Comprehensive Guide to Deploying Cassandra on AWS

- Covers VPC, EC2, EBS, AMIs, concerns
- Covers Networking
- Covers instance storage vs. EBS
- Covers KMS encryption

Cassandra and AWS Support on AWS/EC2

- Covers VPC, EC2, EBS, AMIs, concerns
- Covers Networking
- Covers instance storage vs. EBS
- Covers KMS encryption
Cassandra and AWS Support on AWS/EC2

Cloudfurable Amazon Cassandra Basics | Support around Cassandra and Kafka running in EC2
Cassandra / Kafka Support in EC2/AWS

Company Overview

How we got our start
Different companies same challenges

- How to setup a Cluster across multiple AZs
- Where does enhanced networking fit it
- Should we use EBS or instance storage
- Monitoring and logging that can be actionable
- Integration with AWS services like CloudFormation, and CloudWatch.
- Best fit for images, VPC setup, peering, subnets, firewalls
Services we provide

- Cassandra Training
- Cassandra Consulting
- Setting up Cassandra in AWS/EC2
- AWS CloudFormations
- Subscription Support around Cassandra running in AWS/EC2
  - AWS CloudWatch monitoring
  - AWS CloudWatch logging
Cloudurable Cassandra AWS Support

AWS Review

Review of key Amazon Services and features
Advice and documents AWS Cassandra

- There is a lot of advice on how to configure a Cassandra cluster on AWS
- Not every configuration meets every use case
- Best way to know how to deploy Cassandra on AWS is to know the basics of AWS
- We start covering AWS (as it applies to Cassandra)
- Later we go into detail with AWS Cassandra specifics
AWS Key Concepts

- EC2 – compute services, virtual servers
  - EC2 instance a virtual server running in a VPC
- EBS – virtual disk drives
- VPC – software defined networks
  - Public Subnets – have InternetGateway
  - Private Subnets – no route to InternetGateway
AWS supports regions around the world and throughout the USA. A region is like a datacenter. Regions are independent of each other. You can place services in a region to be closer to your end consumer to lower latency and to improve reliability.

An Availability Zone (AZ) is isolated but multiple AZs live in a region. Placing your services and application in separate Availability Zones, protects you from outages. Each AZ in region has independent power, backup generators, UPS units, etc. AZs if possible exists in a separate location of a metropolitan area. AZs are redundantly connected together with fast connections that deliver low-latency using multiple tier-1 transit providers.

A VPC lives in a single region and a VPC subnet must live in a single AZ.
Cloudurable Cassandra AWS Support

EC2 Compute

EC2, EC2 instances, Instance types, networking speed
Amazon Elastic Compute Cloud (Amazon EC2)
Amazon EC2 is AWS primary web service that provides resizable compute capacity in the cloud.

EC2 Compute
Compute is computational power needed for your use case. Amazon EC2 allows add compute resources through its Web Service API. EC2 allows you to launch instances. An instance is a server and you can install whatever software you need for your service or web application: NGINX, Apache httpd, Cassandra, Kafka, etc. When you launch a virtual server, an instance in EC2 speak, you can use it as you like just like you would a server in your datacenter. You pay for the compute power that you use. There are different instance types with various ranges of CPU, RAM, IO, and networking power. You pay for compute resources by the hour. You can use more instances and you can reserve instances for longer periods of time for a price break.
EC2 Compute

Amazon Elastic compute cloud (EC2) and Characteristics of an EC2 Instance

Amazon EC2 is a web service that provides resizable compute capacity in the cloud.

- CPU
- Memory
- Storage
- GPU
- Enhanced Networking
EC2 Instance Types

- Defines the size/power of virtual server
- Many types of EC2 instances - Families of instance type
- Virtual CPUs (vCPUs)
  - vCPU is a hyperthread of an Intel Xeon core for M4, M3, C4, C3, R3, HS1, G2, I3, and D2.
- Memory RAM (size and type)
- Network performance

Instance Types
The instance type defines the size of the virtual instance. There are many types of EC2 instances with different levels of:

- Virtual CPUs (vCPUs)
- Memory RAM (size and type)
- Network performance

There are families of instance types. Amazon used its own way to measure compute power called ECU, but has since moved to the more industry standard vCPU. A vCPU is a hyperthread of an Intel Xeon core
for M4, M3, C4, C3, R3, HS1, G2, I2, and D2.
Families of types – Part 1

- **T2** - inexpensive and burst-able (good for less expensive and more sporadic workloads)
- **M4** - new generation of general purpose instances (added clustering and placement groups to M3)
- **C4** - compute optimized like M4 but less memory and more vCPUs (use this if you are not using all of your M4 memory)
- **P2** - GPU intensive applications (Machine learning)
- **G2** - graphics-intensive applications (server-side graphic workloads)

What are the two most likely of these families that you would use with Cassandra?

- T2 - inexpensive and burst-able (good for less expensive and more sporadic workloads)
- M4 - new generation of general purpose instances (added clustering and placement groups to M3)
- M3 - old generation of general purpose instances (don’t use this one, M4 is cheaper and better)
- C4 - compute optimized like M4 but less memory and more vCPUs (use this if you are not using all of your M4 memory)
- C3 - use C4 as C3 does not provide clustering and placement groups C3 is to M3 as C4 is to M4
- P2 - GPU intensive applications (Machine learning)
- G2 - graphics-intensive applications (server-side graphic workloads)
Families of types – Part 2

- **X1** - memory optimized for in-memory computing (SAP, HANA)
- **R3** - memory intensive databases and distributed caches (MongoDB)
- **I3** - High IOPS at lower cost, SSD instance storage (MongoDB, RDBMS)
- **D2** - High IO throughput and large disks at lower cost, magnetic instance storage (MapReduce, Kafka)

What are the two most likely of these families that you would use with Cassandra? Why?

- **M4** - new generation of general purpose instances (added clustering and placement groups to M3)
- **C4** - compute optimized like M4 but less memory and more vCPUs (use this if you are not using all of your M4 memory)
- **X1** - memory optimized for in-memory computing (SAP HANA)
- **R3** - memory intensive databases and distributed caches (MongoDB)
- **I2** - High IOPS at lower cost, SSD storage (MongoDB, Cassandra)
- **D2** - High IO throughput and large disks at lower cost, magnetic storage (MapReduce, Cassandra, Kafka)
AMIs exist under a unique id in one region.

To copy an AMI use the `aws ec2 copy-image` as follows:

```
aws ec2 copy-image --source-region us-west-2 --source-image-id ami-6db3310d --name CassandraClusterAMI
```
<table>
<thead>
<tr>
<th>Cloudurable Cassandra AWS Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon Elastic Block Storage</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual volumes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD</td>
</tr>
<tr>
<td>Magnetic</td>
</tr>
</tbody>
</table>
Getting the most bang for your buck with AWS Elastic Block Store (EBS)

Understanding what AWS/EC2 provides for provisioning on-demand storage is critical for DevOps. Companies waste tons by over provisioning AWS.

Amazon Elastic Block Store

Amazon Web Services (AWS) provides Amazon Elastic Block Store (Amazon EBS) for EC2 instance storage. EBS is the virtual hard drives and SSDs for your servers running in the cloud. Amazon EBS volumes are automatically replicated, and it is easy to take snapshots of volumes to back them up in a known state. The replication happens within an availability zone (AZ).
AWS EBS has lots of advantages like reliability, snapshotting, and resizing.
Amazon Elastic Block Store (Amazon EBS)

EC2 Instance

EBS Volume  EBS Volume  EBS Volume

You can attach multiple volumes to the same EC2 instance in the same availability zone.
EBS Volumes Types

There are many types of volumes. Different types have different performance characteristics. The trick is to pick the most cost-efficient for the workload of your service.

AWS provides four volume types. It provides two types of Hard Disk Drives (HDD), and two types of SSDs. Volumes differ in price and performance. An EC2 instance can have many volumes attached to it, just like a server can have many drives. A volume can only be attached to one EC2 instance at a time. If you wanted to share files between EC2 instances than you would use Amazon Elastic File System or S3.
Magnetic Volumes - HDD

- Magnetic volumes can’t be used as a boot volume.
- Lowest performance for random access
- Least cost per gigabyte
- Highest throughput (500 MB/s) for sequential access
- Magnetic volumes average 100 IOPS, but can burst to hundreds of IOPS.
- Good for services like Kafka which writes to a transaction log in long streams,
- Good for databases which use log structured storage or log structured merge tree
  - LevelDB, RocksDB, Cassandra

**Magnetic Volumes - Hard Disk Drives (HDD)**

Magnetic volumes have the lowest performance for random access. However, they have the least cost per gigabyte. But, they have the highest access for throughput (500 MB/s) for sequential access. Magnetic volumes average 100 IOPS, but can burst to hundreds of IOPS.

IOPS are Input/output operations per second (pronounced eye-ops). IOPS are used to characterize storage devices.

Services like Kafka which writes to a transaction log in long streams, and databases which use log structured storage or an approximate of that using some sort of log structured merge tree (examples LevelDB, RocksDB, Cassandra) might do well with HDD EBS - Magnetic volumes. Application that might employ streaming, or less operations per second but
larger writes could actually benefit from using HDDs throughput performance.
In general, magnetic volumes do best with sequential operations like:

- streaming workloads which require cost effective, fast, consistent I/O
- big data
- data warehouses
- log processing
- Databases which use structured merge tree
There are two types of HDD - Magnetic Volumes:

- **st1** - Throughput Optimized HDD
- **sc1** - Cold HDD and most cost effective

Which would be better for a Cassandra production system with low reads but large rows with frequent writes?
General-Purpose SSD (gp2)

General-purpose SSD (gp2) volumes are cost effective, and useful for many workloads. It is the minivan of EBS. Not sexy but works for a lot of applications, and is common. Performance of \textit{gp2} is three IOPS per gigabyte provisioned, but capped at 10,000 IOPS. The sizes range from 1 GB to 16 TB. Databases that use some form of BTrees (MongoDB, MySQL, etc.) can benefit from using SSD. But \textit{gp2} would be more geared to a lower volume database or one that has peak load times but long periods at rest where IOPS credits can accumulate.
can accumulate.
Under 1 TB these volumes burst to 3,000 IOPS for extended periods of time. For example, if you have a 250 GB volume you can expect a baseline of 750 IOPS. When those 750 IOPS are not used, they are accumulated as IOPS credits. Under heavy traffic, those IOPS credits will be used and this is how you can burst up to 3,000 IOPS. IOPS credits is like a savings account. You use this savings when you get hit hard by a user tornado. But as you are using it, the bank account is being withdrawn from.

- Development and test environments
- A server than does periodic batch or cron jobs
- Recommended for most workloads
- Can be used for boot volumes
- Low-latency interactive apps
- Medium-sized databases
Bursting

Credit Amazon Documentation for both images
Provisioned IO (io1)

- For I/O intensive workloads
- Most expensive EBS option
- IOPS up to 20,000 - you can purchase IOPs

Use Cases

- Mission critical business applications that require sustained IOPS performance
- Databases with large, high-volume workloads
- For developers bad at math

Provisioned IOPS SSD (io1)

Provisioned IOPS SSD volumes are for I/O-intensive workloads. These volumes are for random access I/O throughput. They are the most expensive Amazon EBS volume type per gigabyte. And, they provide the highest performance of random access of any Amazon EBS volume. With this volume type you can pick the number of IOPs (pay to play). The IOPs can be up to 20,000. These volumes are great for high-volume databases or just databases that need a constant level of performance. High volume databases that use some form of BTrees (MongoDB, MySQL, etc.) can benefit from using this SSD volume. The io1 IOPS can be ramped up.

Provisioned IOPS SSD volumes are more predictable (don’t have to store up IOPS like gp2), and for application with higher performance needs like:

- Mission critical business applications that require sustained IOPS performance
- Databases with large, high-volume workloads

Overcoming the performance problems by using Provisioned IOPS is expensive.

Some companies have employed RAID-0 striping using a 4-way stripe and used EnhanceIO to effectively increased throughput by over 50% with no more additional expense.

EnhanceIO

RAID-0 can be employed to increase size constraints of EBS and to increase throughput.
## EBS Type Review

<table>
<thead>
<tr>
<th>Solid-State Drives (SSD)</th>
<th>Hard disk Drives (HDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Volume Type</strong></td>
<td></td>
</tr>
<tr>
<td>General Purpose SSD (gp2)</td>
<td>Provisioned IOPS SSD (io1)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>highest-performance SSD volume designed for mission-critical applications</td>
</tr>
<tr>
<td><strong>Volume Size</strong></td>
<td>1 GiB - 16 TiB</td>
</tr>
<tr>
<td>Max. IOPS/Volume**</td>
<td>10,000</td>
</tr>
<tr>
<td>Max. Throughput/Volume†</td>
<td>160 MiB/s</td>
</tr>
<tr>
<td>Max. IOPS/Instance</td>
<td>65,000</td>
</tr>
<tr>
<td>Max. Throughput/Instance</td>
<td>1,250 MiB/s</td>
</tr>
<tr>
<td>Dominant Performance Attribute</td>
<td>IOPS</td>
</tr>
</tbody>
</table>

How can Cassandra use HDD and get 1,000 IOPs? 3 ways
3 * 200G = 600 IOPs
600 IOPs * 4 EBS VOLUMES = 2,400 IOPs
2,000 GB * 3 IOPs * EBS volumes = 24,000 IOPs
4,000,000,000,000 (TOTAL BYTES) / 1,000,000,000 (1 BILLION/GIG) * 3 IOPs per GIG * 10 EBS volumes = 120,000 IOPs

BUT.... 1 box can only have 65,000 IOPs!

65,000 IOPS.

“Attaching more than 40 volumes can cause boot failures. Note that this number includes the root volume, plus any attached instance store volumes and EBS volumes. If you experience boot problems on an instance with a large number of volumes, stop the instance, detach any volumes that are not essential to the boot process, and then reattach the volumes after the instance is running.”
20,000 IOPs per Volume = 80,000 but exceeds 65,000 so 65,000
EBS Volumes

Now Available – i3 Instances for Demanding, I/O Intensive Applications

On the first day of AWS re:Invent I published an EC2 Instance Update and promised to share additional information with you as soon as I had it.

Today I am happy to be able to let you know that we are making six sizes of our new i3 instances available in fifteen AWS regions.

Designed for I/O intensive workloads and equipped with superior NVMe I/O storage, these instances can deliver up to 7.5 million IOPs at a 1KB block and up to 16 GB/second of sequential disk throughput. This makes them a great fit for any workload that requires high throughput and low latency including relational databases, NoSQL databases, search engines, data warehouses, real-time analytics, and data-based caches. When compared to the i2 instances, i3 instances deliver storage that is less expensive and more dense, with the ability to deliver substantially more IOPS and more network bandwidth per CPU core.

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>vCPU Count</th>
<th>Memory</th>
<th>Instance Storage (NVMe SSD)</th>
<th>Price/Hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>i3.large</td>
<td>2</td>
<td>15.25 GB</td>
<td>0.475 TB</td>
<td>$0.15</td>
</tr>
<tr>
<td>i3.xlarge</td>
<td>8</td>
<td>61 GB</td>
<td>1.9 TB</td>
<td>$0.62</td>
</tr>
<tr>
<td>i3.4xlarge</td>
<td>16</td>
<td>122 GB</td>
<td>3.8 TB (2 disks)</td>
<td>$1.25</td>
</tr>
<tr>
<td>i3.8xlarge</td>
<td>32</td>
<td>244 GB</td>
<td>7.5 TB (4 disks)</td>
<td>$2.50</td>
</tr>
<tr>
<td>i3.16xlarge</td>
<td>64</td>
<td>488 GB</td>
<td>15.2 TB (8 disks)</td>
<td>$4.99</td>
</tr>
</tbody>
</table>

Max IOPs for random read?

825,000 IOPs
NOT EBS: Instance storage

- Don’t forget you don’t have to use EBS
- Instances storage is faster than EBS
- EC2 instance types with instance storage are expensive
- No server area network (SAN) or IO over network
EBS Optimized

- Newer EC2 instance types support EBS Optimized
- Higher throughput
- Less jiggle
- More reliable
- Don’t use C3 or M3 use C4 and M4
- Uses Optimized by default
- New Feature added in Feb 2017: Elastic Volumes!

EBS–optimized instances deliver dedicated bandwidth to Amazon EBS, with options between 500 Mbps and 12,000 Mbps, depending on the instance type you use. When attached to an EBS–optimized instance, General Purpose SSD (gp2) volumes are designed to deliver within 10% of their baseline and burst performance 99% of the time in a given year, and Provisioned IOPS SSD (io1) volumes are designed to deliver within 10% of their provisioned performance 99.9% of the time in a given year. Both Throughput Optimized HDD (st1) and Cold HDD (sc1) guarantee performance consistency of 90% of burst throughput 99% of the time in a given year. Non-compliant periods are approximately uniformly distributed, targeting 99% of expected total throughput each hour. For more information, see Amazon EBS Volume Types.”
d.html#enable-ebs-optimization
Performance testing and monitoring

The best way to make an educated guess to pick the right EBS is to know your tool. If you are deploying Kafka or Cassandra or MongoDB then you must understand how to configure the tool and EBS to get the most bang for the buck. When in doubt, test.

You can make educated guesses about which EBS will fit your application or service the best. However using Amazon CloudWatch and watching the IOPs and IO throughput while load testing or watching production workloads could be the quickest way to to pick the best EBS volume type and get the most bang for your buck. This can also help you decide whether or not to use RAID 0, HDDs (st1 or sc1), provisioned IOPS SSD (io1), SSD general purpose (gp2) or not. There is no point in overpaying, and you do not want a laggy service.
or application that do not meet their SLAs.
Understanding what AWS provides for backing up EBS volumes is an important concept for DevOps.

**Data safety with EBS - Backup/Recovery (Snapshots)**

Amazon EBS allows you to easily backup data by taking snapshots.

- Snapshots are point-in-time backups
- Snapshots provide incremental backups of your data
- Snapshots just saves the blocks that have changed
- Only changed blocks since last snapshot saved in new snapshot
- Only last snapshot needed to restore the volume

A Cassandra node goes down, and its EBS volume is corrupt and you have snapshots for this volume. Would it be faster to spin up an new instance with a volume created from the last snapshot or just let Cassandra repopulate the node?

Even though snapshots are saved incrementally, only the last snapshot is needed in order to restore the volume. You can
delete older snapshot, and still use the latest snapshot.
Taking EBS Snapshots

Snapshots are done with:

- AWS Management Console
- Scheduled snapshots
- AWS API - AWS CLI
- Snapshots backed by S3 but you can’t see them
- Snapshots are stored per region
- Use snapshots to create new EBS volumes
- Snapshots can be copied to other regions

EBS snapshots are backed by S3 in AWS-controlled storage. You can’t see these snapshots your account’s Amazon S3 buckets. It is hidden from you but backed by S3. You use the snapshot tools to manage snapshots not S3.

Snapshots are stored per region. You use snapshots to create new EBS volumes. Snapshots can be copied to other regions.

Using snapshots

Snapshots are used to create new EBS volumes. The volume is created and the data is transferred lazily into the EBS volume. Data accessed before the transfer is restored on request. If you need to increase the size of a volume, you create a snapshot and then recreate a larger volume from the snapshot. Then replace the original volume with the new volume from the snapshot.

Tag snapshots to help manage them later. Describing the original volume of the snapshot with the device name (/dev/sdd).

The AWS console can be used to take snapshots or the command line.

Restoring volumes from snapshots

Amazon EBS volumes persist beyond the EC2 instance lifecycle. Thus you can recover data of an instance that fails. Before the instance is terminated, the volume should be detached. Then the volume can attached as a data volume to another instance and then the data can be recovered.

Backing up root devices

To create a snapshot from a root devices EBS volume, should unmount the volume before taking the snapshot so the OS or application that have outstanding / cached blocks can flush them. To unmount the volume in Linux, use the following command:
umount -d /dev/sdc
Best Practices for Snapshots

- Test the process of recovering your instances from snapshots if the Amazon EBS volumes fail
- Use separate volumes for the operating system versus your data
- Make sure that the data persists after instance termination
- Don’t use instance store for database storage, unless you are using replication

You wrote a Chef or Ansible script to update the JDK and Cassandra. Should you perform a snapshot before you run this?
Cloudurable Cassandra AWS Support

VPC

Software defined networking
VPC: 1 public and two private subnets
Understanding what AWS provides for setting up private networks, security groups and more is important for anyone who calls themselves DevOps. AWS allows you to define a software defined network. You do this with Amazon Virtual Private Cloud (Amazon VPC). You can define subnets, ingress rules, security groups, NAT gateways, Internet gateways, and more.

A VPC is a virtual private cloud. You can create multiple Amazon VPCs within a region that spans multiple availability zones. A VPC is an isolated area to deploy Amazon EC2 instances. A VPC is associated with a CIDR block.

**Amazon VPC DHCP Option Set**
A VPC is associated with a DHCP Option Set.
Dynamic Host Configuration Protocol (DHCP) provides a standard for configuring TCP/IP networks.
DHCP Options allow you to configure DHCP per VPC as follows:
- domain name
- domain name server
- netbios-node-type
By default AWS creates and associates a DHCP option set for your Amazon VPC. The default DHCP option set uses domain-name-servers set to AmazonProvidedDNS (Amazon Domain Name System), and the domain-name set to the domain name for
your region
With CIDR block notation the /# denotes the size of the network or rather how many bits of the address will be used for the network. For example: 10.10.1.32/27 denotes a CIDR range (also known as CIDR block). It denotes that the first 27 bits of address is for the network (32 bits total). 32 - 27 leaves five bits for your servers. 00000-111111

First five addresses are reserved in a subnet, and the last address is reserved for broadcast. This leaves us 26 addresses for our servers. There are tools to help build CIDR based subnets.

From address 10.10.1.37 to 10.10.1.61. VPC address range may be as large as /16 (32-16 = 16 bits which allows for 65,536 available addresses) or as small as 16 addresses (/28 is 32 - 28 = 4 bits which is 16 available addresses). The addresses of two VPC should not overlap if you plan on adding VPC peering.
CIDR Block Diagram

Source Wikipedia
An Amazon VPC is made up of subnets, route tables, DHCP option sets, security groups, and Network ACLs.

An AWS VPC can also have Internet Gateways (IGWs), Virtual Private Gateways (VPGs), Elastic IP (EIP) addresses, Elastic Network Interfaces (ENIs), Endpoints, Peering, and NAT gateways.

A VPC has a router defined by its route tables (per subnet and default).
The VPC CIDR has 65536 hosts.
Avoid the last few and first five hosts in any subnet.

**Amazon CloudFormation**

CloudFormation allows developers, DevOps, and Ops create and manage a collection of related AWS resources. You can create and update items in a predictable fashion. You do this by creating CloudFormation templates which are written in JSON or YAML. Then you can submit the templates to be create stacks which can be updated.

We will use CloudFormation to cover the different components of the VPC in more detail.
VPC Subnets

- Part of an VPC’s IP address range
- Has CIDR blocks
- Associated with availability zones
- Can be public or private
- Private subnet has no route from the IGW (Internet Gateway)

Amazon VPC Subnets
A subnet is a part of an VPC’s IP address range. Just like a VPC you need to specify a CIDR blocks for a subnets. Subnets are associated with availability zones (independent power source and network). Subnets can be public or private. A private subnet is one that is not routable from the IGW.
CloudFormation VPC Subnet

```
"subnetPublic": {
  "Type": "AWS::EC2::Subnet",
  "Properties": {
    "CidrBlock": "10.0.0.0/24",
    "AvailabilityZone": "us-west-2a",
    "VpcId": {
      "Ref": "vpcMain"
    }
  }
},

"subnetPrivate": {
  "Type": "AWS::EC2::Subnet",
  "Properties": {
    "CidrBlock": "10.0.1.0/24",
    "AvailabilityZone": "us-west-2a",
    "VpcId": {
      "Ref": "vpcMain"
    }
  }
},

"subnetPrivate2": {
  "Type": "AWS::EC2::Subnet",
  "Properties": {
    "CidrBlock": "10.0.2.0/24",
    "AvailabilityZone": "us-west-2b",
    "VpcId": {
      "Ref": "vpcMain"
    }
  }
},
```

<table>
<thead>
<tr>
<th>CIDR Range</th>
<th>Netmask</th>
<th>Wildcard Bits</th>
<th>First IP</th>
<th>Last IP</th>
<th>Total Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/24</td>
<td>255.255.255.0</td>
<td>0.0.0.255</td>
<td>10.0.0.0</td>
<td>10.0.0.255</td>
<td>256</td>
</tr>
<tr>
<td>10.0.1.0/24</td>
<td>255.255.255.0</td>
<td>0.0.0.255</td>
<td>10.0.1.0</td>
<td>10.0.1.255</td>
<td>256</td>
</tr>
<tr>
<td>10.0.2.0/24</td>
<td>255.255.255.0</td>
<td>0.0.0.255</td>
<td>10.0.2.0</td>
<td>10.0.2.255</td>
<td>256</td>
</tr>
</tbody>
</table>
Internet Gateways
An Internet Gateway (IGW) enables traffic from the public Internet to your VPC. Subnets that have route tables that target the IGW are public subnets. The IGW does network address translation from public IPs of EC2 instances to their private IP for incoming traffic. When an EC2 instance send IP traffic from a public subnet, the IGW acts as the NAT for the public subnet, and translates the reply address to the EC2 instance’s public IP (EIP). The IGW keep track of the mappings of EC2 instances private IP address and their public IP address. AWS ensures that the IGW is highly available and handles the horizontal scale,
redundancy as needed.
We are covering route tables next.
Subnet Route Tables

- contain set of ingress and egress rules (aka routes)
- rules are applied to subnet
- connect subnets within a VPC so they can communicate
- routes direct network traffic
- routes are specified by CIDR and a target
- most specific route that matches traffic determines traffic route
- if subnet has route to the InternetGateway then public
- Each subnet associated with a route table (default route table)

Amazon VPC Subnet Route Tables

Route tables contain a set of ingress and egress rules called routes. These rules are applied to the subnet. The routes direct network traffic. The route tables connect subnets within a VPC so they can communicate. Routes are specified by CIDR and a target.

The most specific route that matches the traffic determines how to route the traffic. Route tables can specify which subnets are public and which subnets are private (if the subnet does or does not have a route to the InternetGateway). Each subnet is always associated with a route table which dictates routes for that subnet. If a route table for a subnet is not specified then that subnets uses the main route table (which is associated with the VPC).
The above shows the Amazon CloudFormation from the public subnet defined earlier to the Internet Gateway.
Amazon VPGs, CGWs and VPNs
Amazon allows VPCs to be connected to your existing data center to allow AWS to augment your existing IT infrastructure. You can connect your existing datacenter to an Amazon VPC using VPG (Virtual Private Gateways) and CGW (Customer Gateways).

Think of the VGW like the IGW but it sends traffic to your corporate network instead of the public Internet. VPGs connect to your companies Virtual Private Network (VPN) connector. The VPG is the Amazon side of the VPN connection. The CGW is the customer side of the VPN connector. CGWs are processes running on a server or network device.

You connect a VPG and a CGW with a VPN tunnel, which allows traffic between your corporate network and your Amazon VPCs. The VPN connection uses the IPSec (Internet Security Protocol) tunnels for higher availability to the AWS VPC.

You can setup the VPN connection to use dynamic routing if the CGW supports it (via Border Gateway Protocol). If your CGW does not support dynamic routing, use static routes to decide which traffic is meant for the VPC. Routes are propagated to the VPC to allow traffic back to your corporate network via the VGW.
The above shows that we are covering Elastic IPs and NatGateways next.
Amazon Elastic IP (EIP)

AWS has a pool of public IP addresses available to rent per region. These public IP addresses are called Elastic IP Addresses (EIPs). You check out an EIP like a library book. As long as you have the EIP checked out, no one else can use it. You can keep the EIP as long as you want but you pay for it. Unused EIPs are more expensive than EIPs that you are using with an EC2 instance. You can assign an EIP to an instance (and only one). You could spin up a new upgraded version of the instance from a snapshot, and the reassign the EIP to the new upgraded instance.

EIPs allow using a set of fixed public IP addresses that can be reassigned to underlying infrastructure which could change over time. EIPs are allocated in a VPC, but can be
moved to another VPC in the same region. EIPs can be assigned to resources like EC2 instances.
Amazon NAT Gateways

Amazon EC2 instances launched in a private subnet cannot access the Internet unless there is a NAT. A NAT is a network address translator. Even if you wanted to update your instances with `yum install foo`, you could not do it because instance by default have no route to the public Internet. NAT gateways which are similar to IGW but unlike IGWs they do not allow incoming traffic, but rather only allow responses to outgoing traffic from your Amazon EC2 instances.

NAT gateways are simple to manage and highly available. A VPC subnet lives in a single Availability Zone (AZ). To maximize failover you will want to deploy a NAT gateway per AZ.

To allow Amazon EC2 instances within a private subnet to access Internet resources through the IGW using a NAT gateway, you must do the following:

- Set up the route table by connecting the private subnet to direct Internet traffic to the NAT gateway
- Associate the NAT gateway with an EIP (covered shortly - elastic IP)

They added an egress only gateway which is like a NAT gateway but only works with IPv6.
CloudFormation for NAT GW

```
"natGateway": {
  "Type": "AWS::EC2::NatGateway",
  "Properties": {
    "AllocationId": {
      "Fn::GetAtt": [
        "eipForNatGateway",
        "AllocationId"
      ]
    },
    "SubnetId": {
      "Ref": "subnetPublic"
    }
  }
},

"eipForNatGateway": {
  "Type": "AWS::EC2::EIP",
  "Properties": {
    "Domain": "vpc"
  }
},

"routeNatGatewayPrivate": {
  "Type": "AWS::EC2::Route",
  "Properties": {
    "DestinationCidrBlock": "0.0.0.0/0",
    "NatGatewayId": {
      "Ref": "natGateway"
    },
    "RouteTableId": {
      "Ref": "routeTablePrivate"
    }
  }
},
```
Amazon Enhanced networking: Placement groups and networking speed

Instance types m4, c4, p2, g2, r3, g2, x1, i2 and d2 support *placement groups* which are essential for server to server performance which is important for clustering.

To achieve maximum throughput, placement groups must be placed in the same AZ – 10Gbits.

Amazon EC2 instances can achieve speeds of up to 10 Gbits if both instances are in the same placement group and in the same AZ.

**Why would this be important for Cassandra? Other systems?**
Elastic Network Interface ENI

- ENI is a virtual network interface - *network interface* in AWS speak
- Can *attach* to EC2 instance in a VPC - *detach* an ENI and attach to another EC2 instance
- Attributes: description, primary private IPv4 address, multiple secondary private IPv4 addresses, EIP per private address, public IPv4 address, multiple IPv6 addresses, multiple security groups (at least one), MAC address, source/destination check flag
- Keeps its attributes no matter which EC2 instance it is attached to
- If an underlying instance fails, the IP address (MAC, public IP, EIPs, etc.) are preserved
- Makes EC2 instances *replaceable* - low-budget, high-available solutions

Amazon Elastic Network Interface (ENIs)

An ENI is an Elastic Network Interfaces. An ENI is often just called a network interface in AWS speak. An ENI is a virtual network interface that you can attach to an instance in a VPC. You can also detach an ENI and attach to another EC2 instance. ENIs don’t work with EC2 classic (no VPC EC2).

An ENI can have the following properties:

- description
- primary private IPv4 address
- potentially multiple secondary private IPv4 addresses
- EIP per private address
- public IPv4 address
- multiple IPv6 addresses
- multiple security groups (at least one)
- MAC address
- source/destination check flag

Again, the what makes these ENIs elastic is that you can create an ENI, attach it to an EC2 instance, detach it from an EC2 instance, and attach it to another. The ENI keeps its properties no matter which instance it is attached to.

If an underlying instance fails, the IP address (MAC, public IP, EIPs, etc.) are preserved by attaching the ENI to a new replacement EC2 instance. ENIs can be
used to create low-budget, high-available solutions.
We are covering Amazon Security Groups next.
Security Groups (SG)

- **stateful firewall** - controls inbound and outbound network traffic to EC2 instances and AWS resources
- Stateful means an Amazon instance (or resource) is allowed to respond to an inbound traffic with outbound traffic
- EC2 instances have to be associated with a security group
- Rules are only *allow* rules
- Rules consist of the following attributes:
  - Source (CIDR or SG id)
  - Protocol (TCP, ICMP, UDP, HTTP, HTTPS, SSH, etc.)
  - Port range (8000-8080)

**Amazon VPC Security Groups**

A security group (SG) is a stateful firewall that controls inbound and outbound network traffic to EC2 instances and AWS resources like Elastic Load Balancers. Security groups being stateful means an Amazon instance (or resource) is allowed to respond to an inbound traffic with outbound traffic. AWS EC2 instances have to be associated with a security group if not specified then it is associated with the default security group for the VPC. AWS EC2 instances can be associated with security groups after they are already running. Each VPC can have up to 500 security groups.

Rules are only *allow* rules. Rules consist of the following attributes:
- Source (CIDR or SG id)
- Protocol (TCP, ICMP, UDP, HTTP, HTTPS, SSH, etc.)
- Port range (8000-8080)

Security groups specify up to fifty inbound and 50 outbound rules using CIDRs or other security groups. AWS will evaluate every rule
before deciding to permit traffic.
Source can be a CIDR or other security groups in the same VPC.

Notice the SG is tied to VpcMain defined before.

Note that 0.0.0.0/0 equates to all public traffic.
The above allows all traffic from the VPC’s CIDR to access this box.
-1 means all ports.

**Ingress in incoming traffic.**
**Egress is outgoing traffic.**
This is taken from an example that uses Cassandra running in multiple regions. The Cassandra EC2 instances are running in a public subnet.
We are covering Amazon Network ACL Control List (NACL) next.
Amazon Network Access Control List (NACL)

- Stateless firewall
- Provides a number ordered list of rules
- Lowest number rule evaluated first
- First rule that allows or denies wins
- Has both *allow* rules and *deny* rules
- Return traffic must be allowed (stateless)
- Applies to the whole subnet

How does this compare to Security Groups?

**Amazon Network Access Control List (NACL)**

Network ACL (NACL) is a stateless layer of security. NACLs act as a stateless firewall. NACLs provide a number ordered list of rules. The lowest number rule is evaluated first. First rule that allows or denies wins. NACLs support both *allow* rules and *deny* rules. Return traffic must be allowed (stateless). NACL applies to the whole subnet.
CloudFormation for NACL

```
    "networkACL": {
        "Type": "AWS::EC2::NetworkAcl",
        "Properties": {
            "VpcId": {
                "Ref": "vpcMain"
            }
        }
    },

    "aclEntryAllowAllEgress": {
        "Type": "AWS::EC2::NetworkAclEntry",
        "Properties": {
            "CidrBlock": "0.0.0.0/0",
            "Egress": "true",
            "Protocol": "-1",
            "RuleAction": "allow",
            "RuleNumber": "100",
            "NetworkAclId": {
                "Ref": "networkACL"
            }
        }
    },

    "aclEntryAllowAllIngress": {
        "Type": "AWS::EC2::NetworkAclEntry",
        "Properties": {
            "CidrBlock": "0.0.0.0/0",
            "Protocol": "-1",
            "RuleAction": "allow",
            "RuleNumber": "100",
            "NetworkAclId": {
                "Ref": "networkACL"
            }
        }
    }
```
VPC Peering

- Networking connection between two VPCs
- Enables routing traffic between private IP addresses
- As if VPCs are on the same network
- Both VPC must be in the same AWS Region
- CIDR addresses of VPCs can’t have conflict/overlaps
<table>
<thead>
<tr>
<th>Cloudurable Cassandra AWS Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related important AWS Concepts</strong></td>
</tr>
<tr>
<td>Important AWS concepts helpful for DevOps of clustered software</td>
</tr>
</tbody>
</table>
AWS Important Concepts

- **Auto Scaling**
  - used scale Amazon EC2 capacity up or down automatically
  - autoscale a group of instances based on workload
  - used to recover when instances go down by automatically spinning up an instance to take its place

- **Amazon Route 53**
  - DNS as a service. Route 53 is highly available and scalable
  - Easily assign DNS names instead of configuring with public IP addresses (internal and external)

- **Amazon CloudFormation**
  - allows developers, DevOps, and Ops create and manage a collection of related AWS resources

Could you use Route53 instead ENI for seed servers?

**Auto Scaling**
Auto Scaling is used scale Amazon EC2 capacity up or down automatically. You can autoscale a group of instances based on workload. It can be used to recover when instances go down by automatically spinning up an instance to take its place. Auto Scaling groups can span multiple AZs.

**Amazon Route 53**
Amazon Route 53 is a DNS as a service. Route 53 is highly available and scalable. You can use easily assign resources DNS names instead of configuring with public IP addresses.

**Amazon CloudFormation**
CloudFormation allows developers, DevOps, and Ops create and manage a collection of related AWS resources. You can create and update items in a predictable fashion. You do this by creating CloudFormation templates which are written in JSON or YAML. Then you can submit the templates to be create stacks which can be
updated.
Amazon makes it easy to create immutable infrastructure with its AWS command line tools and CloudFormation.

Get into the habit of using CloudFormation and AWS command line instead of the console to launch instances.
IAM
AWS Identity and Access Management (IAM) enables secure control access to AWS Cloud services and resources for their users. IAM defines, users, roles, and allows you to apply this to EC2 instances as well as users or groups of users (example in notes).

KMS
AWS Key Management Service (KMS) allows you to create and control the encryption keys. KMS uses Hardware Security Modules (HSMs) to protect the security of your keys. KMS can be used to encrypt Amazon EBS volumes, Amazon S3 buckets and other services. KMS can be used for compliance encryption operations for SOC1, SOC2, SOC 3, PCI DSS Level, ISO 27017/20018, and for FIPS 140-2. KMS also provides an REST API to encrypt data on an application basis.

AWS Certificate Manager
AWS Certificate Manager removes the time-consuming manual process of purchasing, uploading, and renewing SSL/ TLS certificates. AWS Certificate Manager allows provision, manage, and deploy Secure Sockets Layer/ Transport Layer Security (SSL/ TLS) certificates for use with AWS Cloud services like ELB or CloudFront (an Amazon CDN). No longer do you have to purchase, upload and manually update/renew SSL/TLS certificates for the ELB or CDN.

Amazon CloudFront
Amazon’s CDN to put resources closer to end users of your applications and services.

Amazon S3
Amazon Simple Storage Service to store your backups and big data. Good for backups.
Amazon CloudWatch

- Monitoring service it uses for its AWS Cloud resources and services
- Can be used for your services and applications
- Track key performance indicators (KPIs) and metrics
- Log aggregation, and can easily create alarms.
- Trigger AWS Lambda functions based on limits of an KPI or how often an item shows up in log stream in a give period of time
- Provides system-wide visibility into resource utilization, and operational health
- Not passive – Action oriented
- Integration with the entire Amazon ecosystem **integration!
- Actionable: Triggers and events to keep everything running smoothly

**Amazon CloudWatch**

Amazon CloudWatch is a monitoring service it uses for its AWS Cloud resources and services. However you can use CloudWatch for your services and applications.

CloudWatch can track key performance indicators (KPIs) and metrics, allow log aggregation, and can easily create alarms. You can even trigger AWS Lambda functions based on limits of an KPI or how often an item shows up in log stream in a give period of time.

Amazon CloudWatch provides system-wide visibility into resource utilization, and operational health. Unlike many monitoring systems, CloudWatch integration with the entire Amazon ecosystem so you not only have insights into your systems but you can react to triggers and events to keep everything running smoothly.
AWS Guidelines

Details on how to configure Cassandra in EC2
Cassandra and AWS Support on AWS/EC2

Cloudurable Amazon Cassandra Guidelines

Support around Cassandra and Kafka running in EC2
Cassandra / Kafka Support in EC2/AWS

Cassandra on AWS

Best Practices!
Apache Cassandra is extensively deployed in AWS. An estimated 60,000 AWS customers also use Cassandra. Over 1/3 of Cassandra deployments are on AWS. More AWS users use Cassandra than DynamoDB. AWS recently published guide: Deploying Cassandra on AWS in 2016.
<table>
<thead>
<tr>
<th>Cassandra / Kafka Support in EC2/AWS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cassandra AWS Overview</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
What is Cassandra?

- Linearly scalable, open source NoSQL database
- Uses log-structured merge-tree
- Supports high-throughput writes
- continuous availability, with operational simplicity
- master-less peer-to-peer distributed clustered store
- each node knows about cluster network topology via gossip

Cassandra is a linear scalable, open source NoSQL database. Cassandra uses log-structured merge-tree, which makes Cassandra one of the best options for high-throughput writes. Cassandra delivers continuous availability, with operational simplicity. Unlike many other NoSQL solutions, Cassandra is a master-less peer-to-peer distributed clustered store. Each node knows about the cluster network topology via the gossip protocol.
A Cassandra node is one server in a Cassandra cluster. Cassandra nodes store partitions of data according. Cassandra nodes deploy into Cassandra clusters, the largest unit of deployment. In AWS, it is normal for a Cassandra cluster to span Availability Zones and AWS regions to improve disaster recovery and speed client throughput.

Cassandra clusters consist of racks and data centers. Nodes get deploy as members racks which get deployed into data centers. In Amazon, Availability Zones (AWS AZ) equate to Cassandra racks, while Amazon regions equate to Cassandra data centers.

Cassandra nodes deploy to racks (AWS AZs). The number of racks/AZs should equate to a multiple of the replication level. Nodes on a rack or in an AZ have fast connectivity. AZs, in the same region, have low-latency links which benefit replication and data consistency checks.
A commit log is a transaction log on every node in the cluster. All Cassandra write operations are written to the commit log first. Commit logs are written sequentially-append only. Only during Cassandra node recovery, Cassandra reads the commit log. Cassandra replays commit logs to perform Cassandra node recovery. Commit logs are either flushed periodically by the OS or written in batches. Care must be taken to keep the commit log on a different EBS volume if using HDD.

A memtable is an in-memory version of an SStable. The memtable is a write-back cache of data rows that is looked up by key. One memtable links to one Cassandra table. Memtables are flushed to disk when the node reaches global memory thresholds, full commit log event, or Cassandra table level interval arrived event.

In EC2, Cassandra nodes must run on EC2 instances that have enough memory to support Cassandra caches, and memtables while leaving enough room for Linux OS buffers for TCP/IP stack. Cassandra also has key caches and row caches. Use cases that have high read to write ratios will benefit from large caches and EC2 instances that enough system memory (DRAM).
AWS Cassandra: SSTable, Keyspace

- SSTable and Commit log should be stored on separate EBS volume (especially if not SSD)
- SSTable disk representation of Memtable
- Type of compaction can range between 20% and 50% overhead
  - Account for this when allocating EBS volumes
- Keyspaces dictate replication factors, more can mean more IO (IOPs and network bandwidth needs)

An SSTable is the disk representation of a memtable. SSTable is a write-only data structure. When a memtables flush, they are written key sorted, sequentially to form an SSTable. Later SSTable is compacted by being merge-sorted and rewritten as a new larger SSTable. Type of compaction of SSTables can take between 20% and 50% overhead disk space. This space has to be accounted for when allocating EBS volumes.

A keyspace is like a database schema. Keyspaces have many tables and define replication strategy, replication factor, and rules. The more replication, the more IO needs which impact IO costs (more IOPs and more network bandwidth needs). A Cassandra table is like a SQL table except it can contain complex columns (maps, sets, lists). Cassandra tables are collection of ordered columns fetched by row. A table row key is known as the partition key. The row key determines the data distribution across a cluster.
AWS Embrace Change!

- AWS rules keep changing
- Faster, more reliable, more competition
- Prices keep changing (dropping)
- AWS knowledge of EC2, EBS, etc. three or five years ago is ancient now

🌟 What AWS or EC2 rule has changed since you started working with AWS?
<table>
<thead>
<tr>
<th>AWS EC2 Instance Types for Cassandra</th>
<th>AWS EC2 Instance types to consider for Cassandra</th>
</tr>
</thead>
</table>

Cassandra / Kafka Support in EC2/AWS

AWS Cassandra Storage Req.
## Compute Optimized

**C4**

C4 instances are the latest generation of Compute-optimized instances, featuring the highest performing processors and the lowest price/compute performance in EC2.

**Features:**
- High frequency Intel Xeon E5-2666 v3 (Haswell) processors optimized specifically for EC2
- EBS-optimized by default and at no additional cost
- Ability to control processor C-state and P-state configuration on the c4.8xlarge instance type
- Support for Enhanced Networking and Clustering

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
<th>Storage</th>
<th>Dedicated EBS Bandwidth (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>3.75</td>
<td>EBS-Only</td>
<td>500</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>7.5</td>
<td>EBS-Only</td>
<td>750</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>15</td>
<td>EBS-Only</td>
<td>1,000</td>
</tr>
<tr>
<td>c4.4xlarge</td>
<td>16</td>
<td>30</td>
<td>EBS-Only</td>
<td>2,000</td>
</tr>
<tr>
<td>c4.8xlarge</td>
<td>36</td>
<td>60</td>
<td>EBS-Only</td>
<td>4,000</td>
</tr>
</tbody>
</table>

High CPU to memory ratio
M4 instances are the latest generation of General Purpose Instances. This family provides a balance of compute, memory, and network resources, and it is a good choice for many applications.

Features:

- 2.3 GHz Intel Xeon® E5-2686 v4 (Broadwell) processors or 2.4 GHz Intel Xeon® E5-2676 v3 (Haswell) processors
- EBS-optimized by default at no additional cost
- Support for Enhanced Networking
- Balance of compute, memory, and network resources

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
<th>SSD Storage (GiB)</th>
<th>Dedicated EBS Bandwidth (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m4.large</td>
<td>2</td>
<td>8</td>
<td>EBS-only</td>
<td>450</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>16</td>
<td>EBS-only</td>
<td>750</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>32</td>
<td>EBS-only</td>
<td>1,000</td>
</tr>
<tr>
<td>m4.4xlarge</td>
<td>16</td>
<td>64</td>
<td>EBS-only</td>
<td>2,000</td>
</tr>
<tr>
<td>m4.10xlarge</td>
<td>40</td>
<td>150</td>
<td>EBS-only</td>
<td>4,000</td>
</tr>
<tr>
<td>m4.16xlarge</td>
<td>64</td>
<td>256</td>
<td>EBS-only</td>
<td>10,000</td>
</tr>
</tbody>
</table>

If in doubt, start with M4
### Storage Optimized

**I3 – High I/O Instances**

This family includes the High Storage Instances that provide Non-Volatile Memory Express (NVMe) SSD backed instance storage optimized for low latency, very high random I/O performance, high sequential read throughput and provide high IDPS at a low cost.

**Features:**
- High Frequency Intel Xeon E5-2686 v4 (Broadwell) Processors with base frequency of 2.3 GHz
- NVMe SSD Storage
- Support for TRIM
- Support for Enhanced Networking
- High Random I/O performance and High Sequential Read throughput

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
<th>Networking Performance</th>
<th>Storage (TB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i3.large</td>
<td>2</td>
<td>15.25</td>
<td>Up to 10 Gigabit</td>
<td>1 x 0.475</td>
</tr>
<tr>
<td>i3.xlarge</td>
<td>4</td>
<td>30.5</td>
<td>Up to 10 Gigabit</td>
<td>1 x 0.95</td>
</tr>
<tr>
<td>i3.2xlarge</td>
<td>8</td>
<td>61</td>
<td>Up to 10 Gigabit</td>
<td>1 x 1.9</td>
</tr>
<tr>
<td>i3.4xlarge</td>
<td>16</td>
<td>122</td>
<td>Up to 10 Gigabit</td>
<td>2 x 1.9</td>
</tr>
<tr>
<td>i3.8xlarge</td>
<td>32</td>
<td>244</td>
<td>10 Gigabit</td>
<td>4 x 1.9</td>
</tr>
</tbody>
</table>

### D2 – Dense-storage Instances

D2 instances feature up to 48 TB of HDD-based local storage, deliver high disk throughput, and offer the lowest price per disk throughput performance on Amazon EC2.

**Features:**
- High-frequency Intel Xeon E5-2676v3 (Haswell) processors
- HDD storage
- Consistent high performance at launch time
- High disk throughput
- Support for Amazon EC2 Enhanced Networking

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
<th>Storage (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d2.xlarge</td>
<td>4</td>
<td>30.5</td>
<td>3 x 2000 HDD</td>
</tr>
<tr>
<td>d2.2xlarge</td>
<td>8</td>
<td>61</td>
<td>6 x 2000 HDD</td>
</tr>
<tr>
<td>d2.4xlarge</td>
<td>16</td>
<td>122</td>
<td>12 x 2000 HDD</td>
</tr>
<tr>
<td>d2.8xlarge</td>
<td>36</td>
<td>244</td>
<td>24 x 2000 HDD</td>
</tr>
</tbody>
</table>

**Use Cases**
Massively Parallel Processing (MPP) data warehousing, MapReduce and Hadoop distributed computing, distributed file systems, network file systems, log or data-processing applications

Could be used in an analytics Cassandra DC that is connected to another DC that uses SSD for frequent small reads.
Starts at 6TB per node. IoT device monitoring.
Looks like the pricing for the storage optimized is closer than when that benchmark was done with I2, and the I3 specs are more impressive.

We won’t cover C4 vs. M4 yet.

For now C4, is like M4 with about ½ the system memory (DRAM).
| Cassandra / Kafka Support in EC2/AWS | Cassandra AWS Storage Guidelines | Cassandra storage requirements for Amazon EC2 and EBS |
Cassandra AWS - Storage Requirements

- Cassandra does lots of sequential disk IO (commit logs, SSTables)
- Cassandra writes large streams of data to commit logs SSTable, index files and bloom filter files
- Put commit log on separate disk (EBS volume) than SSTables
  - Maybe ok if using SSD
  - Never if HDD, always separate volume if HDD

Cassandra AWS Storage Requirements

Cassandra does a lot sequential disk IO for the commit log and writing out SSTable. You still need random I/O for read operations. The more read operations that are cache missing, the more your EBS volumes need IOPS.

Cassandra writes to four areas
- commit logs
- SSTable
- an index file
- a bloom filter

It makes sense if possible to have commit logs on a separate disk if using magnetic disks. SSTables are written to in streams but are read from using random access if data is not found in cache. SSTables can benefit from SSD drives due to random access.
Consider EC2 Instance Storage

- AWS provides EC2 instance local storage called instance storage
- Available with some EC2 instance types
- Instance storage unlike EBS does not go over a SAN or Intranet – less worry from traffic congestion, noisy neighbors
- Instance storage uses local hardware bus – can be as fast as real disks on real server
- More Expensive! - Less flexible
- More reliable than EBS ** see notes (EBS bad reputation)
- Historically the only real option for running Cassandra nodes in EC2

**Consider EC2 instance store instead of EBS**

AWS provides *local storage* called *instance storage* which is not available with all *EC2 instance types*, and *Elastic Block Store* (*EBS*). *Instance storage* does not have to go over a SAN or Intranet, instead it uses the local hardware bus. Instance storage is right there on the server you are renting. The downside of *EC2 instance storage* is the expense, and it is not as flexible as EBS. Due to historic problems with EBS, it used to be the only real option for running Cassandra in AWS. EBS has a reputation for degrading performance over time. Some of this has likely been fixed with *enhanced EBS*, but instance storage is more reliable.
Using EBS with Cassandra did not work very well in the past, and you had to use more expensive EC2 instances with *instance storage*. Until recently (2015) using Cassandra and AWS EBS was not a good idea. The latest generation of *EBS-optimized* instances offer a good mix of performance and improved reliability. EBS volumes are usually the best pick for price for performance. If in doubt start with EBS-optimized instances. EBS has nice features like snapshots, and redundancy that make it preferred if performance is close or horizontal scale out is an option. Also with *EBS elastic volumes, provisioned IO* and *enhanced EBS*, it would be hard not to pick EBS. It is just a lot more flexible, and less expensive.
EBS-Optimized Instances

- C4, M4, and D2 instances use EBS-Optimized by default
- Delivers up to between 500 and 4K Mbps throughput
- Dedicated connection minimizes contention between Amazon EBS I/O and other traffic from your EC2 instance
- Standard (10K IOPs) and Provisioned IOPS (20K IOPs) Amazon EBS volumes
- 65K IOPs max per EC2 instance
- Provisioned IOPS volumes (expensive) can achieve single digit millisecond latencies and are designed to deliver within 10% of the provisioned IOPS performance 99.9% of the time
- Standard IOPs supported is a function of EBS volume size

From AWS docs:

“EBS-optimized Instances
For an additional, low, hourly fee, customers can launch selected Amazon EC2 instances types as EBS-optimized instances. For C4, M4, P2, and D2 instances, this feature is enabled by default at no additional cost. EBS-optimized instances enable EC2 instances to fully use the IOPS provisioned on an EBS volume. EBS-optimized instances deliver dedicated throughput between Amazon EC2 and Amazon EBS, with options between 500 and 4,000 Megabits per second (Mbps) depending on the instance type used. The dedicated throughput minimizes contention between Amazon EBS I/O and other traffic from your EC2 instance, providing the best performance for your EBS volumes. EBS-optimized instances are designed for use with both Standard and Provisioned IOPS Amazon EBS volumes. When attached to EBS-optimized instances, Provisioned IOPS volumes can achieve single digit millisecond latencies and are designed to deliver within 10% of the provisioned IOPS performance 99.9% of the time. We recommend using Provisioned IOPS volumes with EBS-optimized instances or instances that support cluster networking for applications with high storage I/O requirements.”
https://aws.amazon.com/ec2/instance-types/
<table>
<thead>
<tr>
<th></th>
<th>Solid State Drives (SSD)</th>
<th>Hand Disk Drives (HDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Volume Type</strong></td>
<td>EBS Provisioned IOPS (ss1)</td>
<td>EBS General Purpose SSD (gp2)</td>
</tr>
<tr>
<td><strong>Short Description</strong></td>
<td>Highest performance SSD volume designed for latency-sensitive transactional workloads</td>
<td>General Purpose SSD volume that balances price performance for a wide variety of transactional workloads</td>
</tr>
<tr>
<td><strong>Use Cases</strong></td>
<td>VO-intensive NoSQL and relational databases</td>
<td>Boot volumes, low-latency interactive apps, dev &amp; test</td>
</tr>
<tr>
<td><strong>API Name</strong></td>
<td>ss1</td>
<td>gp2</td>
</tr>
<tr>
<td><strong>Volume Size</strong></td>
<td>4 GB - 16 TB</td>
<td>1 GB - 16 TB</td>
</tr>
<tr>
<td><strong>Max IOPS/Volume</strong></td>
<td>20,000</td>
<td>10,000</td>
</tr>
<tr>
<td><strong>Max Throughput/Volume</strong></td>
<td>320 MB/s</td>
<td>100 MB/s</td>
</tr>
<tr>
<td><strong>Max IOPS/Instance</strong></td>
<td>65,000</td>
<td>65,000</td>
</tr>
<tr>
<td><strong>Max Throughput/Instance</strong></td>
<td>1,250 MB/s</td>
<td>1,250 MB/s</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>$0.125/GB-month</td>
<td>$0.10/GB-month</td>
</tr>
<tr>
<td><strong>Dominant Performance Attribute</strong></td>
<td>IOPS</td>
<td>IOPS</td>
</tr>
</tbody>
</table>
Cassandra EC2 instance good picks

- M4 (or C4) and I3 family mainly used
- I3 came out end of last year (still rolling out in some regions)
- I3 has instance storage
- I3 super fast IO, fairly low cost
- Benchmark of M4 against I2 (previous generation)
  - I2 was 8x perf of M4 for tiny read/writes
  - About the same for medium size read/writes
- Consider D2 for large storage needs and high throughput (less expensive) uses HDD instance storage
  - What is the max total IOPS with M4?
  - What is the max total IOPS with I3?

**EC2 instances we work with for the right IO for your Cassandra cluster**

EC2 instances we use tend to be from the M4 family and the I3 family (released Nov 30, 2016). M4 is AWS EC2s newest generation of general purpose instances with EBS optimized storage whilst the I3 family includes fast SSD-backed instance storage optimized for very high random I/O performance. I3s provide high IOPS at a low cost. For tiny read/writes benchmarking I3 EC2 instances are better instances than m4s (EC2 instances) at 8x the read speed (note benchmark was I2 vs. M4, but I3 is the latest). For medium read/writes, m4 are equivalent (EBS optimized) but at 8x less cost than i3s (keep in mind price goes down and performance goes up over time). There have been some reports of EBS storage degrading over time. But for 8x the cost, and with some monitoring and replication, you could automate the retirement of degrading EC2 instances using optimized EBS that are degrading.

**EC2 I3 instances** go up to 3.3 million random IOPS (great for reads) at 4KB block size, and the I3 throughput goes up to 16 GB/s. It is a beast. The max IOPs for an instance using EBS is 65K with a max throughput of 1,250 MB.

An advantage of the M4 family is the ability to use EBS to create snapshots and simply spin up new instances by attaching EBS volume to a new instance. If you are not sure, start with m4.2xlarge. (You can use optimized EBS with I3 as well.)

You can consider D2 family of EC2 instances for mostly write operations or offline analytics that performs large queries. The D2 family offers the highest throughput for cost. If you are keeping a lot of logs or even approaching big data uses cases, this might be a great option for high throughput (mostly writes and mostly batch reads). We will talk about C4 when we cover...
CPU resources.
### SSD vs HDD

<table>
<thead>
<tr>
<th>AWS Cassandra Storage Reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD has higher read speed / IOPS (random access, seeking key)</td>
</tr>
<tr>
<td>You can use SSD with provision IOPS to improve read access (expensive)</td>
</tr>
<tr>
<td>Magnetic disks (HDD) have higher throughput but lower IOPS</td>
</tr>
<tr>
<td>AWS guide says don’t use, benchmarks say works for some use cases</td>
</tr>
<tr>
<td>Increase HDD IOPS by using instance store or JBOD</td>
</tr>
<tr>
<td>Cheapest throughput and storage of volume options</td>
</tr>
</tbody>
</table>

If in doubt use SSD EBS volumes. SSDs provide low-latency response times for random read operations and supply enough throughput for long sequential writes performance for compaction operations, writing SSTables and commit logs. Magnetic disks in EC2 have greater throughput but less IOPS which is good for SSTables compaction but not good for random reads. HDD are the cheapest per byte of storage and cheapest for byte throughput.

If in doubt, use SSD volumes. You can change it later after observing load test and production KPIs for IOPs and throughput. You can used provisioned IOPs with SSDs to buy IOPs for Cassandra clusters that are doing a lot of reads.

**Separate EBS volume for Cassandra commit log**

It makes sense if possible to have commit logs on a separate disk if using magnetic disks. SSTables are written to in streams but are read from using random access if data is not found in cache.
Take replication and compaction overhead into account. The compaction process of SSTable data makes heavy use of the disk. LeveledCompactionStrategy may need 10 to 20% overhead for compaction (faster reads). SizeTieredCompactionStrategy worse case is 50% overhead needed to perform compaction.

Keep this in mind while sizing disks. If you are doing a high-update use case, LeveledCompactionStrategy is the best solution if you want to limit the total disk size used at any point in time and to optimize reads as the row will be spread across less (up to ten times less) SSTables. LeveledCompactionStrategy requires more IO and processing time for compactions. If in doubt, use LeveledCompactionStrategy.
If you use RAID, RAID 0, which focuses on speed, is sufficient for Cassandra because Cassandra has replication and data-safety built-in. With Cassandra 3.x you should use JBOD (just a bunch of disks) instead of RAID 0 for throughput speed. JBOD is preferred, and it can help with random read speeds.
EBS Elastic Volumes and Linux File Systems

- New **EBS elastic volumes** goes well with ext4 and XFS
- **AWS Elastic volume** added 2/2017, you can change EBS type on running node!
- Prefer XFS Linux file system or Ext4 is ok
- For ext4, you will need to **expand the volume** using
  - `sudo resize2fs /dev/xvda1`
- For XFS expand volume with
  - `sudo xfs_growfs -d /mnt`

**XFS** is the preferred file system since it has less size constraints (**sudo mkfs.xfs -K /dev/xvdb**) and excels at writing in parallel. You can use EX4 as well but avoid others.

Using the new **EBS elastic volumes** goes well with ext4 and XFS. For ext4, you will need to **expand the volume** using **sudo resize2fs**

/dev/xvda1 and use this for XFS **sudo xfs_growfs -d /mnt**.

The key point here is that the OS will not automatically expand. You will have to tell it.
Cassandra Encryption at rest use KMS

- If you need *data at rest* encryption, use encrypted EBS volumes / KMS
- AWS KMS uses hardware-assisted encryption (Hardware Security Modules)
- Faster than JDK based encryption and built into EBS
- Same IOPS performance as unencrypted volumes
- Use KMS so you can rotate keys and expire them
- KMS encryption does not work with EC2 instance storage

Cassandra Encryption at rest use Amazon KMS

If you need *data at rest* encryption, use encrypted EBS volumes / KMS if running in EC2, and use dm-crypt file system if not. Since AWS KMS uses hardware-assisted encryption, it is going to be much faster than the encryption that comes with the JDK. Next fastest would be Linux based file system encryption. Avoid encryption from Cassandra JDK. Encrypted volumes have the same IOPS performance on as unencrypted volumes.

EBS easily supports KMS encryption, and it integrates well. However instance storage requires that you use an encrypted file system like dm-crypt. This is another clear advantage for KMS.
Also, KMS allows you to easily rotate keys and expire them.
EBS problems and workarounds

- **EBS has been known to degrade over time**
- Watch for EBS issues like poor throughput, performance degrading over time, and instances not cleanly dying
- Watch with CloudWatch
- Cloudurable provides AMIs which can be monitored using Amazon CloudWatch (installs systemd processes)
  - Linux OS log aggregation, and Cassandra log aggregation into CloudWatch logs
  - OS metrics and Cassandra metrics into CloudWatch metrics
- Have a plan to retire problem Cassandra Nodes and spin up new ones – yes we have experienced these problems as recent as 2016

**EBS has been known to degrade over time**

With EBS, you need to keep an eye out for EBS issues like poor throughput, performance degrading over time, and instances not cleanly dying. This is where system monitoring like CloudWatch comes into play and one reason we build images AMIs which can be monitored using Amazon CloudWatch. We support Linux OS log aggregation, and Cassandra log aggregation into CloudWatch. We also support OS metrics and Cassandra metrics into CloudWatch.
Ways to Improve Cassandra Read Speed

- Horizontally scale Cassandra (more nodes)
- Use instance store (super fast IO)
- Buy provisioned IOPs – or bigger SSDs
- Add more disks to each node using JBOD (more disks)
  - More EBS volumes or
  - EC2 instances with more SSDs or Disks
- Use a bigger key-cache, row-cache (more memory)
- More disk space for SizeTieredCompactionStrategy

Scaling Cassandra read speeds:
- Horizontally scale Cassandra (more nodes)
- Use instance store (super fast IO)
- Buy provisioned IOPs – or bigger SSDs
- Add more disks to each node using JBOD (more disks / EBS volumes)
- EC2 instances with more SSDs or Disks if using
- Use a bigger key-cache, row-cache (more memory / more cache)
- More disk space for SizeTieredCompactionStrategy
- Don’t forget to optimize query and partition keys
- Add more tables or materialized views to optimize queries
| Cassandra AWS CPU Guidelines | Cassandra CPU requirements with Amazon EC2 |
Cassandra CPU requirements in AWS Cloud

- Cassandra is highly concurrent
- Cassandra nodes can use as many CPU cores as available *if configured correctly*
- An Amazon EC2 vCPU is a hyper thread
  - often referred to as a virtual core
  - Physical thread of execution worker
Cassandra High writes can be CPU bound

Cassandra clusters workloads that do a lot of writing (insert heavy), can be CPU-bound. This can be multiplied if using JBOD, where a single node is managing 4 or 8+ volumes. Cassandra is efficient for writes, but this is largely due to doing a structured merge sort during compaction, which is CPU intensive. Often the CPU becomes the limiting factor for writes.

Since writes are almost never IO bound, the ideal number of concurrent_writes is dependent on the number of cores in Cassandra node. Thus set concurrent_writes in cassandra.yaml to 8 x vCPU is a good rule of thumb for EBS and 4 x vCPU for EC2 instance storage.

The concurrent_compactors should be set to the # of vCPUs if using SSDs, and set to the number of attached EBS volumes / disks for JBOD. The point, that the more CPU resources your Cassandra node has, the faster the compaction throughput. See this Cassandra tuning guide for more information, and this JIRA ticket.
Compaction strategy and CPU usage

- **Compaction strategy can influence CPU Spikes**
  
  - *SizeTieredCompactionStrategy* works with larger *SSTables* so has more spikey CPU
  
  - Need for faster writes
  
  - *LeveledCompactionStrategy* will use a more even level of CPU
  
  - Needed for more consistent CPU usage
  
  - Needed for faster reads

**Compaction strategy can influence vCPU usage**

*SizeTieredCompactionStrategy* works with larger *SSTables* so has more spikey CPU usage. Supports faster writes.

*LeveledCompactionStrategy* will use a more even level of CPU. Supports faster reads.
In general use 4 to 8 vCPUs for Cassandra Nodes minimum
You need at least 4 cores but prefer 8 cores for a production
machine (add more as needed per use case)
Why 8 to start? Compaction, compression, key lookup based on
bloom filters, SSL if enabled, all need CPU resources
m4.xlarge ok for development testing (4 vCPUs)
m4.2xlarge 8 vCPUs should handle most production loads nicely
Use i3.2xlarge (for high random and small reads)
Use d2.xlarge for high writes and long sequential
Use c4.2xlarge for use cases with few cache hits

In general use 4 to 8 vCPUs for Cassandra Nodes minimum
You need at least 4 cores but prefer 8 cores for a production
machine. Compaction, compression, key lookup based on bloom
filters, SSL if enabled, will all need CPU resources. The m4.xlarge falls
a bit behind for this as it only has 4 vCPUs (4 cores).
The m4.2xlarge has 8 vCPUs which should be able to handle most
production loads nicely. The i2.xlarge (for high random read)
and d2.xlarge for high writes and long sequential reads are also a little
light on CPU power. Consider i3.2xlarge and d2.2xlarge for production
workloads as they have 8 vCPUs.
CPU Usage for GC1 Garbage collector and CMS

- **GC1 Garbage collector** and the CMS garbage collector benefit from having more threads (i.e., may need more EC2 vCPUs)
- When working with large Java heap sizes, GC1 and CMS can benefit from parallel processing which requires more EC2 vCPUs
- CMS can really use more CPU!
  - Habit of turning memory eventually into Swiss cheese
  - Eventually needing a full, stop the world garbage collection
  - GC1 does not have this same problem with memory fragmentation, which is why CMS is deprecated in Java 9.

What happens if a node under heavy load has a stop the world GC?

CPU Usage for GC1 Garbage collector and CMS

Both the **GC1 Garbage collector** and the CMS garbage collector benefit from having more threads. When working with large Java heap sizes, GC1 and CMS can benefit from parallel processing which requires more CPUs.

CMS has a habit of turning memory into Swiss cheese and over time, needing a full, stop the world garbage collection. GC1 does not have this same problem with memory fragmentation, which is why CMS is deprecated in Java 9.
Multi Region/DC need more EC2 vCPUs

- If you are using multiple regions, i.e., a multi-dc deployment increase `max_hints_delivery_threads` as cross DC handoffs are slower
- Also keep in mind for cluster/storage communication that there is more CPU overhead, which might be a wash if the DC to DC communication has a lot of latency, using SSL, then more too
- How many threads do you need? How eventually consistent do you want to be between data-centers? And how long will these threads be waiting for IO?
- Don’t just consider happy case, but unhappy case

Multi-DC / Multi Region need more CPU resources
If you are using multiple regions, i.e., a multi-dc deployment then you will want to increase the `max_hints_delivery_threads` as cross DC handoffs are slower. Also keep in mind for cluster/storage communication that there is more CPU overhead, which might be a wash if the DC to DC communication has a lot of latency. Cassandra allows one outbound hint thread per Cassandra node. The maximum inbound hint streaming per node will still be `hinted_handoff_throttle_in_kb`. You can safely increase `max_hints_delivery_threads` without worrying about overwhelming a single node. See Bandwidth Required for Cassandra Hinted Handoff for more details about the math.

How many threads do you need? How eventually consistent do you want to be between data-centers? (And how long will these threads be waiting for IO? Depends on the latency of the network and your throttle rate.) Increase this to 16 or more for two DCs. Increase it to 32 or more for multiple DCs. For single DC deployments, set it to \( \frac{1}{2} \) the number of nodes in the system or less. With Cassandra, it is important not to just consider the happy case, but the unhappy case like a DC went down for a few hours, and came back up right during peak usage. It is good to have the extra CPU. The more nodes or
latency between, the more vCPU you might want to have.
Cassandra workloads with large datasets

- Cassandra workloads that can’t fit into main memory,
- Cassandra’s bottleneck will be reads that need to fetch data from disk (EBS Volume or local storage)
- concurrent_reads should be set to (16 * number_of_drives)
- potential with JBOD 4 of having 64 read threads
- Use EC2 instance with more memory?

Cassandra workloads with large datasets
For Cassandra workloads that can’t fit into main memory, Cassandra’s bottleneck will be reads that need to fetch data from disk (EBS Volume or local storage). The concurrent_reads should be set to (16 * number_of_drives) so you have the potential with JBOD 4 of having 64 read threads.
Doing a lot of streaming writes between nodes?

Increase `memtable_flush_writers`

If you are streaming a lot of data from many nodes

- you need to increase the number of flush writers (`memtable_flush_writers`).
- If you do not have enough writers to deal with a (larger than normal) amount of data hitting them you can cause your streams to fail
- recommendation is to set `memtables_flush_writers` equal to the number of vCPUs on the EC2 Cassandra node instance
- EC2 instance with more vCPUs allows more throughput for writes

Instance Storage SSD increase: `memtable_flush_writers` * 
`data_file_directories <= # of vCPU`.

Instance storage HDDs or EBS SSD use `memtable_flush_writers`: #vCPUs

Read Scale it to Billions — What They Don’t Tell you in the Cassandra README for more details.

Recall, if your data directories are backed by instance storage SSD, you can increase this using `memtable_flush_writers` * 
`data_file_directories <= # of vCPU`. If you are using instance storage HDDs or EBS SSD use `memtable_flush_writers`: #vCPUs. Do not leave this set to 1.
Horizontal scale is not always possible with Cassandra

- When you are using Cassandra for super high-speed writes or using it with very large datasets
- You may have to scale up your Cassandra nodes and add more vCPU and memory by using larger EC2 instances
- Vertical scale-up is also needed in some cases
- When horizontally scaling, Cassandra does a lot of streaming, you could make instances larger (gradually), add your nodes, and then gradually make the Cassandra instances smaller

Horizontal scale is not always possible with Cassandra

When you are using Cassandra for super high-speed writes or using it with very large datasets, you may have to scale up your Cassandra nodes and add more vCPU and memory. Vertical scale-up is also needed in some cases.

When horizontally scaling, Cassandra does a lot of streaming, you could make EC2 instances larger (gradually), add your nodes, let Cassandra stream data to new nodes, and then gradually make the Cassandra instances smaller again. You would employ EBS snapshots, take nodes offline, resize them, bring them back online, let them recover, repeat.
The i3.8xlarge, m4.10xlarge+ and above EC2 instance types use more than 1 CPU, which means NUMA controls are available. Transferring memory between CPU sockets more expensive
Cassandra uses numactl --interleave by default
Add -XX:+UseNUMA to cassandra-env.sh
JVM can handle NUMA directly
GC divide GC efforts across domains
Cassandra uses modified SEDA most likely NUMA pin not as needed
prefer using a smaller EC2 instance before NUMA pin
If you are running running other JVM processes (examples in notes), Force JVM bind to NUMA node, so all memory is local, and all threads will execute on the same core (speed)

Read Al Tolbert blog post on Cassandra tuning

AWS Cassandra and NUMA
The i3.8xlarge, c4.8xlarge, m4.10xlarge, and above EC2 instance types use more than 1 CPU, which means NUMA controls are available. A good read on this is from Al Tolbert’s blog post. The quickest way to tell if a machine is NUMA is to run “numactl –hardware”. -Al Tolbert blog post on Cassandra tuning. NUMA stands for Non-Uniform Memory Architecture. Modern x86 CPUs contain an integrated memory controller. Multi-socket system, have two memory controllers. Each CPU gets a share of the memory. If one CPU socket needs memory that another CPU socket has, the memory is transferred. Transferring this memory between CPUs is more expensive than if the memory only existed in one CPUs memory. When a JVM thread only uses memory local to one CPU, things go fast, and if not slower (10 CPU cycles vs. 100 or some order of magnitude). The Cassandra startup script sets JVM numactl --interleave.

One is to comment out the numactl --interleave in bin/cassandra and add -XX:+UseNUMA to cassandra-env.sh. –Al Tolbert
This setting allows the Cassandra JVM to handle NUMA directly. The GC (GC G1) will divide GC efforts across domains, which should make GC more efficient.

What about CPU pinning? Rather than using numactl --cpunodebind to pin a JVM to a particular node, you should consider running your EC2 instance on a smaller instance type. Also since Cassandra uses modified SEDA, it should do okay with thread memory boundaries. However, if you are running on a larger EC2 instance and you are running other JVM processes on the same instance (perhaps pairing it with SOLR, Spark or others for some locality/integration benefits), then consider using numactl --cpunodebind. The Cassandra JVM gets bound to NUMA node, so all memory is local, and all threads will execute on the same core. Use with caution, and since the Cassandra uses a modified SEDA, it may not benefit as
much as other Java applications. Please read Al Tolbert’s complete Cassandra tuning guide. It is a great resource.
CPU Cache sizes

- Use latest generation of EC2 instances as they have larger L1 and L2 cache sizes (between 25 MB and 30 MB of on CPU cache)
<table>
<thead>
<tr>
<th><strong>Cassandra / Kafka Support in EC2/AWS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS Cassandra System Memory Guidelines</strong></td>
</tr>
</tbody>
</table>
Basic System Memory Guidelines

- Do not use less than 8GB JVM Heap
- More RAM the better for Cassandra
- Use G1GC
- EC2 instance m4.xlarge has 16GB of memory use this or higher (m4.xlarge is light on CPU)
- i2.xlarge and d2.xlarge exceed min memory requirements

System Memory Guidelines for Cassandra AWS

**Basic guidelines**

*Do not use less than 8GB of memory for the JVM.* The more RAM the better. SSTable are first stored in memory and then written to disk sequentially. The larger the SSTable the less scanning that needs to be done while reading and determining if a key is in an SSTable using a bloom filter. In the EC2 world this equates to an m4.xlarge (16GB of memory), and you need some memory for the OS, specifically the IO buffers. The i2.xlarge and d2.xlarge are the smallest in their family and exceed the min memory requirement (and then some).
Cassandra maintains these components in Java heap memory:

- Bloom filters
- Partition summary
- Partition key cache
- Compression offsets
- SSTable index summary

Some of these component’s Java heap usage grows as you increase the Java heap size.
Cassandra uses memory 4 ways:

- Java heap
- Off-heap memory
- OS page cache
- OS TCP/IP stack I/O cache
The more memory the better. If the memory is available, Cassandra and the Linux OS can use it. In the Java heap, Cassandra can use memory for the key cache which can speed up queries. For smaller tables, that are read often, you can use the row cache which uses off-heap.

If cache hit rate is high, then there is less read IO.

In the NoSQL world, Cassandra is king for writes - Cache helps it have good marks for reads.

Read-heavy system in EC2 Cassandra world,
- Might make sense to go into 60 GB to 120 GB (e.g., m4.4xlarge, i3.2xlarge)
- Above this range in EC2, and you have to worry about NUMA concerns

For a read-heavy system in EC2, it could make sense to go into 60 GB to 120 GB (e.g., m4.4xlarge, i3.2xlarge). Above this range in EC2, and you have to worry about NUMA concerns, see NUMA Cassandra AWS guidelines.
Cassandra relies heavily on the Linux OS page cache for caching of data stored on EBS and local instances volumes. Every read that the OS gets a cache hit on, means the data is read from RAM not the EC2 volume, and we take the IOPs or throughput of the EBS out of the equation. This means you must leave memory for the Linux OS. You are not running a stateless servlet engine. You must also leave some space for Linux IO buffers. You are running a stateful NoSQL database. The OS memory left over after the JVM should be 2x to 6x the size of the JVM.
Cassandra makes uses off-heap memory as follows:

- Page cache
- Bloom filter
- Compression offset maps
- Row caches

Think of off-heap memory as OS memory that is not managed by the JVM garbage collectors.
Since Cassandra uses OS/off-heap memory, you have quite a bit more OS memory than allocated to the JVM for Cassandra to be effective.
Starting breakdowns of JVM heap size vs OS RAM for Cassandra
Taking our recommendation for at least 8 vCPUs per Cassandra EC2 instance and using the 2x to 6x ratio of Cassandra JVM memory vs. Linux system memory gives us this table.
JVM Garbage Collector for Cassandra

Due to the fact that Cassandra has some long lived objects on the heap, the choice in GCs come down to CMS and GCG1. Choose GCG1.

The general rule is don't use GGC1 if your heap is under 5GB (some say 8GB, some 6 GB, Oracle says under 1GB).

You will notice on the preceding chart above, there is no JVM configuration with a heap less than 5GB.

In General DON'T USE CMS! (see notes)

- Especially, do not use JVM CMS garbage collector if your JVM is over 16 GB (fragmented memory, long stop the world GC)
- CMS is deprecated in JDK 9
- A case could be made for a heap size between 5 GB and 8 GB for CMS

JVM Garbage Collector for Cassandra
Due to the fact that Cassandra has some long lived objects on the heap, the choice in GCs come down to CMS and GCG1. Choose GCG1.

The general rule is don’t use GGC1 if your heap is under 5GB some say 8GB as GCG1 (Oracle says under 1GB).

You will notice on the preceding chart above, there is no JVM configuration with a heap less than 5GB.

Do not use JVM CMS garbage collector if your JVM is over 16 GB. CMS is deprecated in JDK 9. You could make a case with a heap size between 5 GB and 8 GB for CMS.

The promise of G1 on smaller systems vs CMS is more robust performance across a range of workloads without manual tuning. GCG1 probably won’t perform as well in terms of ops/s, etc. Using GCG1 under a 8GB heap you are trading some speed against CMS pain once you start having cascading IO and heap pressure through the system. There are benchmarks that clearly show G1 beating CMS at 8GBE.

If you want to take CMS out of the picture all together, use this table on the next slide as a guide.
Image Credit: Ranjith Ramachandran from YouTube talk on GC
https://www.youtube.com/watch?v=UnaNQgzw4zY
Image Credit: Ranjith Ramachandran from YouTube talk on GC
https://www.youtube.com/watch?v=UnaNQgzw4zY
Image from Cloudera can be found here: http://blog.cloudera.com/blog/2014/12/tuning-java-garbage-collection-for-hbase/
Image from Oracle can be found here: http://www.oracle.com/technetwork/tutorials/tutorials-1876574.html
Image from Intel from here: https://www.slideshare.net/HBaseCon/dev-session-7-49202969
There are benchmarks that clearly show G1 beating CMS at 8GBE. If you want to take CMS out of the picture all together, use this table above as a guide.

Note that the Apache Cassandra on AWS: Guidelines and Best Practices has a mistake. It says the max heap size you should use for Cassandra is 8GB, and it says the DataStax Documentation says this. The DataStax documentation says use between 14GB and 64GB of heap. 8GB is only for older computers. There is no 8GB cap on Cassandra JVM heaps.

**To set heap size for m4.2xlarge**

-Xms12G
-Xmx12G

-Xms sets min heap size and -Xms sets max heap size
You should prefer an easily tuned and stable setting over one that has the issues that CMS does.
Guide for using GCG1 with Cassandra JVM

The XX:G1RSetUpdatingPauseTimePercent=5 sets a percent target amount (defaults 10) that G1GC spends in updating RSets during a GC evacuation pause. An RSets is Remembered Sets, per-region entries that allow G1GC to track outside references to heap region. This is so GCG1 does not have to scan the whole heap for references into a region. Read tips for tuning the GCG1. By decreasing G1RSetUpdatingPauseTimePercent, the JVM will spend less time in updating the RSets during the stop-the-world (STW) GC pause, and the RSets will be updated in the refinement threads.

InitiatingHeapOccupancyPercent defaults to 45% of your total Java heap. You can drop the value starts the marking cycle earlier. It is another way to start GC earlier to avoid STW. If you want to maximize throughput, and are less concerned with pauses. Here is another way to configure GCG1.

Guide for using GCG1 with Cassandra JVM for faster throughput but longer pauses
-XX:+UseG1GC
-XX:MaxGCPauseMillis=1000
-XX:InitiatingHeapOccupancyPercent=60

Just make sure your Cassandra timeouts are more than 1000ms. If you are experiencing long pauses try –XX:G1HeapWastePercent:20 (it defaults to 5%). See https://www.slideshare.net/HBaseCon/dev-session-7-49202969
GC settings common to both CMS and GCG1

Guide setting for both GCG1 and CMS

-XX:ParallelGCThreads=#vCPU / 2
-XX:ConcGCThreads=#vCPU / 2
-XX:+ParallelRefProcEnabled
-XX:+AlwaysPreTouch # allocate and zero (force fault) heap memory on startup
-XX:+UseTLAB # thread local allocation blocks
-XX:+ResizeTLAB # auto-optimize TLAB size
-XX:+UseBiasedLocking # disable biased locking for cassandra

Increasing ParallelGCThreads and ConcGCThreads is useful for any parallel garbage collector. Turning on ParallelRefProcEnabled helps collect reference objects (e.g., WeakReference) in parallel which will be faster if there is a lot. “Reference processing isn’t usually a big deal for Cassandra, but in some workloads it does start to show up in the GC logs. Since we pretty much always want all the parallel stuff offered by the JVM, go ahead and enable parallel reference processing to bring down your p99.9’s.” – Al Tobey Writes Blog:

Cassandra tuning guide

Use +AlwaysPreTouch to allocate and zero, which does a force fault, heap memory on startup. This ensures all memory is faulted and zeroed on startup, and prevents soft faults making hugepage allocation more effective. Use XX:+UseTLAB to add thread local allocation blocks. Use +ResizeTLAB to allow JVM to auto-optimize TLAB size. Then we disable bias locking for Cassandra with -UseBiasedLocking. Biased locking was introduced in Hotspot 1.5 to reduce locking in systems that use locks efficiently (single-writer locks). Cassandra has contended locks in frequently used areas which makes this optimization a net loss when
Cassandra is under load.
Don’t use CMS. If you really have to have a JVM that is between 5GB and 8GB, and having an easy to configure, reliable systems does not win out over raw speed, then maybe CMS.

Remember, CMS does well until it doesn’t. (Personal experiences with CMS are bad.)
The Parallel copy collector (ParNew) is responsible for young collection in CMS. Use \texttt{ParGCCardsPerStrideChunk (default 256)} to increase granularity of tasks distributed between worker threads.

Use \texttt{CMSScavengeBeforeRemark} triggers a Young GC (STW) before running CMS Remark (STW) phase to reduce the duration of the Remark phase.

Use \texttt{CMSWaitDuration} so that once CMS detects it should start a new cycle, it will wait this long for a Young GC cycle to occur. This reduces the duration of the Initial-Mark (STW) CMS phase.

The \texttt{SurvivorRatio=N} divides the young generation by \( N+2 \) segments, take \( N \) segments for Eden and 1 segment for each survivor.

The \texttt{MaxTenuringThreshold} defines number of young GC an object survives before it gets pushed into the old generation. The idea here is if this is too low that it will increase pressure on CMS.

CMS GC usually uses heuristic rules to trigger garbage collection making it less predictable for production JVM options.

The \texttt{UseCMSInitiatingOccupancyOnly} initiates CMS GC in advance to avoid full, stop-the-world, GC. \texttt{CMSInitiatingOccupancyFraction} sets the trigger level for CMS, i.e., the Cassandra JVM should use less that 70\% of old generation. Note that the -Xmn sets the heap size for young generation.

Depending on how you have compaction workers setup, you want this to
be $\frac{1}{4}$ to $\frac{1}{3}$ size of your total heap.
After is after tuning G1.
Before is before tuning G1.

Image from Intel from here: https://www.slideshare.net/HBaseCon/dev-session-7-49202969
Cassandra uses native libraries to allocate memory if available. Ensure JNA and JEMALLOC are installed on Linux machine Amazon AMI. If you are creating an Amazon AMI image for Cassandra, then you want to install both of these.

In cassandra.yaml set memtable_allocation_type to offheap_objects if JNA and jemalloc are installed and heap_buffers if not. Modify the memtable space by changing the memtable_heap_space_in_mb and memtable_offheap_space_in_mb in cassandra.yaml can reduce the amount of Java heap space that Cassandra uses.

“jemalloc is a general purpose malloc(3) implementation that emphasizes fragmentation avoidance and scalable concurrency support.” --jemalloc jemalloc.net/

“Java Native Access (JNA) JNA’s design aims to provide native access in a natural way with a minimum of effort. No boilerplate or generated glue code is required.” --Java Native Access – Wikipedia

Credit: Image is from a FaceBook talk on JEMalloc
Cassandra / Kafka Support in EC2/AWS

Cassandra Networking Guidelines

Cassandra Networking requirements with Amazon EC2
Networking:
AWS EC2 has placement groups and enhanced networking which allow high-speed throughput for clustered software like Cassandra (10GBE). Networking is important to Cassandra due to replication of data. Most deployments an AZ is treated like a rack, and Cassandra tries to store replica data on nodes that are in a different rack (in EC2’s case a different AZ). EC2 placement groups and enhanced networking only works per AZ. Thus the most common use case of Cassandra cluster network would not use enhanced networking (placement groups) at all. Now if you replicate higher than 2 then some replication will happen within the same AZ and placement groups (enhanced networking) could speed that up. Go ahead and use enhanced networking.
Cluster Networking

- R4, X1, M4, C4, C3, I2, CR1, G2, HS1, P2, and D2 support cluster networking aka enhanced networking
- Cassandra EC2 Instances launched into *placement group* form a *logical EC2 cluster*
  - provides high-bandwidth, low-latency networking between all Cassandra EC2 instances in cluster
  - Placement group per cluster per EC2 region
  - Can Utilize up to 10 Gbps for single-flow
  - 20 Gbps for multi-flow traffic in each direction
  - Network traffic outside placement group limited to 5 Gbps

From EC2 docs: Cluster Networking
https://aws.amazon.com/ec2-instance-types/

“R4, X1, M4, C4, C3, I2, CR1, G2, HS1, P2, and D2 instances support cluster networking. Instances launched into a common cluster placement group are placed into a logical cluster that provides high-bandwidth, low-latency networking between all instances in the cluster. The bandwidth an EC2 instance can utilize in a cluster placement group depends on the instance type and its networking performance specification. When launched in a placement group, select EC2 instances can utilize up to 10 Gbps for single-flow and 20 Gbps for multi-flow traffic in each direction (full duplex). Network traffic outside a cluster placement group (e.g. to the Internet) is limited to 5 Gbps (full duplex). Cluster networking is ideal for high performance analytics systems and many science and engineering applications, especially those using the MPI library standard for parallel programming.”
Enhanced networking vs. none

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>95th percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single AZ</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td>Multi AZ (3 AZs)</td>
<td>1.5</td>
<td>2.8</td>
</tr>
<tr>
<td>READ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single AZ</td>
<td>1.0</td>
<td>2.6</td>
</tr>
<tr>
<td>Multi AZ (3 AZs)</td>
<td>1.5</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Enhanced Networking

Use enhanced networking!

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>95th percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single AZ</td>
<td>0.9</td>
<td>2.4</td>
</tr>
<tr>
<td>Multi AZ (3 AZs)</td>
<td>1.1</td>
<td>2.3</td>
</tr>
<tr>
<td>READ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single AZ</td>
<td>0.7</td>
<td>1.5</td>
</tr>
<tr>
<td>Multi AZ (3 AZs)</td>
<td>1.0</td>
<td>1.9</td>
</tr>
</tbody>
</table>

10 GBE

- 10 GBE is the recommendation for high-performance Cassandra clusters
- With M4 series you need *m4.10xlarge to get 10GBE*
  - *m4.16xlarge has 20GBE (two nics)*
- *c4.8xlarge* supports 10GBE as well
- All I3 series support 10GBE (bonus point for I3)

🌟 What bandwidth do you think you would get with a m4.4xlarge?
Subnet per AZ for quorum

- What happens if the AZ goes down?
- Do your reads stop working?
- Do your writes stop working?
- Availability Zones <= Replication Factor
The image for this slide came from this article: **How To Setup A Highly Available Multi-AZ Cassandra Cluster On AWS EC2** by Todd Hoff for HighScalability.
Cassandra Networking guidelines

- Deploy Cassandra nodes spread across multiple AZs availability zones
- VPC subnet lives in a AZ, you need subnet per AZ
- Start with a replication factor of 3 combined with 3 availability zones
  - Allows you to use local quorum reads and writes even during a single AZ failure
- Use EC2 instances that support enhanced networking and deploy them in the same placement group
A snitch determines where nodes belong with regards to datacenters and racks. Think of a snitch as the configuration of your network topology. Cassandra uses this information to distribute replicas. Cassandra can also use this information, for example, to determine locality with local quorum writes and reads. Why go to another datacenter to do a consistent read when there is a server on the same rack backplane with a very low latency connection? Also, when replicating data, Cassandra will not put the replica on the same rack. For Cassandra to achieve proper replication and high-speed reads/writes from the client, and now about local quorums then it will need to know about the network topology.
Review types of Snitches

- DynamicSnitch – Determines locality of Cassandra nodes by performance of reads (runtime snitch)
- SimpleSnitch – single deployment datacenters – not rack aware or data-center aware
- RackInffering Snitch – determines rack and datacenter by IP. 10.1.1.10 (Datacenter 1, Rack 1, Node 10) 10.2.3.10 (Datacenter 2, Rack 3, Node 10)
- GossipingPropertyFileSnitch – property file exposes rack, server setup. Servers can be added through gossip.
- Cloud specific snitches
AWS: Stiches for Snitches

- There are two snitches specific for Cassandra running on AWS
- **EC2Snitch**
  - Treats AZs as Racks
- **EC2MultiRegionSnitch**
  - Treats AZs as Racks
  - Treats Regions as Datacenters
Pitfalls of EC2 * Snitch

- Relies on EC2 API
  - EC2 API has gone down for a time
  - EC2 APIs are rate limited
    - If you exceed, limit, no calls for you
    - Keep Production and Development on two different accounts

It has happened to people.
EC2Snitch AWS Setup P1

- AWS NAT Gateways are needed for security patches, and software updates of nodes
  - No incoming traffic just outgoing but can respond
- Private subnets are used for Cassandra Cluster
- Limit access to private subnets via route table, NACL and security groups
  - Only apps that need it, bastion, should have access/routes

Remember Cassandra is stateful and limiting full roll out of a brand new instance with updates preinstalled will cause lots of streaming between Cassandra nodes and is not ideal.
EC2Snitch AWS Setup P2

- Use Bastion Host to attach via ssh for public and corporate access
- Bastion can be in a VPN Subnet or a Public Subnet
- Public Subnet can use IGW, host NAT Gateway
- Cassandra Seed servers
  - At least one seed per AZ in case of outage
  - Can use ENIs to keep same private IP
  - Seed servers can be assigned internal DNS hostnames via Route53

Remember Cassandra is stateful and limiting full roll out of a brand new instance with updates preinstalled will cause lots of streaming between Cassandra nodes and is not ideal.
EC2Snitch layout 1 Region

VPC
- Private Subnet 1 for Cassandra Cluster
  - AZ: us-west-2a
  - Seed CN1, CN2, CN3, CN4
- Private Subnet 2 for Cassandra Cluster
  - AZ: us-west-2b
  - Seed CN10, CN12, CN13, CN14
- Private Subnet 3 for Cassandra Cluster
  - AZ: us-west-2c
  - Seed CN20, CN22, CN23, CN24

AWS Region: us-west-2
- Public Subnet 1
  - NAT Gateway
- Public Subnet 2
  - NAT Gateway
  - Bastion Host
- Public Subnet 3
  - NAT Gateway

AWS Internet Gateway
Steps

- Step 1: Create AMI with all the goodies
- Step 2: Run cloud formation to form VPC
- Step 3: Launch EC2 instances
- Step 4: Login and make sure cluster is healthy
Step 1: create AMI

- Tools to monitor [Linux OS to CloudWatch metrics](#)
- Tools to monitor [Cassandra KPIs to CloudWatch metrics](#)
- Tools to send logs from [Linux OS to CloudWatch logs](#)
- Tools to send logs from [Cassandra to CloudWatch logs](#)
- Cassandra, and config files
- AWS command line tools for S3 backup of Cassandra snapshots
- Tools to configure Cassandra with EC2
Metrics daemon to collect OS metrics, Cassandra metrics, and send results to AWS CloudWatch Metrics.

metricsd
Cassandra Cloud

Utility for configuring Cassandra for CloudDeployments

cassandra  ec2  aws-ec2  Manage topics

resources/user-data/cassandra - modify user-data for EC2 instances

#!/bin/bash
set -e


/opt/cloudurable/bin/cassandra-cloud --cluster-name test \\
  --client-address $BIND_IP \\
  --cluster-address $BIND_IP \\
  --cluster-seeds 192.168.1.18,192.168.2.18 \\
  --snitch Ec2Snitch

/bin/systemctl restart cassandra
Cassandra Image (examples)

- **cassandra-image**
- Packer (create AMI image)
- Vagrant (create cluster)
- CloudFormation (create cluster)
- Ansible (manage cassandra, updates)
- AWS-cli
**Step 1: Create AMI: Use Packer to create AMIs**

Packer is used to create machine and container images for multiple platforms from a single source configuration. We use Packer to create AWS EC2 AMIs (images) and Docker images. (We use Vagrant to setup dev images on Virtual Box.) Packer like Vagrant is from HashiCorp. Packer can use Ansible playbooks. Cloudurable developers are big fans of HashiCorp. We love Consul, Vagrant, Packer, Atlas, and the rest.

Notice that we are using a packer amazon-ebs builder to build an AMI image based on our local dev boxes EC2 setup. Also, notice that we use a series of Packer provisioners. The packer file provisioner can copy files or directories to a machine image. The packer shell provisioner can run shell scripts. Lastly the packer ansible provisioner can run ansible playbooks. We covered what playbooks/ssh-addkey.yml does in the previous article, but in short it sets up the keys so we use ansible with our Cassandra cluster nodes.
Before we started using ansible to do provisioning, we used **bash scripts** that get reused for packer/docker, packer/aws, and vagrant/virtual-box. The script 000-ec2-provision.sh invokes these provisioning scripts which the first three articles covered at varying degrees (skim those articles if you are curious or the source code, but you don't need it per se to follow). This way we can use the same provisioning scripts with AMIs, VirtualBox, and AWS EC2.

Those scripts tune the OS, install packages, install Cassandra, install tools to monitor OS and Cassandra and more.
metricsd to send OS metrics to AWS
We are using metricsd to read OS metrics and send data to AWS CloudWatch Metrics. Metricsd gathers OS KPIs for AWS CloudWatch Metrics. We install this as a systemd process which depends on cassandra. We also install Cassandra as a systemd process. We use systemd unit quite a bit. We use systemd to start up Cloudurable Cassandra config scripts. We use systemd to start up Cassandra/Kafka, and to shut Cassandra/Kafka (this article does not cover Kafka at all) down nicely. Since systemd is pervasive in all new mainstream Linux distributions, you can see that systemd is an important concept for DevOps.
Metricsd gets installed as a systemd service by our provisioning scripts. We use systemctl enable to install metricsd to start up on system start. We then use systemctl start to start metricsd.
We could write a whole article on metricsd and AWS CloudWatch metrics, and perhaps we will. For more information about metricsd please see the metricsd github project.
The metricsd systemd unit depends on the Cassandra service. The unit
file is above.
systemd-cloud-watch to send OS logs to AWS log aggregation

We are using `systemd-cloud-watch` to read OS logs from systemd/journald and send data to AWS CloudWatch Log. The systemd-cloud-watch daemon journald logs and aggregates them to AWS CloudWatch Logging. Just like metricsd we install systemd-cloud-watch as a systemd process which depends on cassandra. Remember that we also install Cassandra as a systemd process, which we will cover in a moment.

The systemd-cloud-watch daemon gets installed as a systemd service by our provisioning scripts. **Installing systemd-cloud-watch systemd service from our provisioning scripts** see above.

We use systemctl enable to install systemd-cloud-watch to start up when the system starts. We then use systemctl start to start systemd-cloud-watch. The systemd-cloud-watch system unit depends on the Cassandra service. The unit file (`/etc/systemd/system/systemd-cloud-watch.service`) is listed above.

Note to use metricsd and systemd-cloud-watch we have to set up the right AWS IAM roles, and then associate that IAM instance role with our instances when we start them up. The systemd-cloud-watch.conf is set up to use the AWS log is listed above.
(systemd-cloud-watch.conf). To learn more about systemd-cloud-watch, please see the systemd-cloud-watch GitHub project.
Running Cassandra as a systemd service
If Cassandra stops for whatever reason, systemd can attempt to restart it. The systemd unit file can ensure that our Cassandra service stays running. The systemd-cloud-watch utility will be sure to log all restarts to AWS CloudWatch.
Here is the systemd unit file for Cassandra.

```
[Unit]
Description=Cassandra Service

[Service]
Type=forking
PIDFile=/opt/cassandra/PID

ExecStartPre=/sbin/swapoff -a
ExecStartPre=/bin/chown -R cassandra /opt/cassandra
ExecStart=/opt/cassandra/sbin/cassandra -p /opt/cassandra/PID

WorkingDirectory=/opt/cassandra
Restart=always
RestartSec=60
TimeoutStopSec=60
TimeoutStartSec=60
User=cassandra

[Install]
WantedBy=multi-user.target
```

The above will tell systemd to restart Cassandra in one minute if it goes down. Since we are using OS log aggregation to AWS Cloudwatch every time Cassandra goes down or is restarted by systemd, we will get log messages that we can create alerts and trigger in CloudWatch to then run AWS Lambdas that work with the rest of the AWS ecosystem. Critical bugs in queries or UDF or UFA could cause Cassandra to go down. These could be hard to track down and sporadic. Logging aggregation helps.
AWS CLI is a must!

✧ You must use the AWS CLI!
✧ You can launch instances, query instances for their public IP
✧ AWS CLI is the Swiss army knife of AWS cloud

Using AWS CLI to create our Cassandra EC2 instance

The AWS Command Line Interface is the Swiss army knife of utilities to manage your AWS services.

"With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts." --AWS CLI Docs

The AWS command line tool slices and dices from VPCs to running CloudFormations to backing up Cassandra snapshot files to S3. If you are working with AWS, you need the AWS CLI.
AWS CLI: Common scripts we use

- `bin/ec2-env.sh` - sets up common AWS references to subnets, security groups, key pairs
- `bin/create-ec2-instance.sh` - uses `aws command line` to create an ec2 instance
- `bin/login-ec2-cassandra.sh` - Uses ssh to log into Cassandra node we are testing
- `bin/get-IP-cassandra.sh` - Uses `aws command line` to get the public IP address of the cassandra instance
- `bin/run-vpc-cloudformation.sh` - Launch cloud formation that forms VPC, subnets, IGW, NGW, route tables, security groups
- Many more

**Automating EC2 image creation with AWS CLI**

Starting up an EC2 instance with the right, AMI id, IAM instance role, into the correct subnet, using the appropriate security groups, with the right AWS key-pair name can be tedious. We must automate as using the AWS console (GUI) is error prone (requires too much human intervention).

Instead of using the AWS console, we use the `aws command line`. We create four scripts to automate creating and connecting to EC2 instances:

`bin/ec2-env.sh` - sets up common AWS references to subnets, security groups, key pairs
`bin/create-ec2-instance.sh` - uses `aws command line` to create an ec2 instance
`bin/login-ec2-cassandra.sh` - Uses ssh to log into Cassandra node we are testing
`bin/get-IP-cassandra.sh` - Uses `aws command line` to get the public IP address of the cassandra instance

Note to parse the JSON coming back from the `aws command line` we use `jq`. Note that `jq` is a lightweight command-line JSON processor. To download and install `jq` see the [jq download documents](https://stedolan.github.io/jq/download/).
Note that we created an AWS key pair called cloudurable-us-west-2. You will need to create a VPC security group with ssh access. You should lock it down to only accept ssh connections from your IP. At this stage, you can use a default VPC, and for now use a public subnet. Replace the ids above with your subnet (SUBNET_CLUSTER), your key pair (KEY_NAME_CASSANDRA), your AMI (AMI_CASSANDRA), and your IAM instance role (IAM_PROFILE_CASSANDRA). The IAM instance role should have access to create logs and metrics for AWS CloudWatch.

The login script (login-ec2-cassandra.sh) uses ssh to log into the instance, but to know what IP to use, it uses get-IP-cassandra.sh

Note to parse the JSON coming back from the aws command line we use jq. Note that jq is a lightweight command-line JSON processor. To download and install jq see the jq download documents.

Notice we use the aws ec2 wait to ensure the instance is ready before we tag it and before we log into it.
All of the ids for the servers AWS resources we need to refer to are in scripts/ec2-ens.sh.
Ensure you create a key pair in AWS. Copy it to ~/.ssh and then run `chmod 400` on the pem file. Note the above login script uses `bin/get-IP-cassandra.sh` to get the IP address.
Interactive session showing everything running

```
$ pwd
~/.github/cassandra-image
$ bin/create-ec2-instance.sh
1-0130ec36f1137f8c is being created
1-0130ec36f1137f8c was tagged waiting to login
The authenticity of host '54.282.110.114 (54.282.110.114)' can't be established.
ECDSA key fingerprint is SHA256:osmoj82 Randolph.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.282.110.114' (ECDSA) to the list of known hosts.

[centos@ip-172-31-5-57 ~]$ systemctl status cassandra
* cassandra.service -- Cassandra Service
  Loaded: loaded (/etc/systemd/system/cassandra.service; enabled; vendor)
  Active: active (running) since Wed 2017-03-01 02:15:03 UTC; 14min ago
  Process: 416 ExecStart=/opt/cassandra/bin/cassandra -p /opt/cassandra
  Main PID: 5240 (java)
  CGROUP: /system.slice/cassandra.service
      -- 5240 java -Xlogc:../opt/cassandra/bin/.../logs/c.log -ea
Mar 01 02:14:13 ip-172-31-3-183.us-west-2.compute.internal systemd[1]: S
Mar 01 02:15:10 ip-172-31-5-57 systemd[1]: Started Cassandra Service.

[centos@ip-172-31-5-57 ~]$ systemctl status metricsd
* metricsd.service -- Metricsd DM Metrics
  Loaded: loaded (/etc/systemd/system/metricsd.service; enabled; vendor)
  Active: active (running) since Wed 2017-03-01 02:15:10 UTC; 14min ago
  Main PID: 5243 (metricsd).
```


Notice that we use systemctl status systemd-cloud-watch, systemctl status cassandra, and systemctl status metricsd to ensure it is all working.
Ansible, Cassandra and EC2/AWS

- Configuration automation needed
- Puppet, Ansible, Chef, Boto, etc.
- Ansible is an agentless architecture and works over ssh (secure shell)
  - See Setting up Ansible for our Cassandra Cluster for DevOps for more details
- Many useful Ansible/AWS (beyond scope)
- The Ansible framework allows DevOps staff to run commands against Amazon EC2 instances as soon as they are available
- Ansible is useful provisioning hosts (works with Packer)
- Ansible is useful for DevOps tasks like replacing a failed node, backing up a node, profiling Cassandra, performing a rolling upgrade and more.
- Since Ansible relies on ssh, we should make sure that ssh is working for us.

**Ansible and EC2**

Although we have base images, since Cassandra is stateful, we will want the ability to update the images in place.

The options for configuration and orchestration management are endless (Puppet, Chef, Boto, etc.). This article and Cloudrable uses Ansible for many of these tasks. Ansible is an agentless architecture and works over ssh (secure shell) as we covered in our last article (Setting up Ansible for our Cassandra Cluster to do DevOps tasks). There are some very helpful Ansible/AWS integrations which will try to cover in future articles.

The Ansible framework allows DevOps staff to run commands against Amazon EC2 instances as soon as they are available. Ansible is very suitable for provisioning hosts in a Cassandra cluster as well as performing routine DevOps tasks like replacing a failed node, backing up a node, profiling Cassandra, performing a rolling upgrade and more.

Since Ansible relies on ssh, we should make sure that ssh is working for us.
We covered the CloudFormation the whole time we were covering AWS basics.
You can watch the CloudFormation logs. The logs are very helpful if something goes wrong.

After you are done running the CloudFormation, modify your ec2-env.sh with the output of the CloudFormation.
After you are done running the CloudFormation, modify your `ec2-env.sh` with the output of the CloudFormation.

You also need the AMI that packer created.
You can find more details here as well:

And here
How does it know which IP to listen on?

- We could listen on *listen_interface* (*cassandra.yml*)
- We listen on *listen_address*
- *But how do we know the address?*
Using EC2Snitch
The cassandra-cloud utility allows us to modify the cassandra.yaml file via EC2 User Data. The cassandra-cloud utility has an option called -snitch. The snitch options allows you to describe the Cassandra snitch type, examples, GossipingPropertyFileSnitch, PropertyFileSnitch, Ec2Snitch, etc. The cassandra-cloudutility defaults to "SimpleSnitch". If we instead specify Ec2Snitch, Cassandra will recognize AWS AZs as Cassandra racks.
EC2 Instance UserData and MetaData

- **Instance metadata** is data about the EC2 instance
- You can use the data to configure things
- *user data* is part of *meta-data*
  - Can be anything you want
  - If it is a script that then AWS will launch it when the instance first initializes
  - you use to configure an EC2 instance when its User Data script runs
# EC2 Instance Data

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>curl http://169.254.169.254/latest/meta-data/ami-id</code></td>
<td>am1-12345678</td>
</tr>
</tbody>
</table>

This example gets the list of available public keys.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>

This example shows the formats in which public key 0 is available.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
Checking the status of the Cassandra cluster

```plaintext
$ ssh-agent
$ ssh-add ~/.ssh/test_rsa
Identity added: ~/.ssh/test_rsa

$ ssh -F ssh/config bastion
Last login: Sat Mar 18 01:05:30 2017 from myhost.com

$ ssh 10.0.1.18
Last login: Sat Mar 18 01:03:37 2017 from ip-10-0-0-118.us-west-2.compute.internal

$ sudo systemctl status cassandra
● cassandra.service - Cassandra Service
   Loaded: loaded (/etc/systemd/system/cassandra.service; )
   Active: active (running) since Sat 2017-03-18 00:14:15 UTC; 57min ago
   Mar 18 00:14:18 ip-10-0-0-118 systemd[1]: Starting Cassandra Service...
   Mar 18 00:14:18 ip-10-0-0-118 systemd[1]: Started Cassandra Service.

$ /opt/cassandra/bin/nodetool status
Datacenter: us-west-2
=================================================================
Status=Up/Down    addr     LOAD  Tokens  Owns (effective) Host ID  Rack
UN 10.0.2.18      163.15 Kib  32  50.7%  e5993967-26e8-485b  2b
UN 10.0.1.18      112.84 Kib  32  52.2%  47fb9e8-8bce-468f  2a
UN 10.0.1.11      187.31 Kib  32  47.1%  ac344cc8-4991-47fe  2a
UN 10.0.2.11      221.87 Kib  32  50.1%  5dff0b8e-7cae-4e50  2b
```

STEP 4 MAKE SURE IT WORKS!

Setting up SSH and ansible Part 1
Setting up SSH and ansible Part 2
```
<table>
<thead>
<tr>
<th></th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Host *.us-west-2.compute.amazonaws.com</td>
</tr>
<tr>
<td>2</td>
<td>ForwardAgent yes</td>
</tr>
<tr>
<td>3</td>
<td>IdentityFile ~/.ssh/test_rsa</td>
</tr>
<tr>
<td>4</td>
<td>User ansible</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Host bastion</td>
</tr>
<tr>
<td>7</td>
<td>Hostname bastion.dev.us-west-2.cloudurable.com</td>
</tr>
<tr>
<td>8</td>
<td>ForwardAgent yes</td>
</tr>
<tr>
<td>9</td>
<td>IdentityFile ~/.ssh/test_rsa</td>
</tr>
<tr>
<td>10</td>
<td>User ansible</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Host cassandra.node</td>
</tr>
<tr>
<td>13</td>
<td>Hostname 10.0.1.10</td>
</tr>
<tr>
<td>14</td>
<td>ForwardAgent yes</td>
</tr>
<tr>
<td>15</td>
<td>IdentityFile ~/.ssh/test_rsa</td>
</tr>
<tr>
<td>16</td>
<td>ProxyCommand ssh bastion ~W @@hp</td>
</tr>
<tr>
<td>17</td>
<td>User ansible</td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Host 10.0.2.1+</td>
</tr>
<tr>
<td>20</td>
<td>ForwardAgent yes</td>
</tr>
<tr>
<td>21</td>
<td>IdentityFile ~/.ssh/test_rsa</td>
</tr>
<tr>
<td>22</td>
<td>ProxyCommand ssh bastion ~W @@hp</td>
</tr>
<tr>
<td>23</td>
<td>User ansible</td>
</tr>
<tr>
<td>24</td>
<td>ControlMaster auto</td>
</tr>
<tr>
<td>25</td>
<td>ControlPath ~/.ssh/ansible-4@@hp</td>
</tr>
<tr>
<td>26</td>
<td>ControlPersist 5m</td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Host 10.0.1.1+</td>
</tr>
<tr>
<td>29</td>
<td>ForwardAgent yes</td>
</tr>
<tr>
<td>30</td>
<td>IdentityFile ~/.ssh/test_rsa</td>
</tr>
<tr>
<td>31</td>
<td>ProxyCommand ssh bastion ~W @@hp</td>
</tr>
<tr>
<td>32</td>
<td>User ansible</td>
</tr>
<tr>
<td>33</td>
<td>ControlMaster auto</td>
</tr>
<tr>
<td>34</td>
<td>ControlPath ~/.ssh/ansible-4@@hp</td>
</tr>
<tr>
<td>35</td>
<td>ControlPersist 5m</td>
</tr>
</tbody>
</table>
```
This files /opt/cassandra/conf/cassandra.yaml were generated with the template /opt/cassandra/conf/cassandra-yaml.template by cassandra-cloud.
You can find cassandra-cloud at https://github.com/cloudurable/cassandra-cloud.
Cassandra Cloud is used to automate deployment to EC2 and similar cloud environments.
Questions

- Which servers are the seed servers?
- Which servers are in AZ a?
- Which servers are in AZ b?
Ec2MultiRegionSnitch
Cassandra nodes have to live in a VPC public subnet. You can use AWS NACL and Security group can tighten access. However, traffic is over public Internet so anyone can listen. You should use SSL to the cluster communication. Java performance for SSL is not great. Setup is more difficult, you must setup seeds that are public IPs. You can use AWS ENIs, AWS Route 53 or AWS EIPs to make access to seed nodes more sane. The Cassandra yaml config ssl_storage_port has to be open on the firewall (YUCK). Don’t use storage_port for public traffic unless you do not care who is listening.
Ec2MultiRegionSnitch 1 region
Ec2MultiRegionSnitch 2 regions

USA Oregon

USA Virginia
Ec2MultiRegionSnitch 4 regions

USA West Coast Oregon

USA East Coast Ohio

Brazil – South America

Ireland - Europe
Ec2MultiRegionSnitch

- `broadcast_address` must be set to public IP
- `listen_address` must be set to private IP
- `seed` nodes must be set
  - 1 seed per AZ, assume 3 AZ per region
  - 9 entries for 3 datacenters / regions
- Don’t make all nodes seeds (gossip too chatty)
- Ec2Snitch is easy because you know node IPs ahead of time
- If you use Ec2MultiRegionSnitch, use ENI to make public IP / seed server constant or use Route53 and put domain names as seeds
Let’s set up Cassandra for SSL and CQL clients, as well as installing Cassandra with SSL configured on a series of Linux servers. Cassandra allows you to secure the client transport (CQL) as well as the cluster transport (storage transport). SSL/TLS have some overhead. This is especially true in the JVM world which is not as performant for handling SSL/TLS unless you are using Netty/OpenSSL integration. Understanding SSL/TLS support for Cassandra is important for developers, DevOps and DBAs.

If possible, use no encryption for the cluster transport (storage transport), and deploy your Cassandra Database nodes in a private subnet, and limit access to this subnet to the client transport. Also if possible avoid using TLS/SSL on the client transport and do client operations from your app tier, which is located in a non-public subnet.
Encrypting the Cassandra transports
Data that travels over a network could be accessed by someone you don’t want accessing said data with tools like wire shark.
If data includes private information, SSN number, credentials (password, username), credit card numbers or account numbers, then we want to make that data unintelligible (encrypted) to any and all 3rd parties.
This is especially important if we don’t control the network.
TLS/SSL ensures data has not been tampered.
Cassandra is written in Java. Java defines the JSSE framework and Java Cryptography Architecture (JCA).

Encrypting the Cassandra transports
Data that travels over the client transport or storage transport across a network could be accessed by someone you don’t want accessing said data with tools like wire shark. If data includes private information, SSN number, credentials (password, username), credit card numbers or account numbers, then we want to make that data unintelligible (encrypted) to any and all 3rd parties. This is especially important if we don’t control the network. You can also use TLS to make sure the data has not been tampered with whilst traveling the network. The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols are designed to provide these features (SSL is the old name for what became TLS but many people still refer to TLS as SSL).

Cassandra is written in Java. Java defines the JSSE framework which in turn uses the Java Cryptography Architecture (JCA). JSSE uses cryptographic service providers from JCA. If any of the above is new to you, please take a few minutes to read through the TLS/SSL Java guide.
This article picks up right after this one – Setting up a Cassandra cluster with cassandra image and cassandra cloud project with Vagrant.
Before we go into the details of setting up the cassandra.yaml file, let's create some trust stores, key stores, and export some keys. The following script generates cluster and client keys.

```
#!/bin/bash

KEY_STORE_PATH="$PWD/resources/opt/cassandra/conf/certs"
mkdir -p "$KEY_STORE_PATH"
KEY_STORE="$KEY_STORE_PATH/cassandra.keystore"
PKS_KEY_STORE="$KEY_STORE_PATH/cassandra.pks12.keystore"
TRUST_STORE="$KEY_STORE_PATH/cassandra.truststore"
PASSWORD=cassandra
CLUSTER_NAME=test
CLUSTER_PUBLIC_CERT="$KEY_STORE_PATH/CLUSTER_$CLUSTER_NAME_PUBLIC.cer"
CLIENT_PUBLIC_CERT="$KEY_STORE_PATH/CLIENT_$CLUSTER_NAME_PUBLIC.cer"
```
The `keytool` utility ships with Java SDK. We use this `keytool` command to create the cluster key. Let's break down the script that generates the keys and certificates.

```
keytool -genkey -keyalg RSA -alias "$CLUSTER_NAME" -keystore "$KEY_STORE" -storepass "$PASSWD"
-dname "CN=Cloudurable, O=Cloudurable, C=USA, DC=cloudurable, DC=com"
-validity 36500
```
Once we create the cluster key, we will want to export a public key from it.
Then we will import the public key into the trust store so that nodes can identify each other.

```bash
# Import the identity of the cluster public cluster key into the trust store so that nodes can identify each other.
keytool -import -v -trustcacerts -alias "${CLUSTER_NAME}_CLUSTER" -file "${CLUSTER_PUBLIC_CERT}" -keystore "${TRUST_STORE}"
-storepass "$PASSWORD" -keypass "$PASSWORD" -noprompt
```
We perform the same three tasks for the client keys. Then lastly we create pem files for the client keys by exporting our Java JKS keystore as a PKCS12 trust store.
SSL Files Generated

Cassandra Cert files, key stores, trust stores, private keys for SSL

```
$ pwd
~/github/cassandra-image

$ ls resources/opt/cassandra/conf/certs/
CLIENT_test_PUBLIC.cer  cassandra.pks12.keystore  test_CLIENT.key.pem
CLUSTER_test_PUBLIC.cer  cassandra.truststore
 cassandra.keystore  test_CLIENT.cer.pem
```
SSL Files Explained

- CLIENT_test_PUBLIC.cer
  - public client key for the test cluster.

- cassandra.pks12.keystore
  - PKS12 keystore for client used to generate pem

- test_CLIENT.key.pem
  - private client key in pem format used by csqsh

- CLUSTER_test_PUBLIC.cer
  - public cluster key for the test cluster

- cassandra.truststore
  - Trust store used by cassandra

- cassandra.keystore
  - Key store used by cassandra

- test_CLIENT.cer.pem
  - public client key in pem format used by csqsh

For the cassandra_image project, these files are copied to /opt/cassandra/conf/cert. To learn more about our Vagrant project see Setting up a Cassandra cluster with cassandra image and cassandra cloud project with Vagrant.
As part of the provision script for `cassandra_image` (see Setting up a Cassandra cluster with cassandra_image and cassandra cloud project with Vagrant). We added the above.

This will copy the certs to the right location if you generated a folder in resources (cassandra_image/resources/opt/cassandra/conf/cert), which the last script that we covered does.
Configure Cassandra to use the keys

```
server_encryption_options:
  inter-node_encryption: all
  keystore: /opt/cassandra/conf/certs/cassandra.keystore
  keystore_password: cassandra
  truststore: /opt/cassandra/conf/certs/cassandra.truststore
  truststore_password: cassandra

# More advanced defaults below:
  protocol: TLS

client_encryption_options:
  enabled: true
  # If enabled and optional is set to true encrypted and unencrypted connections are handled.
  optional: false
  keystore: /opt/cassandra/conf/certs/cassandra.keystore
  keystores_password: cassandra
  truststore: /opt/cassandra/conf/certs/cassandra.truststore
  truststore_password: cassandra
  require_client_auth: true
  protocol: TLS
```
Remaining tasks for Ec2MultiRegionSnitch

- Parameterize the CloudFormation so we can pass region
- Parameterize the CloudFormation so we can pass CIDR
- Change Snitch types
- Use ENI or DNS or hack 😊
- Ec2MultiRegionSnitch

Changes to CloudFormation and scripts are described here: [Cassandra Tutorial 7: Setting up Cassandra Cluster in EC2 Part 2 Multi Region with Ec2MultiRegionSnitch](#)
Parameterizing CloudFormation

Add Parameters to Cassandra Cluster CloudFormation

```json
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Setup VPC for Cassandra Cluster for Cassandra Database",
  "Parameters": {
    "vpcCidr": {
      "Description": "Enter VPC CIDR",
      "Type": "String",
      "Default": "10.1.0.0/16",
      "AllowedValues": [
        "10.1.0.0/16",
        "10.2.0.0/16",
        "10.3.0.0/16"
      ]
    },
    "subnetPublicCidr": {
      "Description": "Enter Public Subnet CIDR",
      "Type": "String",
      "Default": "16.1.0.0/24",
      "AllowedValues": [
        "16.1.0.0/24",
        "16.2.0.0/24",
        "16.3.0.0/24"
      ]
    },
    "subnetClusterCidr": {
      "Description": "Enter Cluster Subnet CIDR",
      "Type": "String"
    }
  }
}
```
Parameterize launch script

bin/ec2-env-region-us-west-2.sh

```bash
...
export SUBNET_VPC_CIDR=10.1.0.0/16
export SUBNET_PUBLIC_CIDR=10.1.0.0/24
export SUBNET_CLUSTER1_CIDR=10.1.1.0/24
export SUBNET_CLUSTER2_CIDR=10.1.2.0/24
```

bin/ec2-env-region-us-east-2.sh

```bash
...
export SUBNET_VPC_CIDR=10.1.0.0/16
export SUBNET_PUBLIC_CIDR=10.1.0.0/24
export SUBNET_CLUSTER1_CIDR=10.1.1.0/24
export SUBNET_CLUSTER2_CIDR=10.1.2.0/24
```
Pass CIDR and regions to run-vpc-cloudformation.sh

```
#!/usr/bin/env bash
set -e
source bin/ec2-env.sh

# Set aws-region
if [ -z "$REGION" ];
    then
        AWS_REGION=$REGION
    else
        AWS_REGION=$1
fi

source bin/ec2-env-region.sh

aws --region $AWS_REGION s3 cp cloud-formation/vpc.json s3://$CLOUD_FORMER_S3_BUCKET
aws --region $AWS_REGION cloudformation create-stack --stack-name $(ENV)-vpc-cassandra
    --template-url "https://s3-us-west-2.amazonaws.com/$CLOUD_FORMER_S3_BUCKET/vpc.json"
    --parameters ParameterKey=vpcCidr,ParameterValue=${SUBNET_VPC_CIDR} \
    ParameterKey=subnetPublicCidr,ParameterValue=${SUBNET_PUBLIC_CIDR} \n    ParameterKey=subnetCluster1Cidr,ParameterValue=${SUBNET_CLUSTER1_CIDR} \n    ParameterKey=subnetCluster2Cidr,ParameterValue=${SUBNET_CLUSTER2_CIDR}
```
When launching instances specify the region

Launching 4 Cassandra Database servers and a bastion into each region

# Create Bastion Node for Oregon/us-west-2
bin/create-ec2-instance-bastion.sh us-west-2

# Create Cassandra Nodes for Oregon/us-west-2
bin/create-ec2-instance-cassandra.sh 18.1.1.10 a us-west-2
bin/create-ec2-instance-cassandra.sh 18.1.2.10 b us-west-2
bin/create-ec2-instance-cassandra.sh 18.1.1.11 a us-west-2
bin/create-ec2-instance-cassandra.sh 18.1.2.11 b us-west-2

# Create Bastion for Ohio/us-east-2
bin/create-ec2-instance-cassandra.sh us-east-2

# Create Cassandra Nodes for Ohio/us-east-2
bin/create-ec2-instance-cassandra.sh 18.2.1.10 a us-east-2
bin/create-ec2-instance-cassandra.sh 18.2.2.10 b us-east-2
bin/create-ec2-instance-cassandra.sh 18.2.1.11 a us-east-2
bin/create-ec2-instance-cassandra.sh 18.2.2.11 b us-east-2
Here we use an ansible playbook to run cassandra cloud
The ansible script uses curl and the EC2 metadata to figure out the broadcast address and the private ip address.
Run playbook

Running playbooks/connect.yml

$ ansible-playbook playbooks/connect-regions.yml

PLAY [all-nodes] **********************************************************************************************

TASK [Copy template] **********************************************************************************************
ok: [10.1.1.10]
ok: [10.1.1.11]
ok: [10.1.2.31]
ok: [10.1.2.10]
ok: [10.2.1.10]
ok: [10.2.1.11]
ok: [10.2.2.11]
ok: [10.2.2.10]

PLAY RECAP **********************************************************************************************
10.1.1.10 : ok=5  changed=4  unreachable=0  failed=0
10.1.1.11 : ok=5  changed=4  unreachable=0  failed=0
10.1.2.31 : ok=5  changed=4  unreachable=0  failed=0
10.1.2.10 : ok=5  changed=4  unreachable=0  failed=0
10.2.1.10 : ok=5  changed=4  unreachable=0  failed=0
10.2.1.11 : ok=5  changed=4  unreachable=0  failed=0
10.2.2.11 : ok=5  changed=4  unreachable=0  failed=0
10.2.2.10 : ok=5  changed=4  unreachable=0  failed=0
The ansible inventory list is as follows

[seed1]
10.1.1.10

[seed2]
10.2.1.10

[seeds]
10.1.1.10
10.1.2.10
10.2.1.10
10.2.2.10

[dc1-nodes]
10.1.1.10
10.1.1.11
10.1.2.10
10.1.2.11

[dc2-nodes]
10.2.1.10
10.2.1.11
10.2.2.10
10.2.2.11
Connecting Cluster via VPN
Using VPN instead of public Internet for Multi DC/Region Cassandra Setup

- Enable multi-region cluster communication via Internet Protocol Security (IPsec) tunnels
  - Use VPN
  - Private IP address are used for cross-region communication
- Company VPN is usually connected to VPG
- Setup VPC peering between VPC for Cassandra cluster and Company VPC connected to VPN via VPG
- Recommend to use Ec2Snitch or GossipingPropertyFileSnitch
- The CIDR addresses for the VPNs cannot overlap.
  - Care must be taken in setting up multiple private VPCs that can communicate cross region
GossippingPropertyFileSnitch to VPN
Gossipping Property File Snitch via VPN via VPC Peer
Cassandra / Kafka Support in EC2/AWS

Gossipping Property File Snitch

4 regions / DCs

VPC
Private Subnet 1
for Cassandra Cluster
Seeds CN1, CN2, CN3, CN4
VPN Subnet 1
AZ: us-west-2

AWS Region: us-west-2

Company VPC For VPN

VPC
Private Subnet 2
for Cassandra Cluster
Seeds CN2, CN3, CN4
VPN Subnet 2
AZ: us-west-2

Company VPC For VPN

VPC
Private Subnet 3
for Cassandra Cluster
Seeds CN1, CN2, CN3, CN4
VPN Subnet 3
AZ: us-east-2

Company VPC For VPN

VPC
Private Subnet 4
for Cassandra Cluster
Seeds CN1, CN2, CN3, CN4
VPN Subnet 4
AZ: eu-west-1

Company VPC For VPN

VPC
Private Subnet 5
for Cassandra Cluster
Seeds CN1, CN2, CN3, CN4
VPN Subnet 5
AZ: sa-east-1

Company VPC For VPN

VPC
Private Subnet 6
for Cassandra Cluster
Seeds CN1, CN2, CN3, CN4
VPN Subnet 6
AZ: us-east-1

Company VPC For VPN

VPC
Private Subnet 7
for Cassandra Cluster
Seeds CN1, CN2, CN3, CN4
VPN Subnet 7
AZ: eu-west-1

Company VPC For VPN

VPC
Private Subnet 8
for Cassandra Cluster
Seeds CN1, CN2, CN3, CN4
VPN Subnet 8
AZ: sa-east-1

Company VPC For VPN

VPC
Private Subnet 9
for Cassandra Cluster
Seeds CN1, CN2, CN3, CN4
VPN Subnet 9
AZ: us-east-1

Company VPC For VPN
Final Thoughts
Engineering Tradeoffs

- Less servers and scale up
  - Potential for reduce AWS spend
  - Perhaps harder to horizontal scale if you have to (more streaming when you bring up node)
- There may not be an exact match EC2 instance
  - Do you have lots of spare memory? Bump up the key caches, and IO buffers (use what you have)
  - Do you have spare CPU? Enable compression to minimize IO overhead (first SSTable, then commit log, then internode communication all)
### Company Overview

<table>
<thead>
<tr>
<th>How we got our start</th>
</tr>
</thead>
</table>

Cassandra / Kafka Support in EC2/AWS
Different companies same challenges

- How to setup a Cluster across multiple AZs
- Where does enhanced networking fit it
- Should we use EBS or instance storage
- Monitoring and logging that can be actionable
- Integration with AWS services like CloudFormation, and CloudWatch.
- Best fit for images, VPC setup, peering, subnets, firewalls
Services we provide

- Training
- Consulting
- Setting up Cassandra in AWS/EC2
- AWS CloudFormations
- Subscription Support around Cassandra running in AWS/EC2
  - AWS CloudWatch monitoring
  - AWS CloudWatch logging
Cassandra Cluster Tutorial 3: Part 2 of 2 Setting up Ansible and SSH for our Cassandra Database Cluster for DevOps/DBA Tasks

This tutorial series centers on how DevOps/DBA tasks with the Cassandra Database. As we mentioned before, Ansible and ssh are essential DevOps/DBA tools for common DBA/DevOps tasks whilst working with Cassandra Clusters. Please read part 1 before reading part 2. In part 1, we set up Ansible for our Cassandra Database Cluster to automate common DevOps/DBA tasks.

Cassandra Cluster Tutorial 3: Part 1 of 2 Setting up Ansible/SSH for our Cassandra Database Cluster for DevOps/DBA Tasks

Ansible and ssh are essential DevOps/DBA tools for common DBA/DevOps tasks like managing backups, rolling upgrades to the Cassandra cluster in AWS/EC2, and so much more. An excellent aspect of Ansible is that it uses ssh, so you do not have to install an agent to use Ansible. This article series centers on how DevOps/DBA tasks with the Cassandra Database.
SETTING UP A CASSANDRA CLUSTER WITH SSL FOR CLIENT AND CLUSTER TRANSPORTS FOR DEVOPS
IN CASSANDRA
February 2, 2017
Setting up client and cluster SSL transport for a Cassandra cluster. This article is a Cassandra tutorial on Cassandra setup for SSL and CQL clients, as well as installing Cassandra with SSL configured on a series of Linux servers. Cassandra allows you to secure the client transport (CQL) as well as the cluster transport (storage transport). SSL/TLS have some overhead. This is especially true in the JVM world which is not as performant for handling SSL/TLS unless you are using Netty/OpenSSLI integration.

SETTING UP A CASSANDRA CLUSTER WITH CASSANDRA IMAGE AND CASSANDRA CLOUD PROJECT WITH VAGRANT FOR DEVOPS
IN CASSANDRA
February 1, 2017
The cassandra-image project creates CentOS Cassandra Database images for docker, virtualbox/vagrant and AWS/EC2 using best practices for Cassandra OS setup. It is nice to use vagrant and/or docker for local development. At this time it is hard to develop system services using Docker so we use Vagrant. Since we do a lot of that, we like to use Vagrant. Vagrant is important for developers and DevOps not to mention Cassandra DBAs.

CONTINUE READING