. If you already have the Azure CLI installed on your local workstation, you can use it, though the upcoming commands are written for the Linux Bash terminal. Be sure to run the `az login` command first to get logged in to the right tenant. If you don't have the Azure CLI,

2 For more information about Cromwell on Azure GitHub repository, see *https://oreil.ly/OpD1Y*.

we'll simply use the Azure Cloud Shell as our command line (recommended for this tutorial).

From the top of the Azure Portal, click the Cloud Shell icon. (See the icon at the top of Figure 7-5 if you're having trouble finding it.) If this is your first time using the Cloud Shell, it will ask to create a storage account first, and then you should see the screen in Figure 7-5.

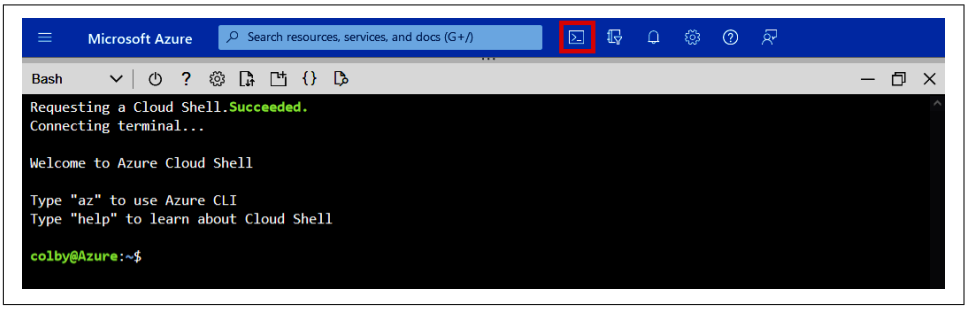


Figure 7-5. Azure Cloud Shell

Run the following command to get the Cromwell on Azure executable. This will create a new folder for the installer, download the installer from the Cromwell on Azure GitHub repository, and then grant permissions to run it:

```
## on Linux (Terminal) or the Azure Cloud Shell
mkdir cromwell-on-azure
cd cromwell-on-azure/
wget https://github.com/microsoft/CromwellOnAzure/releases/download/
            3.0.0/deploy-cromwell-on-azure-linux
chmod 777 deploy-cromwell-on-azure-linux
```

Next, we'll want to create a resource group for our Azure Batch service with Cromwell. From the Azure Portal, create a new resource group, as shown in Figure 7-6. (I called mine "batch-cromwell-dev-rg.") The reason we want to create a new resource group rather than using the one we've been using this whole book is because Azure Batch requires a series of different services that will be provisioned for us. So rather than making our existing resource group cluttered with Cromwell stuff, it's best to make another one.

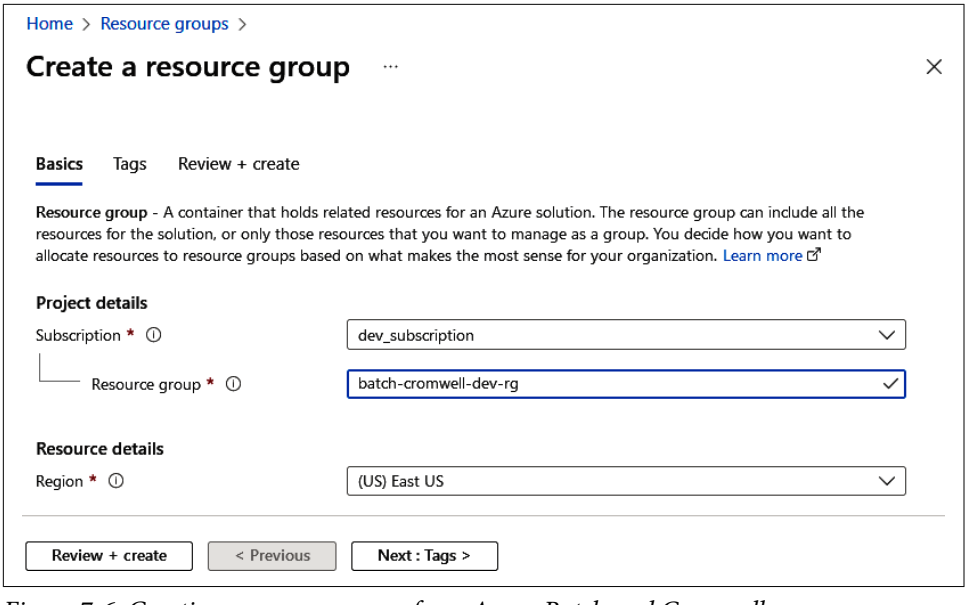


Figure 7-6. Creating a resource group from Azure Batch and Cromwell

Finally, we'll actually provision the Azure Batch service using the following command. You'll need to paste in your Subscription ID that you copied earlier (and change the region or resource group name to suit your needs):

```
./deploy-cromwell-on-azure-linux --SubscriptionId <your-sub-id-here>
  --ResourceGroupName  batch-cromwell-dev-rg
  --RegionName eastus --MainIdentifierPrefix coa
```

This process could take a half an hour or so. Grab a cup of tea (or a nice gin and tonic if you work from home like me), and we can go over what's being created. This installer creates the following resources:

Host VM
: This is an Ubuntu 18.04 LTS VM that runs Docker Compose with Cromwell, MySQL, TES (the orchestration backend), and TriggerService.

Batch account
: The Azure Batch account is used by the orchestration backend to provision VMs that run each task in a workflow.

Storage account
: This is mounted to the Host VM using `blobfuse`, which enables the storage account to be mounted as if it is part of the local filesystem.

Application insights
: This keeps track of the logs from the orchestration backend (useful for debugging) and contains logs from TES and the Trigger Service to enable debugging.

Cosmos DB
: This is the database used by the orchestration backend that houses metadata about each task that gets run in a workflow.

After 30 minutes or so, your Cromwell environment should be ready. You'll see a "Deployment was successful" message, as shown in Figure 7-7. If you hit any permission errors, you may have to work with the subscription owner to do the deployment (or request elevated permissions to be able to create managed identities and other resources in the subscription).

```
Deployment was successful, but Batch account coa456907732f does not have sufficient core quota to run wor
kflows.quota, read the docs to run a test workflow and confirm successful deploRequest Batch core quota:
https://docs.microsoft.com/en-us/azure/batch/batch-quota-limit
After receiving the quota, read the docs to run a test workflow and confirm successful deployment.
Completed in 25.0 minutes.
colby@Azure:~/cromwell-on-azure$
```

Figure 7-7. A successful deployment of Azure Batch and Cromwell

If this is your first Batch account that has been deployed in your tenant, you may get a warning about having an insufficient availability of cores ("quota") to run the test workflow. If this is the case, you can request additional quota by navigating to the Batch service that was just created and clicking "Quotas" on the side menu, as shown in Figure 7-8.

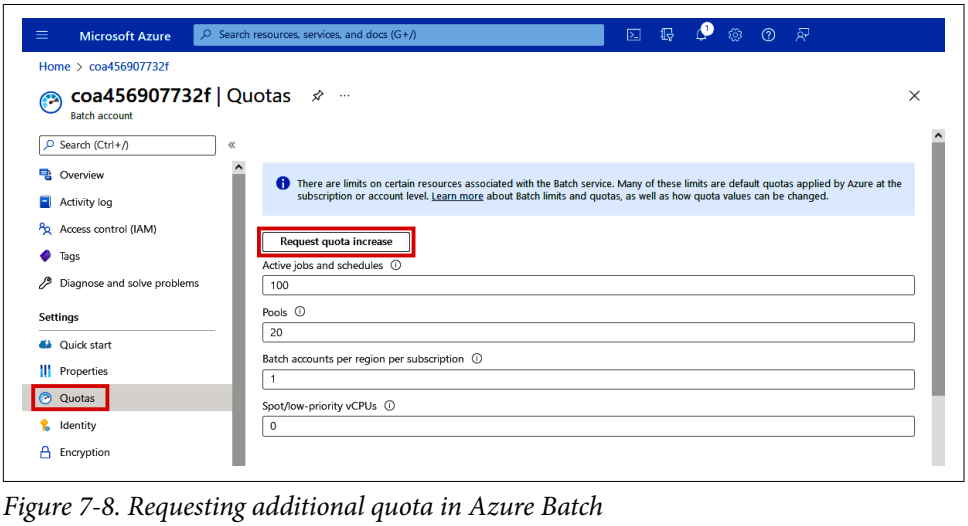


Figure 7-8. Requesting additional quota in Azure Batch

Next, let's create our first pipeline and run it in Cromwell on Azure.

Running your first workflow

Running tasks in Cromwell on Azure is quite simple. We'll basically be placing a few files in the storage account that gets created as part of the Azure Batch deployment.

From the Azure Portal, locate the Cromwell resource group that was just created (mine's called "batch-cromwell-dev-rg") and find the storage account that was created (it'll be called "coa," followed by a bunch of letters and numbers). See Figure 7-9.

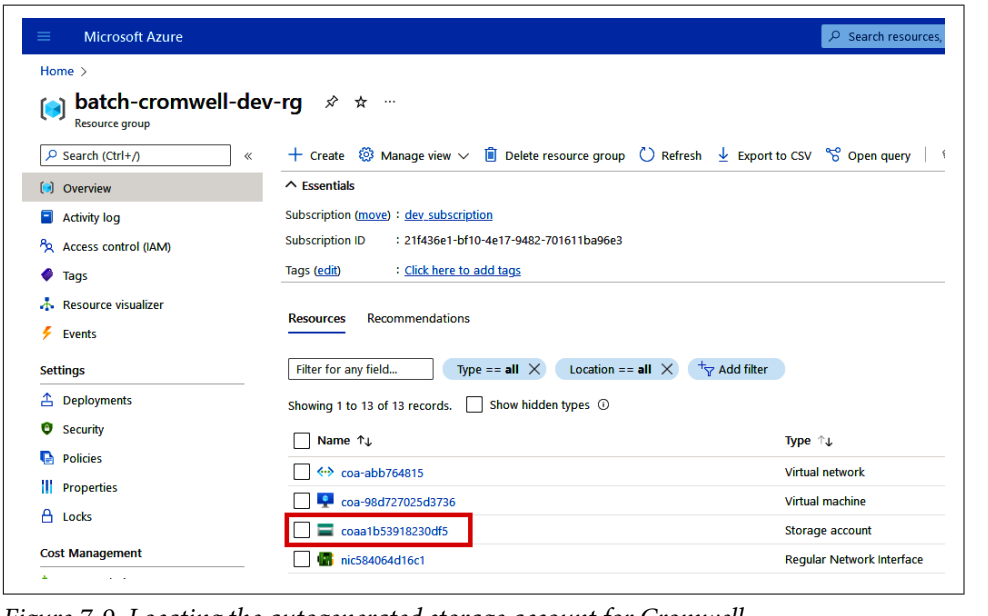


Figure 7-9. Locating the autogenerated storage account for Cromwell

Under "Containers," shown in Figure 7-10, you'll see a few containers such as "inputs," "outputs," "workflows," and more. We'll be dropping a few files in these containers in just a second, which will kick off our Cromwell run.

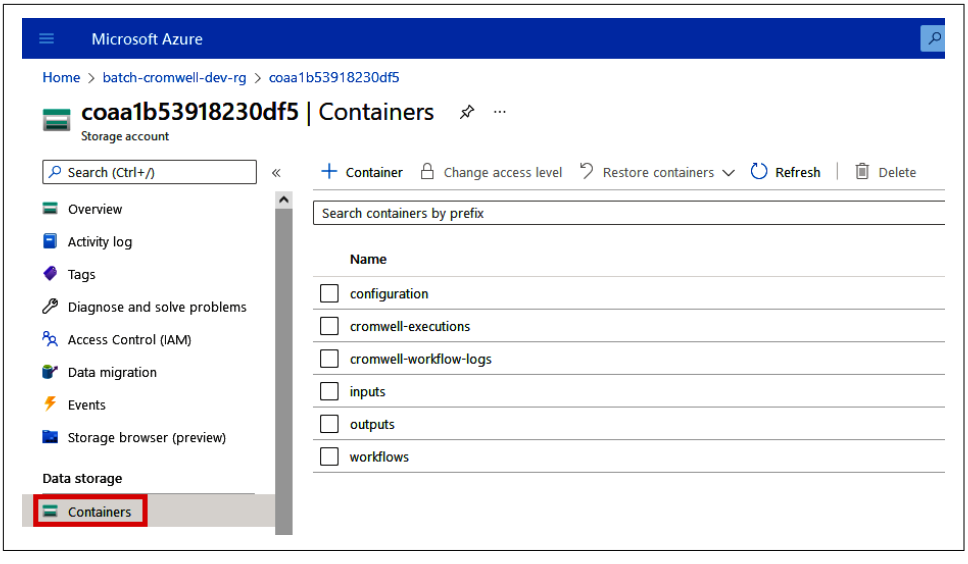


Figure 7-10. Storage account containers for Cromwell

The standard workflow language in Cromwell is WDL (pronounced "widdle"). Along with the WDL file, which defines the workflow steps, you'll also have an inputs file and a trigger file, as shown in Figure 7-11.

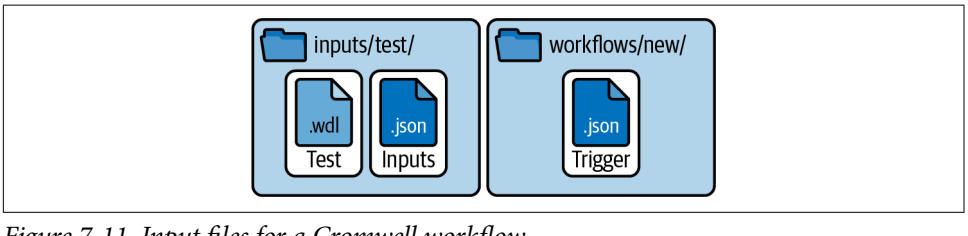


Figure 7-11. Input files for a Cromwell workflow

The following sample workflow files are included in the supplementary code repository under */07_hpc/cromwell/*.

For our example pipeline, the WDL file will look like this:

```
task hello {
  String name

  command {
    echo 'Hello ${name}!'
  }
```

```
  output {
    File response = stdout()
  }
  runtime {
    docker: 'ubuntu:16.04'
  }
}

workflow test {
  call hello
}
```

In the previous WDL script, you can see we are creating a workflow called "test," which has a single task called "hello." The hello task then runs an *echo* command that references a "name" parameter.

Then, to define that input parameter, we'll create an *inputs.json* file:

```
{
"test.hello.name": "World"
}
```

Finally, we can trigger the workflow run in Cromwell by dropping a *trigger.json* file that tells the engine where to find the workflow files (and make sure to update the storage account name with the name of storage account that was created by the Cromwell deployment process earlier):

```
{
"WorkflowUrl": "/<YOUR STORAGE ACCOUNT>/inputs/test/test.wdl",
"WorkflowInputsUrl": "/<YOUR STORAGE ACCOUNT>/inputs/test/inputs.json",
"WorkflowOptionsUrl": null, "WorkflowDependenciesUrl": null
}
```

As soon as you create/upload the *trigger.json* file, the Cromwell system will trigger the workflow. And with that, we've submitted your first workflow in Cromwell on Azure! You can track the progress of the workflow in a couple of ways. As shown in Figure 7-12, from the storage account, you can look at the *workflows/inprogress/* and *workflows/failed/* directories to see the status of a workflow (or if it failed). Also, in the Cosmos DB database that gets created, you can view all of the Cromwell tasks that have been started under the Tasks container (which is like a SQL table).

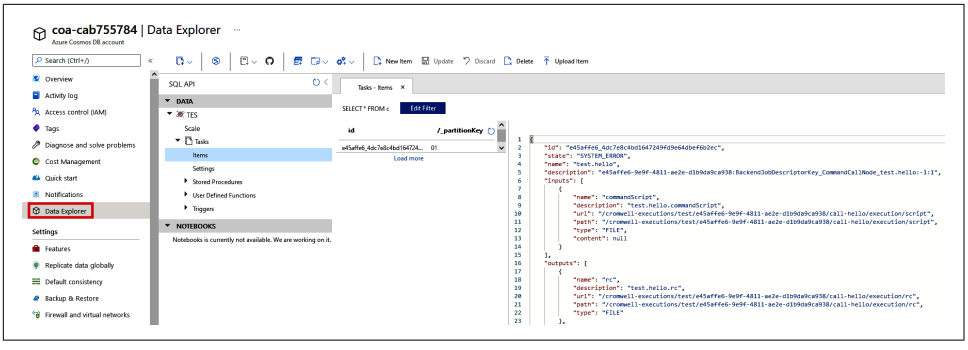


Figure 7-12. Checking on a Cromwell workflow

Once a workflow completes, any outputs (defined in the WDL file) will be written to the *outputs/* container in the storage account.

Now that you have Cromwell up and running in Azure, you can reuse any of the publicly available workflows that have been created by the community. Since Cromwell is a popular tool from the Broad Institute, there are plenty of bioinformatics pipelines that you can use as a starting point. Here are a few examples published by Microsoft for Cromwell on Azure:

- Data preprocessing for variant discovery
- Germline short variant discovery (SNPs + indels)
- Somatic short variant discovery (SNVs + indels)
- RNA-seq short variant discovery (SNPs + indels)
- Variant-filtering with convolutional neural networks
- Sequence data format conversion

In the next section, we'll be using a different HPC service called Azure CycleCloud, which is similar in principle to Azure Batch but a little closer to traditional HPC environments that you may have used in the past.

Azure CycleCloud

Azure CycleCloud serves as the sister product to Azure Batch. The difference? You can bring your own scheduler (like Slurm, Grid Engine, PBS Pro, or others). CycleCloud, in my opinion, is more akin to a traditional HPC environment, where everything is fully configurable, access-controlled, and monitored. Unlike an on-prem HPC cluster, though, CycleCloud is autoscalable.

CycleCloud deploys autoscaling plugins on top of the supported schedulers, making it simple to implement scalability without having to manually configure it yourself.

You just have to interface with the familiar scheduler configurations that you're used to. CycleCloud provides features to manage compute resources more easily and to monitor and analyze jobs. Plus, you can create cluster templates and interact with the service using a RESTful API, which makes this service a perfect tool for automating and standardizing complex bioinformatics tasks in Azure.

Setting Up CycleCloud Clusters

CycleCloud works in a different way than we've experienced with the other Azure services covered in this book. With CycleCloud, we'll spin up a VM with a special image installed. This basically runs a web server with a UI, where we can select a scheduler and create jobs.

To start, navigate to the Azure Portal and in the top search bar, search for "CycleCloud." This will locate the CycleCloud service in the Azure Marketplace, where you can click the "Create" button on the Marketplace page to start the setup process, as shown in Figure 7-13.

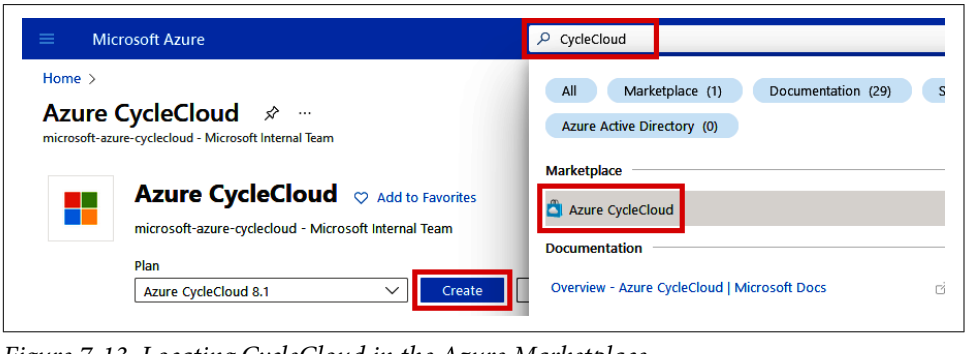


Figure 7-13. Locating CycleCloud in the Azure Marketplace

In the "Create a virtual machine" blade, shown in Figure 7-14, fill out the required fields. I would suggest you create a new resource group (I named mine "cyclecloud-rg") since this process creates a few Azure services and we don't want it cluttering up our main resource group. I named my VM "cyclecloudvm," picked the "Standard_DS1_v2" VM size, and set the "Authentication type" to "Password." Once you're done on the "Basics" tab, we'll leave all of the other tabs' settings as default. Click the "Review + create" button at the bottom and then the "Create" button on the final screen once your choices have been validated.

Home > Azure CycleCloud > Choose recommended defaults that match your workload >

Create a virtual machine ...

Basics | Disks | Networking | Management | Advanced | Tags | Review + create

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. Learn more

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	dev_subscription
Resource group *	(New) cyclecloud-rg Create new

Instance details

Virtual machine name *	cyclecloudvm
Region *	(US) East US
Availability options	No infrastructure redundancy required
Security type	Standard
Image *	Azure CycleCloud 8.1 - Gen1 See all images | Configure VM generation
Azure Spot instance	☐
Size *	Standard_DS1_v2 - 1 vcpu, 3.5 GiB memory ($53.29/month) See all sizes

D-series is recommended for general purpose workloads.

Administrator account

Authentication type	○ SSH public key ◉ Password
Username *	cyclecloudadmin
Password *	•••••••••••••••••••
Confirm password *	•••••••••••••••••••

Review + create | < Previous | Next : Disks >

Figure 7-14. Provisioning the Azure CycleCloud VM

After this setup completes, you can find the IP address of the VM from its page in the Portal. This is shown in Figure 7-15. Copy and paste this IP address into a new browser tab. (If you get any security warnings, simply ignore them. This is due to the VM hosting its own SSL certificate.)

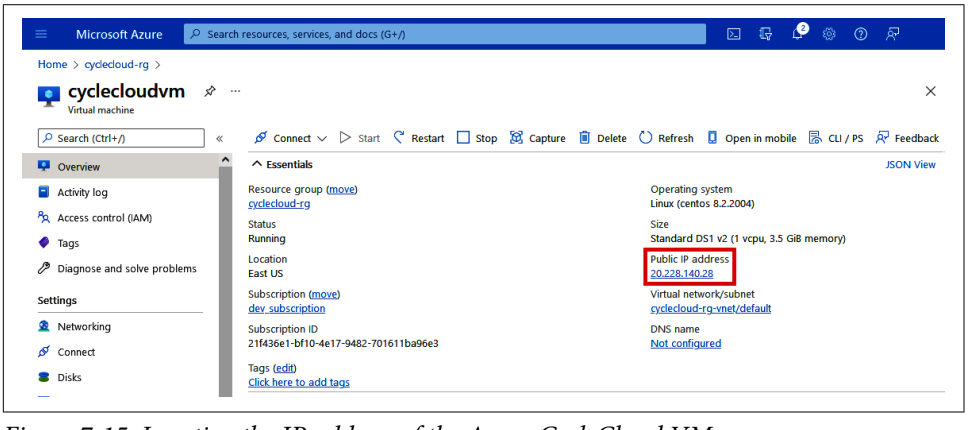


Figure 7-15. Locating the IP address of the Azure CycleCloud VM

In the web UI of the VM that opens, shown in Figure 7-16, give the site a name. This will be used as a prefix for any resource that CycleCloud creates. (I named mine "cyclecloud-01.") Click the "Next" button at the bottom and agree to the license agreement on the next page.

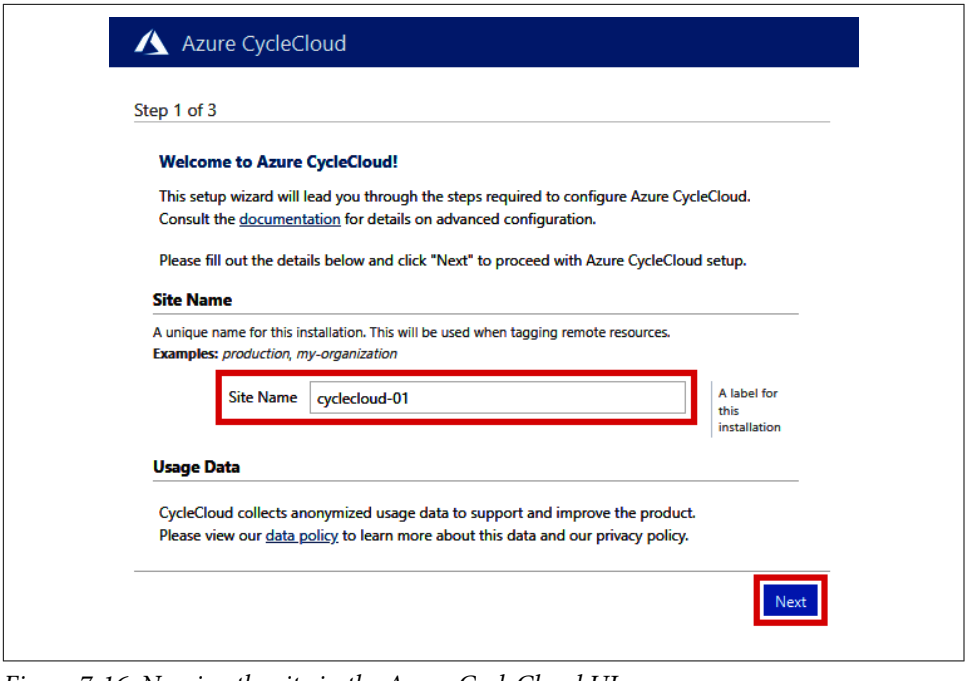


Figure 7-16. Naming the site in the Azure CycleCloud UI

On Step 3, the final step, set the user ID, name, and password for the admin account, as shown in Figure 7-17. I used the same credentials I defined in the original setup (Figure 7-14). You can optionally provide an SSH key if you want to have the ability to SSH into compute nodes. Click the "Done" button when finished.

Azure CycleCloud

Step 3 of 3

Create your Azure CycleCloud administrator account. You will be able to create additional user accounts with different permission levels after setup has been completed.

User ID* cyclecloudadmin

Name Cycle Cloud Administrator

Password* ••••••••••••••••••••

Confirm* ••••••••••••••••••••

SSH Public Key

SSH public key for connecting to nodes.

Begins with one of the following: ssh-rsa, ssh-dss, ssh-ed25519, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384, ecdsa-sha2-nistp521.

Passwords must be **8**-**123** characters long and meet **3** of the following requirements:

- Contains an upper case character
- Contains a lower case character
- Contains a number
- Contains a special character: @ # $ % ^ & * - _ ! + = [ ] { } | \ : ' , . ? ` ~ " ( ) ;

We also support Active Directory and LDAP user authentication.
For more information, see the documentation.

You will be automatically logged in to this account once setup has been completed.

Previous Done

Figure 7-17. Setting up the admin account in the Azure CycleCloud UI

On the "Add Subscription" screen in the CycleCloud UI, shown in Figure 7-18, you'll need a few bits of information about your Azure Tenant.

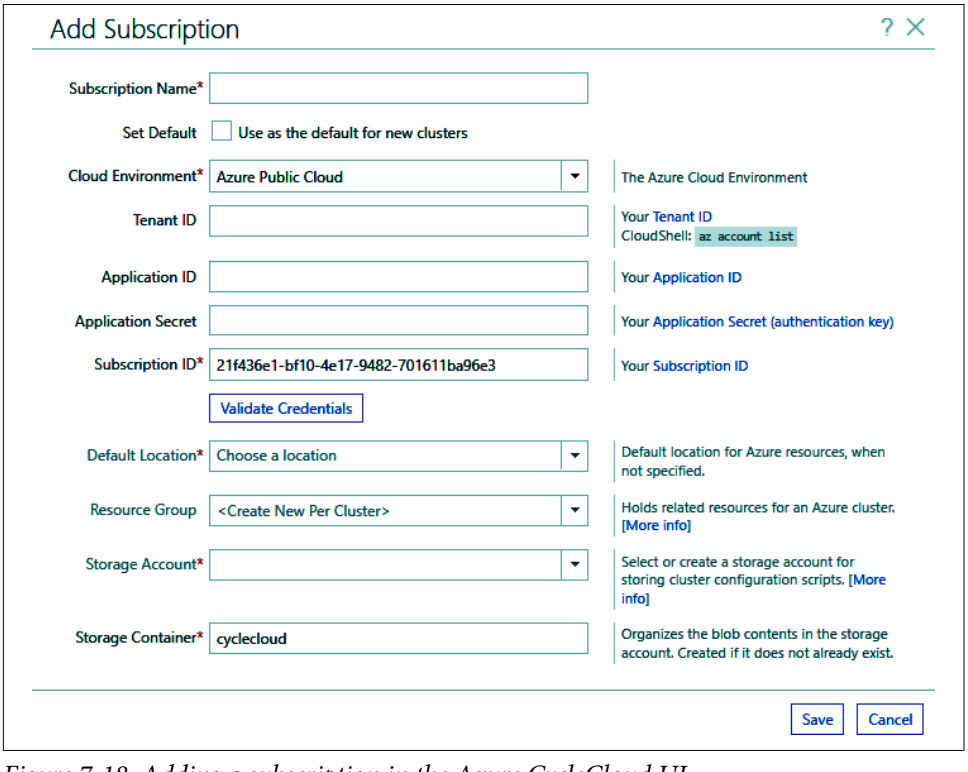


Figure 7-18. Adding a subscription in the Azure CycleCloud UI

You can find your Subscription ID from your subscription's page, as shown in Figure 7-19. (Keep this tab open as you'll need to assign a role in just a second from "Access control (IAM)" on the side menu.)

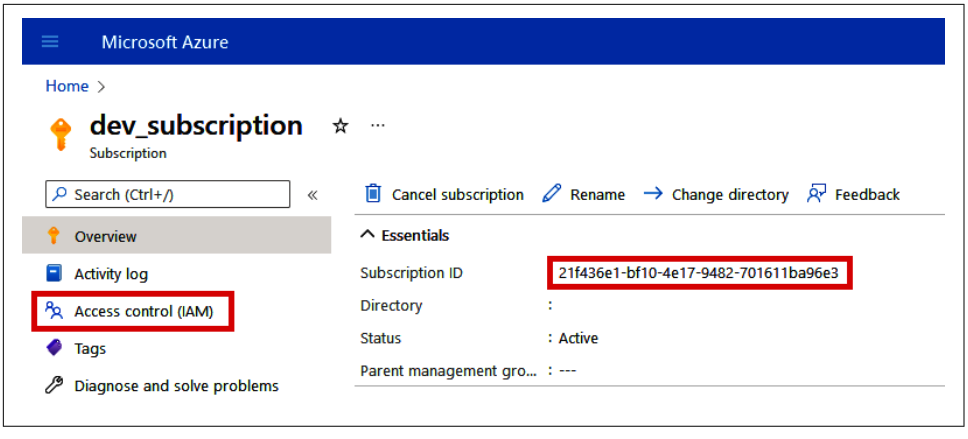


Figure 7-19. Locating your Subscription ID

You'll also need to create a service principal that the CycleCloud service will use to spin up clusters. You can do this from the App Registrations page of Azure Active Directory or using the Azure CLI. To make things easy, navigate to the Azure Cloud shell and run the following CLI command:

```
az ad sp create-for-rbac -name CycleCloudApp -years 1
```

This will give you the application ID, application secret, and tenant ID that you need (called "appId," "password," and "tenant"). You'll then need to add this new service principal as a "Contributor" to your subscription. This can be done by navigating to your subscription in the portal and clicking "Access control (IAM)" on the side menu. See Figure 7-20.

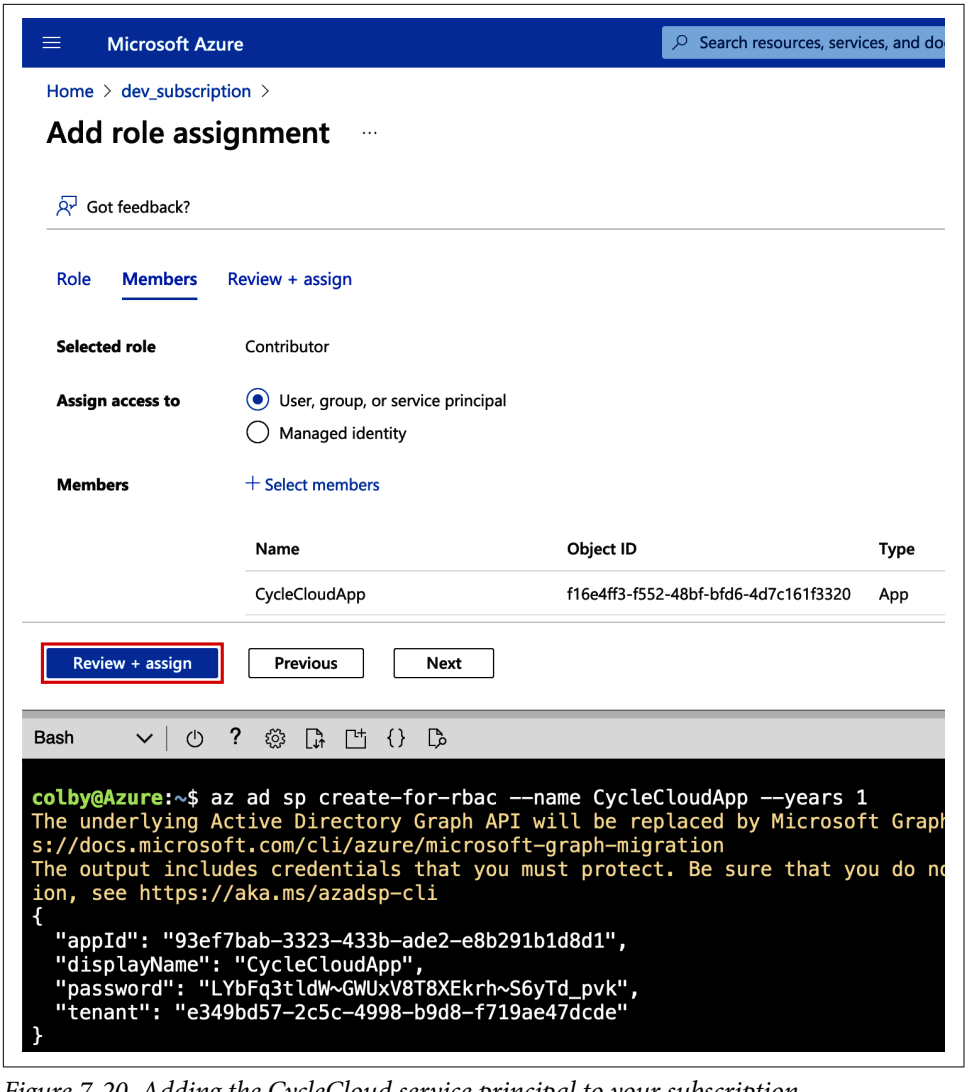


Figure 7-20. Adding the CycleCloud service principal to your subscription

After giving Azure a few minutes to propagate the role you just added, paste in these IDs that we've been collecting, as shown in Figure 7-21. Then click the "Validate Credentials" button. If the test succeeds, click the "Save" button at the bottom.

Add Subscription

Subscription Name* dev_subscription

Set Default ☑ Use as the default for new clusters

Cloud Environment* Azure Public Cloud — The Azure Cloud Environment

Tenant ID e349bd57-2c5c-4998-b9d8-f719ae47dcde — Your Tenant ID CloudShell: `az account list`

Application ID 93ef7bab-3323-433b-ade2-e8b291b1d8d1 — Your Application ID

Application Secret •••••••••••••••••••••••••••••••• — Your Application Secret (authentication key)

Subscription ID* 21f436e1-bf10-4e17-9482-701611ba96e3 — Your Subscription ID

Validate Credentials

Test succeeded

Default Location* East US — Default location for Azure resources, when not specified.

Resource Group cyclecloud-rg — Holds related resources for an Azure cluster. [More info]

Storage Account* genomicsdatalake01 — Select or create a storage account for storing cluster configuration scripts. [More info]

Storage Container* cyclecloud — Organizes the blob contents in the storage account. Created if it does not already exist.

Save Cancel

Figure 7-21. The completed Add Subscription screen in the Azure CycleCloud UI

You should see this new item added to the "Subscriptions" list, as shown in Figure 7-22. If so, you're now ready to create a cluster! Click the "Back to clusters" link at the top.

Azure CycleCloud

Back to clusters

Subscriptions

Show Detail Create Edit Delete Import Retry

Name	Provider	Provider Id	Master Credentials	Default	State	Status	Failed	Disabled
dev_subscription	azure	21f436e1-bf10-4e17-9482-701611ba96e3	dev_subscription	true	Creat...	....	....	....

Figure 7-22. Listing subscriptions in Azure CycleCloud

On the main screen, shown in Figure 7-23, you'll see a list of the different schedulers and filesystems that are available.

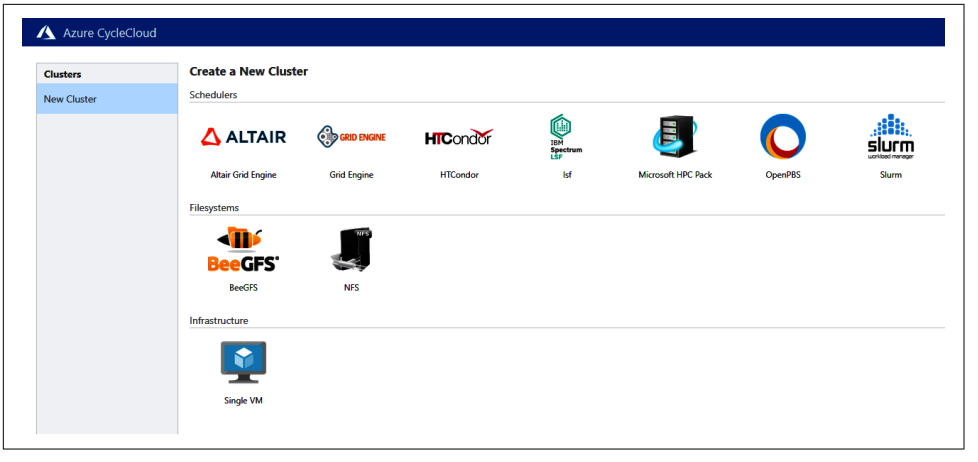


Figure 7-23. Creating a new CycleCloud cluster

In the next step, we'll be creating our first CycleCloud cluster that runs a bioinformatics pipeline.

Creating a cluster for bioinformatics

If you're already using an HPC cluster today, then setting up a similar cluster in the cloud shouldn't have a huge learning curve. You can simply specify the scheduler, the number of nodes, and the libraries to install, and then you can submit jobs just like you're used to.

In this tutorial, we'll be running a FASTQ to VCF alignment pipeline built in Snakemake and run in OpenPBS. To start, click the "OpenPBS" button on the "Create a New Cluster" screen. See Figure 7-24.

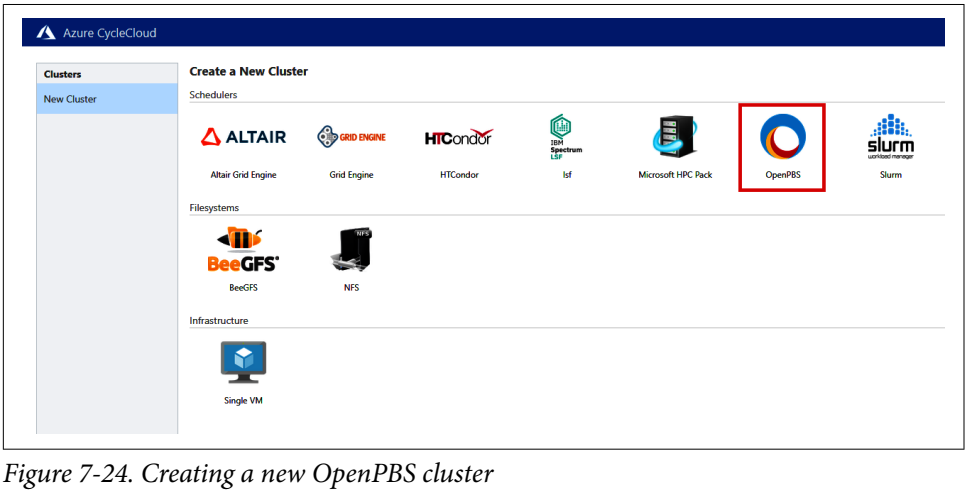


Figure 7-24. Creating a new OpenPBS cluster

On the “About” tab, shown in Figure 7-25, give your cluster a name. (I named mine “PBSGenomicsCluster.”)

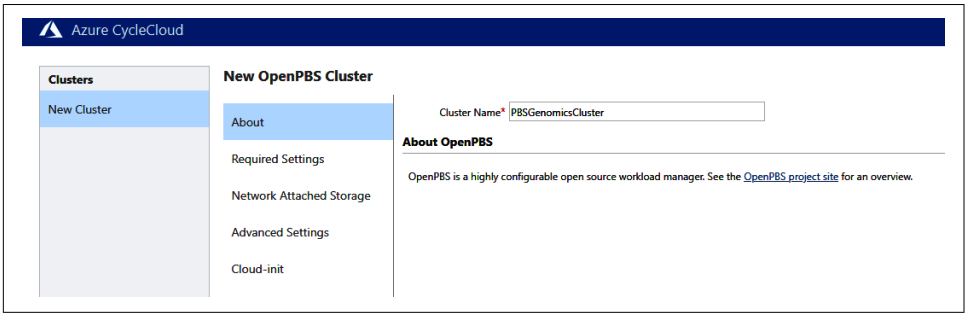


Figure 7-25. Naming your OpenPBS cluster

On the “Required Settings” tab, shown in Figure 7-26, you can select your region and VM types for the server and execution nodes. The default “Max Cores” setting is 100. Reduce this to “24” for now, which is 12 *Standard_F2s_v2* VMs. Select the “Subnet ID” that was created by the CycleCloud deployment. (This should be prefixed with the name of the resource group where you deployed CycleCloud.)

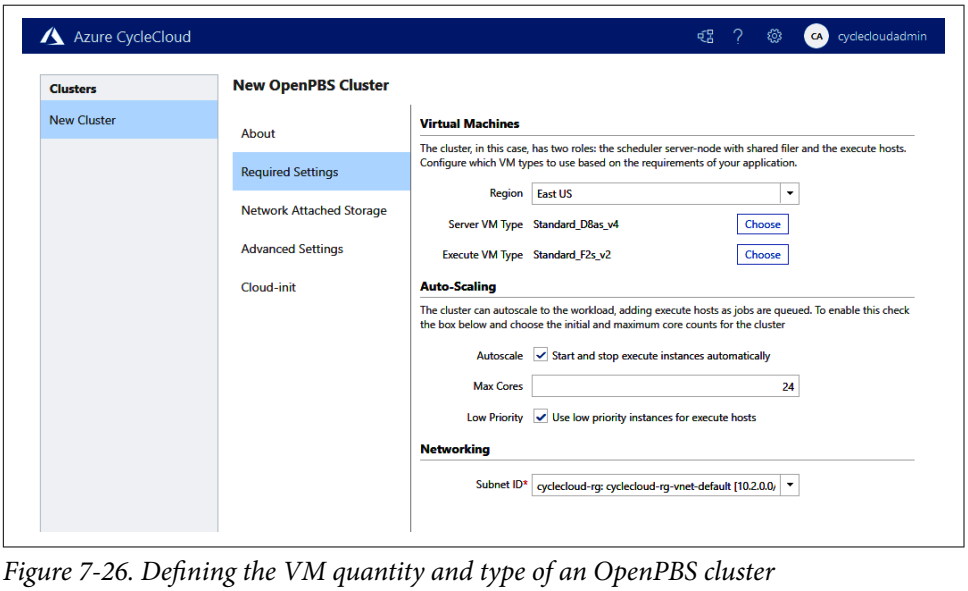


Figure 7-26. Defining the VM quantity and type of an OpenPBS cluster

On the “Network Attached Storage” tab, we can leave the default settings. By default, the cluster will use the built-in NFS storage. However, if you want to mount an Azure Storage Account, this would be configured from this tab. Leave all the default selections on the “Advanced Settings” and “Cloud-init” tabs and click the “Save” button at the bottom of the screen.

On the “Clusters” page, shown in Figure 7-27, click the “Start” button. This will take a couple of minutes to spin up.

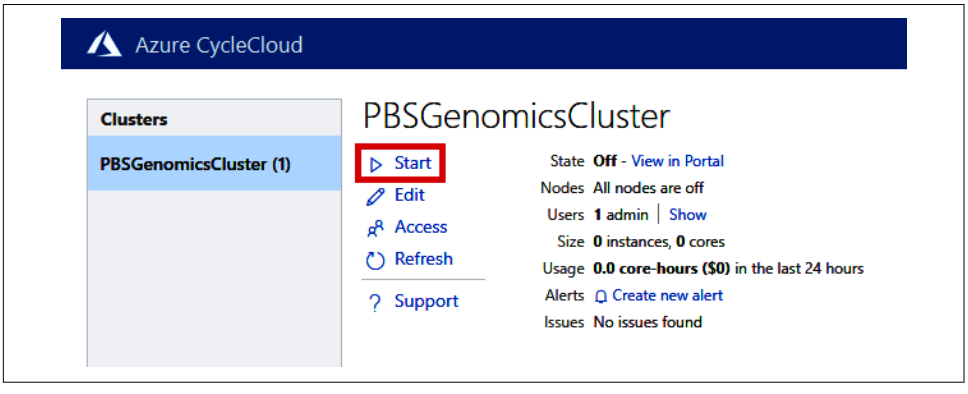


Figure 7-27. Starting the new OpenPBS cluster

If you hit an error mentioning that a “blob does not exist,” this could be an issue with your service principal’s access to the storage account that you code. You may need to grant the Storage Blob Data Contributor role to the service principal.

Once the system has finished acquiring the VMs for the cluster, click the available node under the “Nodes” tab and double-click the server line that pops up below. See Figure 7-28.

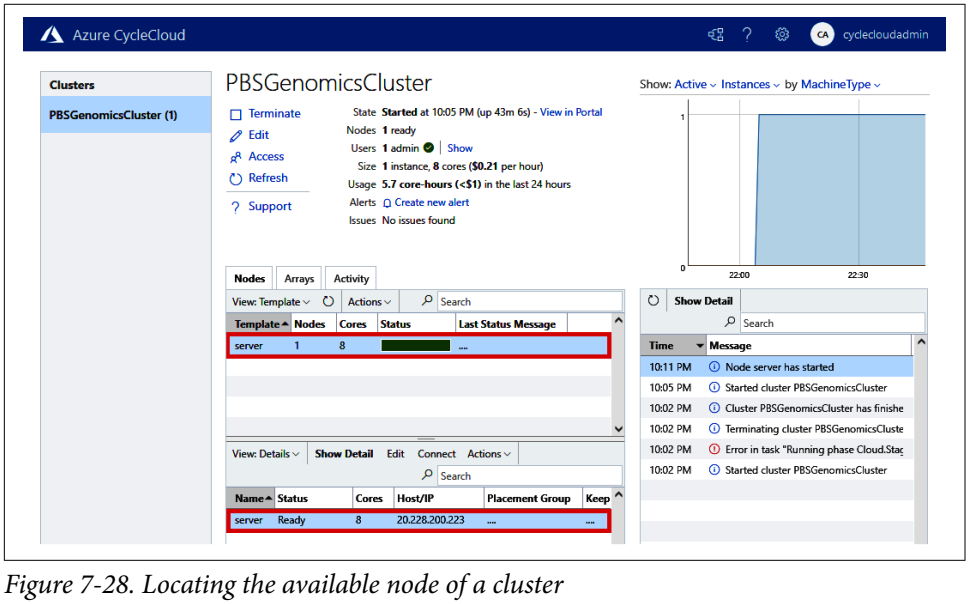


Figure 7-28. Locating the available node of a cluster

Click the "Connect" button on the "Showing server in PBSGenomicsCluster cluster" window, as shown in Figure 7-29.

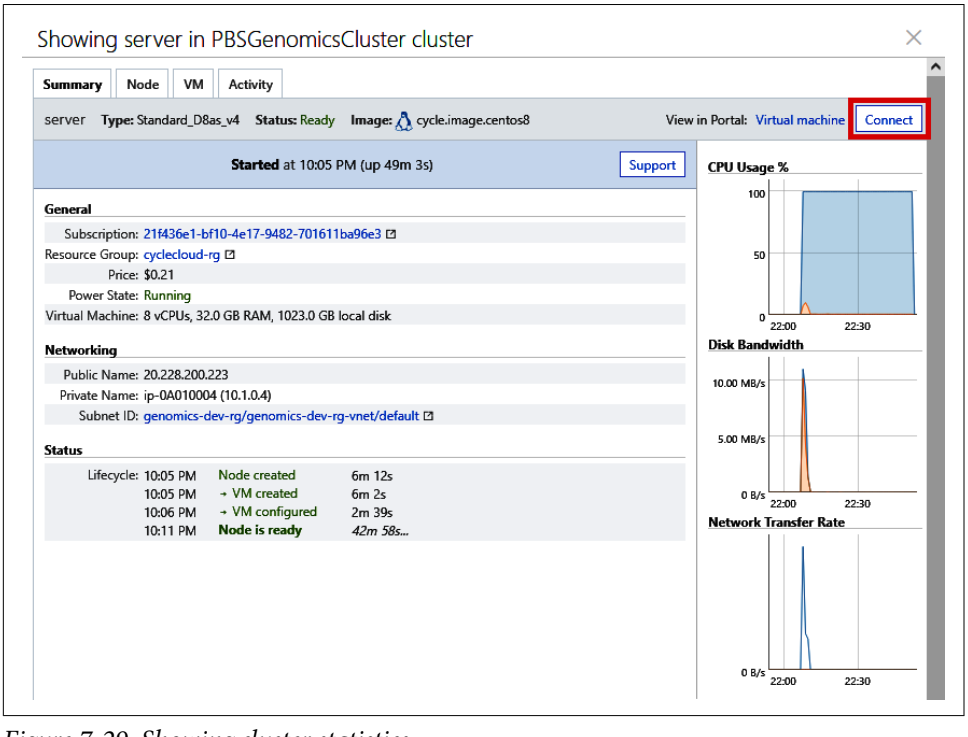


Figure 7-29. Showing cluster statistics

If you configured CycleCloud with an SSH key, you can remote into the node using the `ssh cyclecloudadmin@000.000.000.000` command, as shown in Figure 7-30. There is a CycleCloud CLI, but I think SSH is a bit more universally familiar to everyone. So, let's use that.

Figure 7-30. Connecting to the cluster node from either SSH or the CycleCloud CLI

For those of us who didn't create an SSH key before, we'll do that now. Open the Azure Portal in a new browser tab and open a new Cloud Shell.

From the Cloud Shell, run the `ssh-keygen` command to create an SSH keypair. (I left my key location as the default "id_rsa." Print the public key to the console so that you can copy it.) See Figure 7-31.

```
colby@Azure:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/colby/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/colby/.ssh/id_rsa.
Your public key has been saved in /home/colby/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:1WJtQy2zL+/g0kFTI3Z2dShraPE+zbaajGnMTJ1aioc colby@cc-476c0a44-6676566b7-mtmv5
The key's randomart image is:
+---[RSA 2048]----+
|            .. .+|
|          .+* B o|
|          +==@ o |
|         oooB.   |
|        S. = *   |
|          . O =  |
|         B =.* . |
|        E X=.oo  |
|         oo.=o.  |
+----[SHA256]-----+
colby@Azure:~$ cat /home/colby/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCnH8LL7mJoD1LFgg//82GhEmpz9JPjZYRLrRBsmSvABju1q6JxtpEWgn4duqgS+6V87T
/JqfzmrP2H2tRkjYCPoryMbjASJtZ16Q8zrlbmlwp5ahlBsJ9MiYrVf6ZgLihcOFegf596LMvGEobo96DBKk20N5eesbVyAJUHnpKGWMxp
JaPEEaxZS4VAGt4eeT+Jn+npggCykP9v8VxeZbsZgb2O3PVJrL8351uTvxJChgm/d3ehfCqeJ7oP0fIphvbXBcPbRoxxJTELoBa752UsDi
c2C7CKKcHaME1scumLbWy+vKayt3XTKF98TpO+Ams5b2eLpG5WYw0Yw0gW+CwJ colby@cc-476c0a44-6676566b7-mtmv5
```

Figure 7-31. Creating an SSH keypair

Back in your CycleCloud tab, shown in Figure 7-32, click the settings gear icon at the top of the screen and click "Users." Double-click the user account that you've been using in CycleCloud. (Mine is called "cyclecloudadmin.")

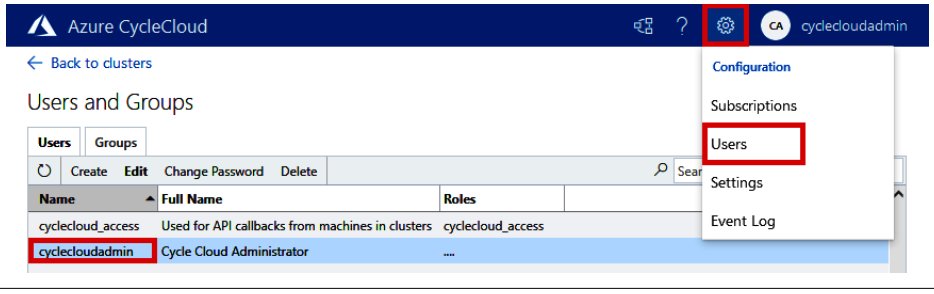


Figure 7-32. Editing users in Azure CycleCloud

As shown in Figure 7-33, copy and paste the SSH key from the Cloud Shell into the "SSH Public Key" text box of the "Edit User" window in CycleCloud. Then click the "Save" button at the bottom.

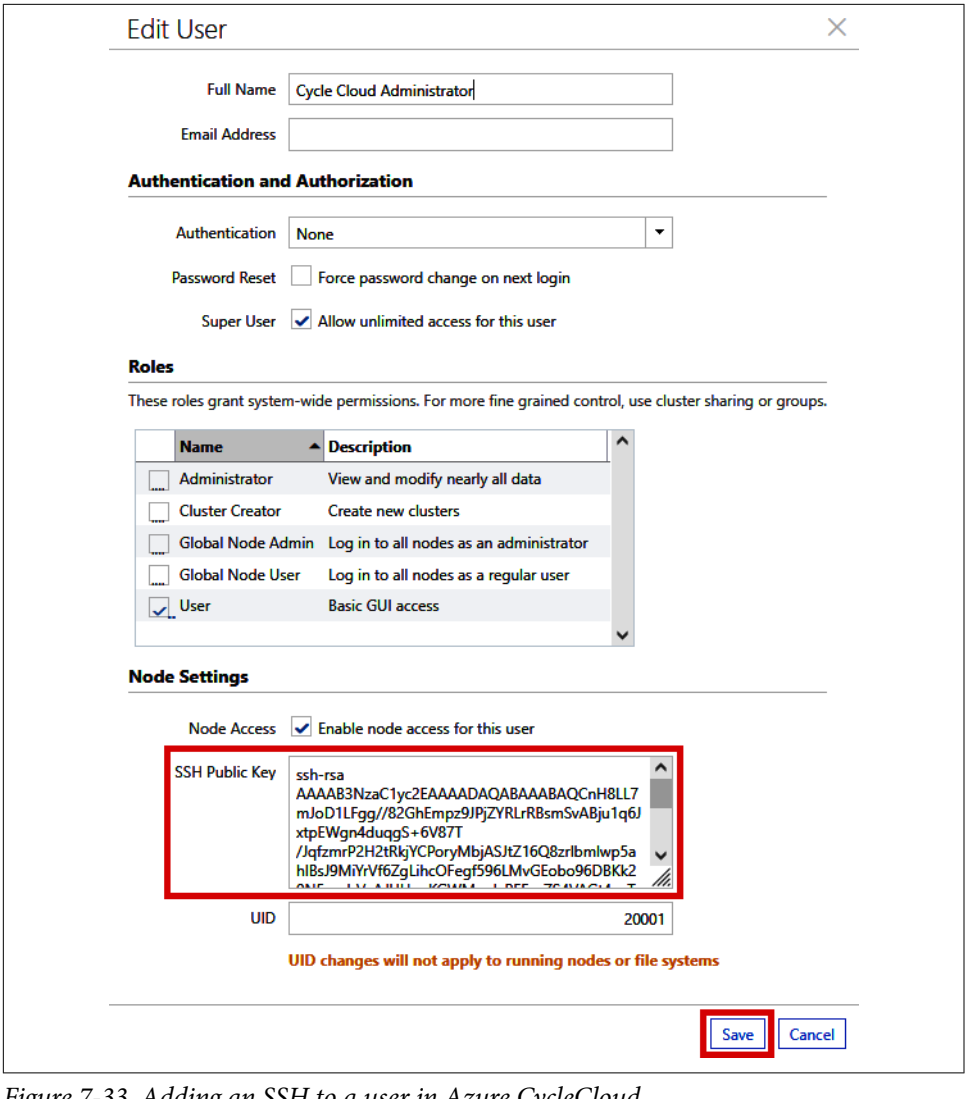


Figure 7-33. Adding an SSH to a user in Azure CycleCloud

Now we should be ready to remote into the cluster. Run the `ssh cyclecloudadmin@...` command (using your CycleCloud's IP address, of course) from the Cloud Shell. If successful, you should now be in the terminal of the cluster, and you should see a screen similar to Figure 7-34.

```
colby@Azure:~$ ssh cyclecloudadmin@20.228.200.223
The authenticity of host '20.228.200.223 (20.228.200.223)' can't be established.
ECDSA key fingerprint is SHA256:plPKcYHiaRHc4pvcYTaTgTcAwTC4XqiKnyKr80Z2dBg.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '20.228.200.223' (ECDSA) to the list of known hosts.
Activate the web console with: systemctl enable --now cockpit.socket

[cyclecloudadmin@ip-0A010004 ~]$
```

Figure 7-34. SSHing into a cluster

Yes, yes. I'm aware that it's been a long haul to get to this point, but I'll make the next few steps easier for you. We'll now prep the cluster for running Snakemake jobs. In the remote cluster terminal, run the following commands:

```
## Install miniconda
wget  https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh\
      -O /miniconda.sh
bash /miniconda.sh -b -p

## Add Conda to PATH (Change username here)
export PATH="/shared/home/cyclecloudadmin/miniconda3/bin/:$PATH"
conda init bash

## Create demo directory
mkdir snakemake_demo
cd snakemake_demo

## Get Snakemake demo files dependency requirements from repo
wget https://github.com/snakemake/snakemake-tutorial-data/
        archive/refs/tags/v7.4.2.tar.gz
tar --wildcards -xf v7.4.2.tar.gz --strip 1 "*/data" "*/environment.yaml"

## Create Snakemake environment (and activate it)
conda activate base
conda env create -n snakemake_env -f environment.yaml
conda activate snakemake_env
```

This will take a bit as it's installing Miniconda (a package/environment manager for Python) and Snakemake. For this example, we're using some really small sample files from a tutorial by the Snakemake team.[3]

3 For Snakemake tutorial data, see *https://oreil.ly/nIJ7o*.

Once the environment is finished being created, we can create a snakefile with the following four steps:

- bwa_map: Aligns a FASTQ file to a reference genome FASTA file and outputs a BAM file.
- samtools_sort: Sorts the reads in the BAM file.
- samtools_index: Indexes the BAM file for easier reading by other tools.
- bcftools_call: Performs variant calling on the BAM file and outputs a VCF file.

Create a new snakefile using your preferred text editing method (e.g., `nano snakefile`) and paste the following lines:

```
SAMPLE_IDS = ["A","B","C"]
rule all:
rule bwa_map:
  input:
    "data/genome.fa",
    "data/samples/sample.fastq"
  output:
    "mapped_reads/sample.bam"
  shell:
    "bwa mem input | samtools view -Sb - > output"

rule samtools_sort:
  input:
    "mapped_reads/sample.bam"
  output:
    "sorted_reads/sample.bam"
  shell:
    "samtools sort -T sorted_reads/wildcards.sample"
    "-O bam input > output"

rule samtools_index:
  input:
   "sorted_reads/sample.bam"
  output:
   "sorted_reads/sample.bam.bai"
  shell:
   "samtools index input"

rule bcftools_call:
  input:
    fa="data/genome.fa",
    bam=expand("sorted_reads/sample.bam", sample=SAMPLE_IDS),
    bai=expand("sorted_reads/sample.bam.bai", sample=SAMPLE_IDS)
  output:
    "calls/all.vcf"
  shell:
```

```
"samtools mpileup -g -f input.fa input.bam | "
"bcftools call -mv - > output"
```

If you peek inside the `/data/samples` folder, shown in Figure 7-35, you'll see that there are three samples to be run and the pipeline has a final step to combine all the outputs as a single VCF file. So we have four tasks to perform here.

```
(base) [cyclecloudadmin@ip-0A010004 snakemake_demo]$ conda activate snakemake_env
(snakemake_env) [cyclecloudadmin@ip-0A010004 snakemake_demo]$ nano snakefile
(snakemake_env) [cyclecloudadmin@ip-0A010004 snakemake_demo]$ ls
data  environment.yaml  snakefile  v7.4.2.tar.gz
(snakemake_env) [cyclecloudadmin@ip-0A010004 snakemake_demo]$ ls data/samples/
A.fastq  B.fastq  C.fastq
```

Figure 7-35. Creating a Snakemake file

Ready to run? To submit this snakefile on the cluster, run the following line:

```
snakemake –cluster qsub -j 4
```

Because this is a tiny set of samples, this will run quickly. Notice that the output points to four nodes to perform the actions as defined in our Snakemake pipeline. When finished, you should see a screen similar to Figure 7-36.

```
(snakemake_env) [cyclecloudadmin@ip-0A010004 snakemake_demo]$ snakemake --cluster qsub -j 4
Building DAG of jobs...
Using shell: /usr/bin/bash
Provided cluster nodes: 4
Job stats:
job      count    min threads    max threads
-----  -------  -------------  -------------
all          1              1              1
total        1              1              1

Select jobs to execute...

[Thu May 12 04:29:07 2022]
localrule all:
    jobid: 0
    resources: mem_mb=1000, disk_mb=1000, tmpdir=/tmp

[Thu May 12 04:29:07 2022]
Finished job 0.
1 of 1 steps (100%) done
Complete log: .snakemake/log/2022-05-12T042907.361460.snakemake.log
```

Figure 7-36. A finished Snakemake run

That's it! Your first Azure CycleCloud run is complete. The setup process is a bit of a chore, but the operationalization of your existing HPC logic should be fairly easy to transfer over. Since CycleCloud can use multiple common schedulers, most existing submission codes should work with only minor adjustments. Plus, since these

cloud-based clusters autoscale, this is a great tool for running tasks that need to elastically grow and shrink based on the workload that you throw at it.

Next, we'll talk about one final tool that is more of an automated offering for scalable secondary analysis bioinformatics pipelines in Azure: Microsoft Genomics.

Microsoft Genomics

Microsoft Genomics was, to my knowledge, the first genomics-specific cloud service on Azure.[4] This service is really interesting as it basically provides a cloud version of the Burrows-Wheeler Aligner (BWA) and the Broad Institute's Genome Analysis Toolkit (GATK) as an API. This service takes either a pair of FASTQ files (which it will align using BWA) or a BAM file as an input and will perform variant calling against a reference genome of your choice. This will return a VCF file of variants.

A benefit of this service is that it provides access to this automated pipeline at a low cost ($1 for the first 10 Gigabases + $0.10/Gb after that). This means that you can process 100Gb for about $10. Also, this service is compliant with multiple regulatory standards (ISOs 27001, 27018, and 9001, as well as HIPAA), if that's important for your particular use case.

One limitation of this service is that it supports only a few human reference genomes (`hg19m1`, `b37m1`, `hg38m1`, and `hg38m1x`) through either the `snapgatk` or `gatk4` process. So, if you're working in nonhuman genomics, this service isn't for you.

Alignment and Variant Calling with the msgen Package

To get started using the Microsoft Genomics service in Azure, you'll first need to spin up a genomics resource in your tenant.

You can find the genomics service just by searching "genomics" from the Marketplace in the Azure Portal, as shown in Figure 7-37. Click the "Create" button.

4 For more information on Microsoft Genomics, see *https://oreil.ly/I9L14*.

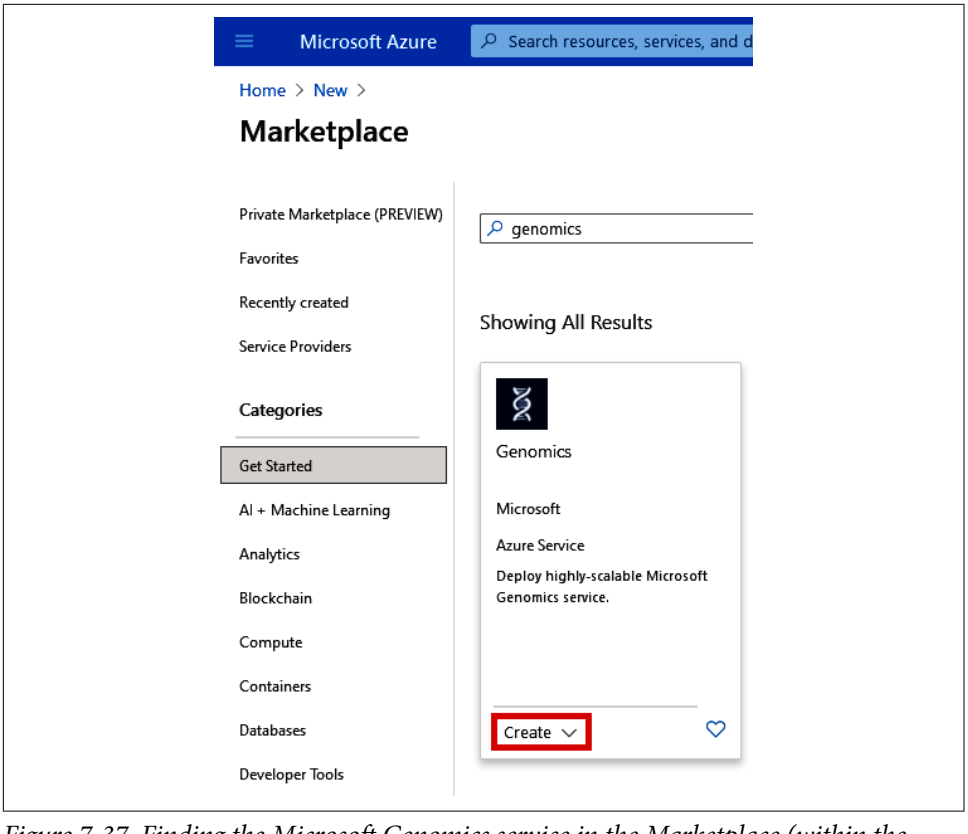


Figure 7-37. Finding the Microsoft Genomics service in the Marketplace (within the Azure Portal)

You'll then specify the account name of your service and in which "Subscription" and "Resource group" it should be placed, as shown in Figure 7-38.

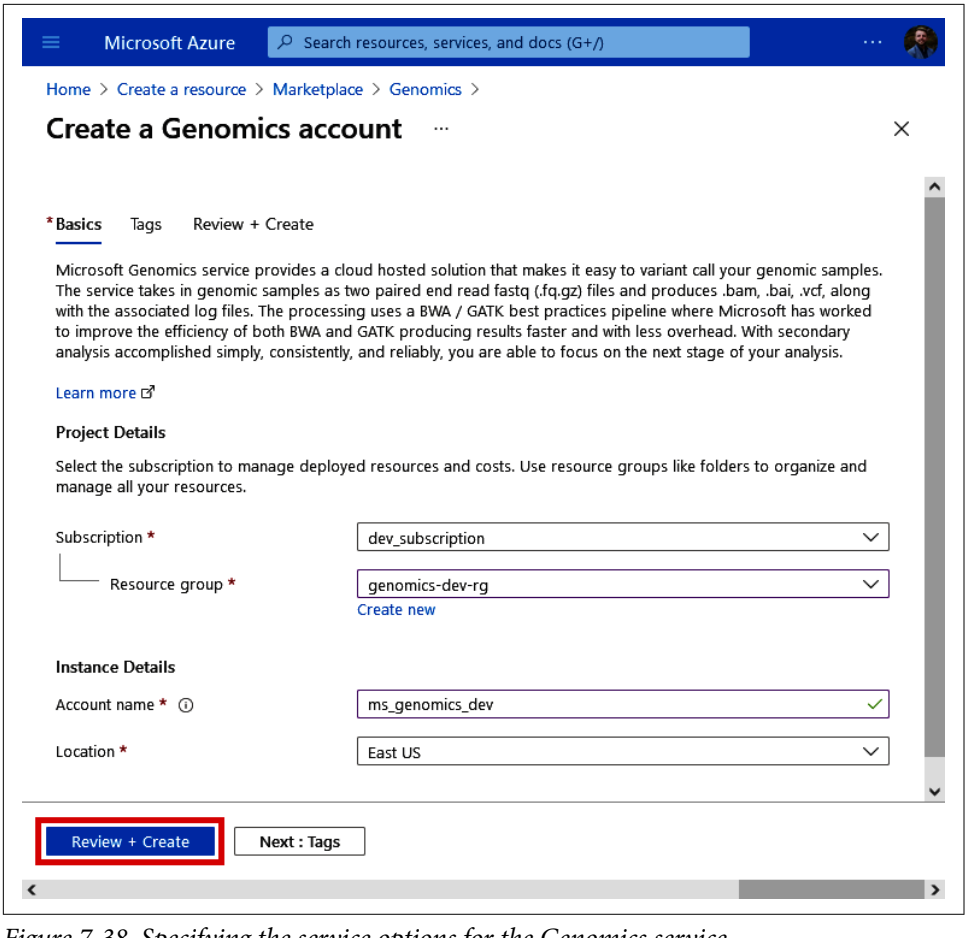


Figure 7-38. Specifying the service options for the Genomics service

Once the service has finished being provisioned in your resource group, you can go into the genomics service and view your access keys. See Figure 7-39. You'll need the resource name and primary access key later to submit workflows to this service.

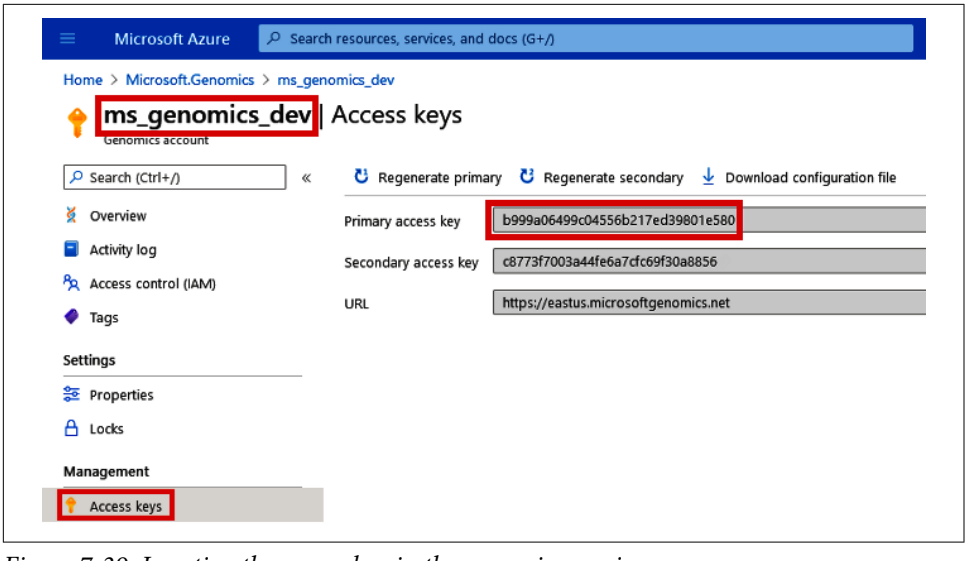


Figure 7-39. Locating the access key in the genomics service

In addition to the genomics service, you'll also provide information about the storage account that the service should be using to grab your input data and output your results.

Let's say you want to place your data in a data lake called "genomicsdls" in a container called "msgen-test." You can upload your data from the Portal UI (or using another tool like Azure Storage Explorer). See Figure 7-40.

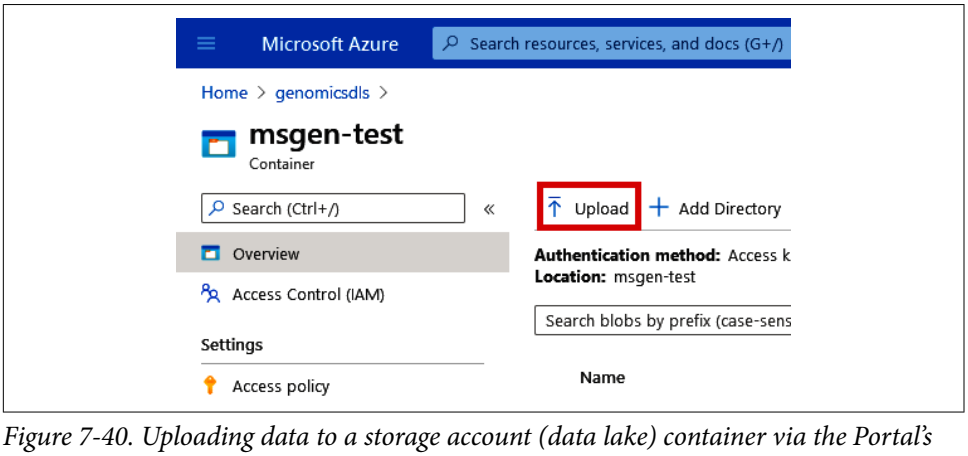


Figure 7-40. Uploading data to a storage account (data lake) container via the Portal's UI

Like the genomics service, you'll also need to grab the access key from the storage account (data lake). This is the key that the genomics service will use to interact with the storage location. See Figure 7-41.

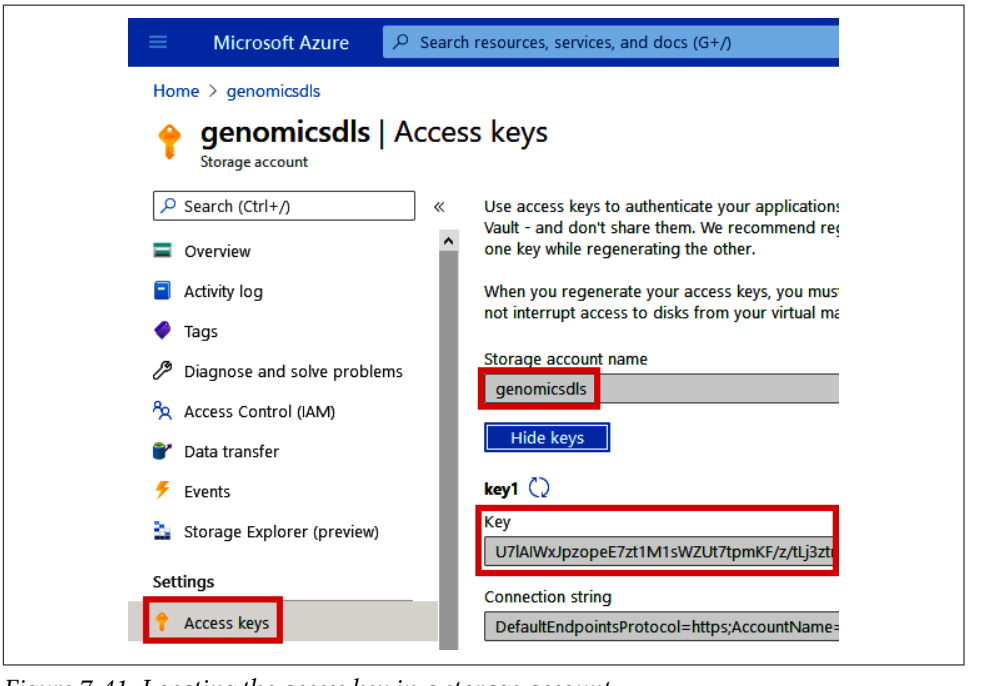


Figure 7-41. Locating the access key in a storage account

The Microsoft Genomics service is currently accessible via a Python 2.7 CLI (which is maintained by Microsoft Research) and an R package (which is maintained by yours truly).

If you prefer to use Python, make sure you have Python version 2.7.12 or later and then run the following:

```
pip install msgen
```

If you prefer to use R, you can install the package from GitHub by running the following:

```
remotes::install_github("colbyford/msgen")
```

Both libraries work by submitting a workflow that defines where the input data is and what the input parameters are. For example, say you have two input FASTQ files named "chr21_1.fq.gz" and "chr21_2.fq.gz" that you want to align and call variants based on the `b37m1` reference genome. You could submit using this code:

```
msgen submit \
  -api-url-base https://eastus.microsoftgenomics.net/api/workflows \
```

```
-access-key b999a06499... \
-process-args R=b37m1 \
-input-storage-account-name genomicsdls \
-input-storage-account-key U7lAlWxJ9z... \
-input-storage-account-container msgen-test \
-input-blob-name-1 chr21_1.fq.gz \
-input-blob-name-2 chr21_2.fq.gz \
-output-storage-account-name genomicsdls \
-output-storage-account-key U7lAlWxJ9z... \
-output-storage-account-container  msgen-test
```

Or you could submit using this code:

```
submit_workflow(subscription_key = "b999a06499...",
                region = "eastus",
                process = "snapgatk",
                reference = "b37m1",
                input_storage_account_name = "genomicsdls",
                input_storage_account_key= "U7lAlWxJ9z...",
                input_container_name = "msgen-test",
                blob_name_1 = "chr21_1.fq.gz",
                blob_name_2 = "chr21_2.fq.gz",
                output_container_name = "msgen-test")
```

Once the workflow has completed successfully, you can look back in your data lake and see the output files, as shown in Figure 7-42.

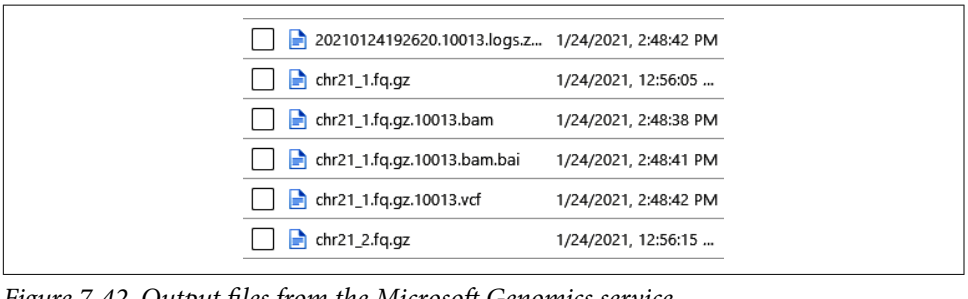


Figure 7-42. Output files from the Microsoft Genomics service

As you can see in Figure 7-42, there is an output VCF file and log. Also, since we provided a pair of FASTQs and inputs, the service will also align them to the reference genome we specified and output a BAM file and accompanying BAM index (BAI) file.

Also note that this entire workflow took less than two hours to complete. These input files are about 1.5GB in size (uncompressed), with just under 700 million bases.

And just like that, we've processed some sequence data through a managed bioinformatics pipeline with only a few commands. While this service isn't exactly "HPC," it is a fully automated pipeline that may meet the needs of groups that use GATK for their

secondary analysis pipelines. No need to reinvent the wheel if your wheel looks like the Azure-based version that's already made!

In this chapter, we've gone through a few different options for high-performance computing in Azure. To restate what I said at the beginning, I wouldn't expect someone to use all of these at once. I would suggest you think about the task you're trying to scale and then pick the service that will work best for you.

Services like Azure Batch or Azure CycleCloud are great for mimicking your existing HPC pipelines in the cloud. For Batch, it has its own basic scheduler. For CycleCloud, you can choose from one of a few popular ones. So, if you already have logic that works with a particular scheduler, CycleCloud may be the way to go.

In Chapters 5 and 6, we looked at Azure Databricks and Azure Machine Learning, respectively, which are both platform services. While these options allow you to scale and automate various compute-heavy tasks, they are quite different than on-premises HPC environments that you may have used in the past. So when would you pick an HPC service like Azure Batch or Azure CycleCloud over a PaaS offering like Azure Databricks or Azure Machine Learning?

Well, my recommendation would be to start with whatever feels the most natural (similar). If you're looking to cloudify existing HPC-based processes, then start with a cloud HPC service. If you're looking to scale some R or Python scripts (that are doing data transformation, bioinformatics, or machine learning tasks), Azure Databricks or Azure Machine Learning may be the better option. PaaS options are a best practice for modern cloud architectures, but getting up and running quickly is often the driver for selecting a tool. Plus, you can always change later.

In summary, Azure Batch provides an HPC service that comes with its own scheduler. This service comes with an API, where you can configure Batch jobs without worrying about managing a cluster. We can run tools like the Broad Institute's Cromwell workflow system on Azure Batch quite easily.

Using Azure CycleCloud, we can quickly create clusters and use common schedulers like OpenPBS and Slurm to operationalize various pipelines. This service is the closest to replicating on-prem HPC environments.

With the Microsoft Genomics service, we can quickly perform alignment and variant calling through a GATK-compliant pipeline with very little code. Using the Python or R libraries, we can simply point to a pair of FASTQ files in our genomics data lake and the service will handle the rest.

CHAPTER 8

Deployment, Security, Compliance, and Potpourri

While you may be a bioinformatics scientist who really cares only about analyzing your sequence data, there is a lot more that goes on when trying to manage a cloud environment. From security to costs, there are lots of facets to ensuring your cloud is scalable, private, and compliant, and doesn't break the bank. In this chapter, I'll be covering some of the ancillary things related to Azure cloud management. Trust me, if you get these things right, your IT team will love you.

Automating the Deployment of Cloud Resources

Throughout this book, we've spun up various cloud services such as a storage account, Databricks, and more. While we can continue to manually spin up these services, a key to scalable, enterprise-ready architectures is to have the entire set of resources be fully deployable in an automated way.

This ensures that if something were to get deleted or if we needed to redeploy everything in a different location across the globe, we could do so with ease. In fact, the latter scenario (deploying in another place on the planet) is a requirement that is becoming more common to meet regulatory requirements, especially with sensitive health data. This concept is known as *infrastructure as code* (IaC), where code dictates what gets deployed rather than relying on a human to remember what/how to deploy critical parts of an architecture.

Dev, Staging, and Prod

In the IT world, it's common to have separate environments for running your data-heavy workloads. The most common paradigm is to have three environments: development, staging (sometimes called *test*), and production.

The development environment serves as the place for us to create new things—a sandbox, you could say. This is for designing new architectures or applications and trying out various services. Dev is where we get things running smoothly. Then, once we're ready, we can promote our code/services/architecture to the next environment up.

Staging serves as the ephemeral area where we test whether our deployment works properly. Plus, we can also run tests surrounding performance and scalability, rollout of security measures, and more. It's sometimes common to destroy the staging area once our deployment has succeeded.

Finally, the production environment is the last stage where our deployment should land, assuming it has passed everything in staging, of course. The production environment is traditionally the most "locked down." Having a separate production environment provides us with an additional barrier so that we don't inadvertently mess up business-critical applications.

Lifting Your Deployment with ARMs and Biceps

While we could always just deploy things manually as we've done up until this point in the book, this gets cumbersome if there are lots of services to be deployed, each with their own set of options and settings.

Inside Azure, there are Azure Resource Management (ARM) templates that can be used to deploy virtually any service. In fact, Azure can automatically create ARM templates for the services you've already deployed. Let's take a look at one.

Navigate back to your data lake in Azure Storage that you created way back in Chapter Chapter 2. On the side panel, click the "Export template" option under "Automation." See Figure 8-1.

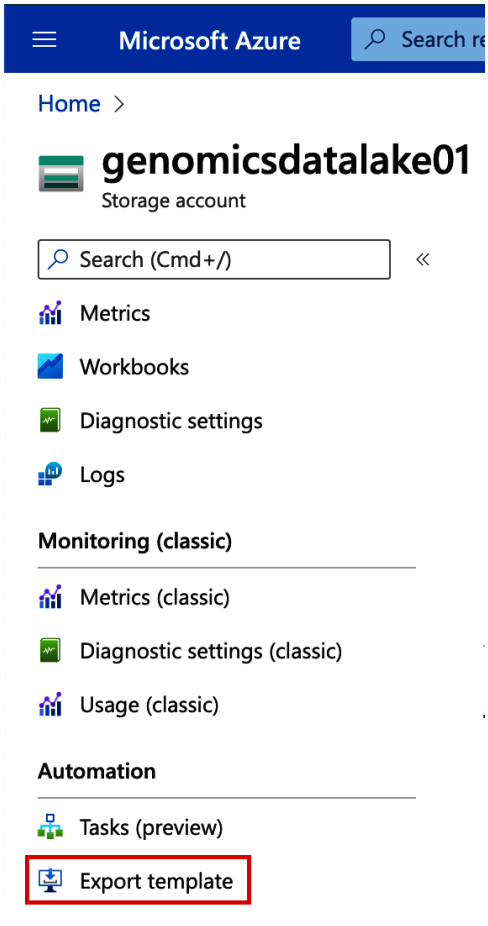


Figure 8-1. Locating the template export option for an Azure resource

In the panel that opens, shown in Figure 8-2, notice the long JSON file. This describes the information about the storage account, along with all the settings that we originally defined pertaining to the account's name, zone redundancy, the hierarchical namespace, containers, etc.

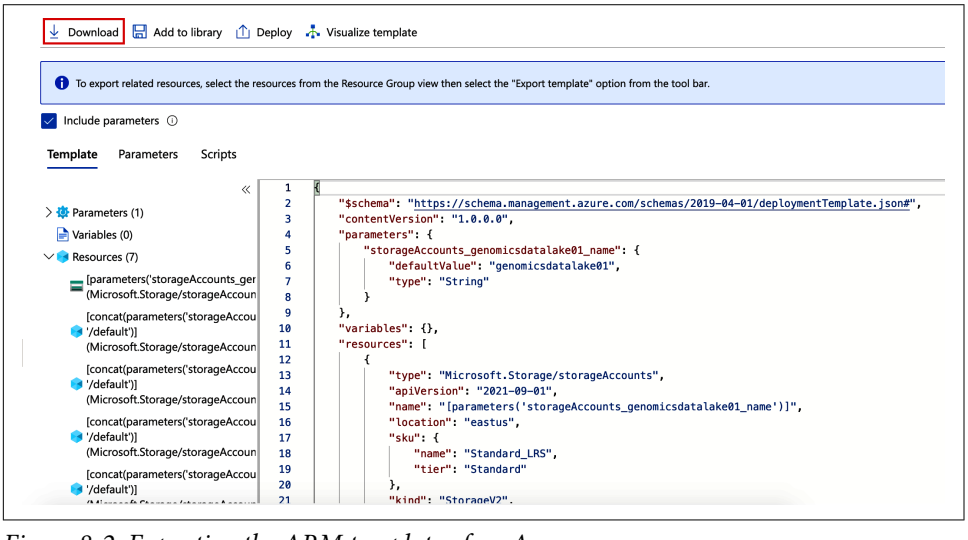


Figure 8-2. Exporting the ARM template of an Azure resource

If you take a look at the JSON body of the ARM template, you'll see the following attributes being defined for the Storage Account:

```
...
"parameters": {
    "storageAccounts_genomicsdatalake01_name": {
        "defaultValue": "genomicsdatalake01",
        "type": "String"
    }
},
"variables": {},
"resources": [
    {
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2021-09-01",
        "name": "[parameters('storageAccounts_genomicsdatalake01_name')]",
        "location": "eastus",
        "sku": {
            "name": "Standard_LRS",
            "tier": "Standard"
        },
        "kind": "StorageV2",
        "properties": {
            "defaultToOAuthAuthentication": false,
            "allowCrossTenantReplication": false,
```

```
            "isSftpEnabled": false,
            "minimumTlsVersion": "TLS1_2",
            "allowBlobPublicAccess": true,
            "allowSharedKeyAccess": true,
            "isHnsEnabled": true,
            "networkAcls": {
                "bypass": "AzureServices",
                "virtualNetworkRules": [],
                "ipRules": [],
                "defaultAction": "Allow"
            },
            "supportsHttpsTrafficOnly": true,
            "encryption": {
                "services": {
                    "file": {
                        "keyType": "Account",
                        "enabled": true
                    },
                    "blob": {
                        "keyType": "Account",
                        "enabled": true
                    }
                },
                "keySource": "Microsoft.Storage"
            },
            "accessTier": "Hot"
    ...
```

Download the ARM template by clicking the “Download” button at the top. This will download a *.zip* file that includes a *template.json* file and a *parameters.json* file.

Back in the Azure Portal, act like you're going to create a new resource. Using the search box, search for "Template deployment" and click the "Template deployment (deploy using custom templates)" option. See Figure 8-3.

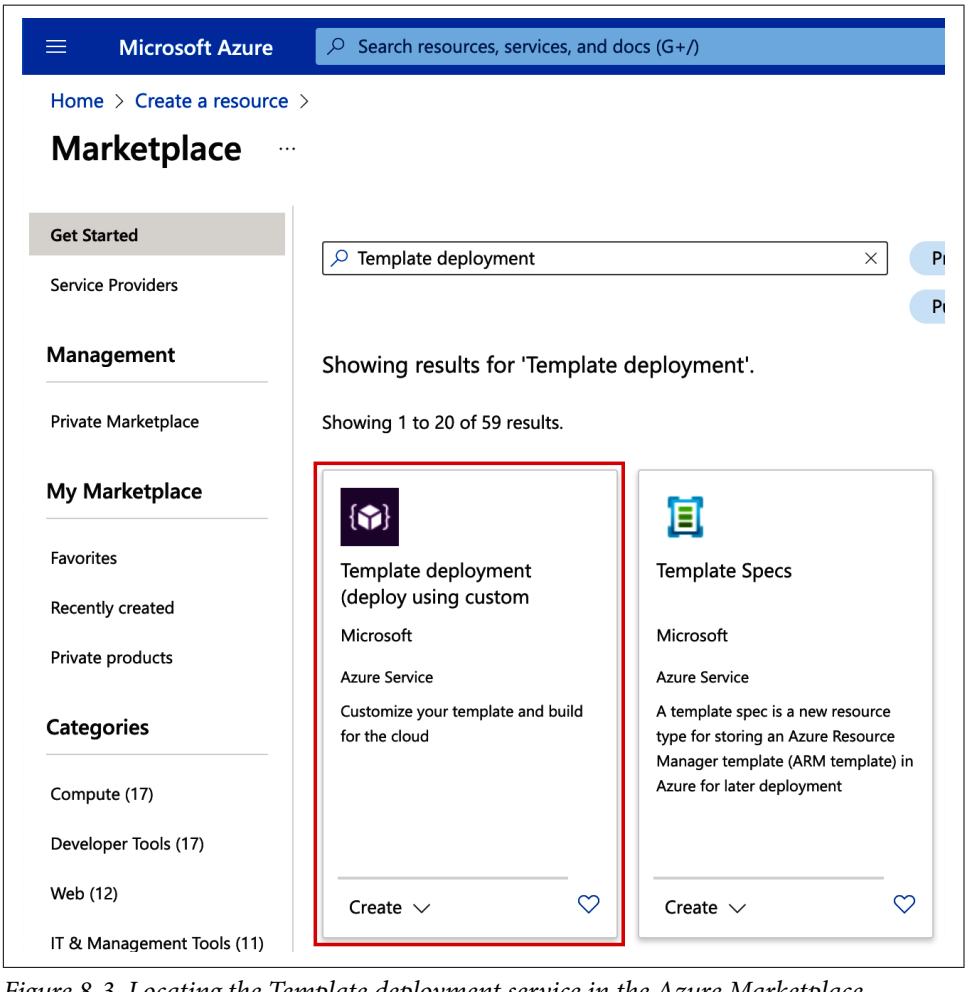


Figure 8-3. Locating the Template deployment service in the Azure Marketplace

In the “Custom deployment” blade that opens, shown in Figure 8-4, click the “Build your own template in the editor” link.

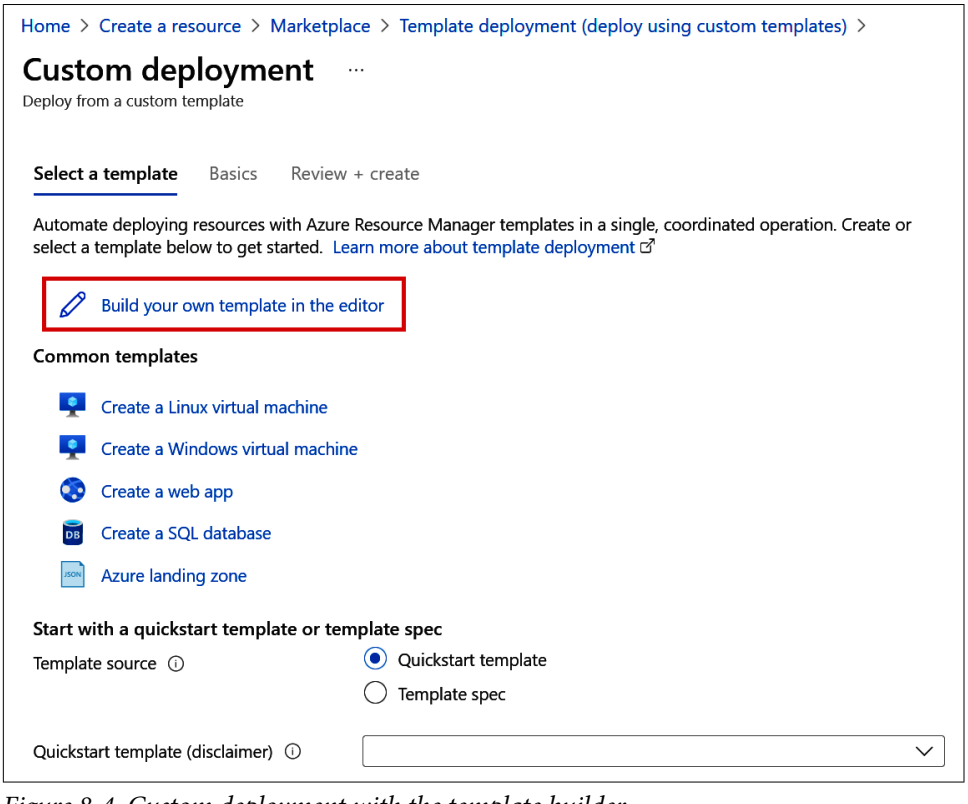


Figure 8-4. Custom deployment with the template builder

Click the “Load file” button at the top and upload your *template.json* file that you just exported. This should fill in the JSON template and parameters on the screen. Once complete, click the “Save” button at the bottom, as shown in Figure 8-5.

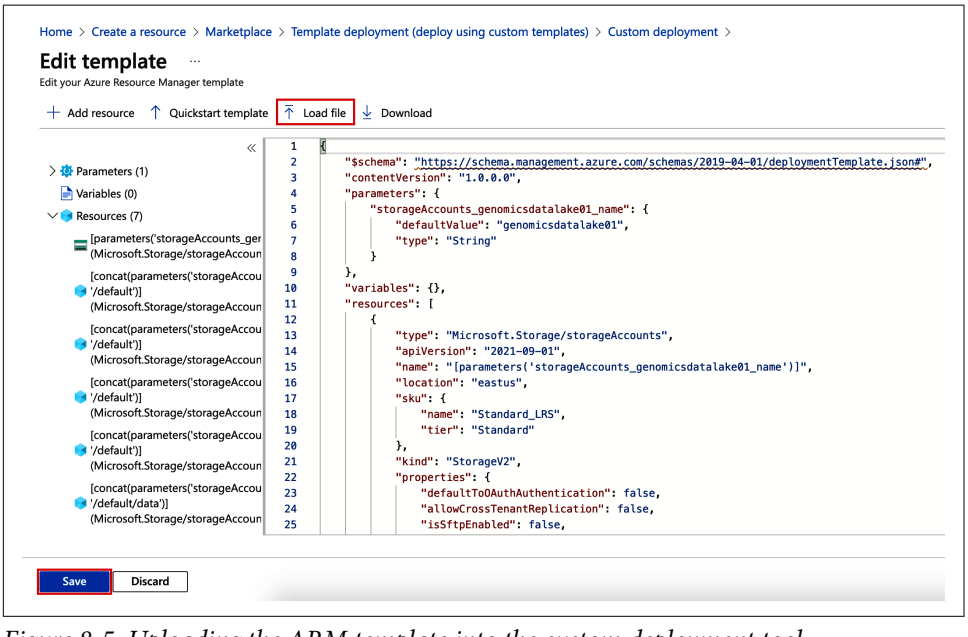


Figure 8-5. Uploading the ARM template into the custom deployment tool

Back on the custom deployment blade, shown in Figure 8-6, you can now see that the template has filled out what is to be deployed. This now shows seven resources to be created, which includes the storage account, containers, etc.

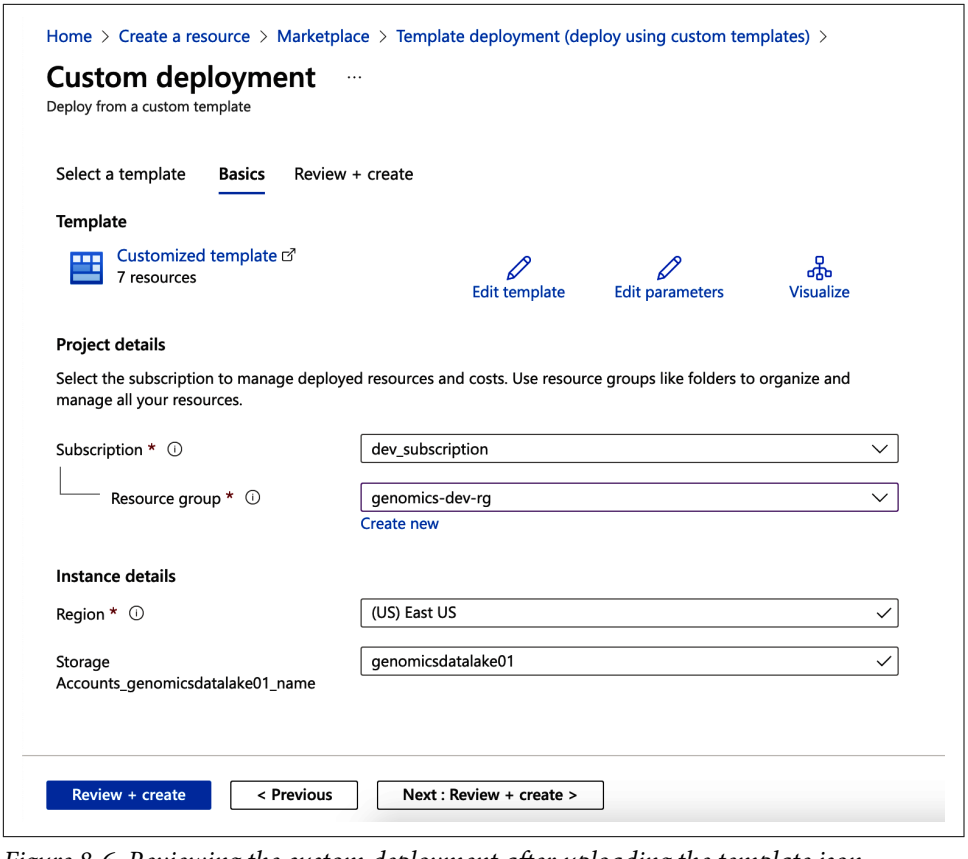


Figure 8-6. Reviewing the custom deployment after uploading the template.json

Note how easy it would be for us to name this data lake something else (like "genomicsdatalake02") and put it in another region. You could then click the "Review + create" button at the bottom to start the deployment (not necessary unless you want to actually deploy this).

Bicep

You may have noticed that the JSON information that describes our storage account is like 160 lines long, and storage accounts are among the simplest services to deploy. While it is convenient that Azure can generate the ARM templates for you, customizing them can prove to be a chore. Microsoft's answer to this is Bicep.

Bicep is a custom language that makes it easier to customize deployments in a more readable format. For example, we could deploy the same data lake in Bicep using the following code:

```
resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' = {
    name: 'genomicsdatalake01'
    location: 'eastus'
    sku: {
        name: 'Standard_LRS'
    }
    kind: 'StorageV2'
    properties: {
        accessTier: 'Hot',
        isHnsEnabled: true
    }
}
```

As you can see, this is considerably shorter than the ARM template equivalent. This is for two reasons: 1) the Bicep code is designed to be more condensed; and 2) the Bicep code is only declaring properties that are changed from the default (whereas the ARM template states everything).

Bicep works with the Azure Resource Manager to deploy as many resources in parallel as possible. Thus, deployment of various resources can occur at the same time without you defining the order in which these resources should be provisioned, as shown in Figure 8-7.

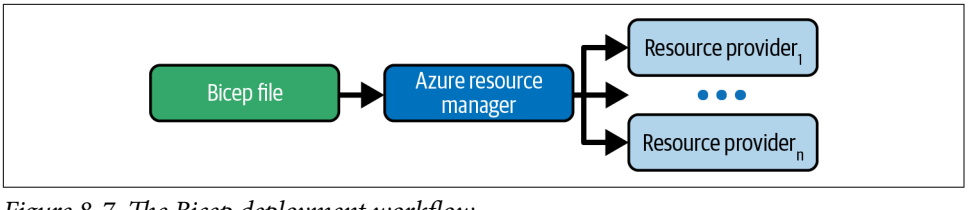


Figure 8-7. The Bicep deployment workflow

If you're interested in getting started with Bicep, it is available as a Visual Studio Code extension[1] and as part of the Azure CLI.

Bicep works well in GitHub Action pipelines and Azure DevOps Build pipelines, where you can deploy changes to your Azure environment upon committing changes to your Bicep code in a Git repository.

For more information about Azure Bicep, you can read the Microsoft documentation at *https://oreil.ly/y8L8f*.

1 For more on the VS Code Bicep extension, see *https://oreil.ly/kPrib*.

Security Planning

Unless you work in a completely transparent, nonprofit, open source research lab, keeping your data safe is super important. For biotech companies that house proprietary data, making sure that the data is accessible only by those who need to access it is paramount.

Some examples of private or proprietary data:

- A plant biotech company's sequence data for a new genetically modified seed
- A biotech company's protein data for a new monoclonal antibody that's in development
- A pharmaceutical company's clinical trial results
- A government research facility's prereleased data on a pathogen (like the United States Centers for Disease Control)

Outside of data that a company might want to keep private for proprietary/profit reasons, another main reason we may want to keep data private is due to its level of sensitivity. This is especially true with human data.

If you participate in any sort of genetic sequencing, whether by participating in a clinical trial or some sort of ancestry/consumer genetics testing, you likely only want your data shared with those who need to see it. At the very least, you would want to opt in to any data-sharing and third-party usage.

All of these notions should be taken into account as part of your overall cloud architecture for bioinformatics and genomics. So we'll cover some ways that we can ensure data security in your Azure environment.

Azure Active Directory

Active Directory is an identity directory service that has been around for decades as an on-premises tool. In fact, it was initially released by Microsoft in 1999. Active Directory helps to map the identities of applications, services, and individuals within a given network.

Azure Active Directory (AAD) is the cloud version of this system that helps to manage user access, roles, groups, and applications in your Azure environment. From the Azure Portal, you can view the Azure Active Directory, which will show you all of the users, groups, and applications in your tenant, as shown in Figure 8-8.

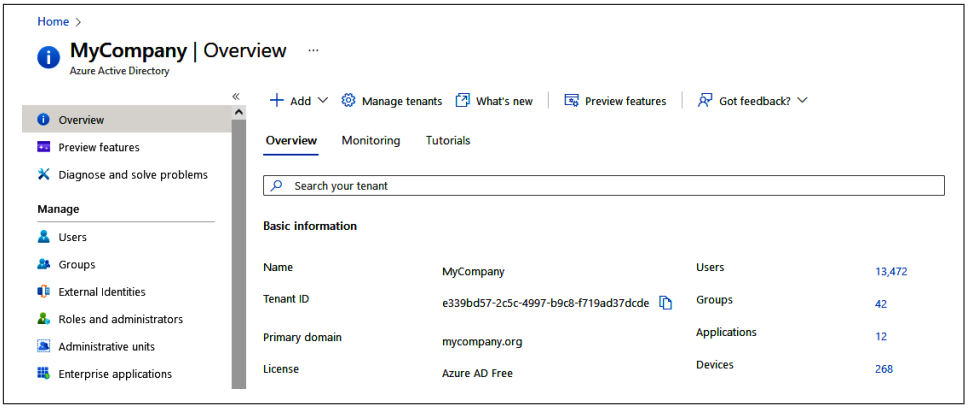


Figure 8-8. The Azure Active Directory service

Roles and groups

In AAD, users are registered into the environment, where we can give them access to resources through each resource's role-based access control (RBAC). There are lots of different roles we can grant to a user, but the three most common are:

Owner
: The user has full access to manage all resources, including the ability to assign roles in Azure RBAC.

Contributor
: The user has full access to manage all resources but does not have the ability to grant access to any other users.

Reader
: The user can view all resources but cannot make any changes.

Aside from these three main user roles, you may also wish to create custom roles within your organization.

Granting a user access is simple (assuming you have permission to do so). From any of the Azure services, locate the "Access Control (IAM)" option on the side menu, as shown in Figure 8-9. Then, click the "+ Add" button and "Add role assignment."

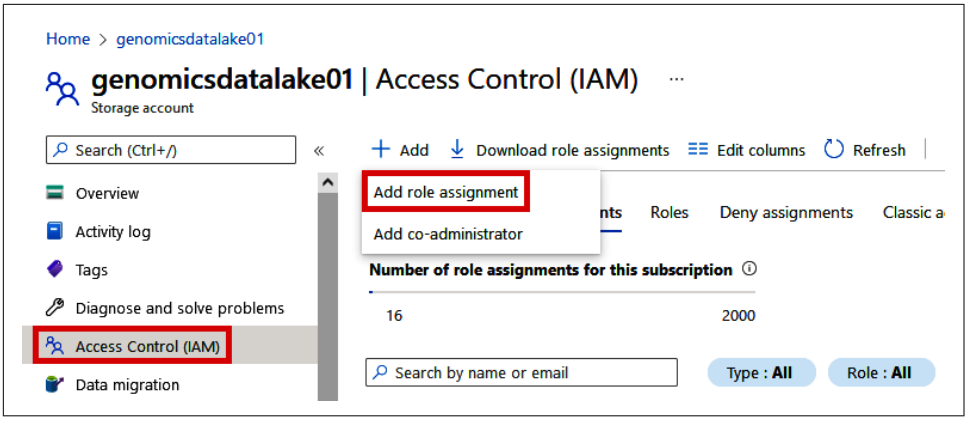


Figure 8-9. The Access Control tool for Azure resources

Then select the role you wish to grant from the Role tab and then pick the user(s) or group(s) to which you want to grant the role. See Figure 8-10.

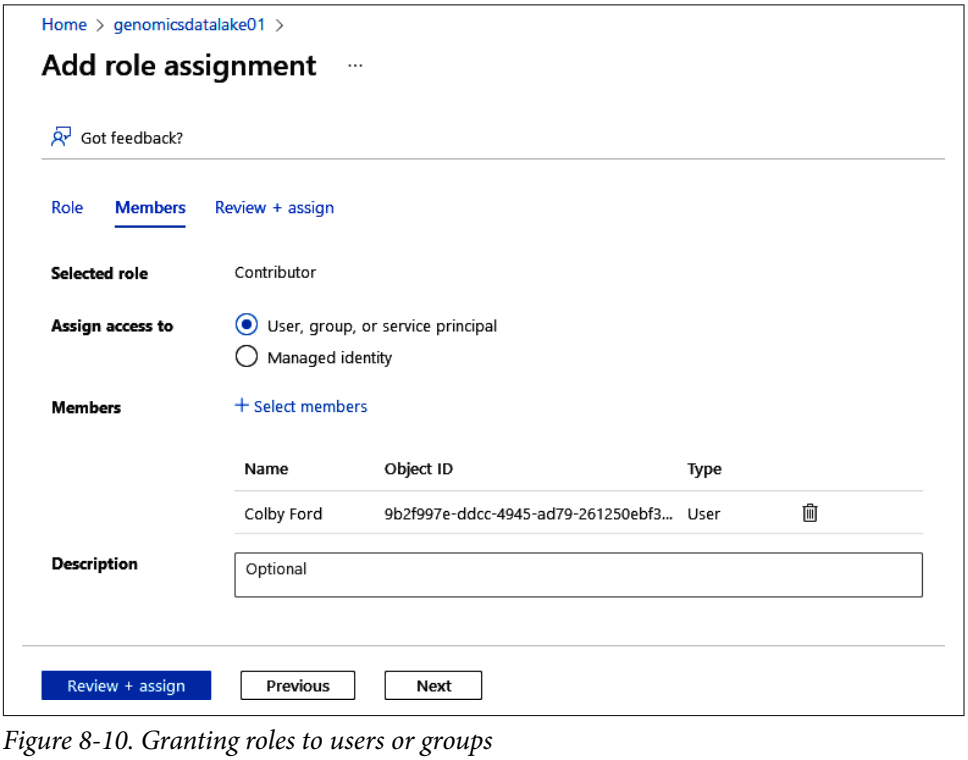


Figure 8-10. Granting roles to users or groups

After clicking the "Review + assign" button, the user(s) or group(s) will be granted the role to the resource at hand.

Setting up custom roles gives you finer controls to define what a user can do across your Azure services. For example, you could make a role that gives users the ability to see certain data lakes, HPC services, and some limited billing information. To learn more about Azure custom roles, visit *https://oreil.ly/ckg7i*.

Note that multiple users can be organized in groups, which makes the assignment of roles even easier. My recommendation is to spend the time to create groups that make sense for your architecture. For example, if you work for a biotech company that develops multiple products across various teams, perhaps it makes sense to create AAD groups to grant access to resources and data as it pertains to their individual needs. Maybe some groups need access to some resources, while others do not. RBAC makes this process quite simple.

Role-Based Access Controls and Access-Control Lists

While role-based access controls (RBACs) grant access to Azure resources, access-control lists (ACLs) allow access to data or directories in your data lake. Inside the data lake you created in Chapter 2, we can control access to individual directories and files by defining the ACL.

If you use Azure Storage Explorer, you can control ACLs by clicking the directory or file of interest and clicking the "Manage ACLs" button in the toolbar. For example, if you want to control access to the NCBI data that we used in Chapter 4, you can click the *ncbidata/* folder and then the "Manage ACLs" button, which will open up a "Manage Access" window. From this window, you can add specific users or groups and define whether you want them to have read, write, or execute permissions. See Chapter 4.

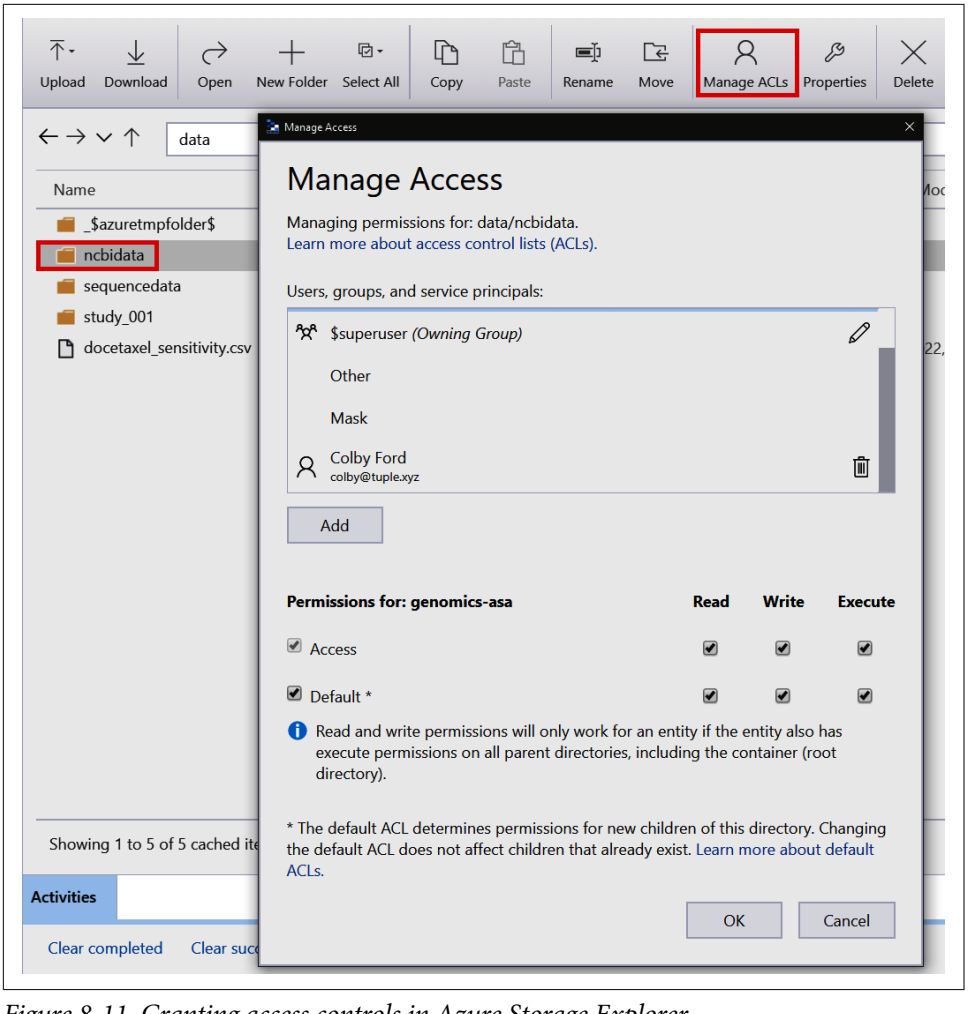


Figure 8-11. Granting access controls in Azure Storage Explorer

If you don't use Azure Storage Explorer, you can set ACLs from within the Azure Portal, shown in Figure 8-12. Navigate into your Azure Storage service (the resource for your data lake) and click the "Manage ACL" option on the side menu.

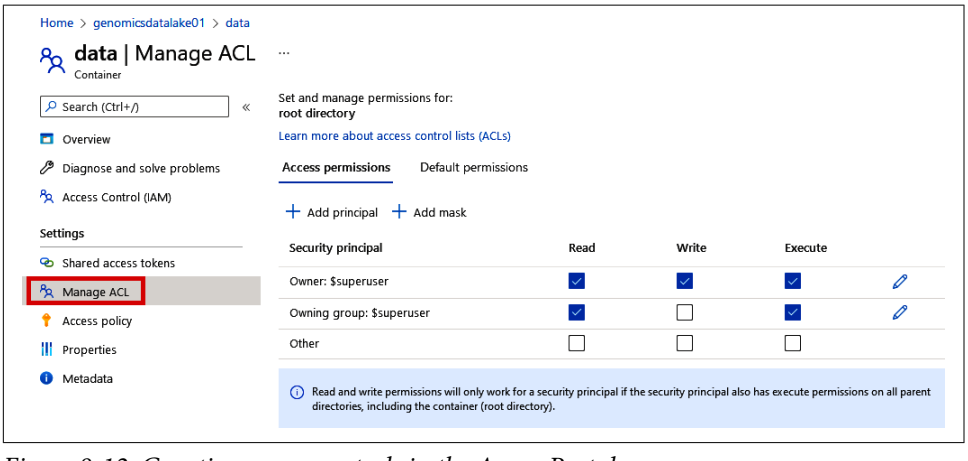


Figure 8-12. Granting access controls in the Azure Portal

With both of these options, we can specify who (or what group) can access and make changes in which directory, further securing our data. This, along with RBACs, can provide the complete access control to resources and fine-grained controls on data. This will then help you guarantee that only the people who *should* see the data *can* see the data.

Compliance

For any bioinformatician in the human genomics space, remaining compliant to certain standards is extremely important when collecting and storing data. In the United States, HIPAA is the main compliance goal to meet. Whether it's HIPAA, GDPR, or other privacy and data compliance standards, Azure can almost always meet these requirements with just a little effort. Many compliance standards have rules around data access, data sharing, and data removal.

Thus, it's important to have your data organized in such a way that you can find the data you want, grant access as needed, and be able to copy/move/update/delete it upon request.

HIPAA, HITECH, and HITRUST

In the US, you'll likely come across these three health-related compliance standards:

- Health Insurance Portability and Accountability Act of 1996 (HIPAA) sets requirements for using, disclosing, and handling protected health information (PHI).
- Health Information Technology for Economic and Clinical Health Act of 2009 (HITECH) extends HIPAA for electronic health records and more modern, data-driven technology in health care.
- Health Information Trust Alliance (HITRUST) is a private company that sets standards, based on HIPAA/HITECH, and provides certification.

Under HIPAA, there are 18 broad types of data that are deemed PHI:

- Name
- Social Security number
- Device identifiers and serial numbers
- Address
- Medical record number
- Web URLs
- Dates (smaller than years)
- Health plan beneficiary number
- IP addresses
- Telephone number
- Account numbers
- Biometric identifiers (fingerprints, voice recording, etc.)
- Fax number
- License number
- Face photos
- Email address
- Vehicle identifiers
- Any other unique ID

Note that while some of these items may not seem all that sensitive, PHI attributes are often things that could link together and identify a person. For example, a telephone

number isn't all that sensitive, but it's easy to look up a person's name from their phone number. So if you had a medical record that happened to have a phone number on it, it would be quite easy to learn who the patient is.

With HIPAA, there are a few main rules:

Privacy Rule
: Defines the limits to which PHI can be shared and describes the authorization process that an entity must go through with the patient to grant permission on sharing their data with other parties. This rule also gives patients the right to obtain copies of their medical records and to request corrections.

Security Rule
: Defines the standards related to the protection of PHI and regulates the methods and procedures regarding its storage, accessibility, and transmission. There are multiple safeguards to this security rule:

- Administrative Safeguards
  - Defines the assignment of a HIPAA security compliance team. In other words, the physical people who should oversee the compliance process.
- Technical Safeguards
  - Covers the encryption and authentication methods for controlling data access.
- Physical Safeguards
  - Handles the protection of the electronic systems, data, or equipment within the facilities of your organization.

Enforcement Rule
: From the HITECH Act, this expands the HIPAA Privacy Rule and Security Rule by increasing the penalties for any violations. It addresses disclosure rules around entities and their third parties. This includes federal privacy and security breach reporting requirements, restrictions on sales and marketing, and more.

Transaction Rule
: Covers the transactions and sets of codes used in HIPAA transactions such that they are used correctly and ensures the safety, accuracy, and security of medical records. This includes common medical coding standards such as ICD-9, ICD-10, HCPCS, CPT-3, CPT-4, and NDC.

Identifiers Rule
: Provides the standards for identifying the payer(s) and provider(s) in health-care transactions. This includes three unique identifiers:

- National Provider Identifier (NPI)

  - A 10-digit number used to identify health-care providers.
- National Health Plan Identifier (NHI)
  - An identifier used to identify health insurance plans and payers with the Center for Medicare and Medicaid Services (CMS).
- Standard Unique Employer Identifier
  - Identifies an employer entity transactions. This is the same as the federal Employer Identification Number (EIN).

Oddly enough, there aren't nearly as many regulations regarding the specific usage of genomics data. Even in the United States, the main genetics-related legislature is from the Genetic Information Nondiscrimination Act (GINA) of 2008, which prohibits companies from discriminating against people based on genetic information. In 2013, HIPAA was updated to include genetic information as PHI. Thus, we should treat genetic data as sensitive as any other type of PHI.

While this is a gross oversimplification of HIPAA, I wanted to include the basics here as a reference. When working with human samples in a clinical trial or research study, the protection of your participant's information should be paramount. So keep an eye out for patient attributes that are inadvertently being shared as you may be violating HIPAA rules.

GDPR

For any European readers, this acronym likely sounds familiar. The General Data Protection Regulation (GDPR) is a newer privacy law from the EU that contains requirements for processing personal data. This law covers individuals ("subjects") who are located in the European Economic Area (EEA) and applies to any organization that is housing or processing these individuals' data.

While GDPR is a large, complex set of rules, here are the basic principles (as summarized by Microsoft's Journey to GDPR Guide):[2]

Discover
: Identify if personal data exists, for whom, and where it resides.

Manage
: Govern how personal data is accessed and used.

Protect
: Establish security controls to prevent, detect, and respond to vulnerabilities and data breaches.

2 See the ebook Accelerate GDPR Compliance with the Microsoft Cloud at *https://aka.ms/gdprebook*.

Report
: Keep required documentation, track the execution of data requests, and report breach notifications.

The trickiest part of GDPR is that it covers any personal data of individuals in the EEA, regardless of your company's activity in EEA countries. If you have data of user/patient/subject/individual in your environment and the person is in the EEA, GDPR compliance applies to you.

A common design pattern today is to replicate your entire architecture for GDPR-covered persons. This means that you could set up the deployment of your entire architecture and redeploy everything in a Europe-based Azure region. All of the compute-based applications would then simply point to the data repositories (data lakes, warehouses, etc.) that are in the European environment, which helps to segregate the environments to further ensure compliance to GDPR.

Azure Blueprints

Reading through the few compliance topics may have you overwhelmed and feeling like it's going to be impossible to be compliant. While I won't lie and act like compliance is easy (or remotely interesting), Azure can help you achieve compliance to a variety of standards using a tool called Azure Blueprints.

Blueprints are a set of organizational standards and requirements for compliance or policy adherence. They allow us to create a set of artifacts, which can include:

Role Assignments
: Using RBAC, we can enforce or limit the granting of certain roles in the Azure environment to help ensure people or systems can only access the things they are supposed to access.

Policy Assignments
: Policies are a system of rule that allow or deny various activities in your cloud environment. For example, we could create a policy that limits the regions users can use or sets a budget for a given subscription.

ARM Templates
: For various Azure services, we can use ARM templates to preset the options to meet our organizational requirements. For example, we could ensure that all storage accounts only allow access from a private network and use a specific encryption key.

Resource Groups
: We can create new resource groups for use by other Blueprint artifacts. This mainly serves as a placeholder to organize resources while the rest of the Blueprints artifacts are put into place.

In fact, Azure already has a set of sample Blueprints for various compliance standards. You can simply click the sample Blueprint for the standard you're looking to achieve or start from scratch and define your own set of rules. See Figure 8-13 for a list of example Blueprints.

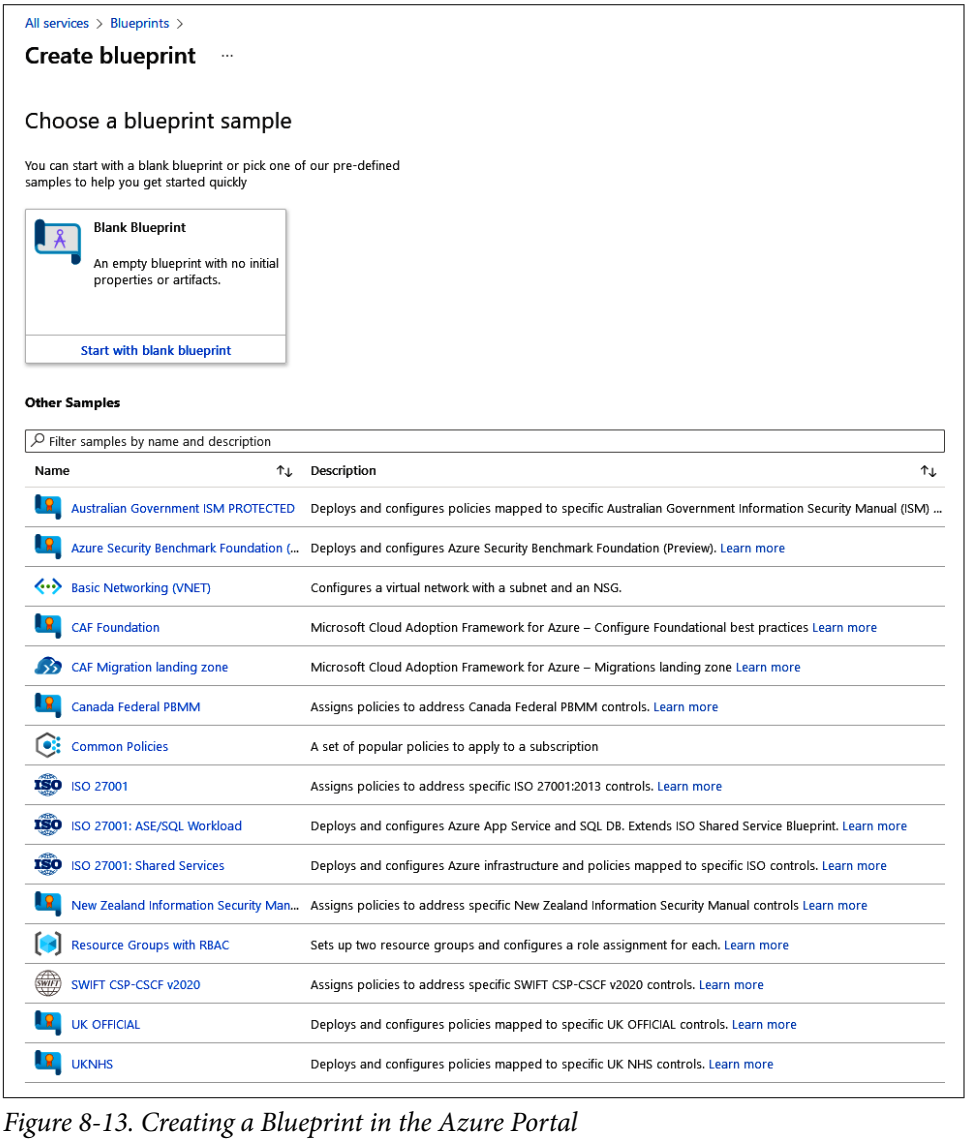


Figure 8-13. Creating a Blueprint in the Azure Portal

You can begin creating your own Blueprints by searching for "Blueprints" in the Azure Portal. To learn more about all the compliance offerings in the Azure cloud, visit *https://oreil.ly/nvmKy*.

For the full set of Azure policies for HIPAA HITRUST 9.2, see *https://oreil.ly/DYhqj*.

Cost Considerations

When architecting your cloud environment for genomics, understanding how cloud resources are charged to your subscription is key to planning and budgeting. In fact, there are subtle differences in the way different types of cloud services charge you, and knowing these differences can help you avoid surprises when your Azure bill shows up.

Azure Pricing Calculator

A very useful tool for cost planning is the Azure pricing calculator, shown in Figure 8-14. This online calculator gives us the ability to add various services and then select the options to simulate their monthly costs.

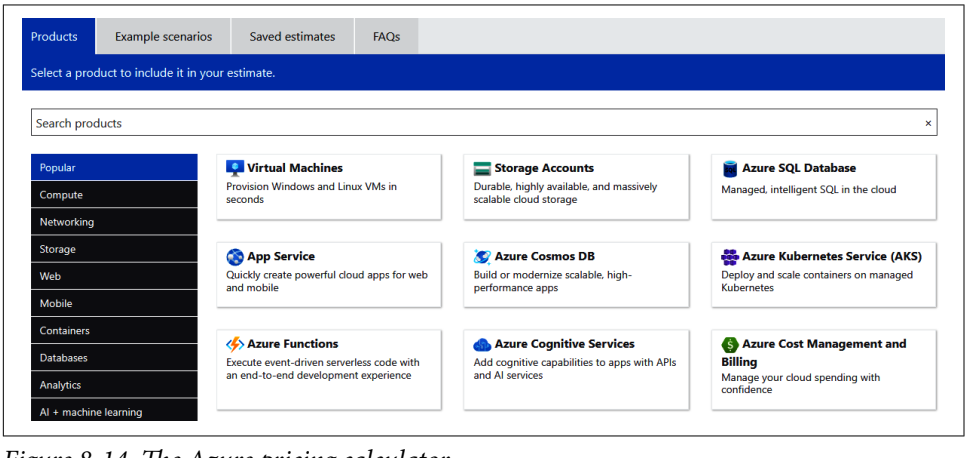


Figure 8-14. The Azure pricing calculator

For example, if you click the storage account item on the calculator and change all the options to match the storage account that we created for our data lake in Chapter 2, we can see the estimated monthly costs per terabyte of data. See Figure 8-15.

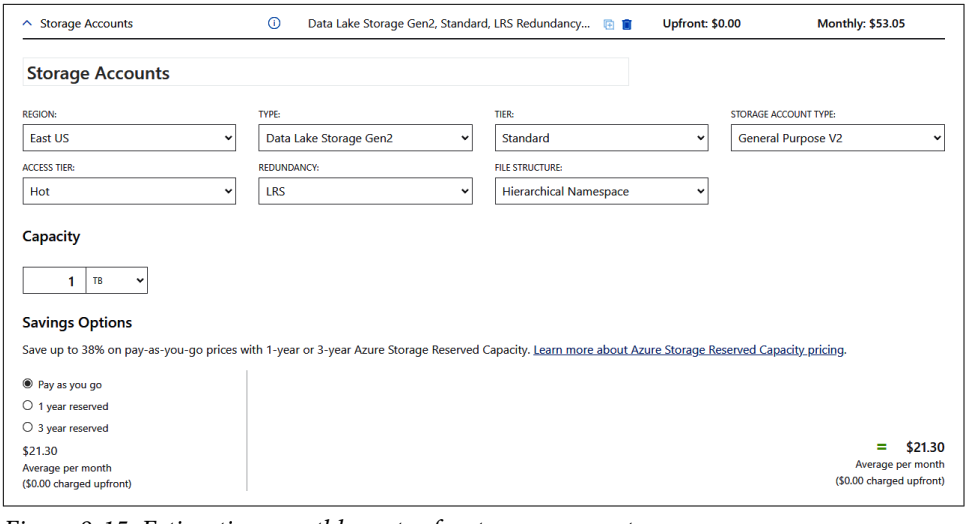


Figure 8-15. Estimating monthly costs of a storage account

This tool also helps to simulate other options. For example, we can estimate the costs of running certain analyses in Databricks (from Chapter 5). We could toggle through the different VM types to see how our costs may change per hour. See Figure 8-16.

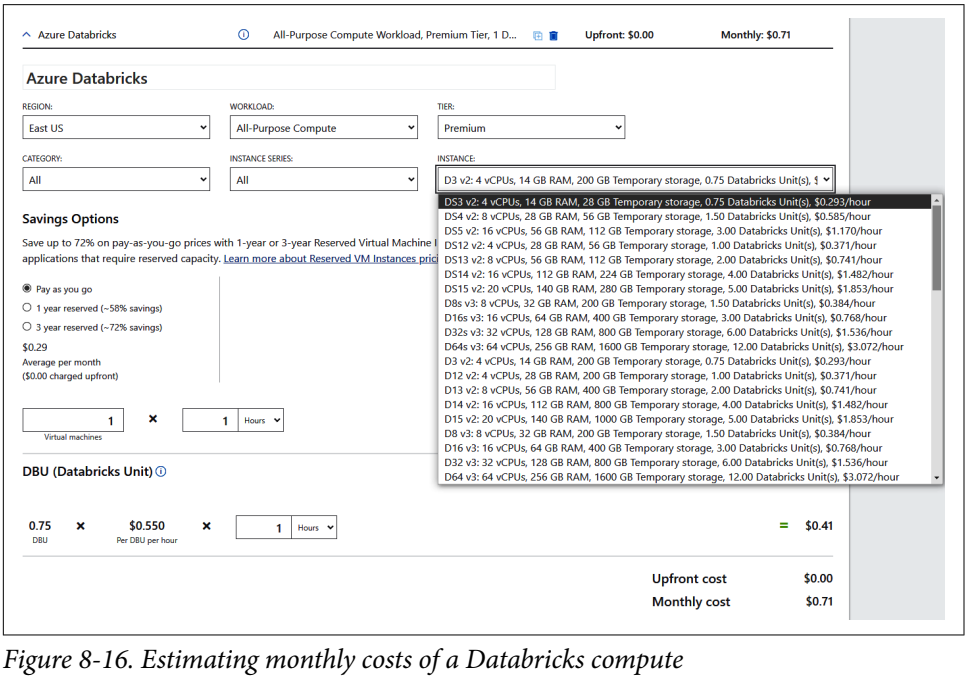


Figure 8-16. Estimating monthly costs of a Databricks compute

This is especially useful if we're running completely distributed tasks in PySpark that scale linearly. That is, if I have double the amount of cores, the process takes half the amount of time. If this is true, then increasing the VM size may increase the per-hour cost, but you'll end up running the VMs for less time. So this calculator is a good tool for understanding the cost implications when balancing scalability and selecting options.

Retail Pricing Versus Enterprise Agreements

Way back in Chapter 1, we created a PAYG subscription. A PAYG subscription is subject to full retail pricing, which includes the prices that you see on the Azure pricing calculator.

If you've been spinning up resources in your company's Azure tenant (and you work for a large company), you may be using a different type of subscription from your company's Enterprise Agreement.

Enterprise Agreements are a special type of arrangement between your organization and Microsoft; they offer different levels of pricing discounts on compute or storage and provide better support options. This is all dictated by the amount of money that the organization is expected to spend on a yearly basis in Azure.

If you've been playing with the Azure pricing calculator, you may have noticed the "Savings Options" section for each resource type. As part of an Enterprise Agreement, Microsoft may discount the prices on various Azure resources if you commit to using a certain amount on a yearly (or three-year) basis. See Figure 8-17.

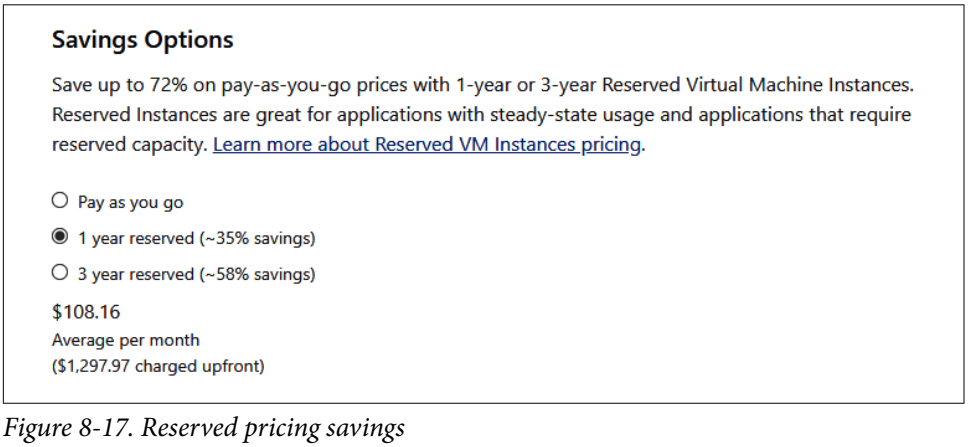


Figure 8-17. Reserved pricing savings

While I don't expect every cloud bioinformatician to have to manage the whole cloud environment at this level of detail, understanding how things cost in the cloud is useful for planning and budgeting. Plus, it helps your IT teammates who actually have to manage subscriptions and the Azure tenant as a whole.

Budgeting Examples

Luckily, even if you aren't on the hook for managing the entire Azure tenant, you can easily manage costs in your group's subscription by using the built-in tools in Azure.

From the Azure Portal, navigate to your subscription, as shown in Figure 8-18. (Note: You'll need to have at least contributor access at the subscription level to the cost management tools for the subscription.) To start, we'll be creating a budget.

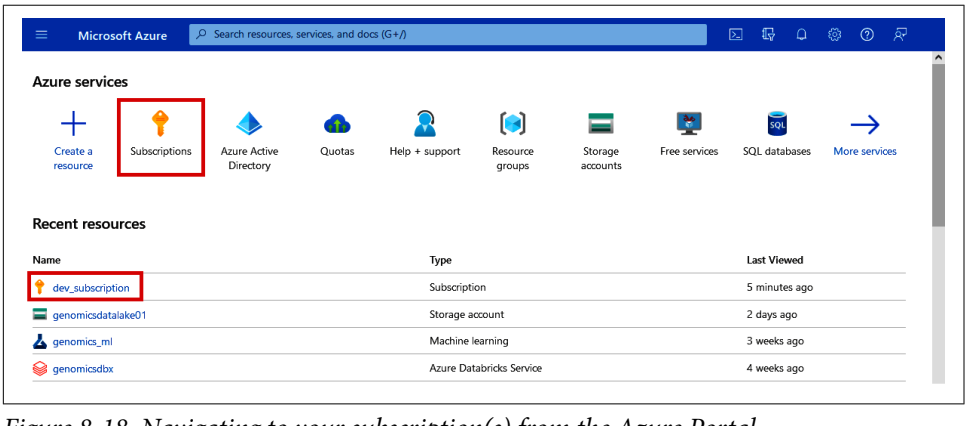


Figure 8-18. Navigating to your subscription(s) from the Azure Portal

Once in the subscription, look for the "Budgets" item under "Cost Management" on the side menu. Then, click the "+ Add" button at the top, as shown in Figure 8-19.

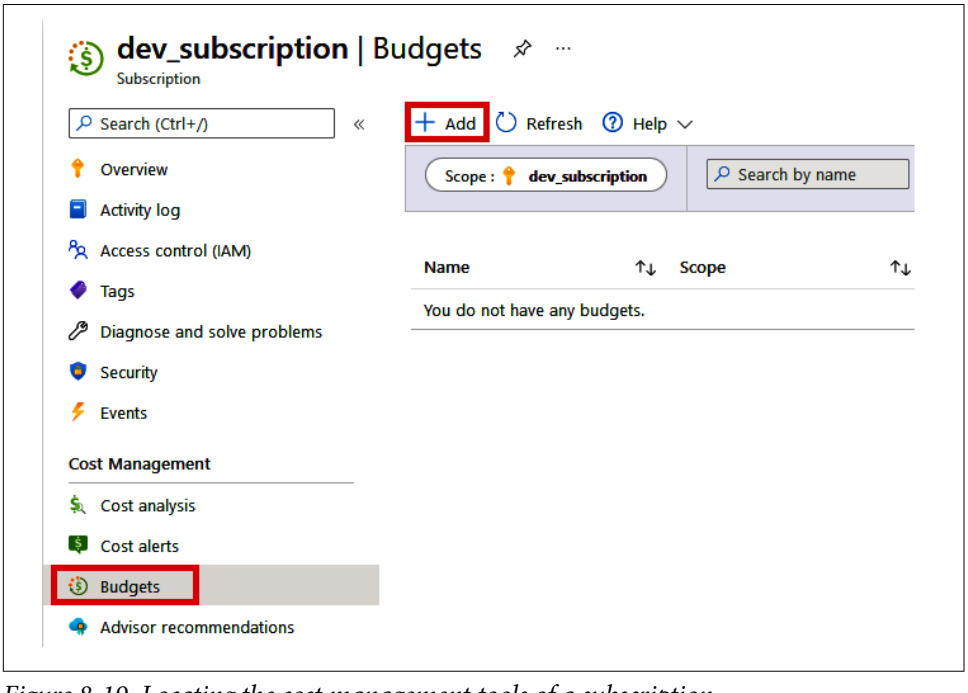


Figure 8-19. Locating the cost management tools of a subscription

In the "Create budget" blade, shown in Figure 8-20, you can set the scope of the budget. For example, we could set a subscription-wide limit or set budgets for each of the resource groups. This is useful if individual teams have their own resource group(s) and you want to manage costs at that level. We can then set the name and expiration date of the budget and set the budget amount. Once you've set these items, click the "Next >" button at the bottom.

Home > Subscriptions > dev_subscription >

Create budget

Budget

✓ Create a budget ② Set alerts

Create a budget and set alerts to help you monitor your costs.

Budget scoping

The budget you create will be assigned to the selected scope. Use additional filters like resource groups to have your budget monitor with more granularity as needed.

Scope: dev_subscription
Change scope

Filters: ResourceGroupName : **genomics-dev-rg** ✕ Add filter

Budget Details

Give your budget a unique name. Select the time window it analyzes during each evaluation period, its expiration date and the amount.

* Name	monthly_genomics_budget		
* Reset period ⓘ	Monthly		
* Creation date ⓘ	2022	May	1
* Expiration date ⓘ	2024	April	30

Budget Amount

Give your budget amount threshold

Amount ($) * 1000

Previous Next >

Figure 8-20. Setting a budget for a subscription and resource group

Then, on the final page, you can set the conditions on which you want to be alerted about the budget usage. You can set a list of emails to be contacted when the budget is partially or fully spent or is forecasted to exceed a certain percentage. Once finished, click the "Create" button to create the budget. See Figure 8-21.

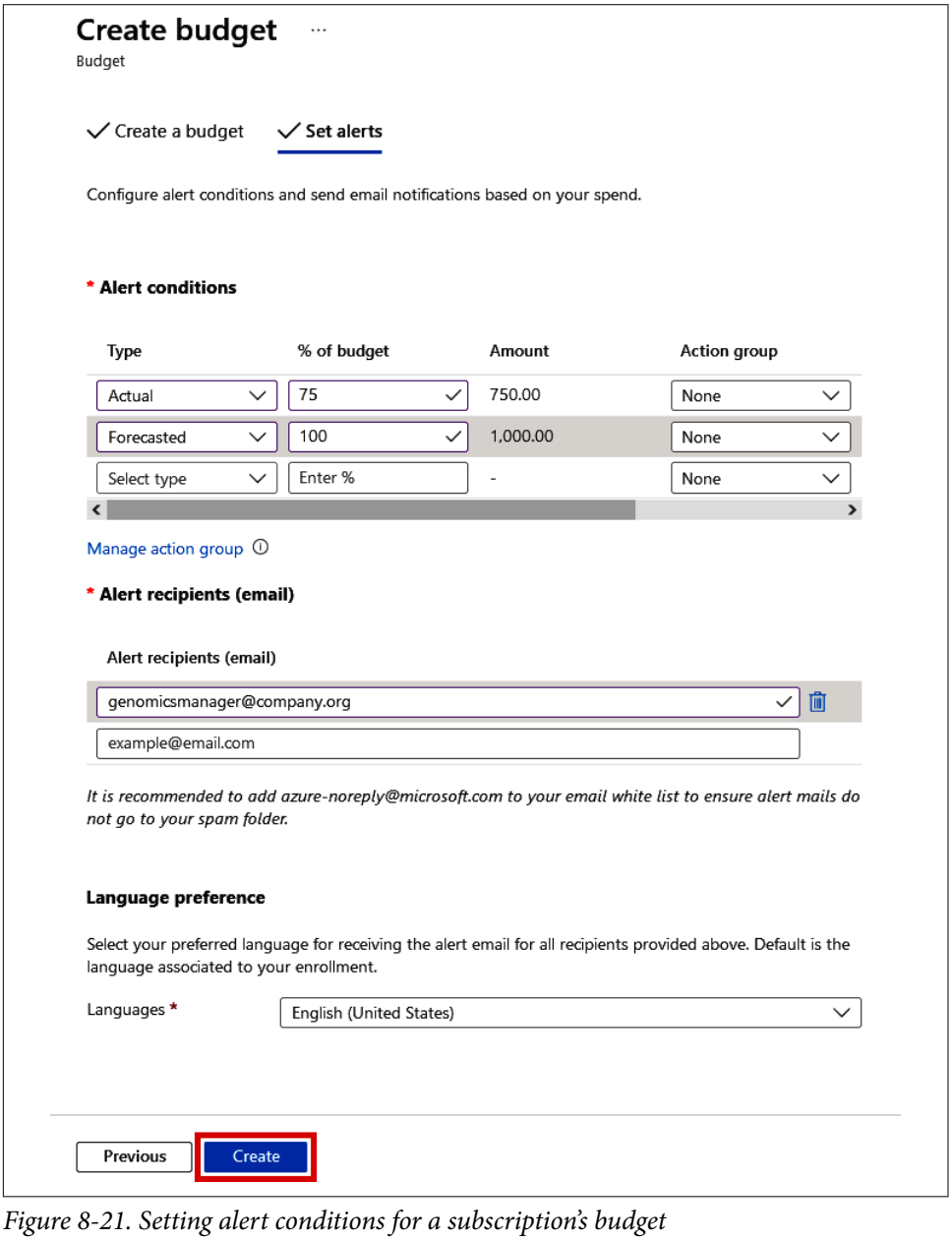


Figure 8-21. Setting alert conditions for a subscription's budget

Lastly, the "Cost analysis" tool, shown in Figure 8-22, is extremely useful when you wish to see the current costs of your the entire Azure tenant, a subscription, or some resource group(s).

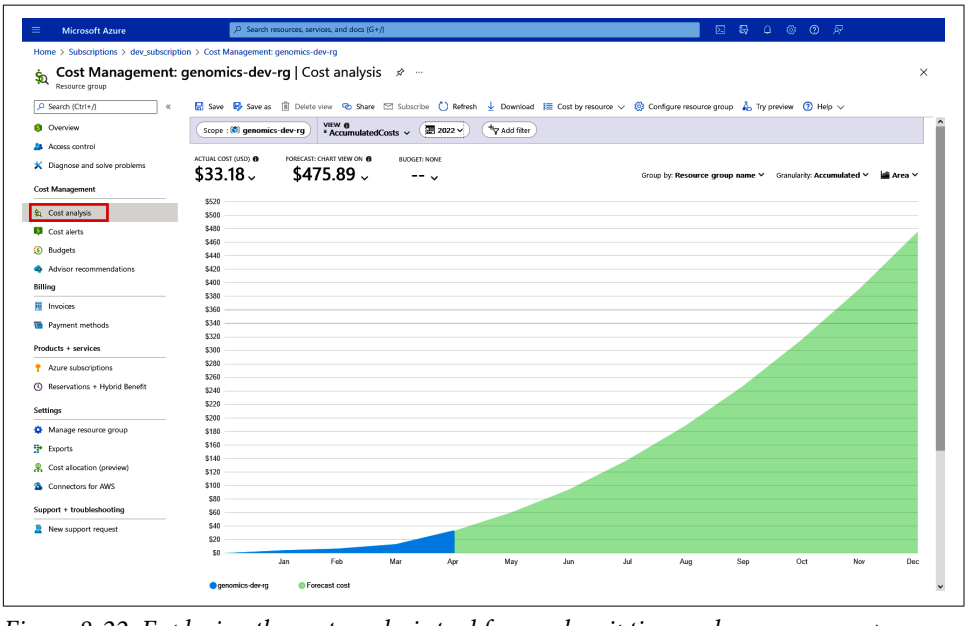


Figure 8-22. Exploring the cost analysis tool for a subscription and resource group

If you're the person in charge of a departmental budget, setting a subscription budget with alerts in Azure and keeping an eye on the cost analysis tool will help keep you out of trouble by limiting the risks of blowing your budget.

Quota Problems

As covered in the compute-heavy chapters (5, 6, and 7), services like Azure Machine Learning, Databricks, and CycleCloud will attempt to spin up one or more virtual machines on your behalf. This is to create a compute cluster on which to run your code.

If you run into an issue where you can't seem to get your compute environment provisioned, you may not have the adequate quota for the virtual machine vCPUs in your tenant. This is a resource planning thing for Microsoft as it doesn't automatically allow everyone to use thousands of cores immediately.

The number of vCPUs you have as part of your quota is for the entire tenant, meaning you don't get a separate bucket of cores from anyone else in the organization. So it's often useful to request a ton of cores when you start to ramp up your compute usage in Azure. The quota costs nothing to increase and will save you time later.

How do you know if this is the issue? Well, you'll usually get an error that states that the platform cannot provision enough compute resources or that your request would exceed the quota.

For example, in Databricks, the error reads "Subscription currently has X resources and the template contains Y new resources of this type which exceeds the quota," as shown in Figure 8-23.

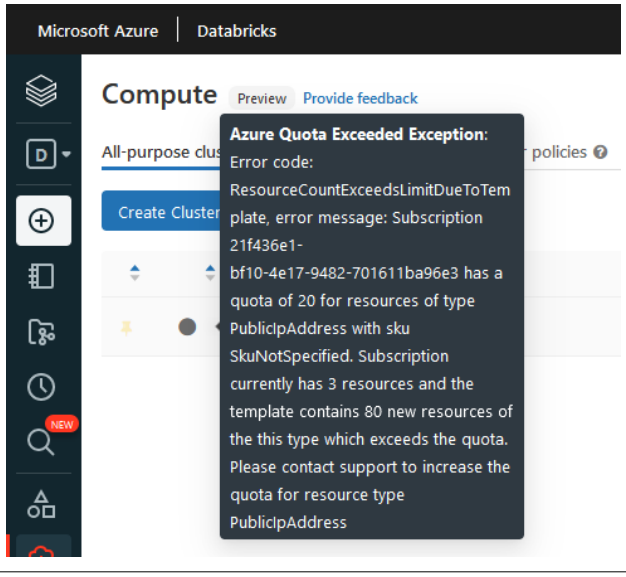


Figure 8-23. Quota issue limiting the creation of a Databricks cluster

For other services, like Azure Machine Learning, you may not get an error since the UI will limit what you can provision based on your quota. For example, if you go to provision a new compute cluster from the Azure Machine Learning Studio, any VM types for which you do not have enough cores available will be grayed out with the error "You do not have enough quota for the following VM sizes. It is possible that you have quota in a different location." This is shown in Figure 8-24.

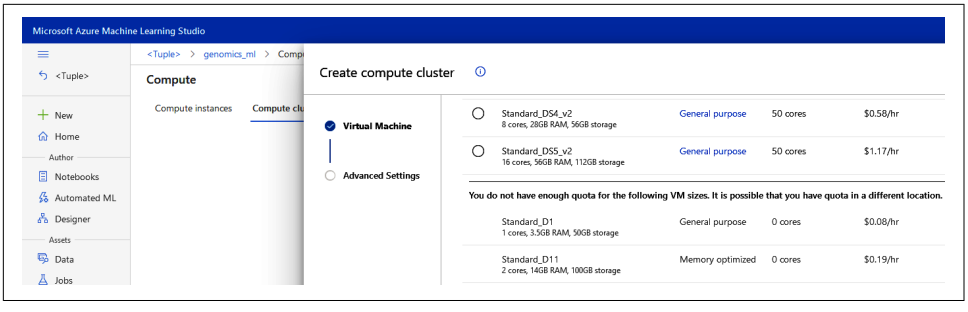


Figure 8-24. Quota issue limiting the creation of an Azure Machine Learning cluster

Don't worry, asking for an increase in your quota is way easier than asking your boss for a raise.

Please, Sir, Can I Have Some More (vCPUs)?

If you need to request an increase to your vCPU's quota, this is done by creating a support request from the Azure Portal.

Search for "quotas" in the Azure Portal's search box and locate the "Quotas" option. From the "Quotas" screen, click the "My quotas" item in the side menu (under "Settings"), as shown in Figure 8-25.

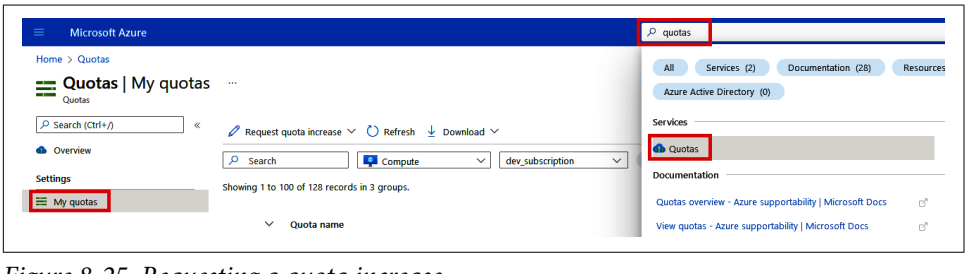


Figure 8-25. Requesting a quota increase

Scroll through the list of VM types and find the one that you need. Note that these quotas are region-specific, so pay attention to the "Region" column.

Once you've found the one you need, click the checkbox beside it. (You can also select multiple types at a time.) Then, click the "Request quota increase" button at the top. Select the "Enter a new limit" option from the drop-down, as shown in Figure 8-26.

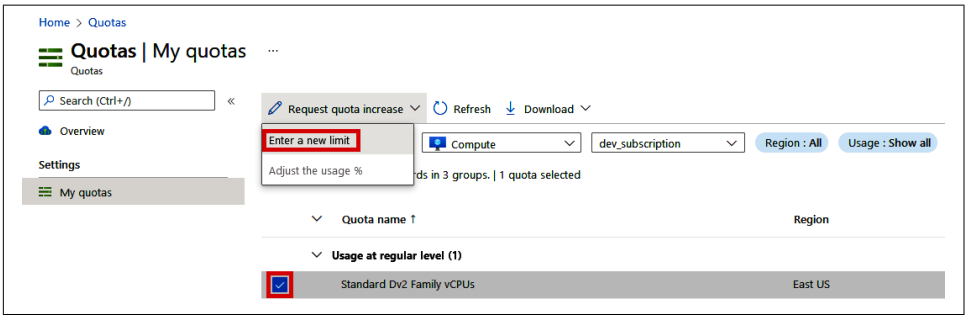


Figure 8-26. Selecting a VM type for the quota increase request

Finally, in the "Request quota increase" panel that pops up on the right of the screen, enter the new quota you'd like to have for the VM type(s) you chose. Then, click the "Submit" button at the bottom.

That's it! For smaller quota increase requests, they are often immediately approved. For larger requests, it may take a bit of time for the Azure resource planning team to grant your wishes.

As I mentioned, the vCPU quota is shared across your entire tenant. It's also shared between multiple types of Azure resources. For example, if you have a 100-core limit for Dv2 family vCPUs, then spinning up standalone virtual machines, Databricks clusters, and CycleCloud clusters will chip away at this quota while they are in use.

There are a couple of exceptions to this sharing, though. Azure Machine Learning and Azure Batch both have their own quotas that have their own separate quotas.

For Azure Machine Learning, you can request a quota increase in this same form simply by changing the request type from "Compute" in the drop-down menu to "Machine learning," as shown in Figure 8-27.

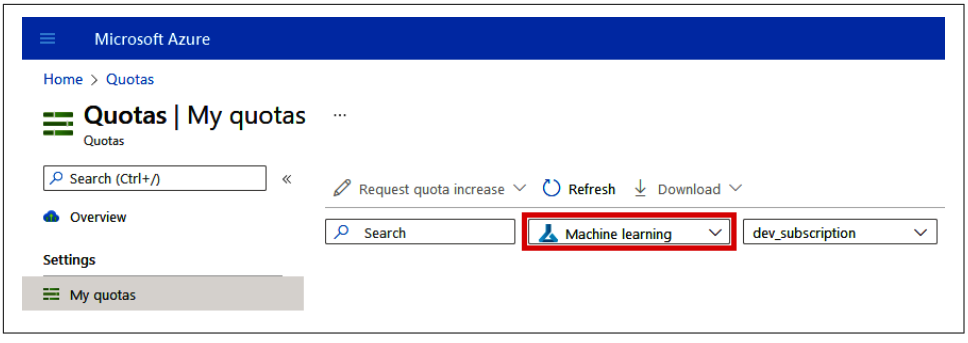


Figure 8-27. Selecting a VM type for an Azure Machine Learning quota increase request

For Azure Batch, we'll cover how to submit a quota increase request in the next section.

Getting General Support

Sometimes things just won't work as planned. If you're having issues building out your solution in Azure and you've exhausted the resources in this book, the online documentation, Stack Overflow, your colleagues, and your IT group, Microsoft Azure has a great support team to help you.

Search for “help” in the portal’s search box and locate the “Help + support” option. From the “Help + support” screen, click the “Create a support request” button at the top. See Figure 8-28.

Microsoft Azure

Home > Help + support >

New support request ...

1. Problem description 2. Recommended solution 3. Additional details 4. Review + create

Tell us your issue, and we'll help you resolve it.

Provide information about your billing, subscription, quota management, or technical issue (including requests for technical advice).

What is your issue related to? *	Azure services
Summary *	Cannot connect to Synapse
Issue type *	Technical
Subscription *	dev_subscription (21f436e1-bf10-4e17-9482-701611ba96e3)
	Can't find your subscription? Show more
Service	My services / All services
Service type *	Azure Synapse Analytics Workspace
Resource *	genomics-asa
Problem type *	Connectivity
Problem subtype *	Firewall Rules

Next

Figure 8-28. Creating a support request

In the "New support request" screen, as shown in Figure 8-29, you can fill out the form to start the support request process. If you set the "Issue type" to "Technical," the portal will list your resources for a specific service type. Then, it will list a few problem types/subtypes to help direct your request to the appropriate support team.

Microsoft Azure

Home > Help + support >

New support request ...

1. Problem description 2. Recommended solution 3. Additional details 4. Review + create

Tell us your issue, and we'll help you resolve it.

Provide information about your billing, subscription, quota management, or technical issue (including requests for technical advice).

What is your issue related to? *	Azure services
Summary *	Can't connect to Synapse
Issue type *	Technical
Subscription *	dev_subscription (21f436e1-bf10-4e17-9482-701611ba96e3)
	Can't find your subscription? Show more
Service	My services / All services
Service type *	Azure Synapse Analytics Workspace
Resource *	genomics-asa
Problem type *	Connectivity
Problem subtype *	Firewall Rules

Next

Figure 8-29. Describing the problem in a support request

For quota needs not covered from the Quotas page from the previous section, these can be requested via a support request in this same form. For the "Issue type," pick "Service and subscription limits (quotas)." Then, under Quota type, find the service for which you need an increase in Quota. If you're reading this section because you can't get the Cromwell to work on Azure Batch, then select "Batch" here. See Figure 8-30.

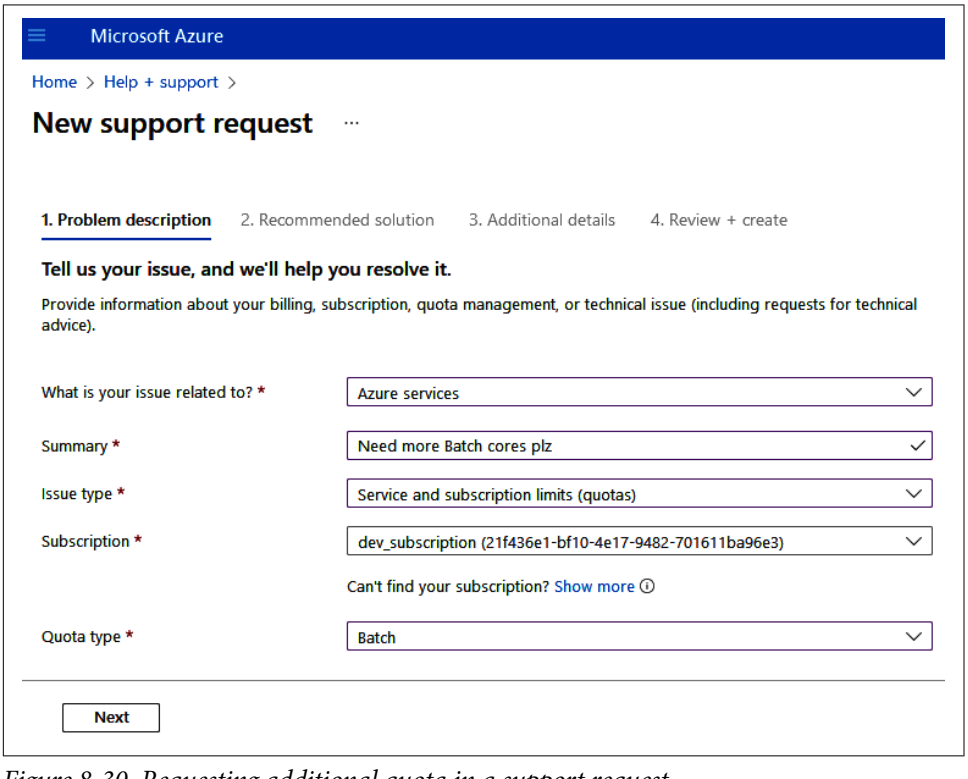


Figure 8-30. Requesting additional quota in a support request

On the second page, the support system will point you to some documentation pages related to your issue. This is likely not super helpful if you've already exhausted the documentation from doing some Google (or Bing) searches. However, it may find something useful, so don't ignore this list.

On the third page, "Additional details" (shown in Figure 8-31), you can provide additional detail about the support request. You'll provide your preferred contact methods (email or phone) and select the severity of the issue.

Microsoft Azure

Home > Help + support >

New support request ...

1. Problem description | 2. Recommended solution | **3. Additional details** | 4. Review + create

Tell us a little more information.

Providing **detailed**, **accurate** information helps us resolve your issue faster.

Problem details

Additional information is required to promptly process your request for a quota increase.

Request details * — **Provide details for the request** Enter details

Advanced diagnostic information

To enable faster resolution, we recommend allowing Microsoft support to access your Azure resources to collect advanced diagnostic information. Access is read-only and is removed when your support request is closed. Learn more

Allow collection of advanced diagnostic information? * — Yes (Recommended) / No

Support method

Support plan — Basic support

Severity — C - Minimal impact

Preferred contact method * — **Email** A Support engineer will contact you over email. / **Phone** A Support engineer will contact you over the phone.

Your availability — Business Hours

Support language * — English

Contact info Edit

Contact name — Colby Ford

Email — colby@tuple.xyz

Additional email for notification — --

Phone — --

Country/region — United States

Previous | Next

Figure 8-31. Providing additional details and contact information in a support request

On the final page, you can review your support request before submitting it. Once you submit the request, you'll either receive a phone call or an email when a support representative has started taking a look at the request. You'll then communicate directly with that person (usually via email) until your problem is resolved.

For enterprise accounts, your subscription may come with a certain level of support for free (or wrapped into the service agreement for your organization). Thus, you can submit business-critical support requests and get help rather quickly.

However, for PAYG subscriptions (like the personal one you may be using to complete this book), technical support is not necessarily included. If you need technical help on your personal tenant, the support starts at $29 per month (and goes all the way up to $1,000 per month for the best support plan).

Quota and billing support requests are included in both types of environments as "basic support."

In summary, using "infrastructure as code" we can automate the deployment of our cloud resources. ARM templates are useful as they are autogenerated by Azure, though they may be a bit cumbersome to read and customize. Thus, we can use the newer language called Bicep to do a similar task, which is much simpler to read.

Managing roles and access controls is key to protecting your data and limiting the resources that people can access and use. Both deployment and access controls are key in meeting compliance standards like GDPR or HIPAA. To make the rest of the compliance facets a bit easier, Azure Blueprint can set policies and standards for your cloud resources.

A key to cloud success is managing the costs of your Azure infrastructure. Tools like the Azure pricing calculator, subscription budgets, and the cost analysis tool can help make cost planning budgeting easier. Getting additional quota to run large compute clusters is as easy as submitting a request through the portal. Plus, getting additional help through the "Help + support" tool in the portal is really useful if you get stuck or if things aren't working as they should.

In this potpourri basket of a chapter, we've tied up a lot of loose ends. From deployment of resources to learning how to protect your data, you now have the ability to create a scalable genomics cloud architecture that will even make your IT team happy. While these topics aren't nearly as exciting as building bioinformatics pipelines, they are necessary to make your cloud enterprise ready.

Conclusion

Well, well, well…if you're reading this sentence, either you're a champion reader who made it through the whole book or you are one of those crazy people who likes to read the end of a book first. Either way, I'm glad you're here as I have some final thoughts for you.

Looking Backward

A fun fact that I learned in graduate school is that much of the high-performance computing hardware and software that we use today was born out of necessity from scientific fields (namely, physics and genomics) that needed to process tons of data. This all happened before the (tacky) *big data* term was coined.

Then when businesses began adopting this notion that data was the new oil, the appetite for "big data" was accelerated overnight. Using cloud-based tools, businesses saw the benefit of flexibility and scale in handling what was becoming a hoarder situation of data.

Baby Azure

I was first introduced to Microsoft Azure in its early days as a consultant at a Microsoft partner firm in 2014. The UI was totally different, and the service offerings were considerably more limited. However, the potential was already there to build flexible and scalable databases, virtual machines, and more.

Since then, it's been exciting to see the adoption of the cloud become a key to success for many organizations. Now, many of the bigger clouds (Azure, Amazon Web Services, and Google Cloud Platform) offer literally hundreds of services to fit virtually any task.

What Else?

In this book, we've covered a ton of Azure services for storing, processing, querying, and analyzing data. However, there are plenty more services that you may wish to check out Here's a quick list:

Azure Virtual Machines
: While I normally recommend the use of PaaS offerings like Azure Machine Learning or Databricks, sometimes you just need a virtual machine to run some software in the cloud.

Azure Web Apps
: Deploy applications (such as R Shiny, JavaScript, Streamlit, or Django applications) in a scalable and secure way on the cloud.

Azure Functions
: Create serverless applications and scalable APIs using your favorite language (like Python, Node.js, PowerShell, .NET, and more).

Azure Health Data Services
: Managed FHIR, DICOM, and MedTech APIs for securely housing and transferring protected health information.

Azure Cognitive Service (Text Analytics for Health)
: Pretrained natural language models available as an API for analyzing health-specific text data.

Azure Cray
: For those interested in super high-performance computing with dedicated Cray® XC™ or CS™ series supercomputers with Cray® ClusterStor™ storage in an Azure data center.

Using Other Web-Based Bioinformatics Platforms

In addition to building your data estate on Azure, there are plenty of non-Azure tools that may be useful for your team.

Many sequencing providers would like for you to use their cloud-based (SaaS) tool for housing and analyzing your data. For example, Illumina's BaseSpace tool makes it easy to house all of your NGS data from the sequencers in Illumina's cloud environment, but what if you wanted to correlate this data with other data you have in Azure?

Remember that centralizing your data in one place (such as in a data lake) is key to harnessing the power of your data. Plus, using tools like Data Factory to pull data

from external sources over to Azure, we can mitigate the risk of siloing your data in multiple places.

However, there are tools, such as Terra, that will allow you to use your own cloud storage and plug into some standardized bioinformatics pipelines using their web portal. Tools like Terra are an exciting new generation of options for performing bioinformatics analyses at scale without having to deploy tons of cloud infrastructure, though we're still in the early stages of such platforms.

Looking Forward

In a health-care presentation from 2020, Microsoft reported that "the future of health-care is powered by genomics." I would extend that statement a bit further. I think the future of multiple areas are connected to genomics. From agriculture to climate change to precision medicine, I believe the success of genomics and scalable bioinformatics will have a huge impact on these areas, and the cloud will help ensure this success.

In that presentation, Microsoft listed the following areas:

- Tailor drugs to patient for more effective treatment.
- Design personalized cancer treatments based on the analysis of tissue from the tumor.
- Use gene therapy to treat or prevent disease.
- Address the influence of microorganisms on health by sequencing the microbiome.
- Determine the cause of developmental issues in newborns sooner.
- Predict inherited disease and its effects.

In my eyes, the key to harnessing the power of genomics data is in the integration of non-omics data to help paint the full picture.

In my research, I've had a lot of success in using heterogeneous datasets to answer questions and to capture the complex relationship between genetics and disease. I urge you to do the same.

Other important data points (for human research) may include:

Lab results
: Metrics such as blood pressure, glucose levels, heart rate, and others could be indicators of disease or disease progression.

Image data
: Derive features from various images such as MRIs, X-rays, and pathology slides.

Demographics
: Factors such as age, sex, location, and socioeconomic factors may affect human health in ways not seen in genes.

Diagnoses
: Whether it's the disease you're working on or comorbidities, understanding their association with genetics factors is often most important.

The next time you're working on a clinical trial or some new investigational research, see what other ancillary data you can dig up. Blend this information with your -omics data in your machine learning or statistical analyses and see what you can find.

Cheaper Sequencing = More Data

In the process of writing this book, a couple biotech startups came out of stealth mode and stated that they can sequence a genome for $100. While the jury is still out as to whether this is really true (or is functionally true at scale), the cost of genomic sequencing has certainly plummeted in recent years.

Over time, this means that we'll begin to see a huge onslaught of genomics data for each of our studies. Instead of a trial having to budget large amounts of money to get some sequencing done, we'll soon be able to have this performed by default for about the same, relatively negligible cost of many typical blood tests.

However, you and I both know that raw sequence information isn't useful in the slightest without being processed. So it's a great time to be a bioinformatician! Your skills in processing vast amounts of data will be needed for years to come. #jobsecurity

I'm excited to see what you build and am also looking forward to seeing the innovative solutions the community creates to solve complex scientific problems using large-scale computation.

I hope you have enjoyed reading this work and have learned a bit along the way!

Until next time, *stay curious*…

C. F.

Index

A

AAD (Azure Active Directory)
  security, 275
    groups, 278
    roles, 276-278
ACID (atomicity, consistency, isolation, durability), 110
ACLs (access control lists), 61-63, 278-280
AlphaFold2, 219-224
Apache Spark (see Spark)
ARM (Azure Resource Management)
  templates, 266-273
atomicity, 110
automated processing, variants, 170-187
automation, 5
AutoML (automated machine learning), 213-218
  ensemble models, 216
autoscaling
  Azure Cycle Cloud, 238
  Databricks, 160
autotermination, Databricks, 160
Azure account signup, 12-13
Azure Batch, 230
  resource groups, 232
  resources, 233-234
  workloads
    automation, 231
    Cromwell, 231-238
Azure Blueprints, 284-286
Azure CLI, Cromwell, 231
Azure Cycle Cloud, 238
  autoscaling, 238
  bioinformatics clusters, 248-258
  clusters, setup, 239-248
  VM creation, 239
Azure Data Lake Storage, 58
  (see also data lakes)
  ACLs (access control lists), 58
  RBAC (role-based access control), 58
Azure Database for PostgreSQL, 7
Azure Databricks (see Databricks)
Azure DevOps
  Bicep, 274
  Data Factory, 115
Azure HDInsight, 48
Azure Key Vault, 175
Azure Machine Learning, 199
  (see also ML (machine learning))
  AlphaFold2, 219-224
  AutoML system, 213-218
  Biopython, 218-219
  child runs, 215
  HyperDrive, 210-212
  notebooks, alignment, 218-219
  quota problems, 294-295
Azure Machine Learning Studio, 199
  cluster-based model training, 209-212
  compute instance, 201
  datasets, 203-208
  datastores, 203-208
  drug sensitivity model, 200-212
Azure Portal
  Azure Synapse Analytics, 77
  free services, 22-23
  resource group setup, 14-18
  resources, creating, 18-22
  Synapse Analytics workspace, 84

Azure pricing calculator, 286-288
Azure SQL DB
database creation, 100-108
elastic pools, 103
Provisioned compute tier, 105
Serverless compute tier, 105
Azure Storage
Blob storage, 48-49
data lake storage, 48-49
data lakes, 43-49
premium performance tier, 44
Azure Storage Explorer, 54-55
ACLs (access control lists), 278
Azure Synapse Analytics, 48, 77
data sources, external, 96-100
services, registering, 81-84
workspace
creating, 77-80
external tables, 89-93
open row sets, 87-89
pools, 93-96

B

BAM (binary alignment map) format, 27
BGEN file format, 30
Bicep, 274
bioinformatics tools, single-node, 188-189
bioinformatics workflow, 24
biostatistics, 30
data lake organization, 40-41
file formats
BGEN, 30
GEN, 30
GFF (general feature format), 31
PDB (Protein Data Bank), 31-35
machine learning, 30
primary analysis, 24
FASTA files, 25
FASTQ files, 25
protein structure modeling, 30
secondary analysis, 26
BAM (binary alignment map) file format, 27
SAM (sequence alignment map) file format, 26-27
VCF (variant call format) file, 27-29
tertiary analysis, 29
Biopython, 218
biostatistics, 30
Blob storage, 48-49
bronze zones (raw data), 38, 39
budgeting examples, cloud costs, 289-293
BWA (Burrows-Wheeler Aligner), 258

C

cloud
benefits, 4
automation, 5
collaboration, 4
scalability, 4
considerations, 2-4
expense, 3
migrating to, 2
cloud horsepower, 1-5
cloud resources, deployment automation, 265-274
cloud services cost
Azure pricing calculator, 286-288
budgeting, 289-293
retail pricing, 288-289
cluster-based model training, 209-212
collaboration, 4
compliance
Azure Blueprints, 284-286
GDPR (General Data Protection, 283
HIPAA (Health Insurance Portability and Accountability Act), 281
rules, 282-283
HITECH (Health Information Technology for Economic and Clinical Health Act), 281
HITRUST (Health Information Trust Alliance), 281
consistency, 110
containers
data lakes, 38
Docker, 157
Cromwell
Azure CLI, 231
workflows, 235-238
workloads, scaling, 231-238

D

D-Series VM type, 229
Data Factory, 113
creating, 114-118
data movement, 119
Copy Data tool, 119-121

NCBI FTP server, 121-129
data transformation, data flows, 130-146
Databricks notebook, 173-185
pipeline, 176-185
pipelines, 147
canvas opening, 131
data flows, data transformation, 130
data lakehouses, Databricks, 109
data lakes, 37
access, 58-63
Azure Storage, 43-49
benefits, 37
bioinformatics workflow, 40-41
bronze zones, 38, 39
containers, 38
data movement, 119-121
Databricks, connecting, 162-165
Delta, 111
flexibility, 38
gold zones, 38, 40
naming, 44
redundancy, 44
scale, 37
silver zones, 38, 40
storage, 48, 49
ACLs (access control lists), 58, 61-63
Hadoop, 48
hierarchical namespace, 48
RBAC (role-based access control), 58, 59-60
versus data warehouses, 108-112
zones, 108
data lifecycle, 50
Azure Storage Explorer, 54-55
rules, 55
data movement, 119
Copy Data tool, 119-121
NCBI (National Center for Biotechnology Information) FTP server, 121-129
data orchestration, 113
(see also Data Factory)
data storage
ACLs (access control lists), 58, 61-63
Azure Storage Explorer, 54-55
cost versus performance, 49-58
RBAC (role-based access control), 58, 59-60
tiers, 49
data transformation
Data Factory, pipelines, 147
data flows, 130
VCF file parsing, 130-146
data warehousing, 69
architecture for genomics, 72-77
dimensions, 69
facts, 69
genomics, 70
Kimball style, 70
lab results, 71
snowflake schema, 70
SQL, 72
star schema, 70
variant, 73
versus data lakes, 108-112
Databricks, 48, 152
access tokens, 174-176
autoscaling, 160
autotermination, 160
bioinformatics tools, single-node, 188-189
clusters
high concurrency, 159
single node, 159
standard, 159
data lakehouses, 109
data lakes, connecting, 162-165
Glow, 161
chromosome coordinates, 169, 169
variant data processing, 165-170
VCF files, 166
Hail, 190-192
Koalas, 189-190
notebooks
Data Factory, 173-185
pipeline, 176-185
workspace setup, 153-165
Databricks Units (DBUs), 155
DataFrames
converting to Parquet format, 173
Spark, 168
datasets
open, 63-67
Python SDK, 208
datastores, registering files, 204
DBUs (Databricks Units), 155
Delta file format, ACIDity, 186-187
Delta lake, 111
deployment
ARM (Azure Resource Management) templates, 266-273

Bicep, 274
cloud resource automation, 265-274
development environment, 266
production environment, 266
staging environment, 266
development environment, 266
dimensions, data warehouses, 69
directory structure, 42
distributed file formats
Delta, ACIDity, 186-187
Parquet, 186
Docker containers, 157, 219-224
drug sensitivity model, 200
compute instance, 201
datasets, 203-208
datastores, 203-208

E

E-Series VM type, 229
EMR (electronic medical record) system, 39
ensemble models, 216
Enterprise Agreements, 10, 288-289
Enterprise Dev/Test subscription, 10
explainable ML, 216-218

F

F-Series VM type, 229
fact-dimension style, data warehousing, 69
facts, data warehouses, 69
FASTA files, 25
FASTQ files, 25
file formats
BGEN file format, 30
distributed
Delta, 186-187
Parquet, 186
GEN file format, 30
GFF (general feature format), 31
Parquet, 109
PDB (Protein Data Bank) files, 31-35
free services, 22-23

G

GATK (Genome Analysis Toolkit), 258
GDPR (General Data Protection Regulation), 283
GDSC (Genomics of Drug Sensitivity in Cancer) database, 205
GDSVM (Genomics Data Science Virtual Machine), 6
general support, 297-302
Genome Analysis Toolkit (GATK), 258
GINA (Genetic Information Nondiscrimination Act), 283
GitHub, Data Factory, 115
Glow, 161
chromosome coordinates, 169
variant data processing, 165-170
VCF files, 166-168
gold zones (curated data), 38, 40
GraphX, 151

H

H-Series VM type, 229
Hadoop Distributed File System (HDFS), 48
Hail, 190-192
HDFS (Hadoop Distributed File System), 48
HIPAA (Health Insurance Portability and Accountability Act), 281
PHI data, 281
rules, 282-283
HITECH (Health Information Technology for Economic and Clinical Health Act), 281
HITRUST (Health Information Trust Alliance), 281
HPC (high-performance computing), 227
access and, 228-230
Azure Batch, 230
workload automation, 231
workload scaling, Cromwell, 231-238
clusters, 149
connectivity, 228-230
hybrid clouds, 2
HyperDrive, 210-212
Docker images, 222

I

IaaS (infrastructure-as-a-service), 5
GDSVM (Genomics Data Science Virtual Machine), 6
IaC (Infrastructure as Code), 265
IAM (Identity and Access Management) service, 59
isolation, 110

J

Jupyter Notebook
- Biopython, 218
- GPU-based training cluster, 221
- reasons to learn, 219

JupyterLab, 209

K

Kimball style, data warehousing, 70
Koalas, 189-190

L

lab results, data warehousing example, 71

M

Microsoft Genomics, 258
- alignment, 258-264
- Python 2.7 CLI, 262
- R package, 262
- variants, calling, 258-264

migration, 2
Miniconda, 255
ML (machine learning), 195
- bioinformatics workflow, 30
- explainable, 216-218
- model training
  - drug sensitivity model, 200-212
  - task distribution, 196
- tasks, scaling, 195-196
- workspace, creating, 197-199

MLlib, 150
model training
- AutoML, 213-218
- cluster-based, 209-212
- drug sensitivity model, 200
  - compute instance, 201
  - datasets, 203-208
  - datastores, 203-208
- ensemble models, 216

N

N-Series VM type, 229
NCBI (National Center for Biotechnology Information) FTP server, 121-129
notebooks, alignment, 218-219

O

open datasets, 63-67

P

PaaS (platform-as-a-service), 6
- Azure Database for PostgreSQL, 7

Pandas, 190
parallelization, local, 196
Parquet file format, 109, 186
pay-as-you-go (PAYG) plan, 9
PDB (Protein Data Bank) files, 31-35
pipelines
- automation, 228
- template pipelines, 228
- tools
  - CWL (Common Workflow Language), 228
  - Nextflow, 228
  - Snakemake, 228
  - WDL (Workflow Description Language), 228

pools, Synapse, 93
- cost considerations, 96
- Spark, serverless, 95
- SQL
  - dedicated, 94
  - serverless, 94

POSIX permissions, 49
primary analysis, 24
- FASTA files, 25
- FASTQ files, 25

private data, 275
production environment, 266
proprietary data, 275
protein structure modeling, 30
Python Miniconda, 255
Python SDK, dataset creation, 208

Q

quotas
- Azure Machine Learning, 294, 295
- vCPUs, 293-294
  - quota increase requests, 295-296

R

RBACs (role-based access controls), 59-60, 278-280
regions, 10

resource groups, 10
setup, 14-18
resources, creating, 18-22
retail pricing versus Enterprise Agreements, 288-289

S

SaaS (software-as-a-service), 8
SAM (sequence alignment map) format, 26-27
sample directories, 42
SAS (shared access signatures) tokens, 58
scalability, 4
scaling ML tasks, 195-196
secondary analysis, 26
BAM (binary alignment map) file format, 27
SAM (sequence alignment map) file format, 26-27
VCF (variant call format) file, 27-29
security, 3
AAD (Azure Active Directory), 275
groups, 278
roles, 276-278
ACLs (access control lists), 278-280
IT team, 3
private data, 275
proprietary data, 275
RBACs (Role-Based Access Controls), 278-280
services, Azure Synapse Analytics registration, 81-84
SHAP (SHapley Additive exPlanations), 217
SHAP values, 217
silver zones (staging/intermediate), 38, 40
single-node bioinformatics tools, 188-189
Snakemake, 255
snakefiles, 256
snowflake schema, data warehousing, 70
Spark, 149
clusters, 151
DataFrames, 168
GraphX, 151
pools, serverless, 95
Spark Core, 150
Spark SQL, 150
Spark Streaming, 151
SQL (Structured Query Language), 150
(see also Spark SQL)
data warehousing, 72
pools
dedicated, 94
serverless, 94
SSH keys, 252-254
staging environment, 266
Storage Blob Data Contributor, 250
Studio UI (Azure Machine Learning), 199
study directories, 42
subject directories, 42
subscriptions, 9
Enterprise Agreement, 10
Enterprise Dev/Test, 10
pay-as-you-go (PAYG), 9

T

tables, Synapse, 89-93
template pipelines, 228
tenants, 9
tertiary analysis, 29

V

variant data warehouse, 73
variants
automated processing, 170-187
Microsoft Genomics, 258-264
VCF (variant call format), 27-29
VCF files, 73
automated processing, 170-187
Glow, 166-168
parsing, 130-146
variant data, 72
vCPU (virtual CPU)
quota problems, 293-294
increase requests, 295-296
VM types, 229

About the Author

Colby T. Ford is a recovering academic who has spent much of his professional career consulting for companies across industries, enabling them to harness the power of the cloud to do more with their data. Currently, he owns and consults for <Tuple>, a Microsoft and Databricks partner consulting firm that focuses on building Azure solutions for genomics. Colby is also the cofounder of Amissa, an NSF- and NIH-funded digital health company looking to use wearable devices to revolutionize caregiving and biomarker research in Alzheimer's disease and beyond.

Colby holds a PhD in bioinformatics and computational biology and a master's degree in data science and business analytics. He teaches the Cloud Computing graduate course in the University of North Carolina at Charlotte's School of Data Science. Colby is passionate about building scalable and elegant solutions in the cloud, especially to solve complex biological problems.

Colophon

The animal on the cover of *Genomics in the Azure Cloud* is a Tamworth pig (*Sus scrofa domesticus*). They are a highly-sociable and intelligent heritage breed of pig that enjoys the attention of humans.

The Tamworth pig is one of the oldest breeds of pig from Britain. It is said that the breed was originally developed by cross-breeding an Irish breed with pigs from Tamworth, Staffordshire in the United Kingdom and the Royal Agricultural Society of England officially recognized it in 1885. They were first imported to the United States in 1882, followed by Canada in 1888. They can also be found in Australia and New Zealand.

Tamworth pigs were bred for their ability to forage for mast (predominantly acorns with the addition of other nuts and fungi) in oak and beech forests during fall and winter months. They are long, lean, athletic pigs with strong legs and sound feet, thus they are excellent at walking long distances. They also have large snouts that are ideal for foraging. The ginger red of their coat helps them avoid sunburn and thrive in colder climates. At peak maturity, they can weigh between 500 and 600 pounds.

Tamworth pigs are considered threatened in the US and vulnerable in the UK. There are less than 300 breeding females currently registered. Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on an antique line engraving from *Brehms Thierleben*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.

O'REILLY®

Learn from experts.
Become one yourself.

Books | Live online courses
Instant Answers | Virtual events
Videos | Interactive learning

Get started at oreilly.com.

©2022 O'Reilly Media, Inc. O'Reilly is a registered trademark of O'Reilly Media, Inc. | 175