
Bioinformatics in the Cloud Workshop Documentation

Release 1.0

Adam Labadorf, Aaron Chevalier, Dileep Kishore, Sebastian Carra

Aug 01, 2018

Contents

1 Prerequisites	3
2 Time & Location	5
3 Registration	7
4 Online Materials	9
4.1 Cloud Concepts Workshop	9
4.2 Cloud App Deployment Workshop	34
4.3 FireCloud Workshop	46
5 Indices and tables	53

Cloud technologies are emerging as a critical tool in Bioinformatics analysis as datasets grow exponentially in number and size. However, the set of cloud technologies and concepts necessary to deploy Bioinformatics analysis is rapidly evolving and complex. This series of workshops will introduce the cloud analysis paradigm using the Amazon Web Services (AWS) platform, cover some current strategies for deploying Bioinformatics data and applications for analysis, and give students some hands-on experience with these topics. The workshop will also highlight [FireCloud](#), a scalable Bioinformatics cloud solution provided by the Broad Institute.

Note: Bioinformatics knowledge is *not* required, as the materials are intended to be sufficiently generic to allow users familiar with the prerequisite concepts to deploy their own applications in the cloud. The workshop simply uses a bioinformatics analysis as the use case for the hands-on materials.

Quick links:

- [Prerequisites](#)
- [Time & Location](#)
- [Registration](#)
- [Online Materials](#)

CHAPTER 1

Prerequisites

This workshop is fairly technical. You will need a good understanding of the following to maximally benefit from the materials:

- Ability to use linux/command line
- Programming in python, Java, C/C++, or other comparable languages
- General familiarity with how the linux operating system works

Attendees are expected to bring their own (preferably Mac/Linux) laptops.

CHAPTER 2

Time & Location

- Session 1 - Cloud Concepts: **Monday July 30th 2PM-5PM**
- Session 2 - Packaging and Deploying Applications: **Wednesday August 1st 2PM-5PM**
- Session 3 - **FireCloud** Case Study: **Thursday August 2nd 2PM-5PM**

Location: **Life Sciences and Engineering Building (LSEB) 103**

CHAPTER 3

Registration

Registration is now closed.

Nota bene

This content is under construction!

4.1 Cloud Concepts Workshop

This is day 1 of the “Bioinformatics in the Cloud” workshop. In this session, you will learn basic cloud concepts and terminologies and work on setting up your own cloud instance and running an application on the cloud.

Workshop Outline:

- Introduction to the cloud (~10min) @Dileep
- Cloud concepts (~15min) @Sebastian
- Deployment Walkthrough: Web Console (~35min) @Sebastian
- Break (~5min)
- Deployment Walkthrough: CLI (~30min) @Dileep
- Working with deployed resources (~10min) @Dileep
- Hands-on section (~40min)
- Machine Learning on the Cloud (~30min) @Gerard

4.1.1 Prerequisites

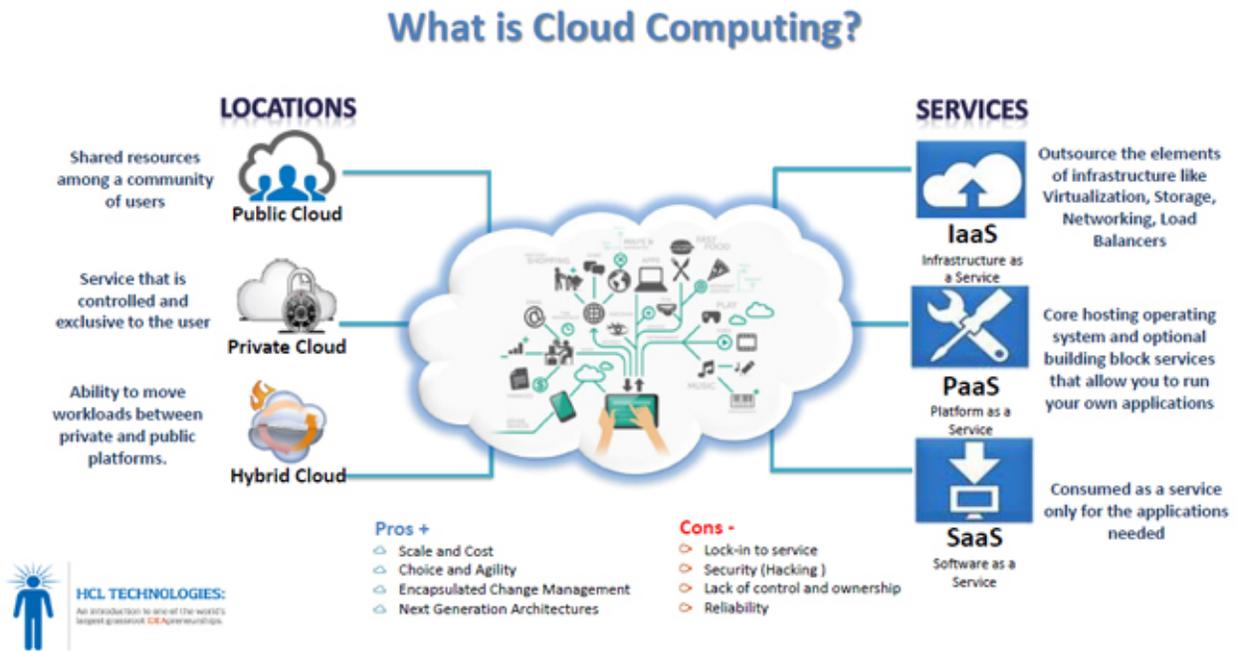
The participants are required to have access to the following resources before attending the workshop

- **AWS account** Access to the AWS account through BU
- **Web browser** A modern web browser is needed to log into the AWS management console

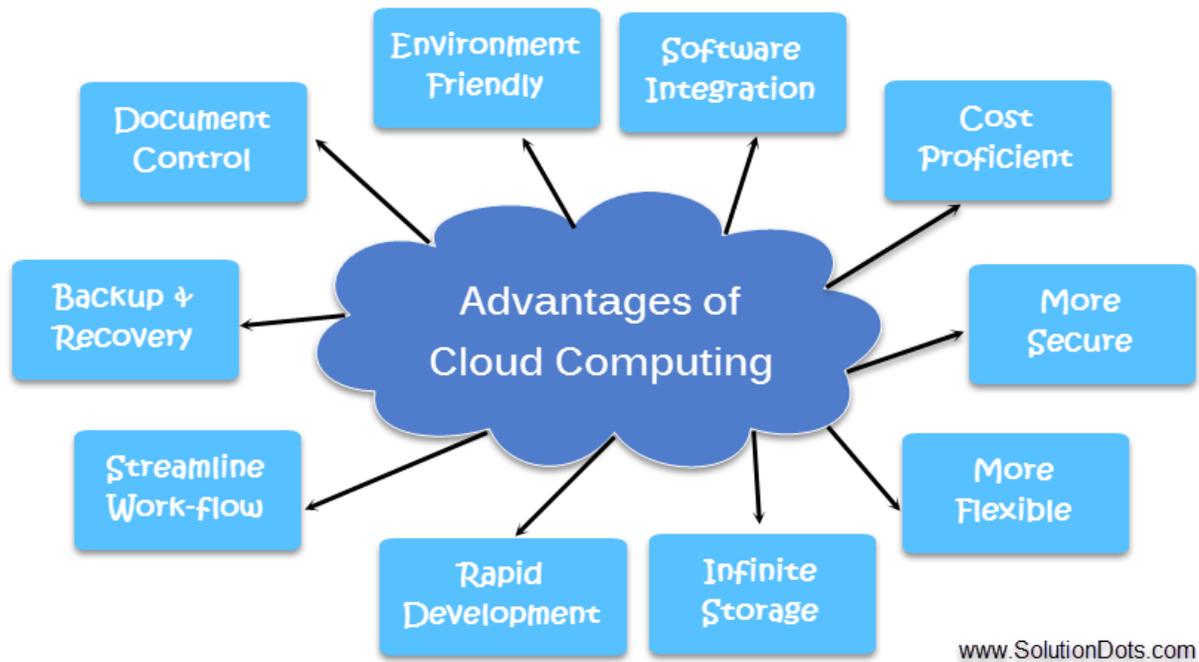
- **A terminal emulator and SSH client** A terminal emulator and ssh client are needed to log in remotely to our AWS instance
- **AWS CLI** A working installation of the AWS CLI

4.1.2 What is Cloud Computing?

Cloud computing allows access to arbitrary amounts of compute resources instantaneously. The computing resources exist on servers managed by the cloud providers, thereby, helping you avoid the hassle of hardware maintenance.



Key advantages



1. High availability - your files are always available across multiple systems
2. Fault tolerant - automatic backups enable recovery from failure
3. Scalability and Elasticity - easily scale compute resources to fit new requirements within minutes

There are various cloud providers, the most popular ones include Amazon (Amazon Web Services), Google (Google Compute Engine) and Microsoft (Azure).

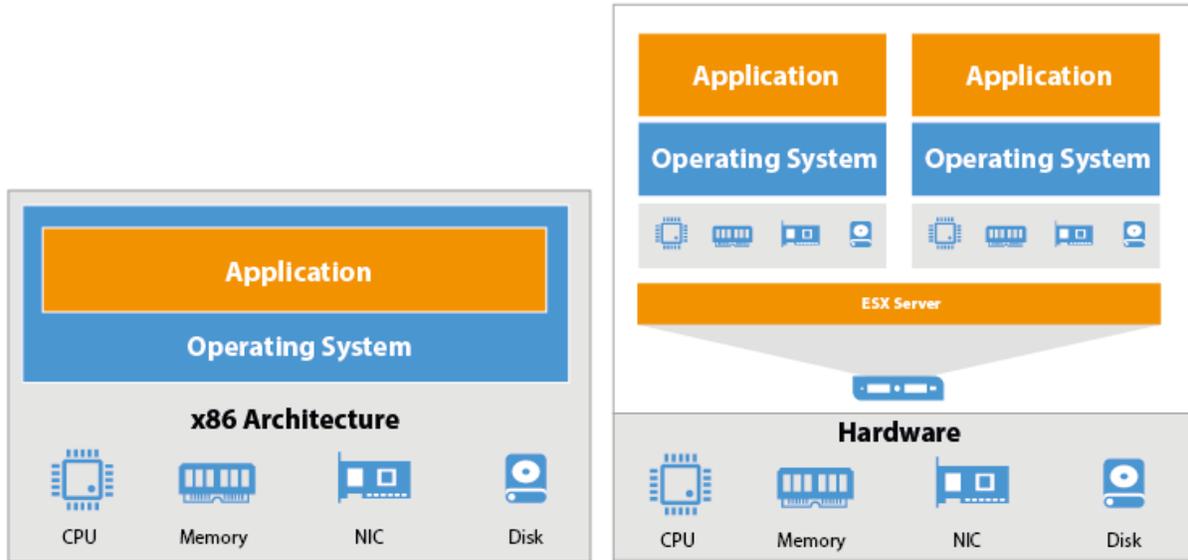
Common use-cases

1. Web hosting
2. Storage
3. Software as a Service
4. Big Data Analytics
5. Test and Development

4.1.3 Cloud concepts

Virtual Machines

Virtual Machines emulate the architecture and functionality of physical computers in the cloud. In AWS, VMs are called Elastic Compute Cloud (EC2), which can be created using different operating systems (i.e. Linux, Windows) and vCPU sizes. Using EC2 eliminates the need to invest in hardware up-front. EC2 can be used to launch as many or as few virtual servers needed, configure security and networking, and manage storage. Amazon EC2 enables scaling up or down to handle changes in requirements or spikes in popularity, reducing the need to forecast traffic.

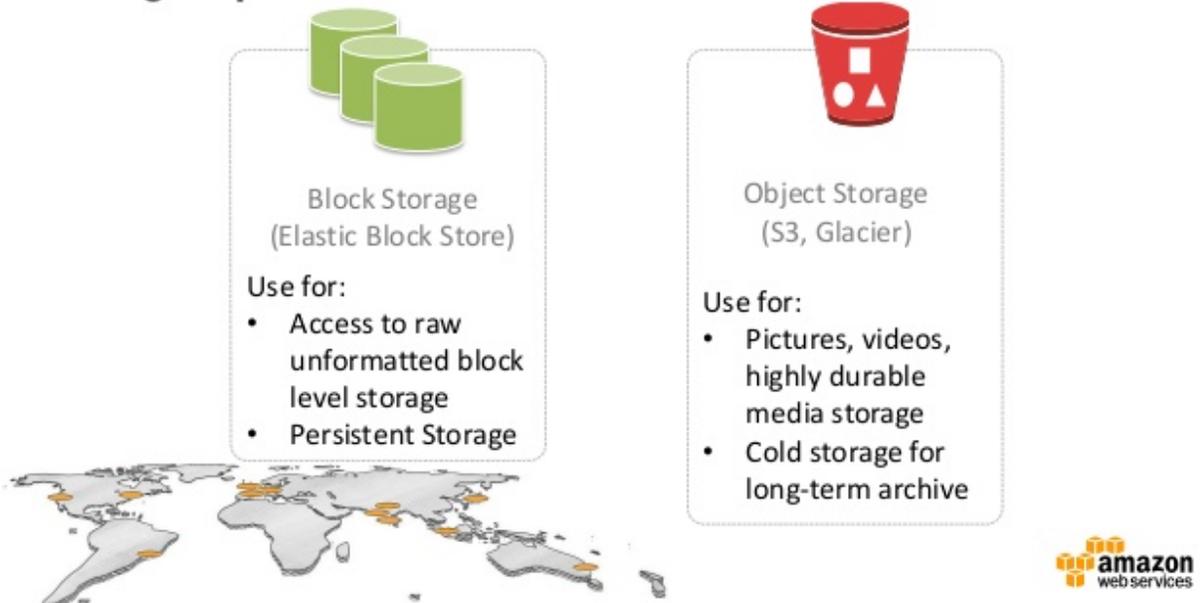


Storage Units

Storage services are also provided for the VMs, in AWS they come in two types depending on your needs:

1. Elastic Block Storage (EBS): block level storage volumes that can be directly mount to EC2
2. Simple Storage Service (S3): bucket of storage accessible through API or command line

Storage Options on AWS



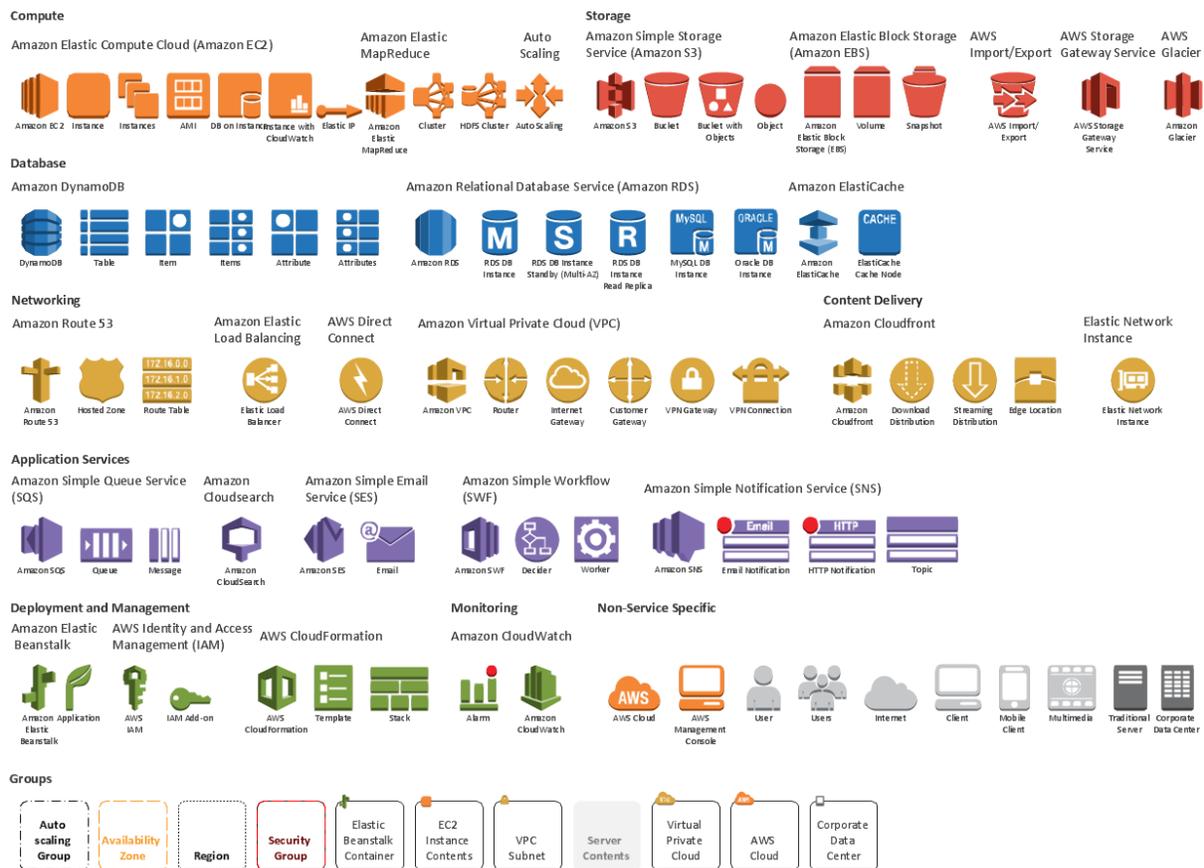
Databases

Relational Database Service (RDS) allows to set up, operate and scale relational databases (i.e. MySQL)

Serverless

Removes the need to worry about managing and operating web servers for applications. It also provides scaling and cost-efficient options.

4.1.4 The AWS infrastructure



This workshop involves working with the Amazon Web Services (AWS) cloud infrastructure, but the concepts in this workshop will apply to other cloud computing services as well. The only difference involves the exact terms used to describe services and actions.

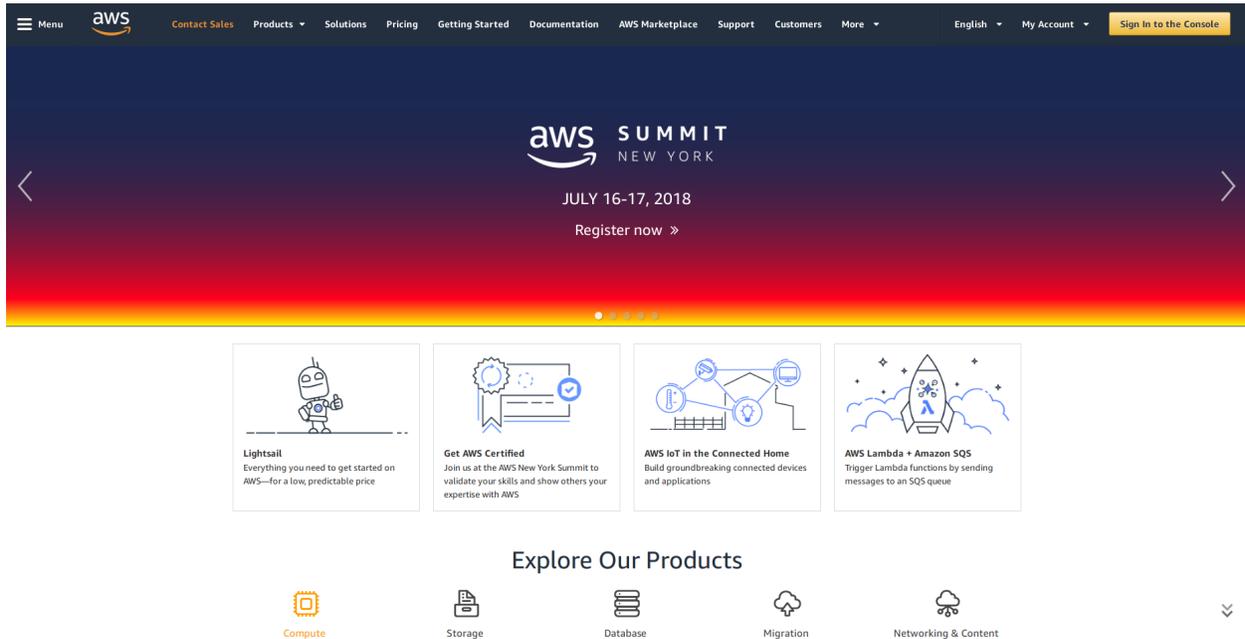
0. The AWS and the web console

1. Creating an account

In order to use AWS you will need to create an account. And in order to create instances and the other services used in this workshop you, will need to associate a credit card with the account. For the purposes of this workshop we will provide you with pre-existing AWS accounts, but you will need to create your own accounts for any future use.

2. Logging into the AWS console

To log into AWS, go to aws.amazon.com and hit the [Sign in to the Console](#) button as shown below.

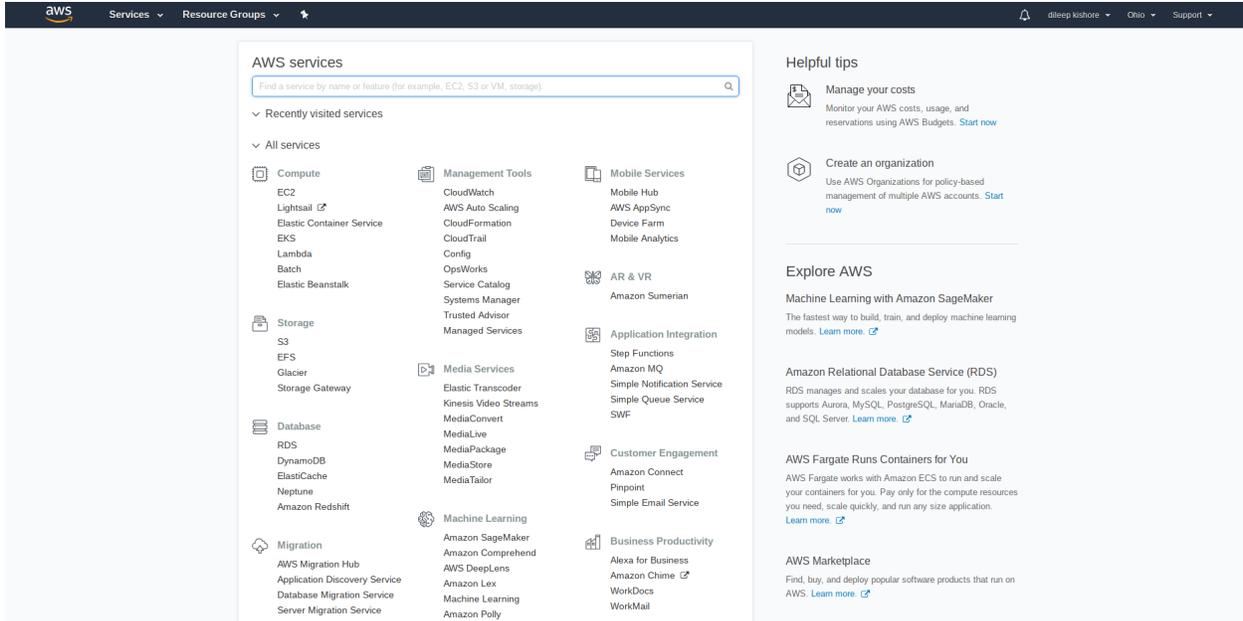


3. AWS regions

An AWS Region is a physical warehouse of servers (data centers) and other computer hardware that Amazon maintains. At any point in time you can only operate in one region. After logging in, the current region is shown in the upper right corner of the console.

Regions are important for several reasons:

1. When you launch a service like an EC2 instance, it will be confined to the region you launched it in. If you switch regions later, you will not see this instance.
2. The cost of usage for many AWS resources varies by region.
3. Since different regions are located in different parts of the world, your choice of region might add significant networking overhead to the performance of your application.



At the time of writing the following AWS regions exist:

Region Name	Region	Endpoint	protocol
US East (Ohio)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS
US East (N. Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS
US West (N. California)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
Asia Pacific (Osaka-Local)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
Canada (Central)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
China (Beijing)	cn-north-1	rds.cn-north-1.amazonaws.com.cn	HTTPS
China (Ningxia)	cn-northwest-1	rds.cn-northwest-1.amazonaws.com.cn	HTTPS
EU (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
EU (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS
EU (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
EU (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
South America (So Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS

VPC: Virtual private cloud. Your private section of AWS, where you can place AWS resources, and allow/restrict access to them.

1. EC2 instances

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud. This service allows you to configure and rent computers to meet your compute needs on an as needed basis. Using EC2 eliminates the need to invest in hardware up-front. EC2 can be used to launch as many or as few virtual servers needed, configure security and networking, and manage storage. Amazon EC2 enables scaling up or down to handle changes in requirements or spikes in popularity, reducing the need to forecast traffic.

Instances come in various shapes and sizes. Some instances might be geared towards running CPU intensive tasks while others might be optimized for memory or storage. Some of the different options available are shown in the figure below and more information can be found [here](#).

[General Purpose](#) [Compute Optimized](#) [Memory Optimized](#)
[Accelerated Computing](#) [Storage Optimized](#)

T2

T2 instances are [Burstable Performance Instances](#) that provide a baseline level of CPU performance with the ability to burst above the baseline.

T2 Unlimited instances can sustain high CPU performance for as long as a workload needs it. For most general-purpose workloads, T2 Unlimited instances will provide ample performance without any additional charges. If the instance needs to run at higher CPU utilization for a prolonged period, it can also do so at a flat additional charge of 5 cents per vCPU-hour.

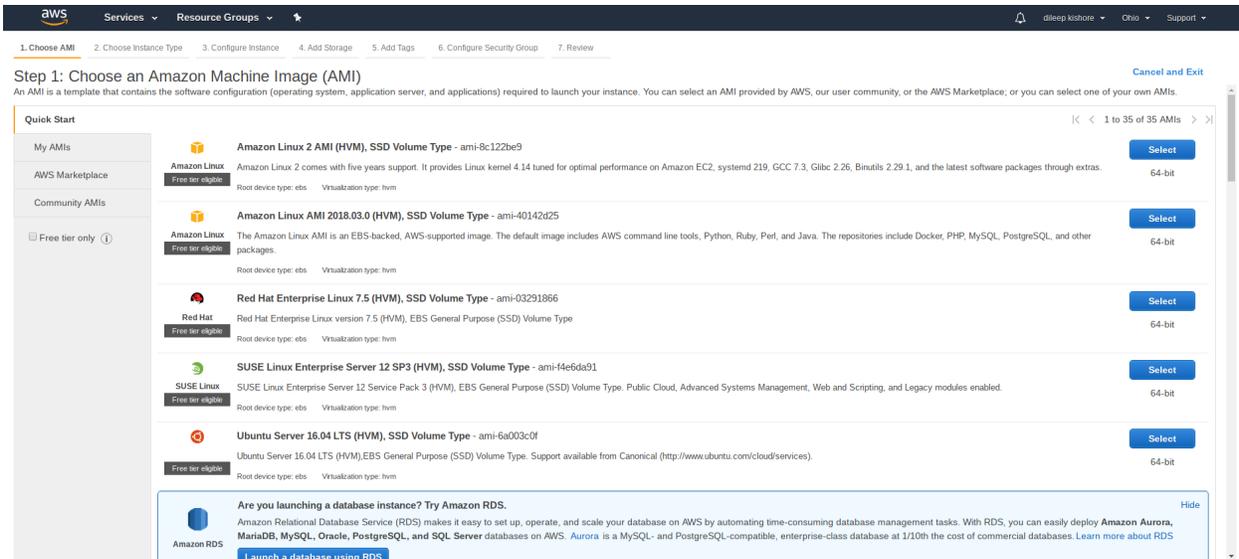
The baseline performance and ability to burst are governed by CPU Credits. T2 instances receive CPU Credits continuously at a set rate depending on the instance size, accumulating CPU Credits when they are idle, and consuming CPU credits when they are active. T2 instances are a good choice for a variety of general-purpose workloads including micro-services, low-latency interactive applications, small and medium databases, virtual desktops, development, build and stage environments, code repositories, and product prototypes. For more information see [Burstable Performance Instances](#).

Model	vCPU	CPU Credits / hour	Mem (GiB)	Storage
t2.nano	1	3	0.5	EBS-Only
t2.micro	1	6	1	EBS-Only
t2.small	1	12	2	EBS-Only
t2.medium	2	24	4	EBS-Only
t2.large	2	36	8	EBS-Only
t2.xlarge	4	54	16	EBS-Only
t2.2xlarge	8	81	32	EBS-Only

The following sections outline the various steps involved in setting up an EC2 instance:

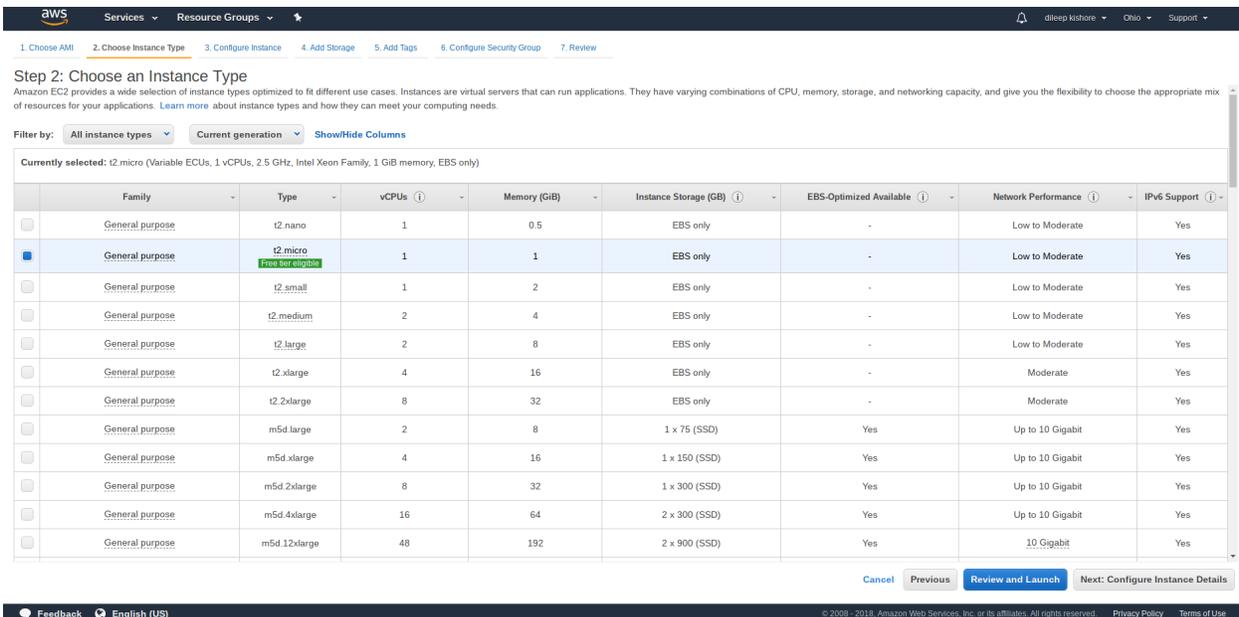
1. AMI selection

An [Amazon Machine Instance \(AMI\)](#) is a preconfigured template for launching an instance. It packages the various applications you need for your server (including the operating system and additional software). There are four main options when selecting an AMI: [Quick Start](#), [My AMIs](#), [AWS Marketplace](#) and [Community AMIs](#). These options can be seen in the image below on the left sidebar. Select the desired AMI and then proceed to the next step.



2. Instance type selection

Once an AMI is selected, the next step is to choose an instance type. If choosing an AMI is equivalent to choosing the software you want on your computer then choosing an instance type is equivalent to choosing the hardware. Broadly speaking the different instance types vary in the number of CPUs, or size of RAM, or storage. The price per hour for each of the options is not listed here. To get the price of a particular instance, look up the name of the instance on the [EC2 pricing list](#). Once you are ready, proceed to the next step by pressing the **Next: Configure Instance Details** button.



3. Instance general configuration

Once you have selected your instance type, the next step is to configure your instance. This step involves many advanced concepts that will not be covered in detail in this tutorial. Using the **Number of instances** option you can launch multiple instances with the same AMI and hardware configuration at the same time. Additionally you could also **Request Spot instances** (spot instances offer spare compute capacity at steep discounts but they

are reclaimed whenever EC2 needs the capacity back) Shutdown behavior determines the behavior of the instance when it is shutdown from within the AMI

For this tutorial we will use the proceed with the default values for all the options.

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances Launch into Auto Scaling Group

Purchasing option Request Spot instances

Network Create new VPC

Subnet Create new subnet

Auto-assign Public IP

Placement group Add instance to placement group.

IAM role Create new IAM role

Shutdown behavior

Enable termination protection Protect against accidental termination

Monitoring Enable CloudWatch detailed monitoring
Additional charges apply.

Tenancy
Additional charges will apply for dedicated tenancy.

T2 Unlimited Enable
Additional charges may apply

Advanced Details

Cancel Previous **Review and Launch** Next: Add Storage

4. Instance storage configuration

The next step is to configure the storage that will be available to the instance. The storage that you start with depends on the type of instance you have selected. In the image below we have an EBS root volume with 8GiB size. This is the Root volume where the operating system will exist. By default, this volume is set to be deleted when the instance is terminated, but, this behavior can be changed. The Add New Volume button can be used to add additional storage to our instance.

- ephemeral or Instance store storage
- EBS storage

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snap-0ea806abc1e13afcd	8	General Purpose SSD (GP2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

Add New Volume

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. [Learn more](#) about free usage tier eligibility and usage restrictions.

Cancel Previous **Review and Launch** Next: Add Tags

5. Instance tagging

When dealing with multiple instances, tagging creates a simpler way to track usage and billing information from groups of related instances

Step 5: Add Tags
 A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver.
 A copy of a tag can be applied to volumes, instances or both.
 Tags will be applied to all instances and volumes. [Learn more](#) about tagging your Amazon EC2 resources.

Key (127 characters maximum)	Value (255 characters maximum)	Instances (1)	Volumes (1)
name	test_instance	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

[Add another tag](#) (Up to 50 tags maximum)

Cancel Previous **Review and Launch** Next: Configure Security Group

6. Instance security

Secure login information for instances using key pairs (AWS stores the public key, and the user stores the private key in a secure place) A firewall that enables you to specify the protocols, ports, and source IP ranges that can reach your instances using security groups

Step 6: Configure Security Group
 A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: Create a new security group Select an existing security group

Security group name:
 Description:

Type (1)	Protocol (1)	Port Range (1)	Source (1)	Description (1)
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop

[Add Rule](#)

Warning
 Rules with source of 0.0.0.0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel Previous **Review and Launch**

7. Instance review

Static IPv4 addresses for dynamic cloud computing, known as Elastic IP addresses

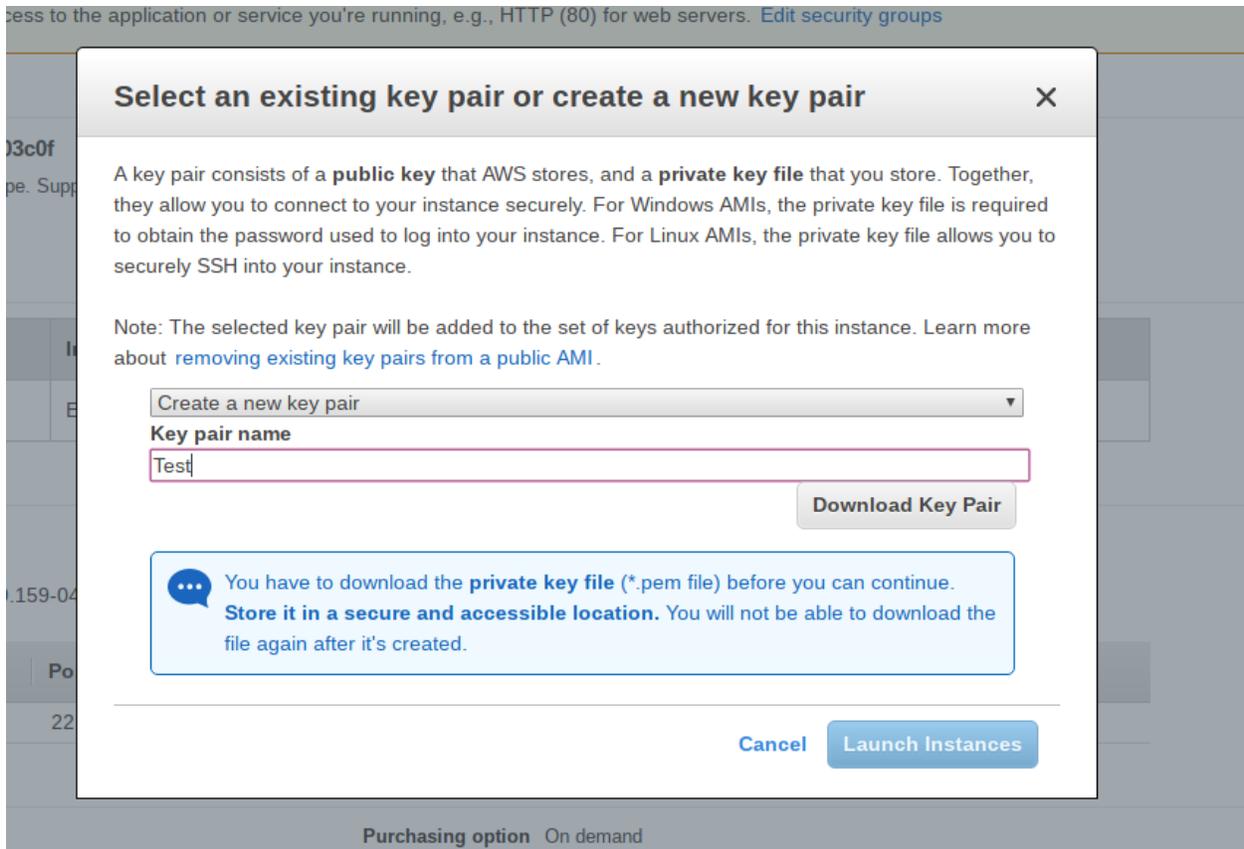
Bioinformatics in the Cloud Workshop Documentation, Release 1.0

The screenshot shows the 'Step 7: Review Instance Launch' page in the AWS console. At the top, there are navigation steps: 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage, 5. Add Tags, 6. Configure Security Group, and 7. Review (highlighted). A warning box states: 'Improve your instances' security. Your security group, launch-wizard-1, is open to the world. Your instances may be accessible from any IP address. We recommend that you update your security group rules to allow access from known IP addresses only. You can also open additional ports in your security group to facilitate access to the application or service you're running, e.g., HTTP (80) for web servers. Edit security groups'. Below this, the 'AMI Details' section shows 'Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-6a003c0f'. The 'Instance Type' section shows a table with columns: Instance Type, ECUs, vCPUs, Memory (GiB), Instance Storage (GB), EBS-Optimized Available, and Network Performance. The row for 't2.micro' shows: Variable ECUs, 1 vCPU, 1 GiB Memory, EBS only storage, not EBS-Optimized, and Low to Moderate network performance. The 'Security Groups' section shows a table with columns: Type, Protocol, Port Range, Source, and Description. The row for 'SSH' shows: SSH Type, TCP Protocol, Port Range 22, Source 0.0.0.0/0, and Description. At the bottom right, there are 'Cancel', 'Previous', and 'Launch' buttons.

Created instance

The screenshot shows the 'Created instance' page in the AWS console. The top navigation bar includes 'Launch Instance', 'Connect', and 'Actions'. A table lists instances with columns: Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, Alarm Status, Public DNS (IPv4), IPv4 Public IP, IPv6 IPs, Key Name, Monitoring, and Launch Time. One instance is listed: Name 'Test', Instance ID 'i-0b214bdd3473df580', Instance Type 't2.micro', Availability Zone 'us-east-2c', Instance State 'running', Status Checks 'Initializing', Alarm Status 'None', Public DNS 'ec2-52-15-214-181.us-east-2.compute.amazonaws.com', IPv4 Public IP '52.15.214.181', and Monitoring 'disabled'. Below the table, the 'Instance: i-0b214bdd3473df580' details are shown, including 'Public DNS: ec2-52-15-214-181.us-east-2.compute.amazonaws.com'. A 'Description' tab is active, showing details: Instance ID 'i-0b214bdd3473df580', Instance state 'running', Instance type 't2.micro', Elastic IPs (none), Availability zone 'us-east-2c', Security groups 'launch-wizard-1', and Scheduled events (none). Other tabs include 'Status Checks', 'Monitoring', and 'Tags'. At the bottom right, there are 'Feedback' and 'English (US)' buttons.

Create a key and save it



Key file example. The file can be saved as Test .pem

```

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAhEpF18lIUouMH8qia/BSB70vrQVq/mTtkiRbsACB78rzy3XGRMfvwUseIsGY
H6SDOAFrRlmTrAArH5A0t2TZ8PKrq7b9FtEAvMCeE7rWEiqBblAWiER0k1pbnIqyKJJCo1YRSUs0
oNMdvjB4CUy1YraSsSNFYJG5gRwcNhBENLDVnDs79geQcPLu/JeEiJ9V+w+CCYAG40f7li/TuULr
rSy6Oq6jgn2Gy7rrHU7XHU5hcEvxuSeoLb8h/bh1N+cN/H7x3ipeJIDdA2ScCkRXum1V6/kTFQFq
vDG01qoTlmTNKgDGpb+rdzJgOg/3QX4RSrX/c0W6aFkV9Ib/jQxT+wIDAQABAoIBADAvWxc6wpQG
bjian0T3mPlmqHnuEkWs9f8yLQ9TcACmVnwr/tbIuISAVu6z8zP7WSxKIAfU0twAh7SMcxclrdh8
m5kFIvRv1kQqKKnPENY3E0PZ+gsSXB/b9qhzQGdUtt8F13BJ61Z07016HA7PEyJ8e7v3q+p7ycTE
N2Zd0GocRIX8zxdRo9GS8ouS0QcFgNF8Kblz1J6Vs0gI7o7mIRZIm9vWkuR9Lp9uEPD2f1UIvN3z
yRmY/FE/R1yc76Uq+g8eywifRAh+GFyyO8PmFoYRni4Ki6+tEIFaq5JauT0JJF66EZeZP8ZKoWm9
1K30Ucti2D518t+CpbBM5JxhmjECgYEAxz1ET42F1sBGYqNn5hmfjRrP+YF3EYz2awRSibOerpJ
Bh1QZeB7/QD3wcB00XFIMu/3haP9xs4eesjSSug+1F59nyzDplNsybz1sYpUQwP9LjX01oUCIb8r
3O2VdLJ5ZJ9dfNgpStC/wi7kkr8xjk5XiHgP6DLk6+H1Lr2d+kMCgYEAqfpUseZ/smlvYt80LlWI
r8ozsUmzuISRspGVUppyDD47Iy j/1mkiWnsFDD107oBcFIUFIEd1rkJNB3gXKSr76kcY0X4lav7a
0dvse2T9PC/pLSFkax9UjVnydCN8ElyNoXI2wT5HuLdjCmHBD/4E9Z00201JICSbrxayk117+kC
gYEAxRiWuxwFiqWq90kxny856LIRJAIvB+2q17Mu84n8/OvL0YCuSBoKjF6nGcSJy6eevUUmV84i
/sho3o5Lek7F2NCg9RYTdjaRKAEGDNwK/0Cy9UPq8fw1X7/+ZE+jyg3EiQYeNaKhNqHLEQ3SkFkT
a1gMv7QGCG5QiAi/w17QyoECgYARcn+VDyrWXsNLK8wIYYE5QhESRpVrADIQUR84DmBcflrEniW8
lWgQT4ZSHeexv300If9Hs+4RZ/7OIHaIJEbdaNTUVBV1KRm+5sscU15m+if+Gopc0Id2RuBLKYVH
wTZMdxPFvCXSGf2q+mxAdGx7Zmj88pw83HGrP3jWQoLzWQKBgQCX5jxy3QXlPpwDppqWKKBQ8cGn
YDDQHCeD5LhrVCUqo5DCobswzmGKU/xEqYsqlik/Mz1Zkvg4FbJwJdGqGkSyAu071Nli006w27dm+
UHuvF5mCDdAhWIrFUBSiebXOpEQnkZ9IPXUUCSC6IQvPFbdGN8G3WjoER6Lw121Q4rJxGA==
-----END RSA PRIVATE KEY-----

```

2. S3 buckets

Simple storage service (S3)

This service allows the storage of large volumes of data, which can be accessed by an API or a command line interface such as `aws-cli`

The screenshot shows the AWS Management Console for Amazon S3. At the top, there's a navigation bar with the AWS logo, 'Services', 'Resource Groups', and user information. Below that, a banner for 'Stream Video to AWS for Analytics' is visible. The main content area is titled 'Amazon S3' and includes a search bar for buckets. There are three buttons: 'Create bucket', 'Delete bucket', and 'Empty bucket'. A summary shows '2 Buckets' and '1 Region'. A table lists the buckets:

Bucket name	Access	Region	Date created
microbial-ai	Public	US East (Ohio)	Dec 2, 2017 9:44:17 AM GMT-0500
microbial-interaction-database	Not public *	US East (Ohio)	Nov 18, 2017 3:16:07 PM GMT-0500

* Objects might still be publicly accessible due to object ACLs. [Learn more](#)

Setting up an S3 bucket

1. Create S3 bucket

o s

Create bucket

1 Name and region 2 Set properties 3 Set permissions 4 Review

Name and region

Bucket name ⓘ

testbucket

Region

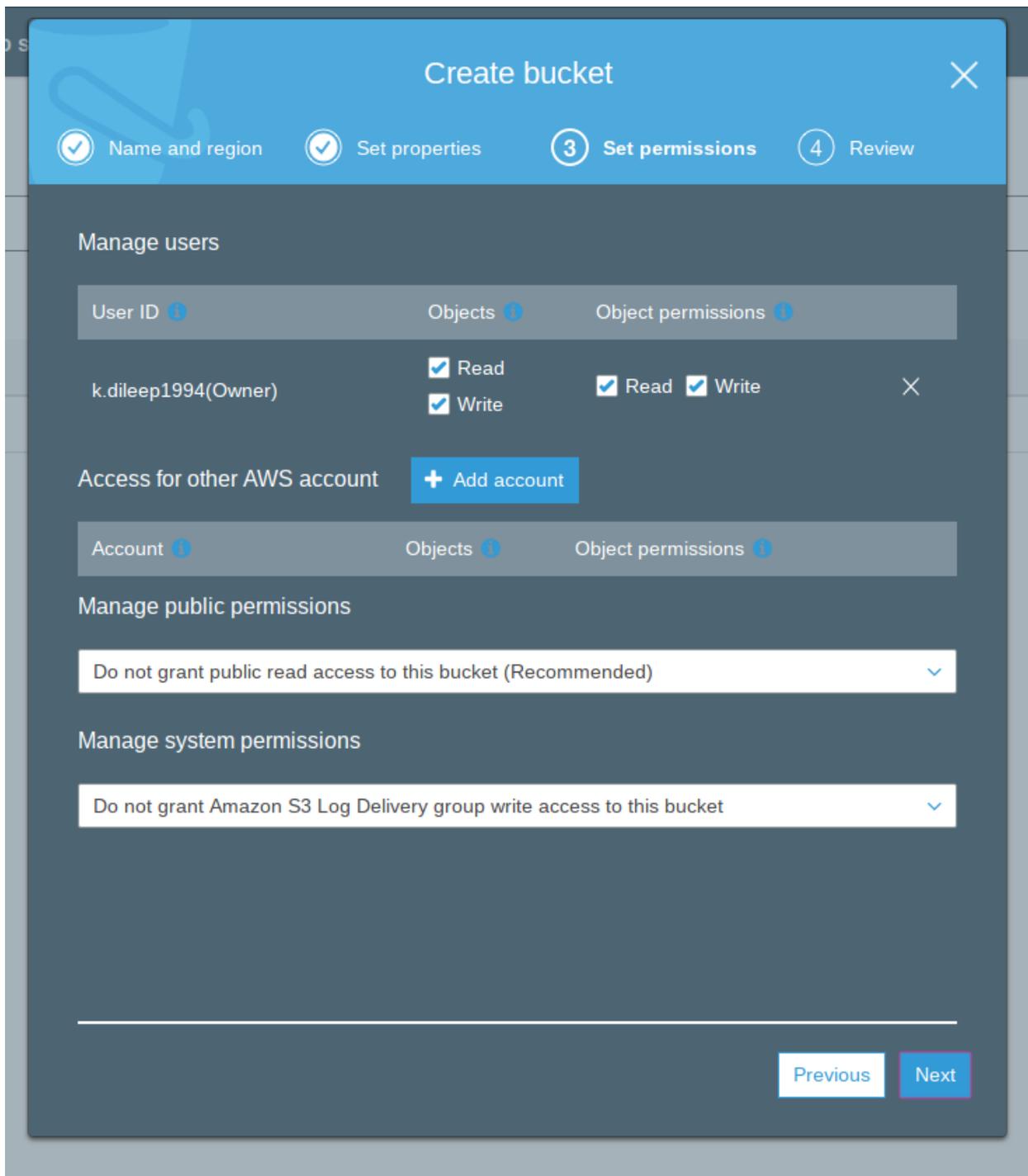
US East (Ohio) ▾

Copy settings from an existing bucket

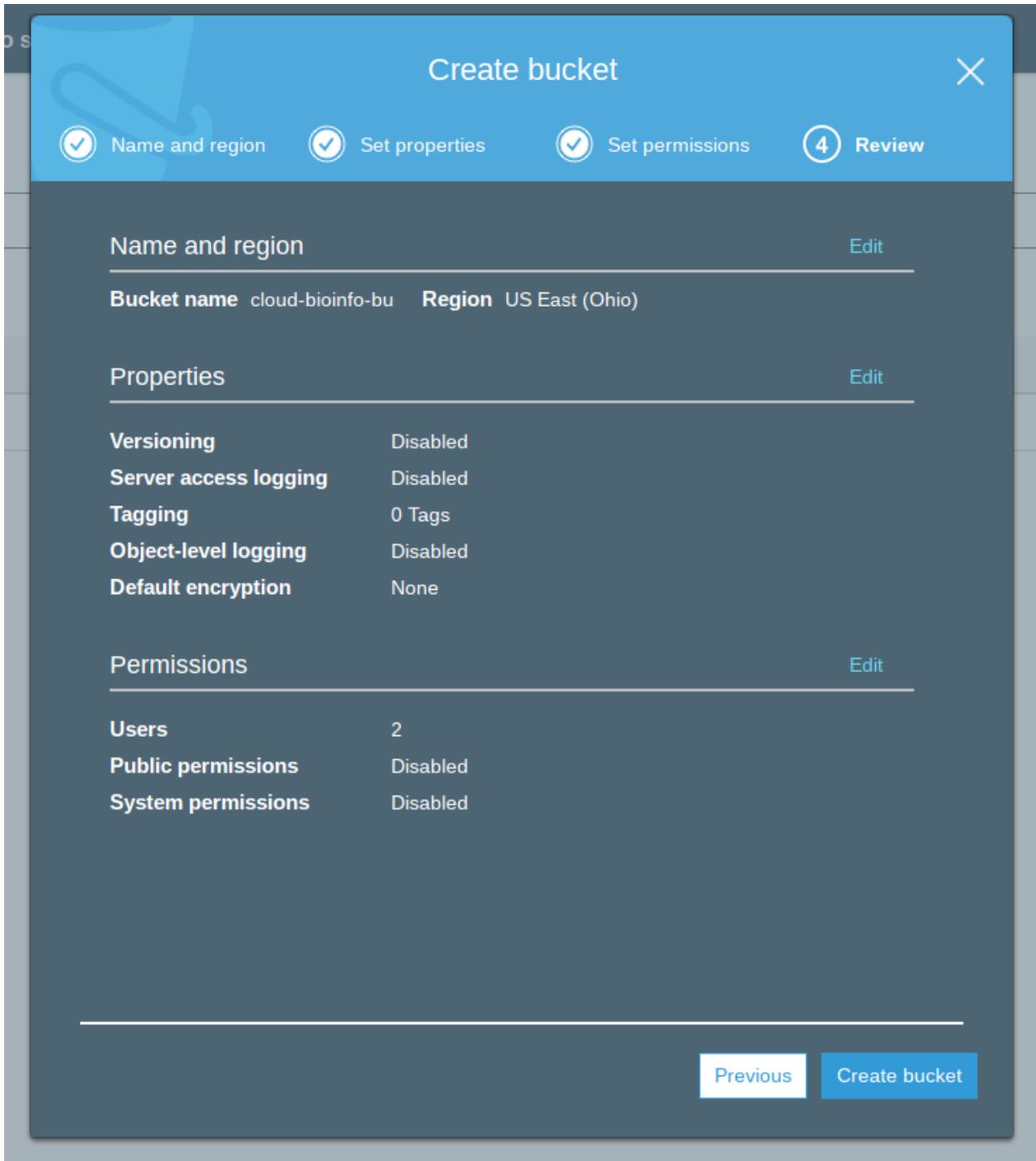
Select bucket (optional) 2 Buckets ▾

Create Cancel Next

2. Change permissions



3. Review



3. EBS

Elastic Block Storage

EBS allows to rent storage and directly mount it to your EC2 instance. In contrast to S3, EBS can only be connected to one EC2 instance at a time and its storage prices are higher.

4. RDS

Relational Database Service

5. AWS Lambda

Run code without thinking about servers. Pay only for the compute time you consume.

6. Pricing

When you create EC2 instances or S3 buckets you are renting computing power from Amazon for which you will be charged. Once you start the instance you will be charged hourly

There is a pricing list Amazon provides a monthly price calculator

4.1.5 CloudFormation

AWS CloudFormation is a service that helps deploy infrastructure as code. You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and AWS CloudFormation takes care of provisioning and configuring those resources for you. You don't need to individually create and configure AWS resources and figure out what's dependent on what; AWS CloudFormation handles all of that. There are similar resources for other services as well, for example, Azure Resource Manager for Microsoft Azure.

Advantages

1. Simplify infrastructure management
2. Quickly replicate your infrastructure
3. Reproducible infrastructure deployment
4. Easily control and track changes to your infrastructure
5. Automatic resource removal

Concepts

- **Templates:** The CloudFormation template is a JSON or YAML formatted text file that contains the configuration information about the AWS resources you want to create. When fed to CloudFormation, the template will direct it to create the required resources on AWS. Templates can also be created using the [AWS CloudFormation Designer](#)
- **Stacks:** When you use AWS CloudFormation, you manage related resources as a single unit called a stack. You create, update, and delete a collection of resources by creating, updating, and deleting stacks. All the resources in a stack are defined by the stack's AWS CloudFormation template. You can create, update or delete stacks by using the AWS CloudFormation [console](#), [API](#), or [AWS CLI](#).
- **Change Sets:** If you need to make changes to the running resources in a stack, you update the stack. Before making changes to your resources, you can generate a change set, which is summary of your proposed changes. Change sets allow you to see how your changes might impact your running resources, especially for critical resources, before implementing them

Template Components

The anatomy of a CloudFormation template:

```
{
  "AWSTemplateFormatVersion": "version date",
  "Description": "description of the template",
  "Parameters": {"set of parameters"},
  "Mappings": {"set of mappings"},
  "Conditions": {"set of conditions"},
  "Resources": {"set of resources"},
  "Outputs": {"set of outputs"}
}
```

All templates consist of the following:

1. **Parameters:** Values to pass to your template at run-time (during stack creation), containing the specifics for the EC2 or S3 needed. A parameter is an effective way to specify sensitive information, such as user names and passwords or unique information, that you don't want to store in the template itself. You can refer to parameters from the Resources and Outputs sections of the template. Multiple parameters can be passed such as the EC2 instance type, SSH security protocols, etc. For example, the code section below defines an InstanceTypeParameter for an EC2 instance.

```
{
  "Parameters": {
    "InstanceTypeParameter": {
      "Type": "String",
      "Default": "t2.micro",
      "AllowedValues": [
        "t2.micro",
        "m1.small",
        "m1.large"
      ],
      "Description": "Enter t2.micro, m1.small, or m1.large. Default is t2.
↪micro."
    }
  }
}
```

2. **Mappings:** A mapping of keys and associated values that you use to specify conditional parameter values, similar to a lookup table. You can match a key to a corresponding value by using the Fn::FindInMap intrinsic function in the Resources and Outputs section. In this example, it will match the corresponding AMI for a given AWS region

```
{
  "Mappings": {
    "RegionMap": {
      "us-east-1": {
        "32": "ami-6411e20d"
      },
      "us-west-1": {
        "32": "ami-c9c7978c"
      },
      "eu-west-1": {
        "32": "ami-37c2f643"
      },
      "ap-southeast-1": {
```

(continues on next page)

(continued from previous page)

```

    "32": "ami-66f28c34"
  },
  "ap-northeast-1": {
    "32": "ami-9c03a89d"
  }
}
}
}
}

```

3. **Conditions:** Conditions that control whether certain resources are created or whether certain resource properties are assigned a value during stack creation or update. For example, you could conditionally create a resource that depends on whether the stack is for a production or test environment.
4. **Resources:** The Resources section specifies the stack resources and their properties, such as an AWS EC2 instance or an AWS S3 bucket. This is the only part of the template that is mandatory. Each resource is listed separately and specifies the properties that are necessary for creating that particular resource. You can refer to resources in the Resources and Outputs sections of the template. The following code section describes an EC2Instance resource and InstanceSecurityGroup resource. The resource declaration begins with a string that specifies the logical name for the resource.

```

{
  "Resources": {
    "EC2Instance": {
      "Type": "AWS::EC2::Instance",
      "Properties": {
        "InstanceType": "InstanceType",
        "SecurityGroups": [
          "InstanceSecurityGroup"
        ],
        "KeyName": "KeyName",
        "ImageId": "ami-08f569078da6ad4c2"
      }
    },
    "InstanceSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Enable SSH access via port 22",
        "SecurityGroupIngress": [
          {
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIp": "SSHLocation"
          }
        ]
      }
    }
  }
}
}

```

5. **Outputs:** Describes the values that are returned whenever you view your stack's properties. For example, you can declare an output for an EC2 instance to display its id and availability zone

```

{
  "Outputs": {
    "InstanceId": {

```

(continues on next page)

(continued from previous page)

```

    "Description": "InstanceId of the newly created EC2 instance",
    "Value": "EC2Instance"
  },
  "AZ": {
    "Description": "Availability Zone of the newly created EC2 instance",
    "Value": {
      "Fn::GetAtt": [
        "EC2Instance",
        "AvailabilityZone"
      ]
    }
  }
}

```

Note:

1. The Resource Type attribute has the format - `AWS::ProductIdentifier::ResourceType`. Eg: The Resource Type for an S3 bucket is `AWS::S3::Bucket` and that for an EBS volume is `AWS::EC2::Volume`.
 2. The `Ref` function returns the value of the object it refers to. The `Ref` function can also set a resource's property to the value of another resource.
 3. Depending on the resource type, some properties are required, other optional properties are assigned default values.
 4. Some resources can have Multiple properties and some properties can have one or more subproperties.
-

Best Practices

Take a look at the official [best practices](#) to be able to use AWS CloudFormation more effectively and securely.

4.1.6 AWS CLI

The AWS CLI is an open source tool built on top of the AWS SDK for Python (Boto) that provides commands for interacting with AWS services. With minimal configuration, you can start using all of the functionality provided by the AWS Management Console from your favorite terminal program.

For installation instructions refer to the [official documentation](#).

Advantages

1. Easy to install
2. Supports all Amazon Web Services
3. Easy to use
4. Can be incorporated in shell scripts for automation and reproducibility

Setting up your profile

Before you can start using the `aws-cli` you need to configure the CLI with your AWS credentials. The `aws configure` command is the fastest way to set this up. This command automatically generates the credentials file at `~/.aws/credentials` and the config file at `~/.aws/config`.

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]: json
```

The AWS CLI will prompt you for four pieces of information. AWS Access Key ID and AWS Secret Access Key are your account credentials.

Alternatively you can manually create and populate these files.

`~/.aws/credentials`

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
```

`~/.aws/config`

```
[default]
region=us-east-1
output=json
```

If you have multiple profiles you can also configure additional named profiles using the `--profile` option

```
$ aws configure --profile user2
AWS Access Key ID [None]: AKIAI44QH8DHBEXAMPLE
AWS Secret Access Key [None]: je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]: text
```

Commands

Help:

To get help when using the AWS CLI, you can simply add `help` at the end of a command or sub-command.

```
$ aws help
$ aws ec2 help
$ aws ec2 describe-instances help
```

The help for each command is divided into six sections: Name, Description, Synopsis, Options, Examples and Output.

Command Structure:

```
$ aws <command> <sub-command> [options and parameters]
```

Specifying parameter values

```
$ aws ec2 create-key-pair --key-name my-key-pair
```

Output:

The AWS CLI supports three different output formats:

- json
- Tab-delimited text
- ASCII formatted table

The default output format is chosen during the configuration step of `aws configure`. This can be changed by editing the `config` file or setting the `AWS_DEFAULT_OUTPUT` environment variable.

Additionally, per command output can be changed using the `--output` option

```
$ aws swf list-domains --registration-status REGISTERED --output text
```

```
# Example output
$ aws ec2 describe-volumes
```

```
{
  "Volumes": [
    {
      "AvailabilityZone": "us-west-2a",
      "Attachments": [
        {
          "AttachTime": "2013-09-17T00:55:03.000Z",
          "InstanceId": "i-a071c394",
          "VolumeId": "vol-e11a5288",
          "State": "attached",
          "DeleteOnTermination": true,
          "Device": "/dev/sda1"
        }
      ],
      "VolumeType": "standard",
      "VolumeId": "vol-e11a5288",
      "State": "in-use",
      "SnapshotId": "snap-f23ec1c8",
      "CreateTime": "2013-09-17T00:55:03.000Z",
      "Size": 30
    },
    {
      "AvailabilityZone": "us-west-2a",
      "Attachments": [
        {
          "AttachTime": "2013-09-18T20:26:16.000Z",
          "InstanceId": "i-4b41a37c",
          "VolumeId": "vol-2e410a47",
          "State": "attached",
          "DeleteOnTermination": true,
          "Device": "/dev/sda1"
        }
      ],
      "VolumeType": "standard",
      "VolumeId": "vol-2e410a47",
      "State": "in-use",
      "SnapshotId": "snap-708e8348",
      "CreateTime": "2013-09-18T20:26:15.000Z",
      "Size": 8
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
}
]
}
```

You can query the resultant output using the `--query` option.

```
$ aws ec2 describe-instances --instance-ids i-0787e4282810ef9cf --query
↳ 'Reservations[0].Instances[0].PublicIpAddress'
"54.183.22.255"
```

Examples:

The following examples show the interface in action performing various tasks and demonstrate how powerful it can be.

```
# deleting an s3 bucket
aws s3 rb s3://bucket-name --force
```

```
# start ec2 instances
aws ec2 start-instances --instance-ids i-34hj23ie
```

Miscellaneous

- Try the `aws-shell` to get a more interactive command line experience.
- Use `jq` to parse the json outputs from various cli commands.

4.1.7 Cheat sheet

AWS CLI

1. Configuring your AWS CLI

```
$ aws configure --profile <profile>
```

CloudFormation

2. Deploying your stack using the AWS CLI via CloudFormation

```
$ aws --profile <profile> cloudformation create-stack --stack-name <stack> [--
↳ template-body <template>] [--parameters <parameters>]
```

Note: Local files need to be prefixed with `file://`

3. Verify and check stack deployment using the AWS CLI

```
$ aws --profile <profile> cloudformation describe-stacks [--stack-name <stack>]
```

4. List resources of a stack using the AWS CLI

```
$ aws --profile <profile> cloudformation list-stack-resources --stack-name <stack>
```

5. Validate your CloudFormation template using the AWS CLI

```
$ aws --profile <profile> cloudformation validate-template --template-body <template>
```

6. Update your stack using the AWS CLI

```
$ aws --profile <profile> cloudformation update-stack --stack-name <stack> [--  
→template-body <template>] [--parameters <parameters>]
```

EC2 Instance

7. Connecting to the deployed EC2 instance via ssh

```
$ ssh -i <key.pem> user@<publicip>
```

To obtain the PublicIpAddress of your instance:

```
$ aws ec2 describe-instances --instance-ids i-0787e4282810ef9cf --query  
→'Reservations[0].Instances[0].PublicIpAddress'
```

Note: The “key.pem” must only allow read access to the user

1. Key - The key specified must be at the path indicated. It must be the private key. Permissions on the key must be restricted to the owner and the key must be associated with the instance.
 2. User - The user name must match the default user name associated with the AMI you used to launch the instance. For an Ubuntu AMI, this is `ubuntu`. For an Amazon Linux AMI, it is `ec2-user`.
 3. Instance - The public IP address or DNS name of the instance. Verify that the address is public and that port 22 is open to your local machine on the instance’s security group.
-

S3 bucket

8. Copy an object from an S3 bucket to EC2 instance or local machine

```
$ aws s3 cp s3://my_bucket/my_folder/my_file.ext my_copied_file.ex
```

9. Copy an object from and EC2 instance or local machine to S3 bucket

```
$ aws s3 cp my_copied_file.ext s3://my_bucket/my_folder/my_file.ext
```

10. Download an entire Amazon S3 bucket to a local directory on your instance

```
$ aws s3 sync s3://remote_S3_bucket local_directory
```

Files

- [EC2 template](#)
- [EC2 parameters](#)

- S3 template
- S3 parameters

Exercise

1. Configure your AWS CLI
2. Run the `cloudformation describe-stacks` and `ec2 describe-instances` to look at existing stacks or instances
3. Try to `--query` the output or display the `--output` in different formats
4. Combine the EC2 and S3 templates to create one template that launches both an EC2 instance and an S3 bucket
5. Validate the template using the `cloudformation validate-template` command
6. Update the `ImageId` to “ami-08f569078da6ad4c2” and run the `cloudformation update-stack` command
7. Connect to the EC2 instance using the pem file
8. Copy the contents of this S3 bucket (`s3://buaws-training-shared/test_reads.fastq.gz`) to the instance
9. Delete the stack using the `cloudformation delete-stack` command

4.2 Cloud App Deployment Workshop

This is day 2 of the “Bioinformatics in the Cloud” workshop. In this session, you will learn about containerization software, how to execute docker containers on an AWS EC2 instance, and how to package your own applications into docker images.

4.2.1 Prerequisites

docker

This workshop assumes you have an environment where `docker` is installed. If you followed *workshop 1*, the EC2 instance you deployed already has docker installed and configured. If not, you may follow this [setup guide](#) to use docker on your own resources.

Creating an CloudFormation Stack

You may use the following template and parameters to create a CloudFormation stack with an EC2 instance that has docker pre-installed:

- Template
- Parameters

You may download these files as-is to create your AWS stack, **be sure to change the stack name to something else!**:

```
$ aws configure
AWS Access Key ID [*****QNQG]:
AWS Secret Access Key [*****z5bv]:
Default region name [us-east-1]:
Default output format [json]:
```

(continues on next page)

(continued from previous page)

```
$ aws cloudformation create-stack --template-body file://main.yaml \
  --parameters file://buaws-training-ec2-parameters.json \
  --stack-name ec2-stack-studentXX
{
  "StackId": "arn:aws:cloudformation:us-east-1:438027732470:stack/test-stack-AL/18c.
  →..."
}
```

When your stack creation is complete, you should ssh to the instance using the appropriate private key:

```
$ ssh -i buawsawstraineec2.pem ec2-user@<IP from cloudformation output>
The authenticity of host 'XX.XXX.XX.XX (XX.XXX.XX.XX)' can't be established.
RSA key fingerprint is 5d:e6:c4:f6:35:a5:9e:85:66:a4:b3:af:56:86:20:93.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'XX.XXX.XX.XX' (RSA) to the list of known hosts.
Last login: Mon Jul 23 16:27:27 2018 from nowhere

  _ | _ | _ )
  _ | ( /   Amazon Linux AMI
  _ | \ _ | _ |

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
2 package(s) needed for security, out of 4 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-19-57 ~]$
```

4.2.2 Containerization

Motivation

Science today faces a [reproducibility crisis](#). Key findings published cross scientific disciplines are not corroborated by other scientists when tested independently. A [survey conducted by Nature](#) asked scientists which factors they thought contributed the most to the crisis. Over 80% of respondents felt the reason of ‘Methods, code unavailable’ contributed to irreproducible research.

For many scientists, software and analysis have become an indispensable and increasingly unavoidable component of their research. Critical findings now arise from the analysis of data that uses tools developed in house as well as tools published by others. These components are usually integrated by custom ‘glue code’ that connects them together.

This environment poses a new set of challenges to scientists who use computational methods in their research:

- **How do we write analysis code that is robust and reproducible?**
- **How can we concisely communicate our code with other researchers?**
- **How do we share analysis code with other researchers in a form that can be easily executed?**

As computational analysis and tools become more complex, so do the environments needed to execute them. Modern software packages often require hundreds of supporting software packages, provided either by a particular operating system or from a third party. Further, each of these software package dependencies has a specific version or set of versions that are needed for the package to run. The author of a package could in principle record all of these packages and their dependencies and provide this list with their software distribution, but maintaining this list of software and ensuring cross-platform compatibility is a major challenge. Environment management software packages such as [miniconda](#) are available to address this challenge, but introduce additional complexity due to the fact that it itself is an additional software dependency, package availability is largely dependent upon community support, and because

third party software packages may not be supported across different platforms. A superior solution to managing and deploying complex software environments is to create **containerized** applications.

Containerization

Containerization, also known as [operating-system-level virtualization](#), is a technology that enables the encapsulation and execution of sets of software and their dependencies in a platform-agnostic manner. A software container is a file that has been built by specific containerization software, e.g. [docker](#) or [singularity](#), to contain all of the necessary software and instructions to run.

What is a container?

Generally speaking, a container is a file that specifies a collection of software that can run in a particular execution environment. The execution environment is provided by the containerization software, e.g. [docker](#), such that the container doesn't have to be aware of the particular machine it is running on. This means that a container will be portable to any environment where the containerization software can run, thus eliminating the need for software authors (i.e. us) to worry about whether or not our code will run on any given hardware/OS/etc.

At the time of writing (July 2018), [docker](#) is by far the most popular containerization software. [docker](#) has been open source since its release in 2013 and an enormous [docker](#) community has grown since. Due to its popularity, this workshop will use [docker](#) exclusively as the vehicle for demonstrating containerization of custom applications.

Another more recent containerization software called [singularity](#) is available that addresses some of the usability shortcomings of [docker](#). If [docker](#) is not available on your computational resources due to security concerns, then [singularity](#) may be an option. The containerization concepts are identical between [docker](#) and [singularity](#), and all of the content of this workshop is easily adaptable to from [docker](#) to [singularity](#).

4.2.3 Introduction to docker

docker

[docker](#) is an open source software project supported and provided for free by [Docker Inc.](#) The software is available for Mac OS, Windows, and Linux operating systems. From its initial open source announcement in 2013, [docker](#) is:

a Linux Container (LXC) technology augmented with a high level API providing a lightweight virtualization solution that runs Unix processes in isolation. It provides a way to *automate* software deployment in a *secure* and *repeatable* environment.

(emphasis added). [docker](#) containers are:

- *automated* because every [docker](#) container contains all of its own configuration is run with the same executable interface, and thus can be started automatically without manual intervention
- *secure* because each runs in its own environment isolated from the host and other containers
- *repeatable* because the container behavior is guaranteed to be the same on any system that runs the [docker](#) software

These three properties make [docker](#) an excellent solution to the problems faced by scientists who wish to write reproducible analysis and applications.

docker concepts

There are four critical concepts needed to get started as a [docker](#) user:

images

A docker image is a description of a software environment and its configuration. The concept of an image is abstract, as images are not run directly. Instead, images are used to instantiate *containers* that are runnable. For those familiar with object oriented programming, an image is to a container as a class is to an object. As such, images are not executed.

docker images are usually created, or *built*, with a *Dockerfile*. Images are often created using other images as a base and adding more application-specific configuration and software. For example, a common base image contains a standard `ubuntu` installation upon which other software is installed. While it is possible to *build an image interactively* without writing a Dockerfile, this practice is highly discouraged due to its irreproducibility.

Images can either be stored locally or in a public or private *Image Registry*. In any case, in order to create a container based off of an image, the image must be resident in the local docker installation. When building an image locally, the image is automatically added to the local registry. When using an image published on a public registry like *Docker Hub*, the image is first *pulled* to the local installation and then used to create an image.

Most docker images have a *version* associated with them. This enables the image to change over time while maintaining backwards compatibility and reproducibility. The image version is specified at build time.

container

A container is an instance created by image. You can think of a container as a physical file that has all of the software described by the image bundled together in a form that can be run. Each container is created using a single image.

By default, containers lack the permissions to communicate with the world outside its immediate docker execution environment. When a container is run, the user can specify locations on the host system that are exposed to the docker container by *binding* files and directories explicitly. The container can only read and write data to locations it is given permission to access. Containers that run services, like web servers, can also be granted access to certain ports on the host system at run time to allow communication outside of the host. In general, a docker container can only be granted access to the resources available to the user running the container (e.g. a normal user without elevated privileges cannot bind to reserved ports 0-1024 on linux).

Dockerfiles

A *Dockerfile* is a text file that contains the instructions for building an image. It is the preferred method for building docker images, over creating them interactively.

Dockerfiles are organized into sections that specify different aspects of an image. The following is a simple Dockerfile from the docs:

```
# Use an official Python runtime as a parent image
# This implicitly looks for and pulls the docker image named 'python'
# annotated with version '2.7-slim' from Docker Hub (if it was not already
# pulled locally)
FROM python:2.7-slim

# Set the working directory to /app inside the container
# The /app directory is created implicitly inside the container
WORKDIR /app

# Copy the current (host) directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
# The file requirements.txt was copied into /app during the ADD step above
```

(continues on next page)

(continued from previous page)

```
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
# This implies that app.py runs a web server on port 80
EXPOSE 80

# Define environment variable $NAME
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

The commands in all capital letters at the beginning of the line are Dockerfile commands that perform different configuration operations on the image.

Image Registry

Image registries are servers that store and host docker images. The software to run a [Docker Registry](#) is freely available, but [Docker Hub](#) is by far the most popular public registry. Docker images for your own apps can be freely published to and listed on [Docker Hub](#) for others to pull and use. Other free registries exist, including [Amazon Elastic Container Registry](#) and [Google Cloud Container Registry](#).

Exercise

Navigate to [Docker Hub](#) and locate the `python` repository. Explore the page until you find the Dockerfile for python version `3.7-stretch` and view it. What parent image was used to build the `python:3.7-stretch` image?

Locate the parent image on Docker Hub and examine its Dockerfile. What parent image was used to build this image?

Continue looking up the parent images of each Dockerfile you find until you reach the root image. What is its name?

4.2.4 Running docker

Nota Bene

You must be using a computer with docker installed to complete the exercises on this page. If you are attending the BU workshop, refer to the page on connecting to your EC2 instance for instructions on how to SSH into your instance.

Your First Docker Container

Containers are run using the command:

```
$ docker run <image name>[:<tag>]
```

The `<image name>` must be a recognized docker image name either on the local machine or on [Docker Hub](#). The optional `:<tag>` specifies a particular version of the image to run.

Exercise

Run a container for the `hello-world` docker image hosted on [Docker Hub](#).

If you need help, try running `docker` and `docker run` without any arguments to see usage information.

Read the text output by the container after it has been run.

Pulling docker images

As part of running a container from a public docker image, the image itself is pulled and stored locally. This only occurs once for each version of an image; subsequently run containers will use the local copy of the image.

If you have never run any docker containers in this environment before, there should be no local images listed by the `docker images` command:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
$
```

To verify that the `hello-world` image has been pulled, we again use the `docker images` command after running the container:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world        latest             2cb0d9787c4d      2 weeks ago       1.85kB
$
```

This output tells us that we have the latest version of the `hello-world` image in our local registry.

We can pull images explicitly, rather than doing so implicitly with a `docker run` call, using the `docker pull` command:

```
$ docker pull nginx
```

This may be useful if we do not want to run a container immediately, or want to perform our own modifications to the image locally prior to running.

Exercise

Pull the `nginx` image using the `docker pull` command. Verify that the latest image of `nginx` has been pulled using `docker images`.

Managing docker containers

Running detached containers

The `hello-world` container runs, prints its message, and then exits. If we were running a docker container that provided a service, we would want the container to persist running until we chose to shut it down. An example of this is the `nginx` web server, which we can run with the command:

```
$ docker run -d -p 8080:80 nginx
```

Here, the `-d` flag tells docker to keep the container running and return control to the command line when it is finished setting up the container. The `-p 8080:80` means forward port 80, the default port for HTTP traffic, on the container to the unrestricted port 8080 on the local machine. When control has returned to the command line, we can verify that the container is still running using the `docker ps` command:

```
$ docker ps
CONTAINER ID          IMAGE          COMMAND          CREATED
↪STATUS              PORTS         NAMES
49af27e82231         nginx         "nginx -g 'daemon of..." 4 minutes ago
↪ Up 4 minutes      0.0.0.0:8080->80/tcp elastic_mcnulty
$
```

Exercise

Run an nginx container as above. Verify that the container is running with `docker ps`.

If specified correctly, the local port 8080 should behave as if it is a web server. Verify that this is the case by running:

```
$ curl localhost:8080
```

Attaching data volumes to containers

Scientific analyses almost always utilize some form of data. Docker containers are intended to execute code, and are not designed to house data. Directories and data volumes that exist on the host machine can be *mounted* in the container at run time to enable the container to read and write data to the host:

```
$ docker run -d -p 8080:80 --mount type=bind,source="$PWD"/data,target=/ nginx
```

The directory named `data` in the current host directory will be mounted as `/data` in the root directory of the container.

Stopping running containers

When a docker container has been run in a detached state, it runs until it is stopped or encounters an error. To stop a running container, we need either the `CONTAINER ID` or `NAMES` attribute of the running container from `docker ps`:

```
$ docker ps
CONTAINER ID          IMAGE          COMMAND          CREATED
↪STATUS              PORTS         NAMES
49af27e82231         nginx         "nginx -g 'daemon of..." 4 minutes ago
↪ Up 4 minutes      0.0.0.0:8080->80/tcp elastic_mcnulty
$ docker stop 49af27e82231 # could also have provided elastic_mcnulty
49af27e82231
$ docker ps
CONTAINER ID          IMAGE          COMMAND          CREATED
↪STATUS              PORTS         NAMES
$
```

Stopping a container sends signals to the container that it should start shutting down, so once a container is stopped it usually cannot be started again.

Nota Bene

Docker maintains a record of all containers that have been run on a machine. After they have been stopped, `docker ps` does not show them, but the containers still exist. To see a list of all containers that have been run, use `docker ps -a`.

It is good practice to remove old containers if they are no longer needed. You can do this with the command `docker container prune`.

Creating docker images

Building a custom image

Chances are there is not an existing docker container that does exactly what you want (but check first!). To create your own image, you must write a `Dockerfile`. As an example, we will create an image that has the python package `scipy` installed for us to use. It is common convention to create a new directory named for the the image you wish to create, and create a text file named `Dockerfile` in it. In the `scipy` directory, our `Dockerfile` contains:

```
# pull a current version of python3
FROM python:3.6

# install scipy with pip
RUN pip install scipy

# when the container is run, put us directly into a python3 interpreter
CMD ["python3"]
```

To build this docker images, we use the `docker build` command from within the `scipy` directory containing the `Dockerfile`:

```
$ docker build --tag scipy:latest .
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM python:3.6
--> 638817465c7d
Step 2/3 : RUN pip install scipy
--> Running in leef65d3b6fd
Collecting scipy
  Downloading https://files.pythonhosted.org/...
Collecting numpy>=1.8.2 (from scipy)
  Downloading https://files.pythonhosted.org/...
Installing collected packages: numpy, scipy
Successfully installed numpy-1.15.0 scipy-1.1.0
Removing intermediate container leef65d3b6fd
--> 7f34e9147bef
Step 3/3 : CMD ["python3"]
--> Running in 5c9d778426e6
Removing intermediate container 5c9d778426e6
--> e27603f4ffaf
Successfully built e27603f4ffaf
Successfully tagged scipy:latest
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
scipy	latest	e27603f4ffaf	About a minute ago	1. ↪15GB
python	3.6	638817465c7d	25 hours ago	922MB

```
$
```

The `--tag scipy:latest` argument gives our image a name when it is listed in `docker images`. Notice also that the `python:3.6` image has been pulled in the process of building the `scipy` image.

Now that we have built our image, we can run and connect to the image using `docker run` with two additional flags:

```
$ docker run -i -t scipy
Python 3.6.0 (default, Jul 17 2018, 11:04:33)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import scipy
>>>
```

The `-i` flag tells docker we want to use the container interactively, and the `-t` flag connects our current terminal to the container so that we may send and receive information to and from the terminal.

Exercise

Create a new Dockerfile where you will install the most recent version of R. Use `ubuntu:bionic` as the base image. You may follow [these instructions](#), without using the `sudo` command.

Hint: Use a different RUN line for each command.

Solution

Passing containers CLI arguments

The CMD Dockerfile command specifies a standalone executable to run when a container starts. However, sometimes it is convenient to be able to pass command line arguments to a container, for example to run an analysis pipeline on different files, or files with filenames that are not known at build time. For instance, if you we might want to run the following:

```
$ docker run python process_fastq.py some_reads.fastq.gz
```

The CMD command does not allow command line arguments to be passed to the run command. Instead, the ENTRYPOINT command is used to prefix a set of commands to any command line arguments passed to docker:

```
FROM python:3.6

# we will mount the current working directory to /cwd when the container is run
WORKDIR /cwd

RUN pip install pysam

# ENTRYPOINT instead of CMD
ENTRYPOINT ["python3"]
```

Any command line arguments passed to docker will be appended to the command(s) specified in the ENTRYPOINT.

If a container is intended to run files that exist on the host, the docker run command must also be supplied with a mount point so the container can access the files. In the example above, the WORKDIR is specified as `/cwd`, so we can bind the current working directory of the host to `/cwd` in the container so it can access the files `process_fastq.py` and `some_reads.fastq` in the current directory:

```
$ docker run -mount type=bind,source=$PWD,target=/cwd process_fastq.py some_reads.
→fastq
```

4.2.5 Packaging your own application

Workflow Overview

The simplest workflow for building a docker container with your own code usually follows these steps:

1. *Identify an appropriate image*
2. Identify additional dependencies needed for your application
3. *Install those dependencies with the appropriate RUN commands*
4. *Add your code to the image, either with ADD or git*
5. Specify an appropriate CMD or ENTRYPOINT specification
6. Build your image, repeating 2-4 if needed until success
7. *Run a container of your image, test behavior*
8. Iterate, if needed

Preparing docker image for your code

Choosing a base image

The first step in creating a docker container is choosing an appropriate base image. In general, picking the most specific image that meets your requirements is desirable. For example, if you are packaging a python app, it is likely advantageous to choose a [python base image](#) with the appropriate python version rather than pulling an [ubuntu base image](#) and installing python using RUN commands.

Installing dependencies

Once a base image is chosen, any additional dependencies need to be installed. For [debian based images](#), the [apt package manager](#) is used to manage additional packages. For [Fedora based images](#), the [yum package manager](#) is used. Be sure to check which base linux image is used for a more specific image to know which package manager to use.

Annoyance Alert

In practice, it can be hard to know all of the additional system packages that need to be installed. Often, building a image to completion and running it to identify errors is the most expedient way to create an image.

Occasionally, a software package dependency, or a specific version of software, is not available in the software repositories for a base image linux distro. In these cases, it might be necessary to download and install precompiled binaries manually, or build a package from source. For example, here is an example Dockerfile that installs a specific version of [samtools](#) from a [source release](#) available on [github](#):

```
FROM ubuntu:bionic

RUN apt update

# need these packages to download and build samtools:
# https://github.com/samtools/samtools/blob/1.9/INSTALL
RUN apt install -y wget gcc libz-dev ncurses-dev libbz2-dev liblzma-dev \
    libcurl3-dev libcrypto++-dev make
```

(continues on next page)

(continued from previous page)

```
RUN wget https://github.com/samtools/samtools/releases/download/1.9/samtools-1.9.tar.  
↪bz2 && \  
    tar jxf samtools-1.9.tar.bz2 && \  
    cd samtools-1.9 && ./configure && make install  
  
CMD ["samtools"]
```

Putting your code into a docker image

Once your dependencies are installed, the final step is to move your own code into your image. There are primarily two different strategies for doing so:

- Copy source files into the image using the `ADD` command in the Dockerfile
- Clone a git repository into the image from a publicly hosted repo like [github](#) or [bitbucket](#)

Nota Bene

In any case, it is a good idea to create a git or other source code versioning system to develop your code, hosted publicly if possible. Your Dockerfile should be developed and tracked along with your code, so that both can be developed over time while maintaining reproducibility.

Locally

The local strategy is convenient when developing software. Running development code in a docker container ensures your testing and debugging environment are consistent with the execution environment where your code will ultimately run. To build from a local source tree:

1. Create a Dockerfile in the root directory where your code resides
2. Prepare the Dockerfile for your code as in *Preparing docker image for your code*
3. Copy all of the source files into a directory (e.g. `/app`) in the container with `ADD . /app`
4. Perform any setup that comes bundled with your package source (e.g. `pip install -r requirements.txt` or `python setup.py`) with the `RUN` command
5. Set the `CMD` entry point appropriately for your app
6. Build your image with an appropriate tag
7. Run and test your application, ideally with unit tests

Assuming we have written a python application named `app.py`, from within the source code directory containing the application we could write the following Dockerfile:

```
# Use an official Python runtime as a parent image  
FROM python:2.7-slim  
  
# Copy the current (host) directory contents into the container at /app  
ADD . /app  
  
# Install any needed packages specified in requirements.txt  
RUN pip install --trusted-host pypi.python.org -r requirements.txt
```

(continues on next page)

(continued from previous page)

```
# mount the current working directory to /cwd when the container is run
WORKDIR /cwd

# Run app.py when the container launches
ENTRYPOINT ["python", "app.py"]
```

When a container is run, `app.py` will be run directly and passed any additional arguments specified to the `docker run` command.

Cloning from github/bitbucket

For software projects hosted on [github](#) or [bitbucket](#), or when it is not desired to include a Dockerfile along with your application source code, the Dockerfile can also be set to clone and install a git repo instead of adding code locally. Instead of using the `ADD` command from *above*, use a `RUN git clone <repo url>` instead:

```
FROM python:3.6

# have to install git to clone
RUN apt install git

# git clone repo instead of ADD
RUN git clone https://bitbucket.org/bubioinformatics/docker_test_app /app
RUN pip install --trusted-host pypi.python.org -r /app/requirements.txt

# mount the current working directory to /cwd when the container is run
WORKDIR /cwd

# use ENTRYPOINT so we can pass files on the command line
ENTRYPOINT ["python", "/app/app.py"]
```

Cloning a public repo into a Docker container in this way has the advantage that the environment where you write your code can be the same or different than the platform where the code is run.

There is one additional caveat to this method of adding code to your image. To save on build time, docker caches the sequential steps in your Dockerfile when building an image, and only reruns the steps from the command where a change has been made. The `ADD` command automatically detects if local file changes have been made and automatically re-copies them into the container on docker build. This method of cloning a repo from bitbucket, however, does not re-trigger a build. When cloning your application from a public git repo, the `--no-cache` flag must be provided to your docker build command:

```
$ docker build --no-cache --tag app:latest .
```

This invalidates all build cache and re-clones your repo on each build.

Running your docker container

Once your code has been loaded into an image, containers for your image can be run in the normal way with `docker run`. Any host directories containing files needed for the analysis must be mounted:

```
$ docker run --mount type=bind,source=/data,target=/data \
  --mount type=bind,source=$PWD,target=/cwd app \
  --in=/data/some_data.txt --out=/data/some_data_output.csv
```

Remember that any time your code changes you will need to rebuild your image, including `--no-cache` if you pull your code from a git repo.

Publishing your docker image

Once your docker image is complete and your app is ready to share, you can create a free account on [Docker Hub](#) and upload your image. Be sure to provide a full description of what the image does, what software it contains, and how to run it, specifying any directories the container expects to be mounted to access data (e.g. `/data`). You might alternatively consider hosting your image on the [Amazon Elastic Container Registry](#) or [Google Cloud Container Registry](#). If your app will primarily be executed in either AWS or GAE environments, it may be preferable to publish your image to the corresponding registry.

Hands On Exercise

Writing the Dockerfile

Write, build, and run a Dockerfile that:

1. Uses the `python:3.6` base image
2. Installs `git` with `apt`
3. Clones the repo `docker_test_app`
4. Installs the dependencies using the `requirements.txt` file in the repo
5. Configures the `ENTRYPOINT` to run the script in the repo with `python3`

Running the Dockerfile with data from an S3 bucket

Nota Bene

When you run this app, you should specify the `-t` flag to your `docker run` command.

Try running the container using `docker run` with no arguments to see the usage.

A fastq file that can be passed to this script has been made available on a shared S3 bucket. You will download this file to your local instance using the `aws cli`. First, you must run `aws configure` and provide your access keys. Specify `us-east-1` as the region. The bucket address of the file is:

```
s3://buaws-training-shared/test_reads.fastq.gz
```

Download the file using the `aws cli` and pass it to the app using `docker run`. You must mount the directory where you downloaded the fastq file using the `--mount` command line option as above.

4.3 FireCloud Workshop

This is day 3 of the “Bioinformatics in the Cloud” workshop. In this session, you will learn about the platform FireCloud. We will learn how to run workflows, upload data, and create methods.

Workshop Outline:

- Introduction to FireCloud from Broad pipeline outreach coordinator Kate Noblett (~10min)

- FireCloud Intro Presentation (~15min)
- FireCloud Guided Tour (~25min)
- FireCloud \$5 Pipeline Hands-On (~15min)
- Break (~5min)
- FireCloud Custom Data and Method Hands-On (~40min)
- Explore FireCloud (Rest of Workshop Time)

4.3.1 Prerequisites

Note: The participants are required to have access to the following resources before attending the workshop

- **FireCloud account** Credits to run workflows (\$300 free on sign up)
- **portal.firecloud.org** Make an account and connect to a google account

4.3.2 Five dollar genome analysis pipeline

Clone and run a featured workspace

Open up portal.firecloud.org

Find the pipeline

1. Navigate to the workspace tab
2. Navigate to “Featured Workspaces”
3. Click on “five-dollar-genome-analysis-pipeline”

The screenshot shows the FireCloud portal interface. At the top, the 'Workspaces' tab is selected, with a red box and '#1' next to it. Below the navigation bar, the 'Featured Workspaces (9)' section is highlighted with a red box and '#2'. A table lists various workspaces, with the first row, 'five-dollar-genome-analysis-pipeline', highlighted in green and marked with a red box and '#3'. The table columns are Status, Workspace, Description, Last Modified, and Access Level. The first row's description is '### GATK Best Practices for Germline SNPs an'. The table footer shows '1 - 9 of 9 results (filtered from 105 total)' and a pagination control set to '20 per page'.

Status	Workspace	Description	Last Modified	Access Level
✓	help-gatk five-dollar-genome-analysis-pipeline	### GATK Best Practices for Germline SNPs an	Jun 18, 2018, 2:53 PM	Reader
✓	help-gatk Germline-SNPs-Indels-GATK4-hg38	### GATK Best Practices for Germline SNPs & I	Jul 27, 2018, 8:10 AM	Reader
✓	help-gatk Seq-Format-Conversion	### Methods for format conversion	Jul 24, 2018, 11:54 AM	Reader
✓	help-gatk Somatic-CNVs-GATK4	### GATK Best Practices for Somatic CNV Disc	Apr 27, 2018, 7:45 PM	Reader
✓	help-gatk Pre-processing_b37_v3	### GATK Best Practices for Germline SNPs Inc	Jun 18, 2018, 2:54 PM	Reader
✓	help-gatk Somatic-SNVs-Indels-GATK4	### GATK Best Practices for Single Tumor-Norr	Jun 21, 2018, 1:55 PM	Reader
✓	help-gatk Pre-processing_hg38_v2	### GATK Best Practices for Germline SNPs Inc	Jun 18, 2018, 2:54 PM	Reader
✓	help-gatk Germline-SNPs-Indels-GATK4-b37	### GATK Best Practices for Germline SNPs Inc	Jul 27, 2018, 10:31 AM	Reader
✓	gmgflobroad gmgf	# GMQL 101: an introduction to the GenoMetric	May 6, 2018, 7:00 PM	Reader

Clone the workspace

1. Append your name to the workspace name (to make it unique)
2. Clone the workspace

Clone Workspace ✕

#1

Workspace Name

Only letters, numbers, underscores, dashes and spaces allowed

Billing Project

Description (optional)

The purpose of this workspace is to provide a fully reproducible example of the GATK Best Practices workflows for Data Pre-processing and Germline Short Variant Discovery. A scientific description of the workflow is available in [Gatk's Best Practices Document](https://software.broadinstitute.org/gatk/best-practices/workflow?id=11145).

Authorization Domain (optional) ⓘ
The cloned Workspace will automatically inherit the Authorization Domain from this Workspace. You may add Groups to the Authorization Domain, but you may not remove existing ones.
You are not a member of any groups. You must be a member of a group to set an Authorization Domain.

Cancel Clone Workspace #2

Find the workflow

1. Navigate to the Method Configurations tab

FireCloud Workspaces Data Library Method Repository ? crzyprlmky@gmail.com

WORKSPACE
help-gatk/five-dollar-genome-analysis-pipeline 🔔

Summary Data Analysis Notebooks BETA Method Configurations Monitor

✓ Complete

[Catalog Dataset...](#)

[Clone...](#)

Workspace Access

Access Level
Reader

Workspace Owner
bshifaw@broadinstitute.org

Authorization Domain
None

Created By
bshifaw@broadinstitute.org
February 14, 2018, 11:16 PM

▼ Tags
[gatk](#) [germline](#) [best-practices](#) [variant discovery](#) [pre-processing](#) [SNP](#) [INDEL](#)

▼ Description
GATK Best Practices for Germline SNPs and Indels as used at the Broad Institute
The purpose of this workspace is to provide a fully reproducible example of the GATK Best Practices workflows for Data Pre-processing and Germline Short Variant Discovery. A scientific description of the workflow is available in [Gatk's Best Practices Document](#).

The "SS Genome Analysis Pipeline" name refers to the cost of running the full pipeline (with all options turned to do the maximum amount of work) on a typical whole genome dataset, on the Google Cloud Platform, as explained on the Broad site in a [Gatk blog post](#).

Storage & Analysis

Google Bucket
[fc-ef9abb30-2a10-4229-b574-895e3ac9cf88](#)

Analysis Submissions
3 Submissions
• 3 Done

Select the pipeline

1. Select the **five-dollar-genome-analysis-pipeline**

FireCloud Workspaces Data Library Method Repository crzyprlmr@gmail.com

WORKSPACE
fcredits-boron-maroon-9691/five-dollar-genome-analysis-pipeline_MY_NAME

Summary Data Analysis Notebooks BETA Method Configurations Monitor

Columns Filter Import Configuration...

Name	Root Entity Type	Method Source	Method
five-dollar-genome-analysis-pipeline	sample	FireCloud	gatk/five-dollar-genome-analysis-pipeline Snapshot ID: 10

1 - 1 of 1 result < Prev 1 Next > 20 per page

BROAD INSTITUTE
© 2015-2018 Broad Institute | Privacy Policy | Terms of Service | User Guide | FireCloud Forum | Firecloud Status

Launch the analysis

1. Launch the analysis

FireCloud Workspaces Data Library Method Repository crzyprlmr@gmail.com

WORKSPACE
fcredits-boron-maroon-9691/five-dollar-genome-analysis-pipeline_MY_NAME

Summary Data Analysis Notebooks BETA Method Configurations Monitor

Edit Configuration Delete Publish...

Method Configuration Name
five-dollar-genome-analysis-pipeline

Referenced Method

Namespace: gatk Created: March 26, 2018, 12:17 PM
 Name: five-dollar-genome-analysis-pipeline Owners: bshifaw@broadinstitute.org
 Snapshot ID: 10 Synopsis: This WDL pipeline implements data pre-processing and initial variant calling
 Entity Type: Workflow Snapshot Comment: disk_size calculation updated
 Source: FireCloud

Documentation:
 Copyright Broad Institute, 2018 This WDL pipeline implements data pre-processing and initial variant calling (GVCF generation) according to the GATK Best Practices (June 2016) for germline SNP and Indel discovery in human whole-genome and exome sequencing data. Requirements/expectations: - Human whole-genome pair-end sequencing data in unmapped BAM (uBAM) format - One or more read groups, one per uBAM file, all belonging to a single sample (SM) - Input uBAM files must additionally comply with the following requirements: - filenames all have the same suffix (we use ".unmapped.bam") - files must pass validation by ValidateSamFile - reads are provided in query-sorted order - all reads must have an RG tag - GVCF output names must end in ".g.vcf.gz" - Reference genome must be Hg38 with ALT contigs Runtime parameters are optimized for Broad's Google Cloud Platform implementation. For program versions, see docker containers. LICENSING: This script is released under the WDL source code license (BSD-3) (see LICENSE in https://github.com/broadinstitute/wdl). Note however that the programs it calls may be subject to different licenses. Users are responsible for checking that they are authorized to run all programs before running this script. Please see the docker page at https://hub.docker.com/r/broadinstitute/genomes-in-the-cloud/ for detailed licensing information pertaining to the included programs.

WDL: [Expand](#)

Launch Analysis...

Select the sample

1. Select either sample_id to run the analysis on
2. Launch

Launch Analysis

Selected: None Estimated wait time: a few seconds
Workflows ahead of yours: 0
Queue status: 0 Queued; 1319 Active

Columns

participant (1 total) sample (2 total)

sample_id	flowcell_unmapp...	output_bqsr_rep...	output_cram	output_cram_ind...	output_cram_md5	output_vcf	output_vcf_index	participant
NA12878	gs://broad-public...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	NA12878
NA12878_small	gs://gatk-test-dat...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	gs://fc-ef9abb30...	NA12878

1 - 2 of 2 results 20 per page

Define Expression

Use Call Caching [Learn about call caching](#) [Cromwell Version: 32](#)

#2

Cancel Launch

Monitor the pipeline

1. Monitor the submitted job, well done!

FireCloud Workspaces Data Library Method Repository crzprpimnky@gmail.com

WORKSPACE
fccredits-boron-maroon-9691/five-dollar-genome-analysis-pipeline_MY_NAME

Summary Data Analysis Notebooks BETA Method Configurations Monitor

Submitted

⚠ Abort

Method Configuration

Namespace: gatk
Name: five-dollar-genome-analysis-pipeline

Submitted by

crzprpimnky@gmail.com
July 27, 2018, 8:45 PM (a few seconds ago)

Total Run Cost

Not Available

Submission Entity

Type: sample
Name: NA12878_small

Submission ID

[30c5a58d-2d4a-49b8-a977-1da1f794af6d](#)

Call Caching

Enabled

Workflows:

Columns

All (1) Queued (1) Launching (0) Submitted (0) Running (0) Aborting (0) Succeeded (0) Failed (0) Aborted (0)

Data Entity	Last Changed	Status	Messages	Workflow ID	Run Cost
NA12878_small (sample)	July 27, 2018, 8:45 PM (a few seconds ago)	Queued			n/a

1 - 1 of 1 result 20 per page

4.3.3 Upload data and run a custom method

Setup

Please download the materials for this section [FireCloud Files](#)

Create a workspace in FireCloud

1. Workspaces > Create a new workspace

- (a) **name:** hello_gatk_fc_YOUR_NAME
- (b) **billing project:** YOUR_PROJECT

Add workspace attributes

1. **Workspaces > Summary > Workspace attributes > Import Attributes**

- (a) data_bundle > FireCloud > workspaceAttributes.tsv
2. When it is uploaded, look at the workspace attributes section to see if the upload was successful

Set up data model

1. **Workspaces > Data > Import Metadata > Import from file**

- (a) **Upload in this order:**
 - i. data_bundle > FireCloud > participant.txt
 - ii. data_bundle > FireCloud > sample.txt
2. When it is uploaded, look at the two tables in the data tab that are filled in to see if it was successfully uploaded.

Put WDL on FireCloud

1. **Method Repository > Create New Method**

- (a) namespace: YOUR_NAME
- (b) name: hello_gatk_fc
- (c) **wdl: load from file**
 - i. This WDL calls HaplotypeCaller in GVCF mode, which takes a BAM input & outputs a GVCF file of variant likelihoods.
 - ii. The FireCloud version has a docker image specified among other runtime settings -- the memory and disk size of the machine we will request from Google's cloud, as well as the number of times we will try to run on a preemptible machine.
 - iii. Notice that you can type in the WDL field to edit if needed.
- (d) **documentation:** We won't be filling this out today, but in general documentation here is highly recommended, as it is helpful for others who may want to run your method.
- (e) Upload

Import configuration to workspace

1. **Method Repository > your method > Export to Workspace**

- (a) **Use Blank Configuration**
 - i. **Name:** hello_gatk_fc
 - ii. **Root Entity Type:** sample
 - iii. **Destination Workspace:** YOUR_PROJECT/hello_gatk_fc_YOUR_NAME

(b) Would you like to go to the edit page now? Yes

(c) _____

Note: If you get popup “Synchronize Access to Method” Grant Read Permission

Fill in method config

1. Workspace > Method Configurations > hello_gatk_fc

2. **Select the Edit Configuration button to fill it in. There are 3 types of inputs.**

(a) **In the data model**

i. You’ll find this value in your data tab. Since it is under the sample section, and your root entity type is sample, simply type `this.` and allow autocomplete to guide you.

ii. eg: `inputBam = this.inputBam`

(b) **In the workspace attributes**

i. You’ll find this value in your workspace attributes section under the summary tab. To find it, type in `workspace.` and let autocomplete guide you.

ii. eg: `refDict = workspace.refDict`

(c) **Hard-coded**

i. These are values which are not in your data model or workspace attributes. They are fixed numbers or strings that are typed in here. You can find the values for these inputs in the `inputs.json` in your data bundle (`data_bundle > hello_gatk > hello_gatk_fc.inputs.json`)

ii. eg: `disk_size = 10`

iii. eg: `java_opt = "-Xmx2G"`

3. Fill in the remaining inputs on your own/helping your neighbors.

4. Fill out the output. It won’t auto-complete, but we want to write it to the data model. The value should be `this.output_gvcf`

5. Save the configuration

Run

1. Refresh the page and check for the yellow refresh credentials banner **BEFORE** running. This isn’t typically an issue for users in a normal setting, but because in a workshop we start and stop a lot, the idle time can cause the credentials to time out. It will throw a Rawls error if you run that won’t pop up until after the job has been submitted and queued, which can be frustrating.

2. Method Config > Launch Analysis > Select sample > Launch

3. Watch & refresh from the monitor tab. Click the view link when it appears, and open the timing diagram to see what’s happening.

CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`