



- March, 22, 2017
- Accurate and up to date
- Comprehensive guide
- Working examples with src
- Ec2MultiRegionSnitch
- EC2Snitch
- Broadcast address
- Using KMS
- SSL config
- Ansible / SSH Config
- Bastions
- Private Subnets
- VPN Multi-Region Cassandra
- Using enhanced networking
- Using new EBS elastic volumes

Cassandra and AWS Support on AWS/EC2

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

Cloudurable 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

Amazon Region and Availability Zones

- ❖ AWS supports regions around the world
- ❖ Regions are independent of each other
 - ❖ place services in a region to be closer to your end consumer to lower latency and to improve reliability
- ❖ *Availability Zone (AZ)* are isolated - Multiple AZs live in a region
- ❖ AZ protects against outage
- ❖ Placing your services and application in different Azs
- ❖ AZs have 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.

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, and often use different utility companies when possible. AZs may 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

EC2 Compute

- ❖ Resizable compute capacity in the cloud
- ❖ Compute: computational power needed for your use case
- ❖ Add compute resources as needed (IaaS)
- ❖ EC2 allows you to launch *instances*
- ❖ instance is a server
- ❖ install whatever software you need: NGINX, Apache httpd, Cassandra, Kafka, etc.
- ❖ Pay for compute power that you use
- ❖ Different instance types with various ranges of CPU, RAM, IO, and networking power
- ❖ Pay for compute resources by hour or longer

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)

AWS EC2 AMI

- ❖ AMI is a Amazon Machine Image (AMI)
- ❖ Contain info on how to launch EC2 instance
- ❖ AMI are specified when an EC2 instance is launched
- ❖ Has template for the root volume for the instance
- ❖ Tied to an account or public

Lives in one region, but can be copied:

```
aws ec2 copy-image --source-region us-west-2 \  
    --source-image-id ami-6db3310d \  
    --name CassandraClusterAMI
```

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
```

Cloudurable Cassandra AWS Support

Amazon Elastic Block Storage

Virtual volumes
SSD
Magnetic

Elastic Block Storage (EBS)

- ❖ Amazon Web Services (AWS) provides **Amazon Elastic Block Store (Amazon EBS)** for EC2 instance storage
- ❖ EBS virtual hard drives and SSDs for your virtual servers (EC2 instances)
- ❖ EBS volumes are automatically replicated in same AZ
- ❖ Easy to take snapshots of volumes (back up)
- ❖ Advantages: reliability, snapshotting, resizing

Getting the most bang for your buck with AWS Elastic Block Store (EBS)

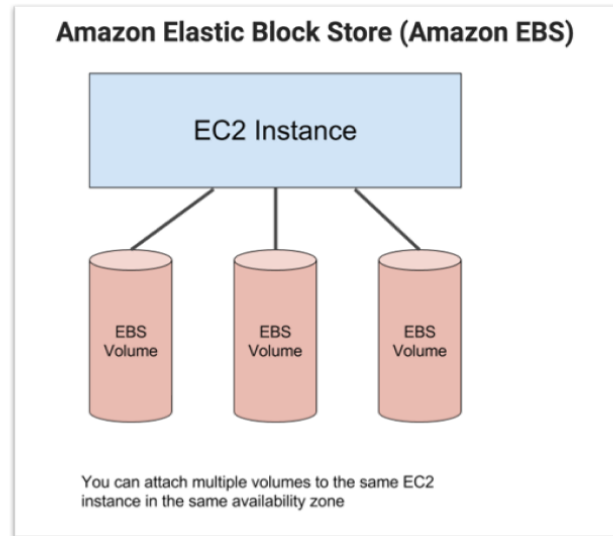
Understanding what AWS/EC2 provides for provisioning on-demand storage is critical for DevOps. Companies waste tons by over provisioning AWS.

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.

AWS EBS



EBS Volume types

- ❖ Four EBS volume types
- ❖ Two types of Hard Disk Drives (HDD) (Magnetic)
- ❖ Two types of SSDs
- ❖ Volumes differ in price and performance
- ❖ EC2 instance can have *many EBS volumes* attached
- ❖ EBS volume can only be attached to *one EC2 instance at a time*

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.

Good use cases for Magnetic Volumes

- ❖ streaming workloads which require cost effective, fast, consistent I/O
- ❖ big data
- ❖ data warehouses
- ❖ log processing
- ❖ Databases which use structured merge tree

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 that employ log structured merge tree

Two types of 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?

There are two types of HDD -
Magnetic Volumes:
st1 - Throughput Optimized HDD
sc1 - Cold HDD and most cost effective

General-Purpose SSD (gp2) 1

- ❖ Cost effective, and useful for many workloads.
- ❖ Minivan of EBS
- ❖ Performance of 3 IOPS per gigabyte provisioned
 - ❖ 250 GB volume you can expect a baseline of 750 IOPS
- ❖ Peak capped @ 10,000 IOPS
- ❖ Sizes range from 1 GB to 16 TB
- ❖ Use Cases: Databases that use some form of [BTrees](#) (MongoDB, MySQL, etc.).
- ❖ Geared to a **lower volume database** or one that has peak load times but long periods at rest where IOPS credits can accumulate

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 **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 **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.

General-Purpose SSD (gp2) 2

- ❖ Can be used for boot volumes
- ❖ Under 1 TB these volumes burst to 3,000 IOPS for extended periods of time
- ❖ Unused IOPS get accumulated as IOPS credits which can be used with bursting
- ❖ IOPS credits is like a savings account
 - ❖ As you are using it, the bank account is being withdrawn from
- Use Case
 - A server than does periodic batch or cron jobs
 - Low-latency interactive apps
 - Medium-sized databases

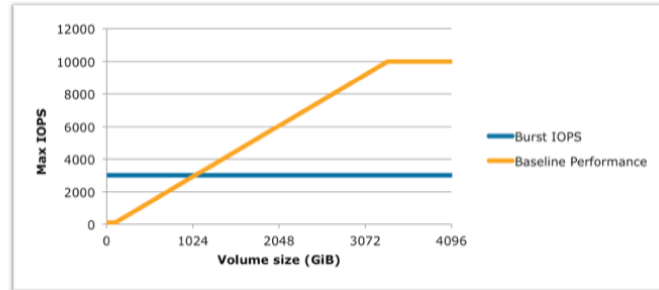
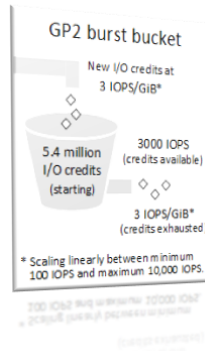


Could you use this for Cassandra? If your Cassandra Cluster had 12 nodes And you got max 12,000 reads per second across the cluster and max 120,000 writes per second what size gp2 would work per node assuming the cluster grows 2 TB per year?

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 B-Trees (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 increase 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.

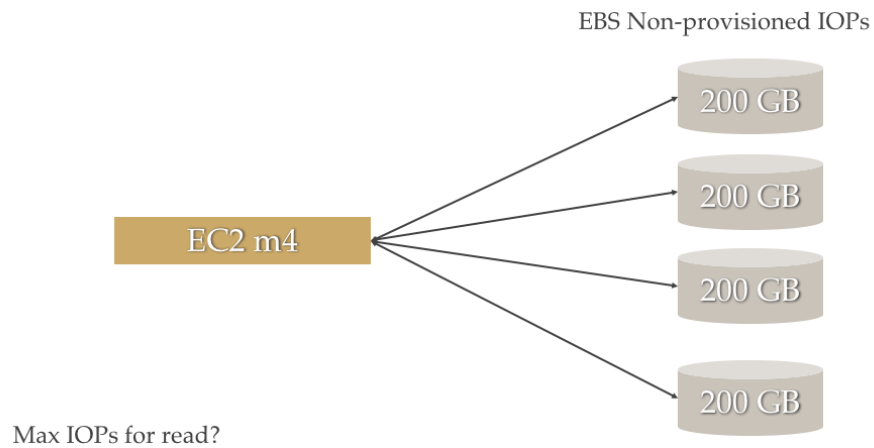
EBS Type Review

	Solid-State Drives (SSD)		Hard disk Drives (HDD)	
Volume Type	General Purpose SSD (gp2)*	Provisioned IOPS SSD (io1)	Throughput Optimized HDD (st1)	Cold HDD (sc1)
Description	General purpose SSD volume that balances price and performance for a wide variety of transactional workloads	Highest-performance SSD volume designed for mission-critical applications	Low cost HDD volume designed for frequently accessed, throughput-intensive workloads	Lowest cost HDD volume designed for less frequently accessed workloads
Volume Size	1 GiB - 16 TiB	4 GiB - 16 TiB	500 GiB - 16 TiB	500 GiB - 16 TiB
Max. IOPS**/Volume	10,000	20,000	500	250
Max. Throughput/Volume†	160 MiB/s	320 MiB/s	500 MiB/s	250 MiB/s
Max. IOPS/Instance	65,000	65,000	65,000	65,000
Max. Throughput/Instance	1,250 MiB/s	1,250 MiB/s	1,250 MiB/s	1,250 MiB/s
Dominant Performance Attribute	IOPS	IOPS	MiB/s	MiB/s



How can Cassandra use HDD and get 1,000 IOPs? 3 ways

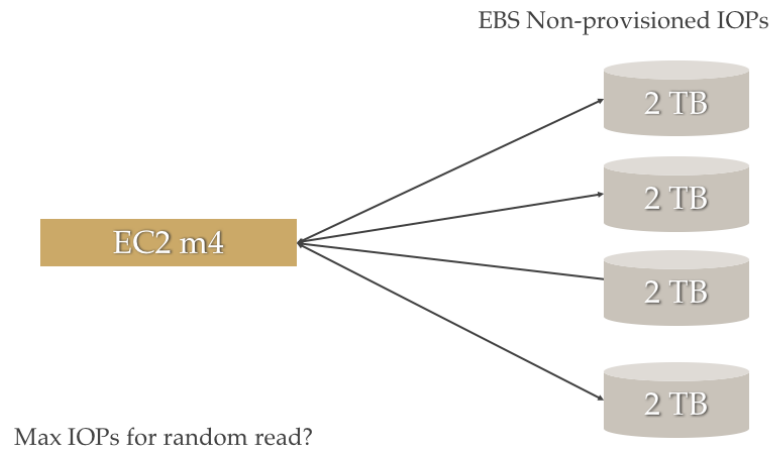
EBS Volumes



$$3 * 200G = 600 \text{ IOPs}$$

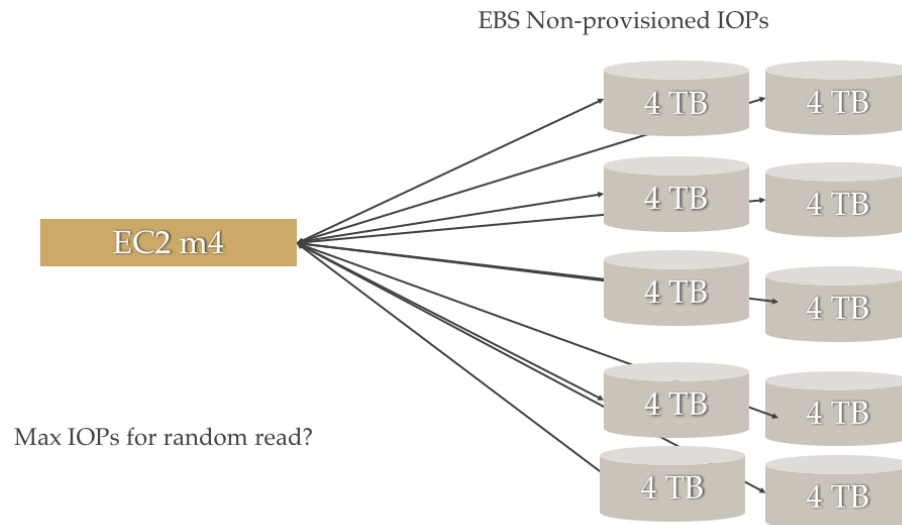
$$600 \text{ IOPs} * 4 \text{ EBS VOLUMES} = 2,400 \text{ IOPs}$$

EBS Volumes



2,000 GB * 3 IOPs * EBS volumes =
24,000 IOPs

EBS Volumes



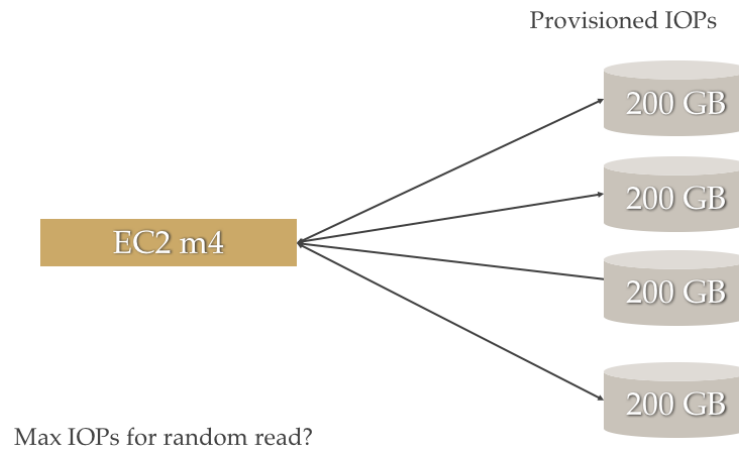
4,000,000,000,000 (TOTAL BYTES) / 1,000,000,000 (1 BILLION/GIG) * 3 IOPs per GIG * 10 EBS volumes = 120,000 IOPs

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.”

EBS Volumes



20,000 IOPs per Volume = 80,000 but
exceeds 65,000 so 65,000

EBS Volumes



Max IOPs for random read?

825,000 IOPs

Now Available – I3 Instances for Demanding, I/O Intensive Applications

by Jeff Barr | on 23 FEB 2017 | in [Amazon EC2](#), [Launch](#) | [Permalink](#) | [Comments](#)

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 super-efficient NVMe SSD storage, these instances can deliver up to 3.3 million IOPS at a 4 KB 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 disk-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.

Instance Name	vCPU Count	Memory	Instance Storage (NVMe SSD)	Price/Hour
i3.large	2	15.25 GiB	0.475 TB	\$0.15
i3.xlarge	4	30.5 GiB	0.950 TB	\$0.31
i3.2xlarge	8	61 GiB	1.9 TB	\$0.62
i3.4xlarge	16	122 GiB	3.8 TB (2 disks)	\$1.25
i3.8xlarge	32	244 GiB	7.6 TB (4 disks)	\$2.50
i3.16xlarge	64	488 GiB	15.2 TB (8 disks)	\$4.99

NOT EBS: Instance storage

- ❖ Don't forget you don't have to use EBS
- ❖ Instance 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](#).”

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSOptimized.html#enable-ebs-optimization>

EBS: Don't just guess measure

- ❖ Make educated guess to pick the right EBS based on workload
 - ❖ Deploying Kafka or Cassandra or MongoDB then you must understand how to configure the tool
 - ❖ Smaller nodes but more of them, or less nodes with larger EBS volumes
 - ❖ JBOD, RAID 1, etc.
- ❖ Use [Amazon CloudWatch](#)
 - ❖ watch IOPs and IO throughput
 - ❖ while load testing or watching production workloads
 - ❖ quickest and best way to pick best EBS volume type

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 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.

Snapshots - EBS backups

- ❖ **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?

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. You do this by taking snapshots. Snapshots are point-in-time backups. Data written to an EBS volume can be periodically used to create a snapshot. Snapshots provide incremental backups of your data. Snapshots just saves the blocks that have changed. Only changed blocks since the last snapshot are saved in the new snapshot.

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 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

Using command line to create a snapshot of a volume

```
aws ec2 create-snapshot --volume-id vol-1234567890abcdef0
```

Taking EBS Snapshots

Snapshots are done with:

- AWS Management Console
- Scheduled snapshots
- AWS API
- AWS CLI

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 be 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



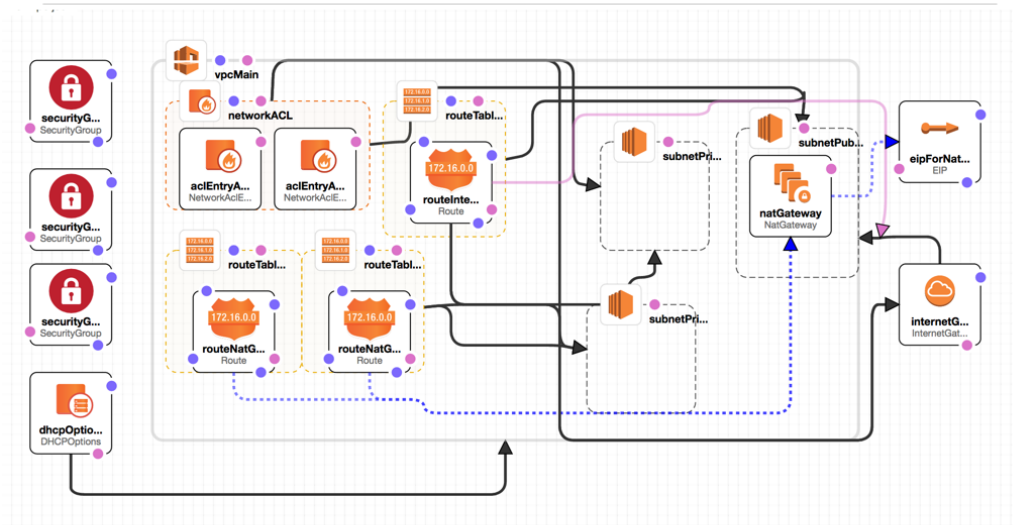
You wrote a Chef or Ansible script to update the JDK and Cassandra.
Should you perform a snapshot before you run this?

Cloudurable Cassandra AWS Support

VPC

Software defined networking

VPC: 1 public and two private subnets



Amazon VPC

- ❖ Software defined networking
- ❖ Virtual private cloud
- ❖ Multiple VPCs can live in a AWS region
- ❖ VPC can span multiple availability zones
- ❖ Isolated area to deploy Amazon EC2 instances
- ❖ Associated with a [CIDR block](#)
- ❖ DHCP Options

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 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

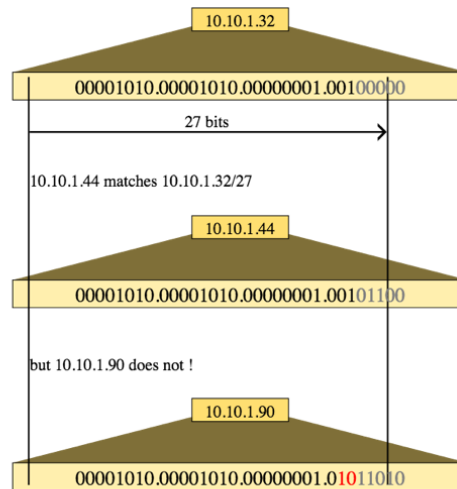
your region

CIDR Block

- ❖ `/#` denotes the size of the network
 - ❖ how many bits of the address will be used for the network
- ❖ Example: `10.10.1.32/27` denotes a CIDR range (also known as CIDR block).
- ❖ First 27 bits of address is for the network (32 bits total)
- ❖ $32 - 27$ leaves five bits for your servers. 00000-11111
- ❖ First five addresses are reserved in a subnet, and the last address is reserved for broadcast
- ❖ Example leaves us 26 addresses for our servers (`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)
- ❖ Addresses of two VPC should not overlap if you plan on adding VPC peering.

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. 00000-11111. The 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

Components of VPC

- ❖ Made up of subnets, route tables, DHCP option sets, security groups, and Network ACLs.
- ❖ 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

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).

CloudFormation for VPC

```
"vpcMain": {
  "Type": "AWS::EC2::VPC",
  "Properties": {
    "CidrBlock": "10.0.0.0/16",
    "InstanceTenancy": "default",
    "EnableDnsSupport": "true",
    "EnableDnsHostnames": "true",
```

CIDR Range	10.0.0.0/16
Netmask	255.255.0.0
Wildcard Bits	0.0.255.255
First IP	10.0.0.0
Last IP	10.0.255.255
Total Host	65536

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"
    },
  },
}
```

```
"subnetPrivate1": {
  "Type": "AWS::EC2::Subnet",
  "Properties": {
    "CidrBlock": "10.0.1.0/24",
    "AvailabilityZone": "us-west-2a",
    "VpcId": {
      "Ref": "vpcMain"
    },
  },
}
```

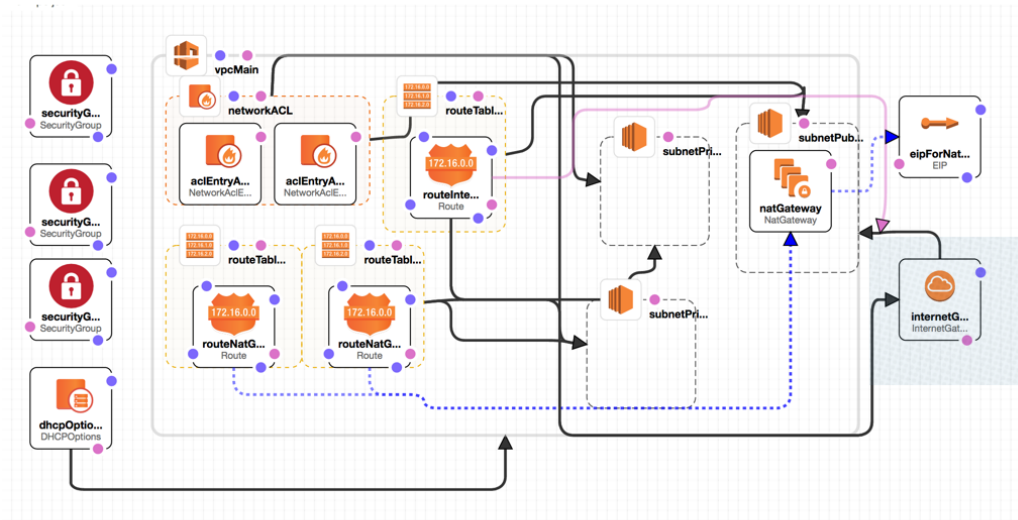
```
"subnetPrivate2": {
  "Type": "AWS::EC2::Subnet",
  "Properties": {
    "CidrBlock": "10.0.2.0/24",
    "AvailabilityZone": "us-west-2b",
    "VpcId": {
      "Ref": "vpcMain"
    },
  },
}
```

CIDR Range	10.0.0.0/24
Netmask	255.255.255.0
Wildcard Bits	0.0.0.255
First IP	10.0.0.0
Last IP	10.0.0.255
Total Host	256

CIDR Range	10.0.1.0/24
Netmask	255.255.255.0
Wildcard Bits	0.0.0.255
First IP	10.0.1.0
Last IP	10.0.1.255
Total Host	256

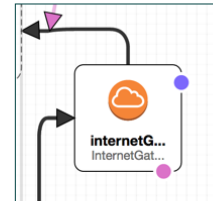
CIDR Range	10.0.2.0/24
Netmask	255.255.255.0
Wildcard Bits	0.0.0.255
First IP	10.0.2.0
Last IP	10.0.2.255
Total Host	256

Internet Gateway



Internet Gateway

- ❖ Internet Gateway (**IGW**) enables inbound traffic from the public Internet to your VPC
- ❖ Public subnets have route tables that target IGW
- ❖ IGW does network address translation from public IPs of EC2 instances to their private IP
- ❖ EC2 instance send IP traffic from a public subnet, the IGW acts as the NAT for public subnet,
 - ❖ translates the reply address to the EC2 instance's public IP (EIP)
- ❖ IGW keep track of the mappings of EC2 instances private IP address and their public IP address
- ❖ Highly available and handles the horizontal scale, redundancy as needed



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.

The diagram illustrates a VPC Main configuration. It includes a central VPC Main box with several subnets: subnetPri..., subnetPri..., and subnetPub... A natGateway NatGateway is connected to subnetPub... and InternetG... InternetG... is connected to eipForNat... EIP. The diagram also shows a networkACL, routeTabl..., routeTabl..., routeTabl..., routeTabl..., routeNatG..., and routeNatG... connected to the VPC Main. Security groups (securityG...) and DHCP options (dhcpOptio...) are also shown.

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).

CF: Route from Pub Subnet to IGW

```
"routeTablePublic": {
  "Type": "AWS::EC2::RouteTable",
  "Properties": {
    "VpcId": {
      "Ref": "vpcMain"
    }
  },
}
```

```
"subnetRouteTableAssociationPublic": {
  "Type": "AWS::EC2::SubnetRouteTableAssociation",
  "Properties": {
    "RouteTableId": {
      "Ref": "routeTablePublic"
    },
    "SubnetId": {
      "Ref": "subnetPublic"
    }
  },
}
```

```
"routeInternetGateway": {
  "Type": "AWS::EC2::Route",
  "Properties": {
    "DestinationCidrBlock": "0.0.0.0/0",
    "RouteTableId": {
      "Ref": "routeTablePublic"
    },
    "GatewayId": {
      "Ref": "internetGateway"
    }
  },
}
```

The above shows the Amazon CloudFormation from the public subnet defined earlier to the Internet Gateway.

VPC VPN Access via VGW and CGW

- ❖ AWS to augment your existing IT infrastructure via VPN
- ❖ Connect existing datacenter to VPC using *VP*G (*Virtual Private Gateways*) and *CGW* (*Customer Gateways*)
- ❖ *VGW* like the *IGW* but it sends traffic to / fro your corporate network instead of the public Internet
- ❖ *VP*Gs connect to your companies - *VP*G is the Amazon side of the *VPN* connection
- ❖ *CGW* is the customer side of the *VPN* connector
- ❖ *CGW*s are processes running on a server or network device.
- ❖ Connect a *VP*G and a *CGW* with a *VPN tunnel*
- ❖ Uses the *IPSec* to connect VPC to corporate network
- ❖ Use dynamic routing or static routes



Which subnets in a given VPC would have access to the corporate internet connected via the VPN?

Amazon *VP*Gs, *CGW*s and *VP*Ns

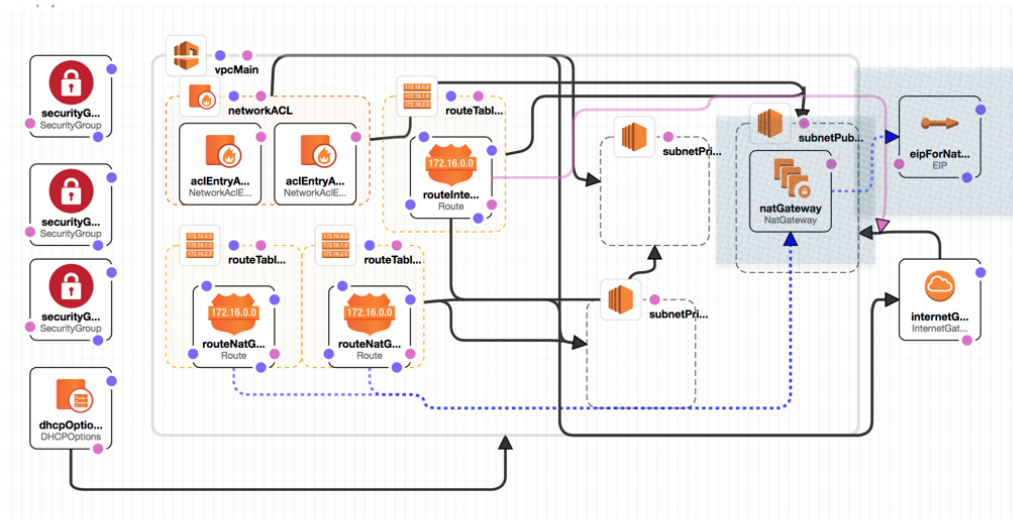
Amazon allows VPCs to be connected to your existing data center to allow AWS to augment your existing IT infrastructure. You can connect your existing datacenter to an Amazon VPC using *VP*G (*Virtual Private Gateways*) and *CGW* (*Customer Gateways*).

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

You connect a *VP*G 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*.

NAT Gateway and EIP



The above shows that we are covering Elastic IPs and NatGateways next.

Elastic IP (EIP)

- ❖ AWS pool of public IP addresses - Available to rent per region
- ❖ Check out EIPs to use and assign - Allows you to keep same Public IP
- ❖ Example: Assign an EIP to an instance (and only one)
 - ❖ Spin up a new upgraded version of the instance from a snapshot or with Ansible, Chef, etc.
 - ❖ Reassign the EIP to the new upgraded instance.
- ❖ Allow public IPs to be reassigned to new underlying infrastructure
- ❖ Allocated in a VPC, can be moved to another same region VPC
- ❖ Assigned to resources like EC2 instances, Nat Gateways, etc.

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.

Nat Gateways

- ❖ Needed so Amazon EC2 instances launched in a private subnet cannot access the Internet
- ❖ NAT is a network address translator
- ❖ Why? *yum install foo*, you could not do it because instance by default have no route to the public Internet.
- ❖ Similar to IGW but unlike IGWs they do not allow incoming traffic
- ❖ Only allow responses to outgoing traffic from your Amazon EC2 instances
- ❖ To maximize failover you will want to deploy a NAT gateway per AZ
- ❖ To setup
 - ❖ Set up the route table by connecting private subnet to direct Internet traffic to the *NAT gateway*
 - ❖ Associate the *NAT gateway* with an EIP (covered shortly - elastic IP)

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 they have no route to the public Internet. AWS provides *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

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"
    }
  }
},
```


Placement groups per AZ

- ❖ **Amazon Enhanced networking by using**
- ❖ **Placement groups**
- ❖ Instance types m4, c4, p2, g2, r3, g2, x1, i2 and d2 support **enhanced networking/placement groups**
- ❖ Essential for high-speed server to server performance which is important for clustering
- ❖ To achieve maximum throughput, placement groups must be placed in the same AZ – 10Gbits



Why would this be important for Cassandra? Other systems?

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.

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.

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



What special Cassandra nodes might benefit from using an ENI to keep their private IP constant even if instance goes down?
How are ENIs different than EIPs? How are they similar?

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:

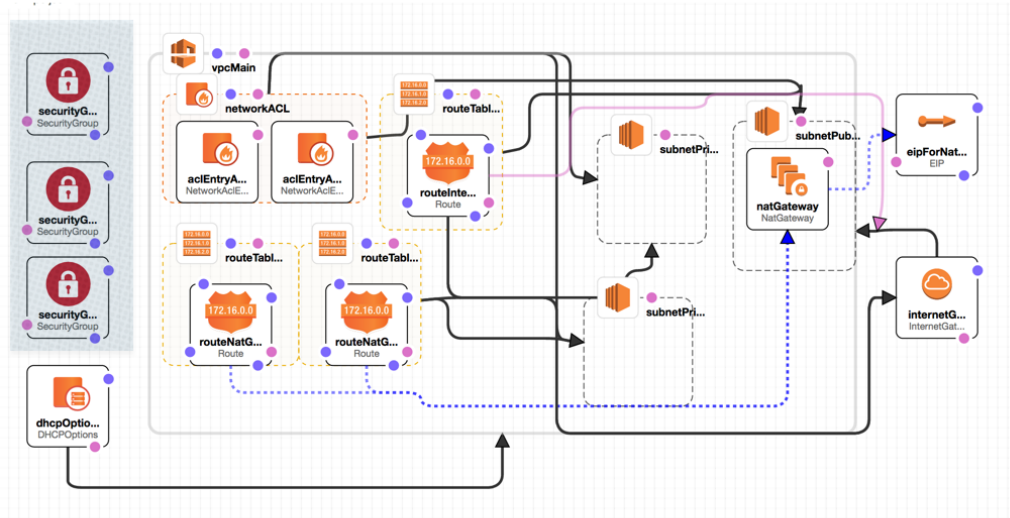
- description
- primary private IPv4 address
- potentially multiple secondary private IPv4 addresses
- EIP per private address
- public IPv4 address
- multiple IPv6 addresses
- multiple security groups (at least one)
- MAC address
- source/destination check flag

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.

Security Groups



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.

CloudFormation SG Bastion

```

"securityGroupBastion": {
  "Type": "AWS::EC2::SecurityGroup",
  "Properties": {
    "GroupDescription": "Security group for bastion server.",
    "VpcId": {
      "Ref": "vpcMain"
    },
    "SecurityGroupIngress": [
      {
        "IpProtocol": "tcp",
        "FromPort": "22",
        "ToPort": "22",
        "CidrIp": "0.0.0.0/0"
      }
    ],
    "SecurityGroupEgress": [
      {
        "IpProtocol": "-1",
        "CidrIp": "0.0.0.0/0"
      }
    ]
  }
},

```

sg-a0dce7d8 | bastionSecurityGroup

Summary Inbound Rules Outbound Rules Tags

Edit

Type	Protocol	Port Range	Source
SSH (22)	TCP (6)	22	0.0.0.0/0

Cancel Save

Type	Protocol	Port Range	Source	Remove
SSH (22)	TCP (6)	22		

Add another rule

sg-05dce77d

sg-a0dce7d8 | bastionSecurityGroup

sg-afdce7d7 | CassandraTestSG

sg-b6dce7ce | cassandraSecurityGroup

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.

CloudFormation SG Cassandra

Private Subnet

```

"securityGroupCassandraNodes": {
  "Type": "AWS::EC2::SecurityGroup",
  "Properties": {
    "GroupDescription": "Security group for Cassandra Database nodes in Cassandra Cluster",
    "VpcId": {
      "Ref": "vpcMain"
    },
    "SecurityGroupIngress": [
      {
        "IpProtocol": "-1",
        "CidrIp": "10.0.0.0/8"
      }
    ],
    "SecurityGroupEgress": [
      {
        "IpProtocol": "-1",
        "CidrIp": "0.0.0.0/0"
      }
    ]
  }
},

```

The above allows all traffic from the VPC's CIDR to access this box.
-1 means all ports.

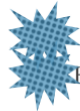
The above allows all traffic from the VPC's CIDR to access this box.
-1 means all ports.
Ingress in incoming traffic.
Egress is outgoing traffic.

CloudFormation SG Cassandra

```

"securityGroupCassandraNodes": {
  "Type": "AWS::EC2::SecurityGroup",
  "Properties": {
    "GroupDescription": "Security group for Cassandra Database nodes in Cassandra Cluster",
    "VpcId": {
      "Ref": "vpcMain"
    },
    "SecurityGroupIngress": [
      {
        "IpProtocol": "-1",
        "CidrIp": "10.0.0.0/8"
      },
      {
        "IpProtocol": "TCP",
        "CidrIp": "0.0.0.0/0",
        "FromPort": "7000", "ToPort": "7001"
      },
      {
        "IpProtocol": "TCP",
        "CidrIp": "0.0.0.0/0",
        "FromPort": "9042", "ToPort": "9042"
      }
    ]
  }
},

```

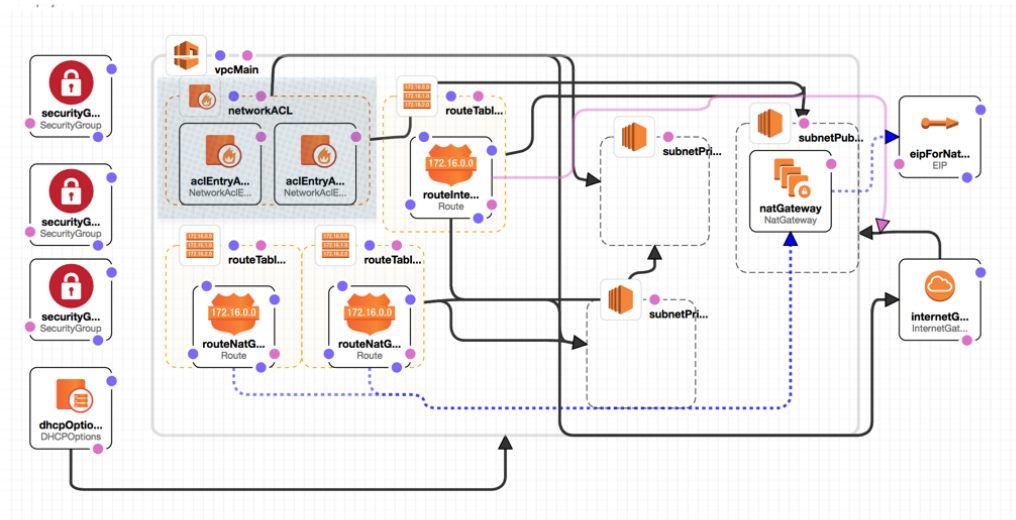


What is different about this SG for Cassandra than the last one?

How could we narrow which EC2 instances access Cassandra nodes?

This is taken from an example that uses Cassandra running in multiple regions.
The Cassandra EC2 instances are running in a public subnet.

CloudFormation: NACL



We are covering Amazon Network ACL Control List (NACL) next.

Network ACL (NACL)

- ❖ **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

Cloudurable Cassandra AWS Support

Related important AWS Concepts

Important AWS concepts
helpful for DevOps of
clustered software

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.

Aws CMD: CloudFormation Route53

Create a new CloudFormation stack (like JSON file from earlier)

```
aws --region ${REGION} s3 cp cloud-formation/vpc.json s3://$CLOUD_FORMER_S3_BUCKET
aws --region ${REGION} cloudformation create-stack --stack-name ${ENV}-vpc-cassandra \
--template-url "https://s3-us-west-2.amazonaws.com/$CLOUD_FORMER_S3_BUCKET/vpc.json" \
```

Create a DNS name for a public IP address from an EC2 instance

```
REQUEST_BATCH="
{
  \"Changes\": [
    {
      \"Action\": \"UPSERT\",
      \"ResourceRecordSet\": {
        \"Type\": \"A\",
        \"Name\": \"${DNS_NAME}\",
        \"TTL\": 300,
        \"ResourceRecords\": [
          {
            \"Value\": \"${IP_ADDRESS}\"
          }
        ]
      }
    }
  ]
}
"

echo "$REQUEST_BATCH"

changeId=$(aws route53 change-resource-record-sets --hosted-zone-id "$HOSTED_ZONE_ID" --change-batch "$REQUEST_BATCH" |
jq --raw-output .ChangeInfo.Id)
```

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.

More important AWS concepts

- ❖ IAM
 - ❖ AWS Identity and Access Management (IAM) enables secure control access to AWS Cloud services and resources for their users
 - ❖ Defines 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
 - ❖ Uses Hardware Security Modules (HSMs) to protect the security of your keys
 - ❖ Used to encrypt Amazon EBS volumes, Amazon S3 buckets and other services.
- ❖ S3
 - ❖ Amazon Simple Storage Service to store your backups and big data.
 - ❖ Good for backups of Cassandra snapshots

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. To use CloudWatch logging or metrics from an application, you would need to assign rights to a role and then associate an IAM role with your EC2 instance.

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!

Cassandra on AWS

[AWS Cassandra Support](#)

- ❖ Apache Cassandra is extensively deployed in AWS
- ❖ Estimated 60,000 AWS customers also use Cassandra
- ❖ Estimated 1/3 of Cassandra are on AWS
- ❖ More AWS users use Cassandra than DynamoDB
- ❖ AWS published guide: Deploying [Cassandra on AWS in 2016](#)

Apache Cassandra is extensively deployed in AWS. 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 on Cassandra called: Apache Cassandra on AWS.

Cassandra / Kafka Support in EC2/AWS

Cassandra AWS Overview

Cassandra Architecture
Review

What is Cassandra?

[AWS Cassandra Architecture](#)

- ❖ 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.

AWS Cassandra Concepts: Cluster

[AWS Cassandra Architecture](#)

- ❖ Cassandra node runs on EC2 instances
- ❖ Cassandra cluster can span AZs and Regions
- ❖ Cassandra Cluster consists of Data centers, racks, nodes
- ❖ AWS AZ equate to Cassandra racks
- ❖ Amazon regions equate to Cassandra data centers
- ❖ AZs should equate to a multiple of replication level

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.

AWS Cassandra Concepts: Commit log and in-memory

[AWS Cassandra Architecture](#)

- ❖ Commit logs are write-ahead logs used for Cassandra node recovery – sequential append write - only read for recovery
- ❖ Must but commit log on separate EBS volume if magnetic
- ❖ Memtables, key cache, row cache, in-memory bloom filters, in-memory index files, OS buffers for disk and TCP/IP, all require lots of memory
- ❖ Pick EC2 instances with enough DRAM

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

[AWS Cassandra Architecture](#)

- ❖ 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 memtable flushes, 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 a 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?

Cassandra / Kafka Support in EC2/AWS

AWS EC2 Instance Types for Cassandra

AWS EC2 Instance types to
consider for Cassandra

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.xlarge instance type
- Support for [Enhanced Networking](#) and Clustering

Model	vCPU	Mem (GiB)	Storage	Dedicated EBS Bandwidth (Mbps)
c4.large	2	3.75	EBS-Only	500
c4.xlarge	4	7.5	EBS-Only	750
c4.2xlarge	8	15	EBS-Only	1,000
c4.4xlarge	16	30	EBS-Only	2,000
c4.8xlarge	36	60	EBS-Only	4,000

High CPU to memory ratio

M4

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

Model	vCPU	Mem (GiB)	SSD Storage (GB)	Dedicated EBS Bandwidth (Mbps)
m4.large	2	8	EBS-only	450
m4.xlarge	4	16	EBS-only	750
m4.2xlarge	8	32	EBS-only	1,000
m4.4xlarge	16	64	EBS-only	2,000
m4.10xlarge	40	160	EBS-only	4,000
m4.16xlarge	64	256	EBS-only	10,000

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 IOPS 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

Model	vCPU	Mem (GiB)	Networking Performance	Storage (TB)
i3.large	2	15.25	Up to 10 Gigabit	1 x 0.475 NVMe SSD
i3.xlarge	4	30.5	Up to 10 Gigabit	1 x 0.95 NVMe SSD
i3.2xlarge	8	61	Up to 10 Gigabit	1 x 1.9 NVMe SSD
i3.4xlarge	16	122	Up to 10 Gigabit	2 x 1.9 NVMe SSD
i3.8xlarge	32	244	10 Gigabit	4 x 1.9 NVMe SSD

Perfect for Cassandra high-speed, small sized reads. Critical path. No KMS encryption.

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

Model	vCPU	Mem (GiB)	Storage (GB)
d2.xlarge	4	30.5	3 x 2000 HDD
d2.2xlarge	8	61	6 x 2000 HDD
d2.4xlarge	16	122	12 x 2000 HDD
d2.8xlarge	36	244	24 x 2000 HDD

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.

General Purpose

Storage optimized

	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage		vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
m4.large	2	6.5	8	EBS Only	\$0.108 per Hour	i3.large	2	7	15.25	1 x 475 NVMe SSD	\$0.156 per Hour
m4.xlarge	4	13	16	EBS Only	\$0.215 per Hour	i3.xlarge	4	13	30.5	1 x 950 NVMe SSD	\$0.312 per Hour
m4.2xlarge	8	26	32	EBS Only	\$0.431 per Hour	i3.2xlarge	8	27	61	1 x 1900 NVMe SSD	\$0.624 per Hour
m4.4xlarge	16	53.5	64	EBS Only	\$0.862 per Hour	i3.4xlarge	16	53	122	2 x 1900 NVMe SSD	\$1.248 per Hour
m4.10xlarge	40	124.5	160	EBS Only	\$2.155 per Hour	i3.8xlarge	32	99	244	4 x 1900 NVMe SSD	\$2.496 per Hour
m4.16xlarge	64	188	256	EBS Only	\$3.447 per Hour	i3.16xlarge	64	200	488	8 x 1900 NVMe SSD	\$4.992 per Hour
						d2.xlarge	4	14	30.5	3 x 2000 HDD	\$0.69 per Hour
						d2.2xlarge	8	28	61	6 x 2000 HDD	\$1.38 per Hour
						d2.4xlarge	16	56	122	12 x 2000 HDD	\$2.76 per Hour
						d2.8xlarge	36	116	244	24 x 2000 HDD	\$5.52 per Hour

General Purpose

Compute Optimized - Current Generation

m4.large	2	6.5	8	EBS Only	\$0.108 per Hour	c4.large	2	8	3.75	EBS Only	\$0.1 per Hour
m4.xlarge	4	13	16	EBS Only	\$0.215 per Hour	c4.xlarge	4	16	7.5	EBS Only	\$0.199 per Hour
m4.2xlarge	8	26	32	EBS Only	\$0.431 per Hour	c4.2xlarge	8	31	15	EBS Only	\$0.398 per Hour
m4.4xlarge	16	53.5	64	EBS Only	\$0.862 per Hour	c4.4xlarge	16	62	30	EBS Only	\$0.796 per Hour
m4.10xlarge	40	124.5	160	EBS Only	\$2.155 per Hour	c4.8xlarge	36	132	60	EBS Only	\$1.591 per Hour
m4.16xlarge	64	188	256	EBS Only	\$3.447 per Hour						



Trivia what is ECU?



What does vCPU equate to?

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).



[AWS Cassandra Storage Reqs.](#)

Cassandra / Kafka Support in EC2/AWS

Cassandra AWS Storage Guidelines

Cassandra storage
requirements for Amazon EC2
and EBS

Cassandra AWS - Storage Requirements

[AWS Cassandra Storage Reqs.](#)

- ❖ 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 Cassandra Storage Reqs.](#)

- ❖ 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.

Prefer EBS

[AWS Cassandra Storage Reqs.](#)

- ❖ Historically EBS did not work well with Cassandra
 - ❖ **You will read old AWS Cassandra guides that tell you to use EC2 instance storage**
- ❖ Until recently (2015) using Cassandra and AWS EBS was not a good idea
- ❖ Latest generation of *EBS-optimized* instances offer performance and improved reliability
- ❖ EBS volumes best pick for price for performance
- ❖ If in doubt start with EBS-optimized instances
- ❖ EBS has nice features like snapshots, and redundancy
- ❖ NEW! *EBS elastic volumes, provisioned IO*
- ❖ EBS is more flexible, and less expensive
- ❖ Other ways to improve reliability with Cassandra and AWS (replication)
- ❖ Supports high-speed encryption (i3 would need OS encryption or JDK)

EBS is ok for Cassandra, Prefer EBS

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 using Cassandra and AWS EBS was not a good idea.

The latest generation of *EBS-optimized* instances offer a good mix of performance and for many use cases rivaling instance storage. 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



If using Cassandra, JBOD how many volumes minimum would I need to get up to 65K IOPs with standard EBS optimized volume? With provisioned IOPs?

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/>

[Credit AWS Docs](#)

	Solid State Drives (SSD)		Hard Disk Drives (HDD)	
Volume Type	EBS Provisioned IOPS SSD (io1)	EBS General Purpose SSD (gp2)*	Throughput Optimized HDD (st1)	Cold HDD (sc1)
Short Description	Highest performance SSD volume designed for latency-sensitive transactional workloads	General Purpose SSD volume that balances price performance for a wide variety of transactional workloads	Low cost HDD volume designed for frequently accessed, throughput intensive workloads	Lowest cost HDD volume designed for less frequently accessed workloads
Use Cases	I/O-intensive NoSQL and relational databases	Boot volumes, low-latency interactive apps, dev & test	Big data, data warehouses, log processing	Colder data requiring fewer scans per day
API Name	io1	gp2	st1	sc1
Volume Size	4 GB - 16 TB	1 GB - 16 TB	500 GB - 16 TB	500 GB - 16 TB
Max IOPS*/Volume	20,000	10,000	500	250
Max Throughput/Volume	320 MB/s	160 MB/s	500 MB/s	250 MB/s
Max IOPS/Instance	65,000	65,000	65,000	65,000
Max Throughput/Instance	1,250 MB/s	1,250 MB/s	1,250 MB/s	1,250 MB/s
Price	\$0.125/GB-month \$0.065/provisioned IOPS	\$0.10/GB-month	\$0.045/GB-month	\$0.025/GB-month
Dominant Performance Attribute	IOPS	IOPS	MB/s	MB/s

Cassandra EC2 instance good picks

[AWS Cassandra Storage Reqs.](#)

- ❖ 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

[AWS Cassandra Storage Reqs.](#)

- ❖ If in doubt use ESB SSD
- ❖ SSD has higher read speed / IOPS (random access, seeking key)
 - ❖ You can use SSD with provision IOPS to improve read access (expensive)
- ❖ Magnetic disks (HDD) have higher throughput but lower IOPS
 - ❖ AWS guide says don't use, benchmarks say works for some use cases
 - ❖ Increase HDD IOPS by using instance store or JBOD
 - ❖ Cheapest throughput and storage of volume options

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 use 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 into Account

[AWS Cassandra Storage Reqs.](#)

- ❖ Keep replication strategy into account when sizing EBS volumes and instance stores
- ❖ SSTable Compaction data makes heavy use of disk
- ❖ *LeveledCompactionStrategy* needs 10 to 20% overhead for compaction (faster reads)
- ❖ *SizeTieredCompactionStrategy* worse case is 50% overhead for compaction (faster writes)
- ❖ Improves read speed (spikier CPU)

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. *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*.

RAID, JBOD, Read speed

[AWS Cassandra Storage Reqs.](#)

- ❖ If RAID, use RAID 0
 - ❖ Not needed for data safety because EBS provides it as does Cassandra
- ❖ Prefer JBOD to RAID 0
 - ❖ Just mount EBS volumes for JBOD
 - ❖ JBOD is just a bunch of disks (added in Cassandra 3)
 - ❖ JBOD helps with read speed (more so than RAID 0)

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

[AWS Cassandra Storage Reqs.](#)

- ❖ 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

[AWS Cassandra Storage Reqs.](#)

- ❖ 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

[AWS Cassandra Storage Reqs.](#)

- ❖ **EBS has been know 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 know 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

[AWS Cassandra Storage Reqs.](#)

- ❖ 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



What would you need to make sure of if you added a materialized view on a large table to improve read speed? WRT to EBS

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 / Kafka Support in EC2/AWS

Cassandra AWS CPU Guidelines

Cassandra CPU requirements
with Amazon EC2

Cassandra CPU Reqs for AWS Cloud

[AWS Cassandra CPU Guidelines](#)

- ❖ 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 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. Think of it as a physical thread of execution. It is able to run one thread at a time (which of course could be swapped out).

Cassandra high-speed writes = CPU bound

[AWS Cassandra CPU Guidelines](#)

- ❖ Cassandra clusters insert heavy workloads can be CPU-bound
- ❖ Potential to be multiplied if using JBOD - 4 or 8+ volumes
- ❖ Cassandra is efficient for writes, but does CPU intensive structured merge sort during compaction
- ❖ Writes are almost never IO bound, *concurrent_writes* (workers) depends on vCPU in EC2 Cassandra node instance
- ❖ *concurrent_compactors* should be set to
 - ❖ # of vCPUs if using SSDs
 - ❖ number of attached EBS volumes for JBOD
 - ❖ unless you are having GC issues than set to max 4
 - ❖ More vCPU resources your Cassandra node has, the faster compaction throughput
 - ❖ See this [Cassandra tuning guide](#) for more information, and this [JIRA ticket](#).

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** [AWS Cassandra CPU Guidelines](#)
- ❖ *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.

Use 8 vCPUs for prod

[AWS Cassandra CPU Guidelines](#)

- ❖ **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

[AWS Cassandra CPU Guidelines](#)

- ❖ [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

[AWS Cassandra CPU Guidelines](#)

- ❖ 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

[AWS Cassandra CPU Guidelines](#)

- ❖ 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 $(16 * \text{number_of_drives})$
 - ❖ 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 $(16 * \text{number_of_drives})$ so you have the potential with JBOD 4 of having 64 read threads.

Cassandra with high volume of writes/streams

[AWS Cassandra CPU Guidelines](#)

- ❖ **Doing a lot of streaming writes between nodes?**
Increase memtable_flush_writers
- ❖ If 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`

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). Avoid the streams all hitting the memtables. 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. The recommendation is to set memtables_flush_writers equal to the number of vCPUs on the EC2 Cassandra node instance. More vCPUs allows more throughput for writes.

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

[AWS Cassandra CPU Guidelines](#)

- ❖ 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.

EC2, Cassandra and NUMA

[AWS Cassandra CPU Guidelines](#)

- ❖ i3.8xlarge+, m4.10xlarge+ - Non-Uniform Memory Architecture([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

[AWS Cassandra CPU Guidelines](#)

- ❖ 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)



[AWS Cassandra System Memory Guidelines](#)

Cassandra / Kafka Support in EC2/AWS

AWS Cassandra System Memory Guidelines

Cassandra system memory
guidelines for Amazon EC2

Basic System Memory Guidelines

[AWS Cassandra 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 JVM Heap Usage

[AWS Cassandra System Memory Guidelines](#)

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

Some grow as JVM heap grows

Java heap usage for Cassandra

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

[AWS Cassandra System Memory Guidelines](#)

- ❖ Java heap
- ❖ Off-heap memory
- ❖ OS page cache
- ❖ OS TCP/IP stack I/O cache

Cassandra uses memory in 4 ways:

- Java heap
- offheap memory
- OS page cache
- OS TCP/IP stack I/O cache

Memory: More is better

[AWS Cassandra System Memory Guidelines](#)

- ❖ More memory the better
- ❖ If memory is available, Cassandra and the Linux OS can use it
- ❖ In 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](#)

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. If the cache hit rate is high, then there is less read IO. In the NoSQL world, Cassandra gets high marks for writes, and lower marks for reads, which use case permitting could benefit from caches.

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 needs OS memory!

[AWS Cassandra System Memory Guidelines](#)

- ❖ **OS memory should be 2x to 6x the size of the JVM!**
- ❖ 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,
 - ❖ Linux OS Page Cache takes IOPs and throughput of the EBS out of the equation
- ❖ *Leave memory for the Linux OS!*
- ❖ Leave some space for Linux IO buffers.

Cassandra relies heavily on the Linux OS page cache for caching of data on 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 Off-heap memory usage

[AWS Cassandra System Memory Guidelines](#)

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

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.

Cassandra JVM vs Linux System Memory table for EC2

[AWS Cassandra System Memory Guidelines](#)

EC2 Instance Type	Instance Size GB	JVM Size Range GB	Linux OS memory Range GB
c4.2xlarge	15	5	10
m4.2xlarge	32	5 to 16	16 to 27
m4.4xlarge	64	10 to 32	32 to 54
m4.10xlarge	160	27 to 80	80 to 133
i3.2xlarge	61	10 to 30	31 to 51
i3.4xlarge	122	20 to 60	62 to 102

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

```
-Xms12G
-Xmx12G
```

-Xms sets min heap size and -Xmx sets max heap size

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

[AWS Cassandra System Memory Guidelines](#)

- ❖ Due to Cassandra has some long lived objects on the heap...
- ❖ Choice in GCs come down to CMS and GCG1. Choose GCG1.
- ❖ General rule is don't use GCG1 if your heap is under 5GB (some say 8GB, some 6 GB, Oracle says under 1GB)
- ❖ You will notice on the preceding chart, 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

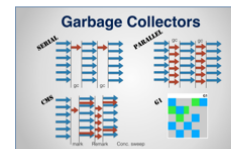


Image Credit: Ranjith Ramachandran

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 GCG1 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>

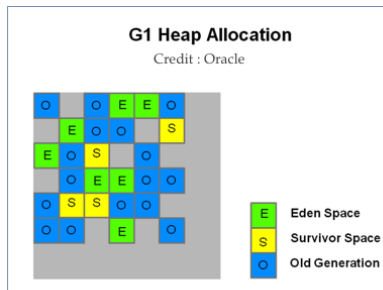
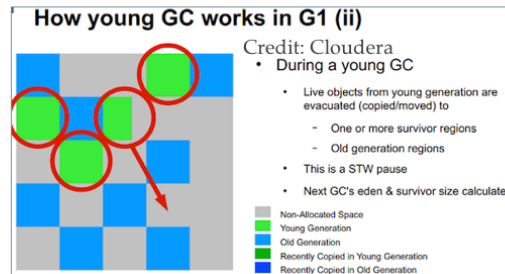
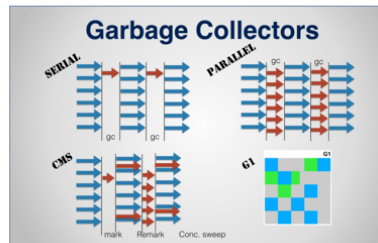


Image Credit: Ranjith Ramachandran from YouTube talk on GC

<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>

Amazon EC2 instances taking CMS out of equation!

[AWS Cassandra System Memory Guidelines](#)

EC2 Instance Type	Instance Size GB	JVM Size Range GB	Linux OS memory Range GB
m4.2xlarge	32	8 to 16	13 to 24
m4.4xlarge	64	10 to 32	32 to 54
m4.10xlarge	160	27 to 80	80 to 133
i3.2xlarge	61	10 to 30	31 to 51
i3.4xlarge	122	20 to 60	62 to 102

- ❖ Note: Amazon has a [Cassandra on Amazon](#) guide that says never use over 8 GB for JVM. **This is a mistake! See note!**



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 -Xmx sets max heap size

You should prefer an easily tuned and stable setting over one that has the issues that CMS does.

Guideline for GCG1 Cassandra JVM

[AWS Cassandra System Memory Guidelines](#)

Guide for using GCG1 with Cassandra JVM

```
-XX:+UseG1GC
-XX:MaxGCPauseMillis=500
-XX:G1RSetUpdatingPauseTimePercent=5
-XX:InitiatingHeapOccupancyPercent=25
```

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

```
-XX:+UseG1GC
-XX:MaxGCPauseMillis=1000
-XX:InitiatingHeapOccupancyPercent=60
```

Just make sure your Cassandra timeouts are more than 1000ms.

```
-XX:+UseG1GC -XX:MaxGCPauseMillis=100
-XX:G1HeapWastePercent=20 -XX:InitiatingHeapOccupancyPercent=75
-XX:ConcGCThreads=32 -XX:ParallelGCThreads=48 Very Large Heap Low Pause
Credit: Intel: HBaseCon
```

GCG1 will self adjust using ergonomics and runtime statistics.
The only setting that you really need to set is `-XX:MaxGCPauseMillis`.

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 for both CMS and GCG1

[AWS Cassandra System Memory Guidelines](#)

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
```

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.”—[AI Tobey Writes Blog:](#)

[Cassandra tuning guide](#)

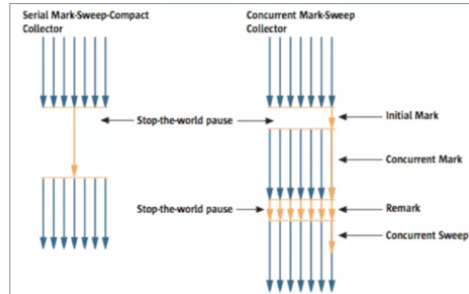
Use +AlwaysPreTouch to allocate and zero, which does a force fault, heap memory on startup. This ensures all memory is faulted and zeroed on startup, and prevents soft faults making hugepage allocation more effective. Use XX:+UseTLAB to add thread local allocation blocks. Use +ResizeTLAB to allow JVM 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

[AWS Cassandra System Memory Guidelines](#)

- ❖ Don't use CMS
- ❖ If you need JVM heap between 5GB and 8GB, and having an easy to configure, reliable systems does not win out over raw speed, then maybe CMS
- ❖ Remember, take CMS out of equation by using m4.2xlarge and i3.2xlarge as smallest EC2 instances for Cassandra
- ❖ **Definitely do not** use CMS above 8GB JVM heap



- CMS is deprecated in Java 9
- CMS is harder to tune
- CMS has problems with fragmented memory
- GCG1 is the replacement
- GCG1 is easier to tune
- Better to learn how to tune GCG1 now
- Harder to reach 99.9 SLAs

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 use this as a guide.

Remember, you can take CMS out of the equation by using m4.2xlarge and i3.2xlarge as the smallest EC2 instances you deploy to. If for some reason you need to go smaller than 8GB, try using GCG1 anyway.

Do not use CMS for anything above 8GB JVM heap. It does have STW pauses.

CMS does well until it doesn't. (Personal experiences with CMS are bad.)

Guide to using CMS

[AWS Cassandra System Memory Guidelines](#)

Guide to setting up CMS for Cassandra running in AWS

```
-XX:+UseConcMarkSweepGC
-XX:ParGCCardsPerStrideChunk=4096
-XX:SurvivorRatio=2
-XX:MaxTenuringThreshold=16
-XX:+CMSScavengeBeforeRemark
-XX:CMSMaxAbortablePrecleanTime=60000
-XX:CMSWaitDuration=30000
-XX:CMSInitiatingOccupancyFraction=70
-XX:+UseCMSInitiatingOccupancyOnly
-Xmn{1/4 to 1/3 the size of the heap}
```

BTW Don't use CMS

The Parallel copy collector (ParNew) is responsible for young collection in CMS. Use [ParGCCardsPerStrideChunk \(default 256\)](#) to increase granularity of tasks distributed between worker threads.

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

Use 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 SurvivorRatio=N divides the young generation by N+2 segments, take N segments for Eden and 1 segment for each survivor.

The 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 UseCMSInitiatingOccupancyOnly initiates CMS GC in advance to avoid full, stop-the-world, GC. CMSInitiatingOccupancyFraction sets the trigger level for CMS, i.e., the Cassandra JVM should use less than 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.

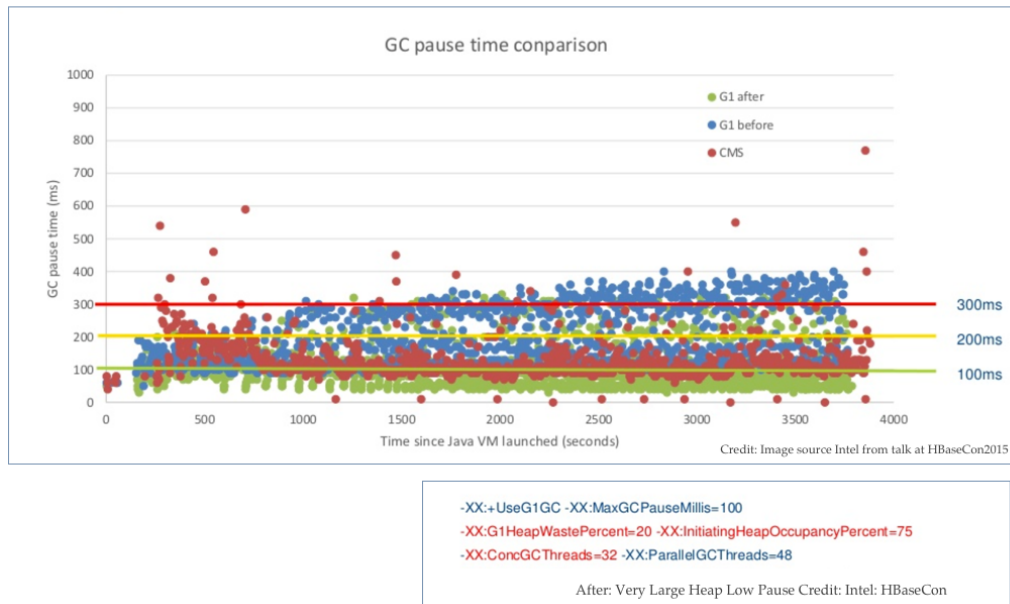


Image from Intel from here: <https://www.slideshare.net/HBaseCon/dev-session-7-49202969>

After is after tuning G1.

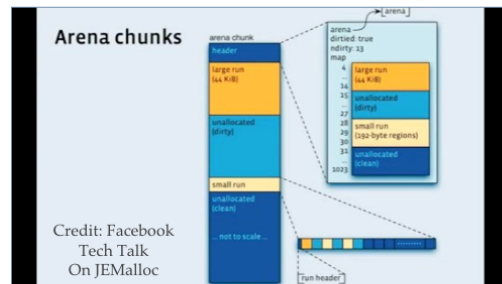
Before is before tuning G1.

Take pressure off JVM GC by using off-heap

[AWS Cassandra System Memory Guidelines](#)

- ❖ **Reduce pressure garbage collection**
- ❖ Cassandra uses native libraries to allocate memory if available.
- ❖ **Ensure JNA and JEMALLOC are installed on Linux machine Amazon AMI**
 - ❖ `yum install -y jna`
 - ❖ `yum install -y jemalloc`
- ❖ If you are creating an Amazon AMI image with both of those
- ❖ Use `offheap_objects`
- ❖ Modify memtable space by changing the `memtable_heap_space_in_mb` and `memtable_offheap_space_in_mb`

```
yum install -y jna
yum install -y jemalloc
```



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/](http://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](https://en.wikipedia.org/wiki/Java_Native_Access)

https://en.wikipedia.org/wiki/Java_Native_Access

Credit: Image is from a FaceBook talk on JEMalloc

<http://www.downvids.net/scalable-memory-allocation-using-jemalloc-tech-talk-1-11-2011--505622.html>



Cassandra / Kafka Support in EC2/AWS

Cassandra Networking Guidelines

Cassandra Networking
requirements with Amazon
EC2

AWS Networking for Cassandra

- ❖ AWS EC2 has placement groups / enhanced networking
 - ❖ 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
- ❖ 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, it helps with quorum reads and writes

Networking:

AWS EC2 has placement groups and enhanced networking which allow high-speed throughput for clustered software like Cassandra. This is where things get tricky in EC2. Networking is important to Cassandra due to replication of data. However, with 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

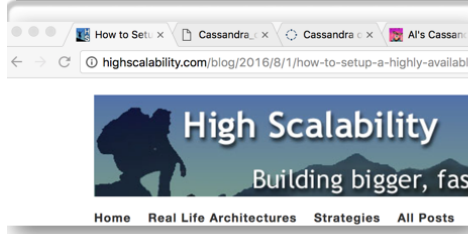
From EC2 docs: Cluster Networking

<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

Enhanced Networking			Use enhanced networking!		
	Median	95th percentile		Median	95th percentile
WRITE					
Single AZ	1.0	2.5	Single AZ	0.9	2.4
Multi AZ (3 AZs)	1.5	2.8	Multi AZ (3 AZs)	1.1	2.3
READ					
Single AZ	1.0	2.6	Single AZ	0.7	1.5
Multi AZ (3 AZs)	1.5	23.5	Multi AZ (3 AZs)	1.0	1.9



See <http://highscalability.com/blog/2016/8/1/how-to-setup-a-highly-available-multi-az-cassandra-cluster-o.html> for more details

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)

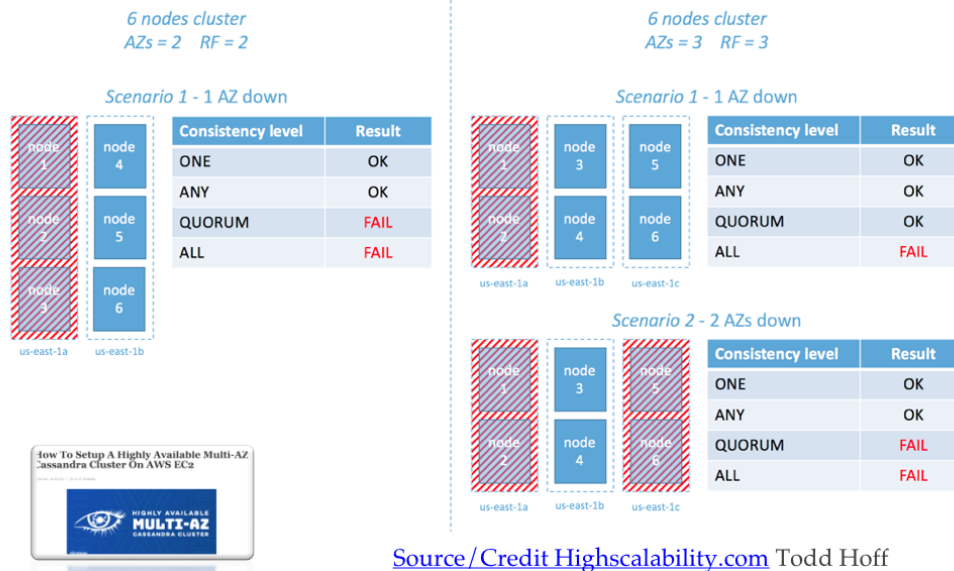


What bandwidth do you think you would get with a m4.4xlarge?

Subnet per AZ for quorum

- ❖ What happens if the AZ goes down?
- ❖ Do your reads stop working?
- ❖ Do your writes stop working?
- ❖ Availability Zones \leq Replication Factor

Failure Scenarios



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

What is a snitch?

- ❖ **Snitch** sets **network topology**
- ❖ **Snitch** determines where nodes live. Which Datacenter? Which Rack?
- ❖ Cassandra uses *network topology* information to *distribute replicas*
- ❖ Cassandra can also use this information
 - ❖ To *determine datacenter locality* with local quorum writes and reads
 - ❖ To *optimize client reads and writes* with data consistency
- ❖ 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?
- ❖ When replicating data, info used to not put replica on the same rack

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

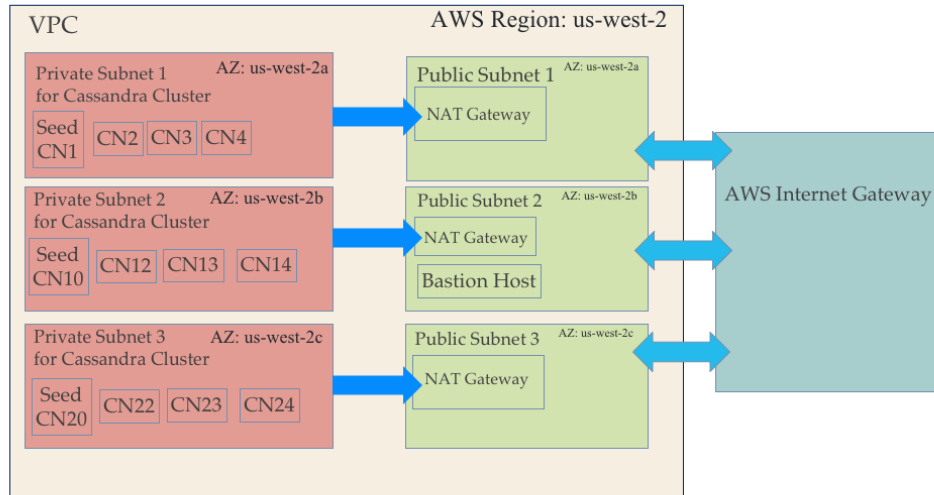
Remember Cassandra is stateful and limiting full roll out of a brand new instance with updates preinstalled will cause lots of streaming between Cassandra nodes and is not ideal.

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



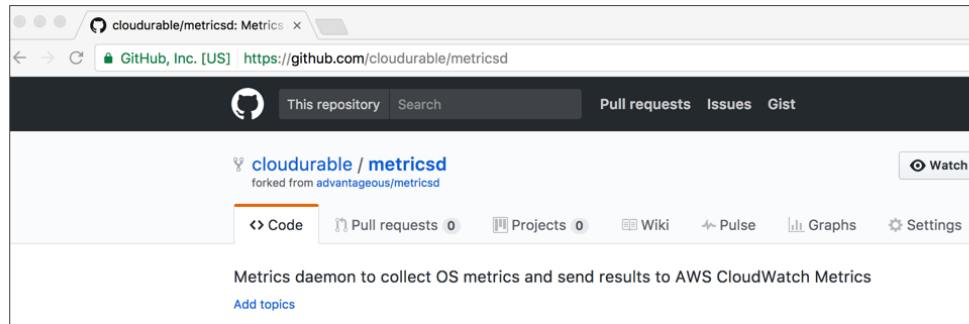
Steps

- ❖ 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

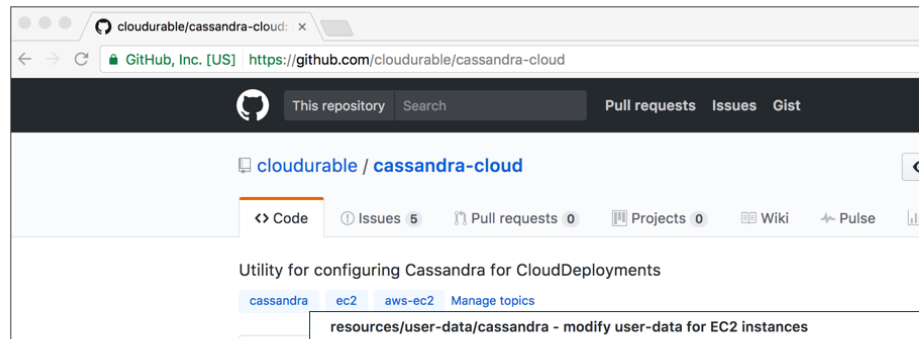
metricsd



Metrics daemon to collect OS metrics, Cassandra metrics,
and send results to AWS CloudWatch Metrics

[metricsd](#)

Cassandra Cloud



Generates
Cassandra YAML
From Template

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

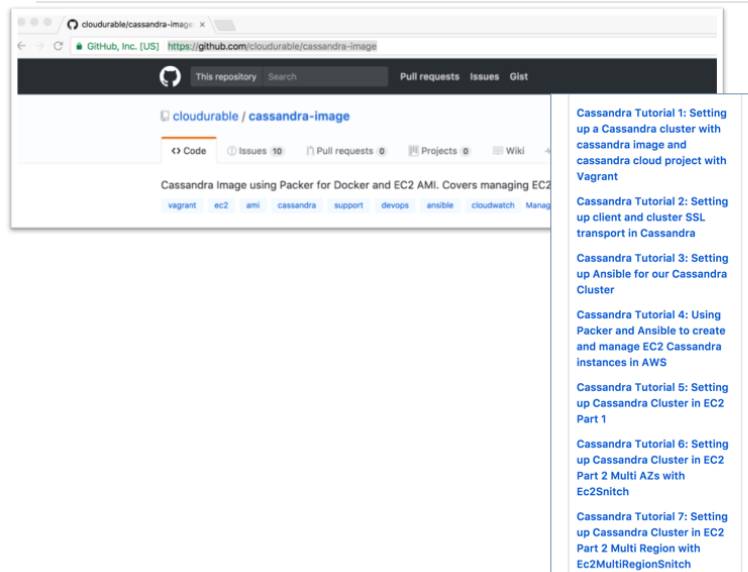
```
#!/bin/bash
set -e

export BIND_IP=`curl http://169.254.169.254/latest/meta-data/local-ipv4`

/opt/cloudurable/bin/cassandra-cloud -cluster-name test \
  -client-address ${BIND_IP} \
  -cluster-address ${BIND_IP} \
  -cluster-seeds 10.0.1.10,10.0.2.10 \
  -snitch Ec2Snitch

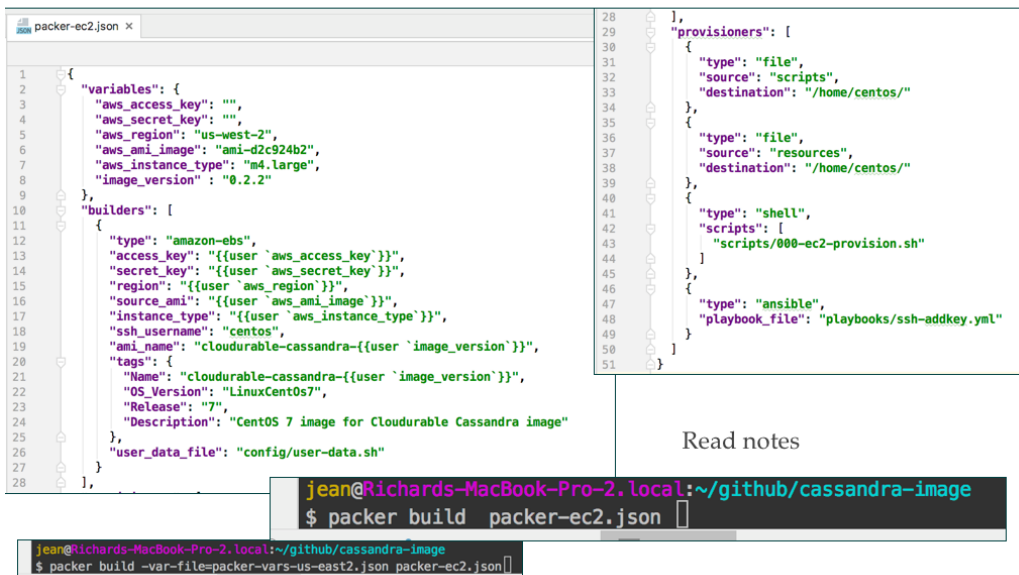
/bin/systemctl restart cassandra
```

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



```

1  {
2    "variables": {
3      "aws_access_key": "",
4      "aws_secret_key": "",
5      "aws_region": "us-west-2",
6      "aws_ami_image": "ami-d2c924b2",
7      "aws_instance_type": "m4.large",
8      "image_version": "0.2.2"
9    },
10   "builders": [
11     {
12       "type": "amazon-ec2",
13       "access_key": "{{user `aws_access_key`}}",
14       "secret_key": "{{user `aws_secret_key`}}",
15       "region": "{{user `aws_region`}}",
16       "source_ami": "{{user `aws_ami_image`}}",
17       "instance_type": "{{user `aws_instance_type`}}",
18       "ssh_username": "centos",
19       "ami_name": "cloudurable-cassandra-{{user `image_version`}}",
20       "tags": {
21         "Name": "cloudurable-cassandra-{{user `image_version`}}",
22         "OS_Version": "LinuxCentos7",
23         "Release": "7",
24         "Description": "CentOS 7 image for Cloudurable Cassandra image"
25       },
26       "user_data_file": "config/user-data.sh"
27     }
28   ],
29   "provisioners": [
30     {
31       "type": "file",
32       "source": "scripts",
33       "destination": "/home/centos/"
34     },
35     {
36       "type": "file",
37       "source": "resources",
38       "destination": "/home/centos/"
39     },
40     {
41       "type": "shell",
42       "scripts": [
43         "scripts/000-ec2-provision.sh"
44       ]
45     },
46     {
47       "type": "ansible",
48       "playbook_file": "playbooks/ssh-addkey.yml"
49     }
50   ]
51 }

```

```

jean@Richards-MacBook-Pro-2.local:~/github/cassandra-image
$ packer build packer-ec2.json

jean@Richards-MacBook-Pro-2.local:~/github/cassandra-image
$ packer build -var-file=packer-vars-us-east2.json packer-ec2.json

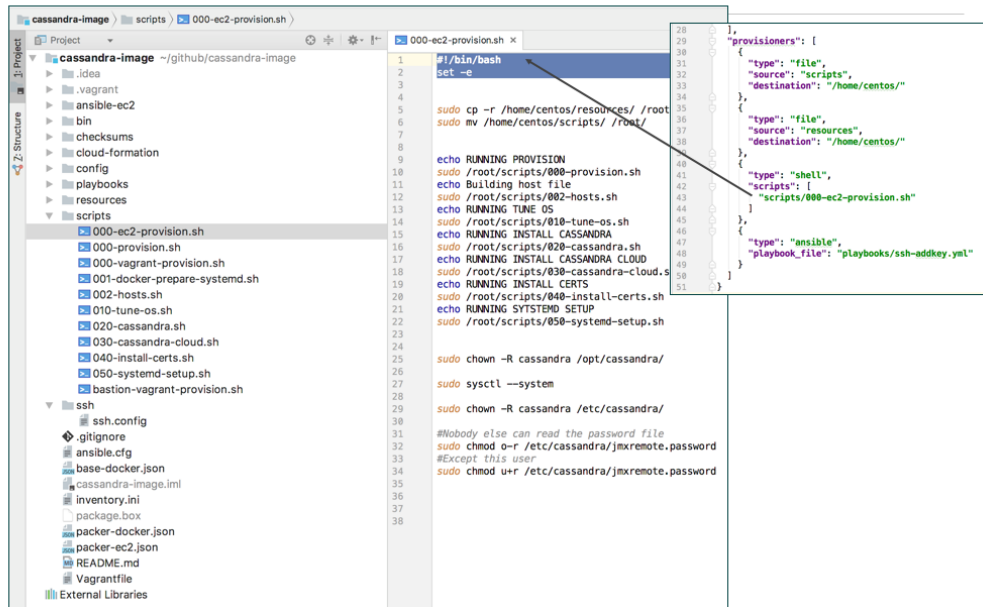
```

[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-ec2 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.

Step 1: Build AMI with Packer

Read notes!



The screenshot shows a code editor with two files open. The left file is `000-ec2-provision.sh`, which contains a series of shell commands for provisioning an EC2 instance. The right file is a Packer JSON configuration for an EC2 AMI, which references the scripts in the `scripts` directory.

```

1 #!/bin/bash
2 set -e
3
4 sudo cp -r /home/centos/resources/ /root/
5 sudo mv /home/centos/scripts/ /root/
6
7 echo RUNNING PROVISION
8 sudo /root/scripts/000-provision.sh
9 echo Building host file
10 sudo /root/scripts/002-hosts.sh
11 echo RUNNING TUNE OS
12 sudo /root/scripts/010-tune-os.sh
13 echo RUNNING INSTALL CASSANDRA
14 sudo /root/scripts/020-cassandra.sh
15 echo RUNNING INSTALL CASSANDRA CLOUD
16 sudo /root/scripts/030-cassandra-cloud.sh
17 echo RUNNING INSTALL CERTS
18 sudo /root/scripts/040-install-certs.sh
19 echo RUNNING SYSTEND SETUP
20 sudo /root/scripts/050-systemd-setup.sh
21
22 sudo chown -R cassandra /opt/cassandra/
23
24 sudo systemctl --system
25
26 sudo chown -R cassandra /etc/cassandra/
27
28 #Nobody else can read the password file
29 sudo chmod o-r /etc/cassandra/jmxremote.password
30 #Except this user
31 sudo chmod u+r /etc/cassandra/jmxremote.password
32
33
34
35
36
37
38

```

```

28 },
29 "provisioners": [
30 {
31   "type": "file",
32   "source": "scripts",
33   "destination": "/home/centos/"
34 },
35 {
36   "type": "file",
37   "source": "resources",
38   "destination": "/home/centos/"
39 },
40 {
41   "type": "shell",
42   "scripts": [
43     "scripts/000-ec2-provision.sh"
44 ]
45 },
46 {
47   "type": "ansible",
48   "playbook_file": "playbooks/ssh-addkey.yml"
49 }
50 ]
51 }

```

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.

Step 1: Build AMI Monitoring tools

Read notes!

Installing metricsd systemd from our provisioning scripts

```
cp ~/resources/etc/systemd/system/metricsd.service /etc/sys
cp ~/resources/etc/metricsd.conf /etc/metricsd.conf
systemctl enable metricsd
systemctl start metricsd
```

/etc/systemd/system/metricsd.service

```
[Unit]
Description=MetricsD OS Metrics
Requires=cassandra.service
After=cassandra.service

[Service]
ExecStart=/opt/cloudurable/bin/metricsd

WorkingDirectory=/opt/cloudurable
Restart=always
RestartSec=60
TimeoutStopSec=60
TimeoutStartSec=60

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

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 informatino about metricsd please see the [metricsd github project](#).

The metricsd system unit depends on the Cassandra service. The unit

file is above.

Step 1: Build AMI : Log aggregation

Read notes!

Installing systemd-cloud-watch systemd service from our provisioning scripts

```
cp ~/resources/etc/systemd/system/systemd-cloud-watch.service /etc/systemd/system/sy
cp ~/resources/etc/systemd-cloud-watch.conf /etc/systemd-cloud-watch.conf
systemctl enable systemd-cloud-watch
systemctl start systemd-cloud-watch
```

systemd-cloud-watch.conf

```
log_priority=7
debug=true
log_group="cassandra"
batchSize=5
```

/etc/systemd/system/systemd-cloud-watch.service

```
[Unit]
Description=SystemD Cloud Watch Sends Journald logs to CloudWatch
Requires=cassandra.service
After=cassandra.service

[Service]
ExecStart=/opt/cloudurable/bin/systemd-cloud-watch /etc/systemd-cloud-watch.conf

WorkingDirectory=/opt/cloudurable
Restart=always
RestartSec=60
TimeoutStopSec=60
TimeoutStartSec=60

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

Creating a AWS CloudWatch log group

```
$ aws logs create-log-group --log-group-name cassandra
```

To learn more about `systemd-cloud-watch`, please see the [systemd-cloud-watch GitHub project](#).

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 journals 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](#).

Step 1: Build AMI : Install Cassandra

```
/etc/systemd/system/cassandra.service

[Unit]
Description=Cassandra Service

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

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

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

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

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.

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*** - setups 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 - setups 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

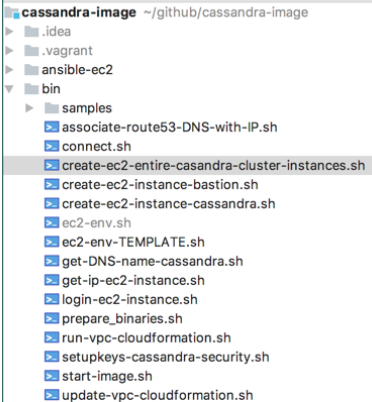
Note to parse the JSON coming back from the ***aws command line** we use jq. Note that [jq](#) is a lightweight command-line JSON processor. To download and install jq see the [jq download documents](#).

AWS CLI: Common Environment

```

1  #!/bin/bash
2  set -e
3
4
5  export REGION=us-west-2
6  export ENV=dev
7  export KEY_PAIR_NAME="cloudurable-$REGION"
8  export PEM_FILE="${HOME}/.ssh/${KEY_PAIR_NAME}.pem"
9  export SUBNET_PUBLIC=subnet-abc1234
10 export SUBNET_PRIVATE=subnet-abc1234
11 export CLOUD_FORMER_S3_BUCKET=abc1234-cloudformer-templates
12
13
14
15 export BASTION_NODE_SIZE=t2.small
16 export BASTION_SECURITY_GROUP=sg-abc1234
17 export BASTION_AMI=ami-abc1234
18 export BASTION_EC2_INSTANCE_NAME="bastion.${ENV}.${REGION}"
19 export BASTION_DNS_NAME="bastion.${ENV}.${REGION}.cloudurable.com."
20
21
22 export CASSANDRA_NODE_SIZE=m4.large
23 export CASSANDRA_AMI=ami-abc1234
24 export CASSANDRA_SECURITY_GROUP=sg-abc1234
25 export CASSANDRA_IAM_PROFILE=IAM_PROFILE_CASSANDRA
26 export CASSANDRA_EC2_INSTANCE_NAME="cassandra-node.${ENV}.${REGION}"
27 export CASSANDRA_DNS_NAME="node0.${ENV}.${REGION}.cloudurable.com."
28
29
30 export HOSTED_ZONE_ID="abc1234"

```



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](#).

AWS CLI: Create an EC2 instance

bin/create-ec2-instance.sh Create an EC2 instance based on our new AMI from Packer

```
#!/bin/bash
set -e

source bin/ec2-env.sh

instance_id=$(aws ec2 run-instances --image-id "$AMI_CASSANDRA" --subnet-id "$SUBNET_ID" \
--instance-type m4.large --iam-instance-profile "Name=IAM_PROFILE_CASSANDRA" \
--associate-public-ip-address --security-group-ids "$VPC_SECURITY_GROUP" \
--key-name "$KEY_NAME_CASSANDRA" | jq --raw-output .Instances[].InstanceId)

echo "${instance_id} is being created"

aws ec2 wait instance-exists --instance-ids "$instance_id"

aws ec2 create-tags --resources "$instance_id" --tags Key=Name,Value="${EC2_INSTANCE_NAME}"

echo "${instance_id} was tagged waiting to login"

aws ec2 wait instance-status-ok --instance-ids "$instance_id"

bin/login-ec2-cassandra.sh
```

See full code listings here: <https://github.com/cloudurable/cassandra-image/wiki/Cassandra-Tutorial-4:-Using-Packer-and-Ansible-to-create-and-manage-EC2-Cassandra-instances-in-AWS>

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-env.sh`.

AWS CLI: Login, Get IP

bin/login-ec2-cassandra.sh Log into new EC2 instance using ssh

```
#!/bin/bash
set -e

source bin/ec2-env.sh

if [ ! -f "$PEM_FILE" ]; then
    echo "Put your key file $PEM_FILE in your .ssh directory."
    exit 1
fi
ssh -i "$PEM_FILE" centos@`bin/get-IP-cassandra.sh`
```

bin/get-IP-cassandra.sh Get public IP address of new EC2 instance using aws cmdline

```
#!/bin/bash
set -e

source bin/ec2-env.sh

aws ec2 describe-instances --filters "Name=tag:Name,Values=${EC2_INSTANCE_NAME}" \
| jq --raw-output .Reservations[].Instances[].PublicIpAddress
```

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.

AWS CLI: Putting it all together

Interactive session showing everything running

```
$ pwd
~/github/cassandra-image
$ bin/create-ec2-instance.sh
i-013daca3d11137a8c is being created
i-013daca3d11137a8c was tagged waiting to login
The authenticity of host '54.202.110.114 (54.202.110.114)' can't be established.
ECDSA key fingerprint is SHA256:asdfsdfasdfsdfasdfsdf.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.202.110.114' (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 preset: enabled)
   Active: active (running) since Wed 2017-03-01 02:15:10 UTC; 14min ago
   Process: 456 ExecStart=/opt/cassandra/bin/cassandra -p /opt/cassandra/P
   Main PID: 5240 (java)
   CGroup: /system.slice/cassandra.service
           └─5240 java -Xloggc:/opt/cassandra/bin/../logs/gc.log -ea -XX:

Mar 01 02:14:13 ip-172-31-22-103.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
Unit metricsd.service could not be found.
[centos@ip-172-31-5-57 ~]$ systemctl status metricsd
● metricsd.service - MetricsD OS Metrics
   Loaded: loaded (/etc/systemd/system/metricsd.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2017-03-01 02:15:10 UTC; 14min ago
   Main PID: 5243 (metricsd)
```

See <https://github.com/cloudurable/cassandra-image/wiki/Cassandra-Tutorial-4:-Using-Packer-and-Ansible-to-create-and-manage-EC2-Cassandra-instances-in-AWS> for more details.

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.

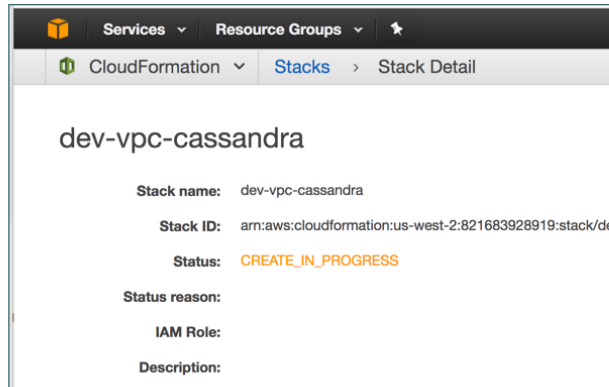
The options for configuration and orchestration management are endless (Puppet, Chef, Boto, etc.). This article and **Cloudurable** uses Ansible for many of these tasks. Ansible is an agentless architecture and works over **ssh** (secure shell) as we covered in our last article ([Setting up Ansible for 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.

Step 2: Run VPC CloudFormation

```
$ ./bin/run-vpc-cloudformation.sh
upload: cloud-formation/vpc.json to s3://cloudurable-cloudformer-templates/vpc.json
{
  "StackId": "arn:aws:cloudformation:us-west-2:821683928919:stack/dev-vpc-cassandra/3"
}
```



We covered the CloudFormation the whole time we were covering AWS basics.

Step 2: Watch CloudFormation logs

▼ Events

2017-03-17	Status	Type	Logical ID
▶ 16:49:33 UTC-0700	CREATE_COMPLETE	AWS::EC2::SubnetNetworkAclAssociation	subnetPrivate2AclAssociation
▶ 16:49:33 UTC-0700	CREATE_COMPLETE	AWS::EC2::SubnetRouteTableAssociation	subnetPrivate2RouteTableAssociation
▶ 16:49:32 UTC-0700	CREATE_COMPLETE	AWS::EC2::SubnetRouteTableAssociation	subnetPrivate1RouteTableAssociation
▶ 16:49:32 UTC-0700	CREATE_COMPLETE	AWS::EC2::SubnetRouteTableAssociation	subnetRouteTableAssociationPublic
▶ 16:49:32 UTC-0700	CREATE_COMPLETE	AWS::EC2::SubnetNetworkAclAssociation	subnetPrivate1AclAssociation
▶ 16:49:32 UTC-0700	CREATE_COMPLETE	AWS::EC2::SubnetNetworkAclAssociation	subnetAclAssociationPublic
▶ 16:49:32 UTC-0700	CREATE_COMPLETE		
▶ 16:49:18 UTC-0700	CREATE_COMPLETE		
▶ 16:49:17 UTC-0700	CREATE_COMPLETE		

Services ▼ Resource Groups ▼ ⭐

CloudFormation ▼ Stacks

Create Stack ▼ Actions ▼ Design template

Filter: Active ▼ By Stack Name

	Stack Name	Created Time	Status
<input type="checkbox"/>	dev-vpc-cassandra	2017-03-17 16:48:27 UTC-0700	CREATE_COMPLETE

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.

Step 2: Modify Env to match output of CloudWatch

▼ Outputs

Key	Value	Description
subnetPrivate1Out	subnet-dfc17696	Subnet Private Id
subnetPrivate2Out	subnet-10e07d77	Subnet Private Id
subnetPublicOut	subnet-dac17693	Subnet Public Id
securityGroupCassandraNodesOut	sg-01525a79	Cassandra Database Node secur...
securityGroupBastionOut	sg-00525a78	Security Group Bastion for mana...

ec2-env.sh x
ec2-env-TEMPLATE.sh x
create-ec2-instal

```

1  #!/bin/bash
2  set -e
3
4  export SUBNET_PUBLIC=subnet-dac17693
5  export SUBNET_PRIVATE1=subnet-dfc17696
6  export SUBNET_PRIVATE2=subnet-10e07d77
7  export BASTION_SECURITY_GROUP=sg-00525a78
8  export BASTION_AMI=ami-a0a729c0
9  export CASSANDRA_AMI=ami-a0a729c0
10 export CASSANDRA_SECURITY_GROUP=sg-01525a79
11
12

```

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.

Step 3: Launch the servers

create-ec2-entire-cassandra-cluster-instances.sh

```

1  #!/bin/bash
2  set -e
3
4  # Create Bastion
5  bin/create-ec2-instance-bastion.sh
6
7  # Create Cassandra seed nodes in AWS AZ a and AWS AZ b.
8  bin/create-ec2-instance-cassandra.sh 10.0.1.10 a
9  bin/create-ec2-instance-cassandra.sh 10.0.2.10 b
10
11
12 # Create two more Cassandra database servers in AZ a and b.
13 bin/create-ec2-instance-cassandra.sh 10.0.1.11 a
14 bin/create-ec2-instance-cassandra.sh 10.0.2.11 b
15

```

Launch Instance

Connect

Actions

Filter by tags and attributes or search by keyword

1 to 5 of 5

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
<input type="checkbox"/>	cassandra-node.dev.us-west-2	i-0060d7caa0d6ee5e6	m4.large	us-west-2a	running	2/2 checks passed
<input type="checkbox"/>	bastion.dev.us-west-2	i-06798823b85a8c375	t2.small	us-west-2a	running	2/2 checks passed
<input type="checkbox"/>	cassandra-node.dev.us-west-2	i-07a642c508a032513	m4.large	us-west-2a	running	2/2 checks passed
	cassandra-node.dev.us-west-2	i-024566165e10b1aec	m4.large	us-west-2b	pending	Initializing
	cassandra-node.dev.us-west-2	i-0d31b2937a6b4b4...	m4.large	us-west-2b	pending	Initializing

You can find more details here as well:

<https://github.com/cloudurable/cassandra-image/wiki/Cassandra-Tutorial-5:-Setting-up-Cassandra-Cluster-in-EC2-Part-1>

And here

<https://github.com/cloudurable/cassandra-image/wiki/Cassandra-Tutorial-6:-Setting-up-Cassandra-Cluster-in-EC2-Part-2-Multi-AZs-with-Ec2Snitch>

How does it know which IP to listen on?

- ❖ We could listen on *listen_interface* (*cassandra.yaml*)
- ❖ We listen on *listen_address*
- ❖ *But how do we know the address?*

We use EC2 meta-data

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

```
#!/bin/bash
set -e

export BIND_IP=`curl http://169.254.169.254/latest/meta-data/local-ipv4`

/opt/cloudurable/bin/cassandra-cloud -cluster-name test \
    -client-address ${BIND_IP} \
    -cluster-address ${BIND_IP} \
    -cluster-seeds 10.0.1.10,10.0.2.10 \
    -snitch Ec2Snitch

/bin/systemctl restart cassandra
```

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-cloud utility 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

```
$ curl http://169.254.169.254/latest/meta-data/ami-id  
ami-12345678
```

```
$ curl http://169.254.169.254/latest/meta-data/reservation-id  
r-fea54097
```

```
$ curl http://169.254.169.254/latest/meta-data/local-hostname  
ip-10-251-50-12.ec2.internal
```

```
$ curl http://169.254.169.254/latest/meta-data/public-hostname  
ec2-203-0-113-25.compute-1.amazonaws.com
```

This example gets the list of available public keys.

```
$ curl http://169.254.169.254/latest/meta-data/public-keys/  
0=my-public-key
```

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

```
$ curl http://169.254.169.254/latest/meta-data/public-keys/0/  
openssh-key
```


Checking the status of the Cassandra cluster

STEP 4 MAKE SURE IT WORKS!

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

$ ssh -F ssh/ssh.config bastion
Last login: Sat Mar 18 01:05:38 2017 from myhost.com

$ ssh 10.0.1.10
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 00:14:10 ip-10-0-0-118 systemd[1]: Starting Cassandra Service...
Mar 18 00:14:15 ip-10-0-0-118 systemd[1]: Started Cassandra Service.

$ /opt/cassandra/bin/nodetool status
Datacenter: us-west-2
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address      Load          Tokens         Owns (effective)  Host ID                               Rack
UN  10.0.2.10    163.15 KiB    32             50.7%             e5993967-26e8-485b 2b
UN  10.0.1.10    112.04 KiB    32             52.2%             47fbd9e8-0bcf-468f 2a
UN  10.0.1.11    107.31 KiB    32             47.1%             ac744cc8-4d91-47fe 2a
UN  10.0.2.11    221.87 KiB    32             50.1%             5dff0b8e-7cae-4e50 2b
```

[Setting up SSH and ansible Part 1](#)[Setting up SSH and ansible Part 2](#)

SSH config

[Setting up SSH and ansible Part 1](#)[Setting up SSH and ansible Part 2](#)

```
1 Host *.us-west-2.compute.amazonaws.com
2   ForwardAgent yes
3   IdentityFile ~/.ssh/test_rsa
4   User ansible
5
6 Host bastion
7   Hostname bastion.dev.us-west-2.cloudurable.com
8   ForwardAgent yes
9   IdentityFile ~/.ssh/test_rsa
10  User ansible
11
12 Host cassandra.node0
13   Hostname 10.0.1.10
14   ForwardAgent yes
15   IdentityFile ~/.ssh/test_rsa
16   ProxyCommand ssh bastion -W %h:%p
17   User ansible
18
19 Host 10.0.2.*
20   ForwardAgent yes
21   IdentityFile ~/.ssh/test_rsa
22   ProxyCommand ssh bastion -W %h:%p
23   User ansible
24   ControlMaster auto
25   ControlPath ~/.ssh/ansible-%r@%h:%p
26   ControlPersist 5m
27
28 Host 10.0.1.*
29   ForwardAgent yes
30   IdentityFile ~/.ssh/test_rsa
31   ProxyCommand ssh bastion -W %h:%p
32   User ansible
33   ControlMaster auto
34   ControlPath ~/.ssh/ansible-%r@%h:%p
35   ControlPersist 5m
```

Cassandra YAML (generated)

<pre>cluster_name: "test" num_tokens: 32 storage_port: 7000 ssl_storage_port: 7001 native_transport_port: 9042 endpoint_snitch: Ec2Snitch # Broadcast address for storage cluster communication # Listen address for client communication rpc_address: 10.0.1.10 # Listen address for storage cluster communication listen_address: 10.0.1.10 max_hints_delivery_threads: 2 data_file_directories: - /opt/cassandra/data commitlog_directory: /opt/cassandra/commitlog seed_provider: - class_name: org.apache.cassandra.locator.SimpleSeedProvider parameters: - seeds: "10.0.1.10,10.0.2.10"</pre>	<pre>cluster_name: "test" num_tokens: 32 storage_port: 7000 ssl_storage_port: 7001 native_transport_port: 9042 endpoint_snitch: Ec2Snitch # Broadcast address for storage cluster communication # Listen address for client communication rpc_address: 10.0.1.11 # Listen address for storage cluster communication listen_address: 10.0.1.11 max_hints_delivery_threads: 2 data_file_directories: - /opt/cassandra/data commitlog_directory: /opt/cassandra/commitlog seed_provider: - class_name: org.apache.cassandra.locator.SimpleSeedProvider parameters: - seeds: "10.0.1.10,10.0.2.10"</pre>
--	--

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

Ec2MultiRegionSnitch Pitfalls

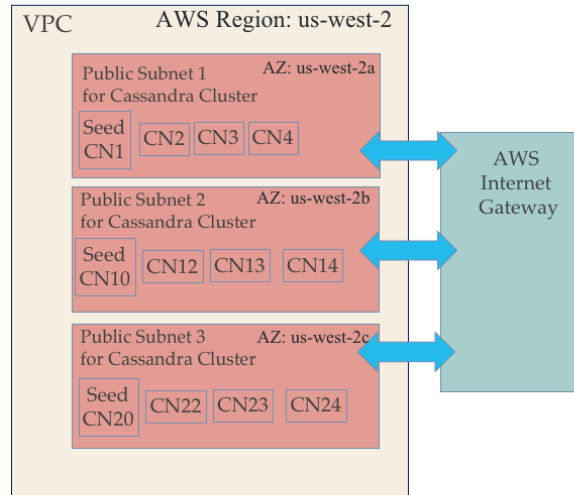
- ❖ Cassandra nodes have to live in a VPC public subnet
 - ❖ NACL and Security group can tighten access
- ❖ Traffic is over public Internet so anyone can listen
 - ❖ You have to use SSL
 - ❖ SSL Java performance is not great
- ❖ Setup is more difficult, you must setup seeds that are public IPs
 - ❖ use AWS ENIs, AWS Route 53 or AWS EIPs to make access to seed nodes more sane



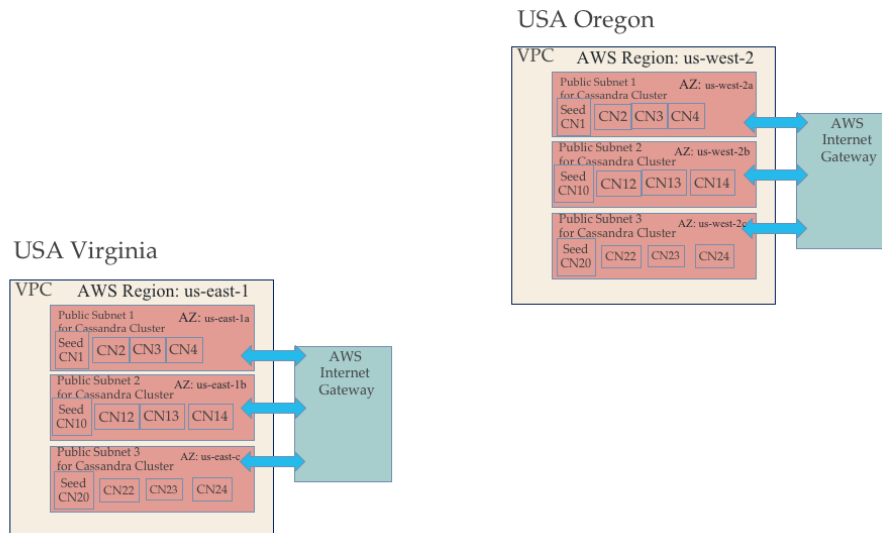
Cassandra nodes have to live in a VPC public subnet. You can use AWS NACL and Security group can tighten access.

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 to use 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.

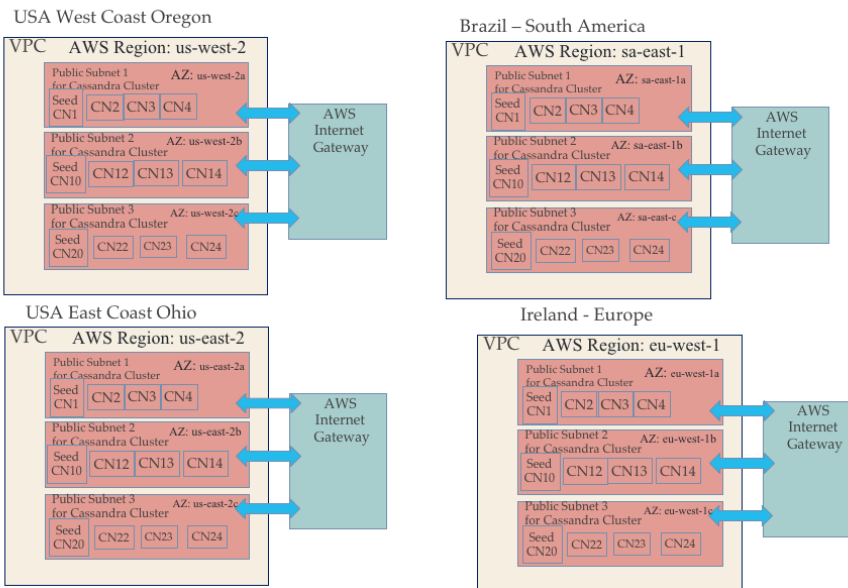
Ec2MultiRegionSnitch 1 region



Ec2MultiRegionSnitch 2 regions



Ec2MultiRegionSnitch 4 regions



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



First step SSL!

[Setting up Cassandra SSL](#)

- ❖ Set up Cassandra for SSL and CQL clients
- ❖ Cassandra allows you to secure the **client transport** (CQL) as well as the **cluster transport** (storage transport).
- ❖ We have to use SSL because we are using *Ec2MultiRegionSnitch* over the public Internet

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.

Why SSL? Just in case

[Setting up Cassandra SSL](#)

- ❖ 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](#).

setupkeys-cassandra-security.sh

[Setting up Cassandra SSL](#)

setupkeys-cassandra-security.sh

```
#!/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"
```

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.

Create the SSL Cassandra Key

[Setting up Cassandra SSL](#)**DBA/DevOps Cassandra Task: Create the Cassandra cluster key**

```
keytool -genkey -keyalg RSA -alias "${CLUSTER_NAME}_CLUSTER" -keystore "$KEY_STORE" -storepass "$PASSWORD" -keypass "$PASSWORD" \
-dname "CN=CloudDurable Image $CLUSTER_NAME cluster, OU=Cloudurable, O=Cloudurable, L=San Francisco, ST=CA, C=USA, DC=cloudurable, DC=com" \
-validity 36500
```

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.

Export a public key for the Cassandra cluster key

[Setting up Cassandra SSL](#)

Export a public key for the Cassandra cluster key.

```
# Create the public key for the client to identify itself.  
keytool -export -alias "${CLUSTER_NAME}_CLIENT" -file "$CLIENT_PUBLIC_CERT" -keystore "$KEY_STORE" \  
-storepass "$PASSWORD" -keypass "$PASSWORD" -noprompt
```

Once we create the cluster key, we will want to export a public key from it.

Import Public key for Cassandra cluster key into trust store

[Setting up Cassandra SSL](#)

Import public key for the Cassandra cluster key into the trust store so nodes can identify each other

```
# 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
```

Then we will import the public key into the trust store so that nodes can identify each other.

Creating client pem files

[Setting up Cassandra SSL](#)

Creating client pem files

```
keytool -importkeystore -srcalias "${CLUSTER_NAME}_CLIENT" -srckeystore "$KEY_STORE" -destkeystore "$PKS_
KEY_STORE" -deststoretype PKCS12 \
-srcstorepass "$PASSWORD" -deststorepass "$PASSWORD"

openssl pkcs12 -in "$PKS_KEY_STORE" -nokeys -out "$KEY_STORE_PATH/${CLUSTER_NAME}_CLIENT.cer.pem" -passin
pass:cassandra
openssl pkcs12 -in "$PKS_KEY_STORE" -nodes -nocerts -out "$KEY_STORE_PATH/${CLUSTER_NAME}_CLIENT.key.pem"
-passin pass:cassandra
```

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

[Setting up Cassandra SSL](#)

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

- 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](#).

Provision Cassandra to Use Keys

```
040-install-certs.sh x
1  #!/bin/bash
2
3  DESTINATION_DIRECTORY=/opt/cassandra/conf/certs
4  cd ~
5  SOURCE_DIRECTORY="$PWD/resources$DESTINATION_DIRECTORY"
6
7  echo $SOURCE_DIRECTORY
8
9
10 if [ -d "$SOURCE_DIRECTORY" ]; then
11     echo "$SOURCE_DIRECTORY was found making directory $DESTINATION_DIRECTORY"
12     mkdir -p "$DESTINATION_DIRECTORY"
13     echo "$SOURCE_DIRECTORY was found"
14     cp "${SOURCE_DIRECTORY}/*" "$DESTINATION_DIRECTORY"
15 fi
16
17
18 if [ ! -d "$SOURCE_DIRECTORY" ]; then
19     echo "UNABLE TO INSTALL CERTS AS THEY WERE NOT FOUND"
20 fi
21
22
23
24
```

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

/opt/cassandra/conf - DevOps task configure yaml with SSL keystore and trust stores

```
server_encryption_options:
  internode_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
  keystore_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

```
{
  "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": [
        "10.0.0.0/16",
        "10.1.0.0/16",
        "10.2.0.0/16",
        "10.3.0.0/16"
      ]
    },
    "subnetPublicCidr": {
      "Description": "Enter Public Subnet CIDR",
      "Type": "String",
      "Default": "10.1.0.0/24",
      "AllowedValues": [
        "10.0.0.0/24",
        "10.1.0.0/24",
        "10.2.0.0/24",
        "10.3.0.0/24"
      ]
    },
    "subnetCluster1Cidr": {
      "Description": "Enter Cluster Subnet Rack 1 CIDR",
      "Type": "String",
```

Parameterize launch script

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

```
...  
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

```
...  
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

bin/run-vpc-cloudformation.sh - load and use CIDRs per AWS Region / Cassandra Datacenter

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

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

source bin/ec2-env-region.sh

aws --region ${REGION} s3 cp cloud-formation/vpc.json s3://$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} \
    ParameterKey=subnetCluster1Cidr,ParameterValue=${SUBNET_CLUSTER1_CIDR} \
    ParameterKey=subnetCluster2Cidr,ParameterValue=${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 10.1.1.10 a us-west-2
bin/create-ec2-instance-cassandra.sh 10.1.2.10 b us-west-2
bin/create-ec2-instance-cassandra.sh 10.1.1.11 a us-west-2
bin/create-ec2-instance-cassandra.sh 10.1.2.11 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-bastion.sh 10.2.1.10 a us-east-2
bin/create-ec2-instance-cassandra.sh 10.2.2.10 b us-east-2
bin/create-ec2-instance-cassandra.sh 10.2.1.11 a us-east-2
bin/create-ec2-instance-cassandra.sh 10.2.2.11 b us-east-2
```

Switch clusters to Ec2MultiRegionSnitch

```

1  connect-regions.yml x
2
3  - hosts: all-nodes
4    gather_facts: no
5    become: true
6    remote_user: ansible
7    vars:
8      cluster_name: test
9      seeds: 52.14.159.203,52.14.180.53,54.202.57.109,54.149.126.26
10     aws_meta: http://169.254.169.254/latest/meta-data
11     bind_ip: "{{aws_meta}}/local-ipv4"
12     broadcast_ip: "{{aws_meta}}/public-ipv4"
13
14  tasks:
15    - name: "Copy template"
16      copy:
17        src: ../resources/opt/cassandra/conf/cassandra-yaml.template
18        dest: /opt/cassandra/conf/cassandra-yaml.template
19        owner: cassandra
20        group: cassandra
21        mode: "u=rw,g=r,o=r"
22
23    - name: "Get bind_ip"
24      command: curl {{bind_ip}}
25      register: bind_ip_contents
26
27    - name: "Get broadcast_ip"
28      command: curl {{broadcast_ip}}
29      register: broadcast_ip_contents
30
31    - name: Configure Cassandra
32      command: "/opt/cloudurable/bin/cassandra-cloud -cluster-name {{cluster_name}} \
33        -client-address {{bind_ip_contents.stdout}} \
34        -cluster-address {{bind_ip_contents.stdout}} \
35        -cluster-seeds {{seeds}} \
36        -cluster-broadcast-address {{broadcast_ip_contents.stdout}} \
37        -snitch Ec2MultiRegionSnitch"
38
39    - name: Restart Cassandra
40      command: "/bin/systemctl restart cassandra"
41
42

```

Here we use an ansible playbook to run cassandra cloud

Using cassandra-cloud from Ansible

```
vars:
  cluster_name: test
  seeds: 52.14.159.203,52.14.108.53,54.202.57.109,54.149.126.26
  aws_meta: http://169.254.169.254/latest/meta-data
  bind_ip: "{{aws_meta}}/local-ipv4"
  broadcast_ip: "{{aws_meta}}/public-ipv4"
```

```
- name : "Get bind_ip"
  command: curl {{bind_ip}}
  register: bind_ip_contents
```

```
- name : "Get broadcast_ip"
  command: curl {{broadcast_ip}}
  register: broadcast_ip_contents
```

```
- name: Configure Cassandra
  command: "/opt/cloudurable/bin/cassandra-cloud -cluster-name {{cluster_name}} \
    -client-address {{bind_ip_contents.stdout}} \
    -cluster-address {{bind_ip_contents.stdout}} \
    -cluster-seeds {{seeds}} \
    -cluster-broadcast-address {{broadcast_ip_contents.stdout}} \
    -snitch Ec2MultiRegionSnitch"
```

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: [10.1.2.11]
```

```
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
```

```
10.1.2.10      : ok=5    changed=4    unreachable=0    failed=0
```

```
10.1.2.11      : 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
```

```
10.2.2.10      : ok=5    changed=4    unreachable=0    failed=0
```

```
10.2.2.11      : ok=5    changed=4    unreachable=0    failed=0
```

Verify the setup

Verify new setup

status on seed1

```
$ ansible seed1 -a "/opt/cassandra/bin/nodetool status"
10.1.1.10 | SUCCESS | rc=0 >>
Datacenter: us-east-2
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address      Load      Tokens     Owns (effective)  Host ID
UN  52.14.159.203 198.57 KiB 32         28.1%             3b268d9b-56dd-48a7-bf
UN  52.14.156.148 241.97 KiB 32         24.5%             c73e4c97-6b3e-4fc5-ac
UN  52.14.108.53  239.15 KiB 32         26.1%             38a314ff-a0a0-40b2-8e
UN  52.14.139.245 195.85 KiB 32         25.4%             9fa3c21d-7d12-4a1a-81
Datacenter: us-west-2
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address      Load      Tokens     Owns (effective)  Host ID
UN  54.202.137.143 194.06 KiB 32         24.1%             4afd0d96-6536-40af-8e
UN  54.202.57.109  234.12 KiB 32         26.5%             32002e7b-2867-47ce-bf
UN  54.187.138.240 220.24 KiB 32         20.1%             f13f3aef-035b-47a1-a1
UN  54.149.126.26  194.18 KiB 32         25.3%             440eef64-1471-4711-9e
```

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

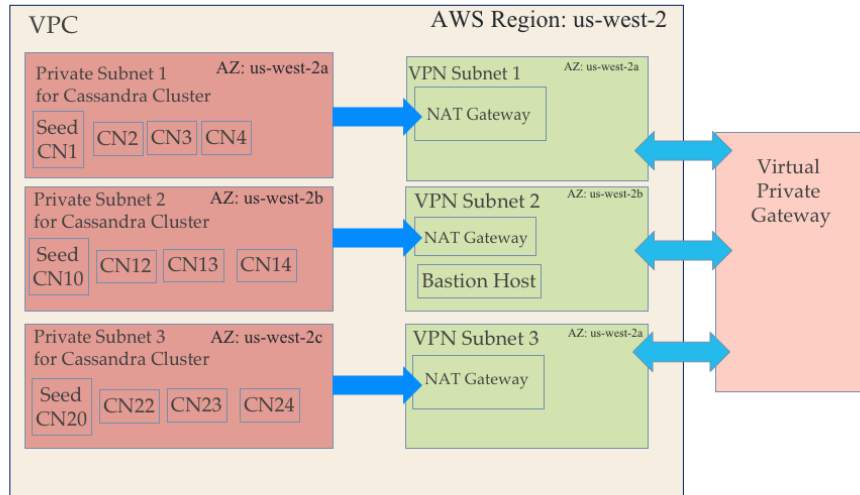
10.2.2.11

Connecting Cluster via VPN

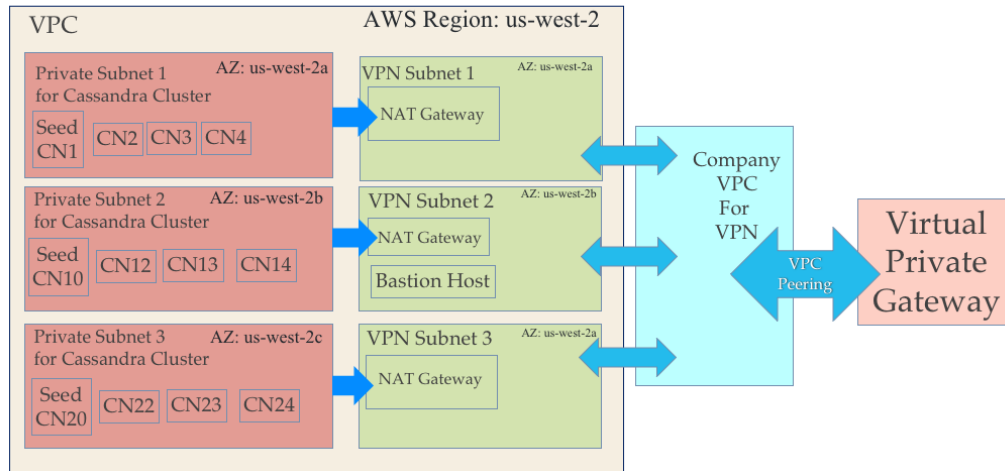
Using VPN instead of public Internet for Multi DC/Region Cassandra Setup

- ❖ 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 communication cross region

GossipingPropertyFileSnitch *to* VPN

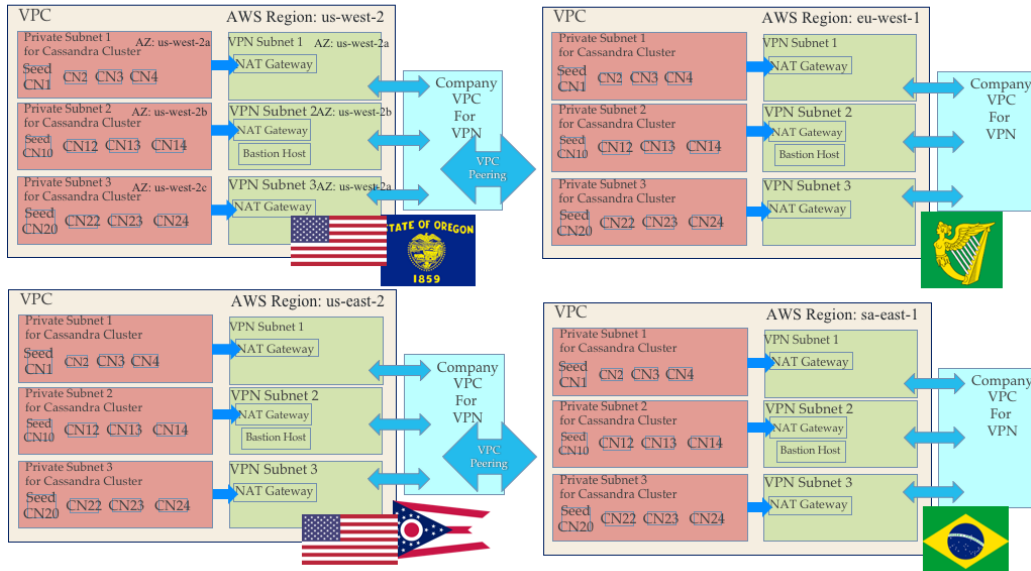


GossipingPropertyFileSnitch via VPN via VPC Peer



GossipingPropertyFileSnitch

4 regions / DCs





[AWS Cassandra Support](#)

Cassandra / Kafka Support in EC2/AWS

Final Thoughts

Some 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)



[AWS Cassandra Support](#)

Cassandra / Kafka Support in EC2/AWS

Company Overview

How we got our start

Different companies same challenges

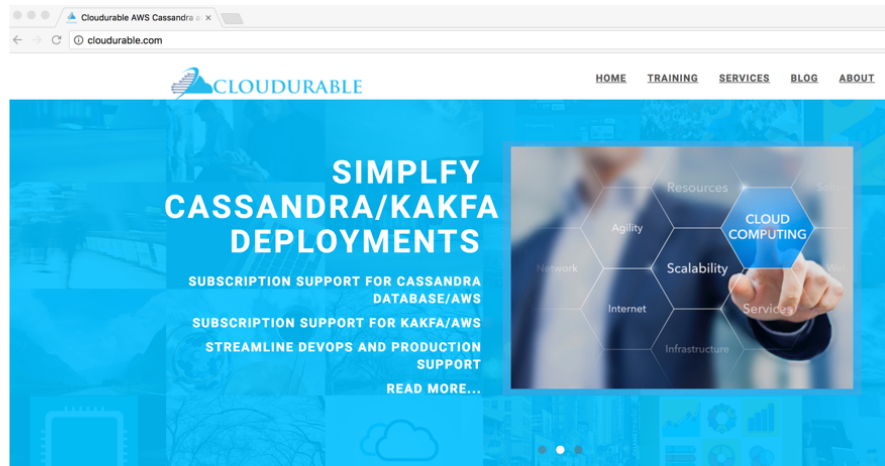
[AWS Cassandra Support](#)

- ❖ 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

[AWS Cassandra Support](#)

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



Visit us: <http://cloudurable.com/>

FOLLOW CLOUDURABLE™

[facebook page](#)

[google plus](#)

[twitter](#)

[linkedin](#)

PART 2 SETTING UP ANSIBLE AND SSH FOR CASSANDRA DATABASE CLUSTER DEVOPS

IN [ANSIBLE](#)

[March 11, 2017](#)

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.

SETTING UP ANSIBLE/SSH FOR CASSANDRA DATABASE CLUSTER DEVOPS PART 1

IN [ANSIBLE](#)

[March 11, 2017](#)

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 articles 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/OpenSSL 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 systemd 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](#)



Cassandra / Kafka Support in EC2/AWS

FOLLOW CLOUDURABLE™

[facebook page](#)

[google plus](#)

[twitter](#)

[linkedin](#)