Cassandra and AWS Support on AWS/EC2

Comprehensive Guide to Deploying Cassandra on AWS

- 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

March 2017 – version 2
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
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 Internet Gateway
  - Private Subnets – no route to Internet Gateway
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.
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
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
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?
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?
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:

```bash
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.
Amazon Elastic Block Store (Amazon EBS)

EC2 Instance

You can attach multiple volumes to the same EC2 instance in the same availability zone.
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*
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
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
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?
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) 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?
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
# EBS Type Review

<table>
<thead>
<tr>
<th></th>
<th>Solid-State Drives (SSD)</th>
<th>Hard disk Drives (HDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Volume Type</strong></td>
<td>General Purpose SSD (gp2)*</td>
<td>Provisioned IOPS SSD (io1)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>General purpose SSD volume that balances price and performance for a wide variety of transactional workloads</td>
<td>Highest-performance SSD volume designed for mission-critical applications</td>
</tr>
<tr>
<td><strong>Volume Size</strong></td>
<td>1 GiB - 16 TiB</td>
<td>4 GiB - 16 TiB</td>
</tr>
<tr>
<td><em><em>Max. IOPS</em>/Volume</em>*</td>
<td>10,000</td>
<td>20,000</td>
</tr>
<tr>
<td><strong>Max. Throughput/Volume</strong></td>
<td>160 MiB/s</td>
<td>320 MiB/s</td>
</tr>
<tr>
<td><strong>Max. IOPS/Instance</strong></td>
<td>65,000</td>
<td>65,000</td>
</tr>
<tr>
<td><strong>Max. Throughput/Instance</strong></td>
<td>1,250 MiB/s</td>
<td>1,250 MiB/s</td>
</tr>
<tr>
<td><strong>Dominant Performance Attribute</strong></td>
<td>IOPS</td>
<td>IOPS</td>
</tr>
</tbody>
</table>

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

EC2 m4

EBS Non-provisioned IOPs

200 GB

200 GB

200 GB

200 GB

Max IOPs for read?
EBS Volumes

Max IOPs for random read?

EC2 m4

EBS Non-provisioned IOPs

2 TB

2 TB

2 TB

2 TB
EBS Volumes

Max IOPs for random read?
EBS Volumes

Max IOPs for random read?

Provisioned IOPs
- 200 GB
- 200 GB
- 200 GB

EC2 m4

Cassandra / Kafka Support in EC2/AWS
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.

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

Max IOPs for random read?
NOT EBS: Instance storage

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

- Newer EC2 instance types support EBS Optimized
- Higher throughput
- Less jiggle
- More reliable
- Don’t use C3 or M3 use C4 and M4
- Uses Optimized by default
- New Feature added in Feb 2017: Elastic Volumes!
EBS: 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
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 a new instance with a volume created from the last snapshot or just let Cassandra repopulate the node?
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
```
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
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.
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
CloudFormation for VPC

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

<table>
<thead>
<tr>
<th>CIDR Range</th>
<th>10.0.0.0/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netmask</td>
<td>255.255.0.0</td>
</tr>
<tr>
<td>Wildcard Bits</td>
<td>0.0.255.255</td>
</tr>
<tr>
<td>First IP</td>
<td>10.0.0.0</td>
</tr>
<tr>
<td>Last IP</td>
<td>10.0.255.255</td>
</tr>
<tr>
<td>Total Host</td>
<td>65536</td>
</tr>
</tbody>
</table>
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)
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" 
      }
    }
  
```

<table>
<thead>
<tr>
<th>CIDR Range</th>
<th>Netmask</th>
<th>Wildcard Bits</th>
<th>First IP</th>
<th>Last IP</th>
<th>Total Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/24</td>
<td>255.255.255.0</td>
<td>0.0.0.255</td>
<td>10.0.0.0</td>
<td>10.0.0.255</td>
<td>256</td>
</tr>
<tr>
<td>10.0.1.0/24</td>
<td>255.255.255.0</td>
<td>0.0.0.255</td>
<td>10.0.1.0</td>
<td>10.0.1.255</td>
<td>256</td>
</tr>
<tr>
<td>10.0.2.0/24</td>
<td>255.255.255.0</td>
<td>0.0.0.255</td>
<td>10.0.2.0</td>
<td>10.0.2.255</td>
<td>256</td>
</tr>
</tbody>
</table>
Internet 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.
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)
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"
        }
    }
}
```
VPC VPN Access via VGW and CGW

- AWS to augment your existing IT infrastructure via VPN
- Connect existing datacenter to VPC using VPG (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
- VPGs connect to your companies - VPG is the Amazon side of the VPN connection
- CGW is the customer side of the VPN connector
- CGWs are processes running on a server or network device.
- Connect a VPG 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?
NAT Gateway and EIP
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.
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)
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?
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?
Security Groups
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)
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"
    }
  ]
}
```

Inbound Rules

<table>
<thead>
<tr>
<th>Type</th>
<th>Protocol</th>
<th>Port Range</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH (22)</td>
<td>TCP (6)</td>
<td>22</td>
<td>0.0.0.0/0</td>
</tr>
</tbody>
</table>

Add another rule

Remove
The above allows all traffic from the VPC’s CIDR to access this box.
-1 means all ports.
What is different about this SG for Cassandra than the last one?

How could we narrow which EC2 instances access Cassandra nodes?
CloudFormation: NACL
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?
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
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?
Create a new CloudFormation stack (like JSON file from earlier)

```bash
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

```json
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)
```
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.
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
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

- 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
Cassandra AWS Overview

AWS Cassandra Architecture

Cassandra Architecture Review
What is Cassandra?

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

AWS Cassandra Architecture
AWS Cassandra Concepts: Cluster

- 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
AWS Cassandra Concepts: Commit log and in-memory

- 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
AWS Cassandra: SSTable, Keyspace

- SSTable and Commit log should be stored on separate EBS volume (especially if not SSD)
- SSTable disk representation of Memtable
- Type of compaction can range between 20% and 50% overhead
  - Account for this when allocating EBS volumes
- Keyspaces dictate replication factors, more can mean more IO (IOPs and network bandwidth needs)
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?
AWS EC2 Instance Types for Cassandra

AWS Cassandra Storage Reqs.
Compute Optimized

C4

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

Features:

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

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

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

Features:

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

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
<th>SSD Storage (GB)</th>
<th>Dedicated EBS Bandwidth (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m4.large</td>
<td>2</td>
<td>8</td>
<td>EBS-only</td>
<td>450</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>16</td>
<td>EBS-only</td>
<td>750</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>32</td>
<td>EBS-only</td>
<td>1,000</td>
</tr>
<tr>
<td>m4.4xlarge</td>
<td>16</td>
<td>64</td>
<td>EBS-only</td>
<td>2,000</td>
</tr>
<tr>
<td>m4.10xlarge</td>
<td>40</td>
<td>160</td>
<td>EBS-only</td>
<td>4,000</td>
</tr>
<tr>
<td>m4.16xlarge</td>
<td>64</td>
<td>256</td>
<td>EBS-only</td>
<td>10,000</td>
</tr>
</tbody>
</table>

If in doubt, start with M4
Storage Optimized

**I3 – High I/O Instances**

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

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

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

D2 – Dense-storage Instances

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

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

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

Use Cases

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

Could be used in an analytics Cassandra DC that is connected to another DC that uses SSD for frequent small reads. Starts at 6TB per node. IoT device monitoring.
# AWS Cassandra Storage Reqs.

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Storage optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance Storage (GB)</strong></td>
<td><strong>Linux/UNIX Usage</strong></td>
</tr>
<tr>
<td>EBS Only</td>
<td>EBS Only</td>
</tr>
<tr>
<td><strong>vCPU</strong></td>
<td><strong>ECU</strong></td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
</tr>
<tr>
<td>m4.4xlarge</td>
<td>16</td>
</tr>
<tr>
<td>m4.10xlarge</td>
<td>40</td>
</tr>
<tr>
<td>m4.16xlarge</td>
<td>64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Compute Optimized - Current Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance Storage (GB)</strong></td>
<td><strong>Linux/UNIX Usage</strong></td>
</tr>
<tr>
<td>EBS Only</td>
<td>EBS Only</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
</tr>
<tr>
<td>m4.4xlarge</td>
<td>16</td>
</tr>
<tr>
<td>m4.10xlarge</td>
<td>40</td>
</tr>
<tr>
<td>m4.16xlarge</td>
<td>64</td>
</tr>
</tbody>
</table>

- **Trivia:**
  - What is ECU? [Answer]
  - What does vCPU equate to? [Answer]
Cassandra / Kafka Support in EC2/AWS

Cassandra AWS Storage Guidelines

Cassandra storage requirements for Amazon EC2 and EBS
Cassandra AWS - Storage Requirements

- Cassandra does lots of sequential disk IO (commit logs, SSTables)
- Cassandra writes large streams of data to commit logs SSTable, index files and bloom filter files
- Put commit log on separate disk (EBS volume) than SSTables
  - Maybe ok if using SSD
  - Never if HDD, always separate volume if HDD
Consider EC2 Instance Storage

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

- 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)
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?
<table>
<thead>
<tr>
<th>Volume Type</th>
<th>Solid State Drives (SSD)</th>
<th>Hard Disk Drives (HDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EBS Provisioned IOPS SSD (io1)</td>
<td>EBS General Purpose SSD (gp2)*</td>
</tr>
<tr>
<td>Short Description</td>
<td>Highest performance SSD volume designed for latency-sensitive transactional workloads</td>
<td>General Purpose SSD volume that balances price performance for a wide variety of transactional workloads</td>
</tr>
<tr>
<td>Use Cases</td>
<td>I/O-intensive NoSQL and relational databases</td>
<td>Boot volumes, low-latency interactive apps, dev &amp; test</td>
</tr>
<tr>
<td>API Name</td>
<td>io1</td>
<td>gp2</td>
</tr>
<tr>
<td>Volume Size</td>
<td>4 GB - 16 TB</td>
<td>1 GB - 16 TB</td>
</tr>
<tr>
<td>Max IOPS/Volume</td>
<td>20,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Max Throughput/Volume</td>
<td>320 MB/s</td>
<td>160 MB/s</td>
</tr>
<tr>
<td>Max IOPS/Instance</td>
<td>65,000</td>
<td>65,000</td>
</tr>
<tr>
<td>Max Throughput/Instance</td>
<td>1,250 MB/s</td>
<td>1,250 MB/s</td>
</tr>
<tr>
<td>Price</td>
<td>$0.125/GB-month</td>
<td>$0.10/GB-month</td>
</tr>
<tr>
<td>Dominant Performance Attribute</td>
<td>IOPS</td>
<td>IOPS</td>
</tr>
</tbody>
</table>
Cassandra EC2 instance good picks

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

🌟 What is the max total IOPS with M4?
🌟 What is the max total IOPS with I3?
SSD vs HDD

- 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
Take Replication into Account

- 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)
RAID, JBOD, Read speed

- 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)
EBS Elastic Volumes and Linux File Systems

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

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

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

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

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
AWS Cassandra CPU Guidelines

Cassandra AWS CPU Guidelines

Cassandra CPU requirements with Amazon EC2
Cassandra CPU Reqs for AWS Cloud

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

- 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](#).
Compaction strategy and CPU usage

- Compaction strategy can influence CPU Spikes
  - `SizeTieredCompactionStrategy` works with larger `SSTables` so has more spikey CPU
    - Need for faster writes
  - `LeveledCompactionStrategy` will use a more even level of CPU
    - Needed for more consistent CPU usage
    - Needed for faster reads
Use 8 vCPUs for prod

- 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
CPU Usage for GC1 Garbage collector and CMS

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

What happens if a node under heavy load has a stop the world GC?
Multi Region/DC need more EC2 vCPUs

- If you are using multiple regions, i.e., a multi-dc deployment increase `max_hints_delivery_threads` as cross DC handoffs are slower

- Also keep in mind for cluster/storage communication that there is more CPU overhead, which might be a wash if the DC to DC communication has a lot of latency, using SSL, then more too

- How many threads do you need? How eventually consistent do you want to be between data-centers? And how long will these threads be waiting for IO?

- Don’t just consider happy case, but unhappy case
Cassandra workloads with large datasets

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

- **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`
Horizontal scale is not always possible with Cassandra

- When you are using Cassandra for super high-speed writes or using it with very large datasets

- You may have to scale up your Cassandra nodes and add more vCPU and memory by using larger EC2 instances

- Vertical scale-up is also needed in some cases

- When horizontally scaling, Cassandra does a lot of streaming, you could make instances larger (gradually), add your nodes, and then gradually make the Cassandra instances smaller
EC2, Cassandra and NUMA

- 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 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
CPU Cache sizes

- Use latest generation of EC2 instances as they have larger L1 and L2 cache sizes (between 25 MB and 30 MB of on CPU cache)
Cassandra / Kafka Support in EC2/AWS

AWS Cassandra System Memory Guidelines

AWS Cassandra System Memory Guidelines

Cassandra system memory guidelines for Amazon EC2
Basic System Memory Guidelines

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

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

Some grow as JVM heap grows
Cassandra uses memory 4 ways

- Java heap
- Off-heap memory
- OS page cache
- OS TCP/IP stack I/O cache
Memory: More is better

- 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
Cassandra needs OS memory!

- **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 Off-heap memory usage

- Page cache
- Bloom filter
- Compression offset maps
- Row caches
# Cassandra JVM vs Linux System Memory table for EC2

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

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

- `-Xms12G`
- `-Xmx12G`

- `-Xms` sets min heap size and `-Xmx` sets max heap size
JVM Garbage Collector for Cassandra

- 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 GGC1 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
Cassandra / Kafka Support in EC2/AWS

Garbage Collectors

- Serial GC
- Parallel GC
- CMS

G1 Heap Allocation

Credit: Oracle

How young GC works in G1 (ii)

- Live objects from young generation are evacuated (copied/moved) to
  - One or more survivor regions
  - Old generation regions
- This is a STW pause
- Next GC's eden & survivor size calculated

Garbage First Collector Architecture Overview

- Heap is divided into 2K non-contiguous regions of Eden, survivor, and old spaces, region size can be 1MB, 2MB, 4MB, 8MB, 16MB, 32MB.
- Humongous regions are old regions for large objects that are larger than 1/2 of region size. Humongous objects are stored in contiguous regions in heap.
- Number of regions in eden and survivor can be changed between GC's.
- Young GC: multi-threaded, low stop-the-world pauses
- Concurrent marking and clean up: mostly parallel, multi-threaded
- Mixed GC: multi-threaded, incremental GC, collects and compacts heap partially, more expensive stop-the-world pauses
- Full GC: Serial, collects and compacts entire heap

Amazon EC2 instances taking CMS out of equation!

Note: Amazon has a [Cassandra on Amazon](https://docs.aws.amazon.com/cassandra/latest/beta2/guide/hr/optimizing-memory.html) guide that says never use over 8 GB for JVM. **This is a mistake! See note!**

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

Young Generation

Old Generation

To set heap size for `m4.2xlarge`

- `Xms12G`
- `Xmx12G`

-Xms sets min heap size and -Xms sets max heap size
Guideline for GCG1 Cassandra JVM

GCG1 will self adjust using ergonomics and runtime statistics. The only setting that you really need to set is –XX:MaxGCPauseMillis.
GC settings for 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
Don’t use CMS

- 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

AWS Cassandra System Memory Guidelines
- 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
Guide to using CMS

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
Cassandra / Kafka Support in EC2/AWS

Credit: Image source Intel from talk at HBaseCon2015

After: Very Large Heap Low Pause Credit: Intel: HBaseCon

-XX:+UseG1GC -XX:MaxGCPauseMillis=100
-XX:G1HeapWastePercent=20 -XX:InitiatingHeapOccupancyPercent=75
-XX:ConcGCThreads=32 -XX:ParallelGCThreads=48
Take pressure off JVM GC by using off-heap

- **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`

Credit: Facebook Tech Talk On JEMalloc
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
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
Enhanced networking vs. none

<table>
<thead>
<tr>
<th></th>
<th>Enhanced Networking</th>
<th>Use enhanced networking!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Median</td>
<td>95th percentile</td>
</tr>
<tr>
<td>Single AZ</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td>Multi AZ (3 AZs)</td>
<td>1.5</td>
<td>2.8</td>
</tr>
<tr>
<td>Read</td>
<td>Median</td>
<td>95th percentile</td>
</tr>
<tr>
<td>Single AZ</td>
<td>1.0</td>
<td>2.6</td>
</tr>
<tr>
<td>Multi AZ (3 AZs)</td>
<td>1.5</td>
<td>23.5</td>
</tr>
</tbody>
</table>

10 GBE

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

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

- What happens if the AZ goes down?
- Do your reads stop working?
- Do your writes stop working?
- Availability Zones <= Replication Factor
Failure Scenarios

6 nodes cluster  
AZs = 2  RF = 2  

Scenario 1 - 1 AZ down

Consistency level | Result
--- | ---
ONE | OK
ANY | OK
QUORUM | FAIL
ALL | FAIL

6 nodes cluster  
AZs = 3  RF = 3  

Scenario 1 - 1 AZ down

Consistency level | Result
--- | ---
ONE | OK
ANY | OK
QUORUM | OK
ALL | FAIL

Scenario 2 - 2 AZs down

Consistency level | Result
--- | ---
ONE | OK
ANY | OK
QUORUM | FAIL
ALL | FAIL

Source/Credit: Highscalability.com  Todd Hoff
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
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
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
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
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
Cassandra / Kafka Support in EC2/AWS

**EC2Snitch layout 1 Region**

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

**AWS Region: us-west-2**
- Public Subnet 1: AZ: us-west-2a
  - NAT Gateway
- Public Subnet 2: AZ: us-west-2b
  - NAT Gateway
  - Bastion Host
- Public Subnet 3: AZ: us-west-2c
  - NAT Gateway

**AWS Internet Gateway**
Steps

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

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

Utility for configuring Cassandra for CloudDeployments

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

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


/opt/cloudurable/bin/cassandra-cloud -cluster-name test \\
  -client-address ${{BIND_IP}} \\
  -cluster-address ${{BIND_IP}} \\
  -cluster-seeds 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

```json
packer-ec2.json

{
  "variables": {
    "aws_access_key": "",
    "aws_secret_key": "",
    "aws_region": "us-west-2",
    "aws_amis": "ami-d2c924b2",
    "aws_instance_type": "m4.large",
    "image_version": "0.2.2"
  },
  "builders": [
    {
      "type": "amazon-ebs",
      "access_key": "{{user `aws_access_key`}}",
      "secret_key": "{{user `aws_secret_key`}}",
      "region": "{{user `aws_region`}}",
      "source_amis": "{{user `aws_amis`}}",
      "instance_type": "{{user `aws_instance_type`}}",
      "ssh_username": "centos",
      "ami_name": "cloudurable-cassandra-{{user `image_version`}}",
      "tags": {
        "Name": "cloudurable-cassandra-{{user `image_version`}}",
        "OS_Version": "LinuxCentos7",
        "Release": "7",
        "Description": "CentOS 7 image for Cloudurable Cassandra image"
      },
      "user_data_file": "config/user-data.sh"
    }
  ],
  "provisioners": [
    {
      "type": "file",
      "source": "scripts",
      "destination": "/home/centos/
    },
    {
      "type": "file",
      "source": "resources",
      "destination": "/home/centos/
    },
    {
      "type": "shell",
      "scripts": [
        "scripts/000-ec2-provision.sh"
      ],
      "playbook_file": "playbooks/ssh-addkey.yml"
    }
  ]
}
```

Read notes

```
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
```
Step 1: Build AMI with Packer

Read notes!
Step 1: Build AMI Monitoring tools

Installing metricsd systemd from our provisioning scripts

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

```
[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
```
Step 1: Build AMI: Log aggregation

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

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

/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
```

systemd-cloud-watch.conf

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

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](https://github.com/cloudurable/systemd-cloud-watch).
Step 1: Build AMI: Install Cassandra

```
[Unit]
Description=Cassandra Service

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

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

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

[Install]
WantedBy=multi-user.target
```
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
AWS CLI: Common scripts we use

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

export REGION=us-west-2
export ENV=dev
export KEY_PAIR_NAME="cloudurable-$REGION"
export PEM_FILE="$HOME/.ssh/$(KEY_PAIR_NAME).pem"
export SUBNET_PUBLIC=subnet-abc1234
export SUBNET_PRIVATE=subnet-abc1234
export CLOUDFORMER_S3_BUCKET=abc1234-cloudformer-templates

export BASTION_NODE_SIZE=t2.small
export BASTION_SECURITY_GROUP=sg-abc1234
export BASTION_AMI=ami-abc1234
export BASTION_EC2_INSTANCE_NAME="bastion.$ENV.$REGION"
export BASTION_DNS_NAME="bastion.$ENV.$REGION.cloudurable.com."

export CASSANDRA_NODE_SIZE=m4.large
export CASSANDRA_AMI=ami-abc1234
export CASSANDRA_SECURITY_GROUP=sg-abc1234
export CASSANDRA_IAM_PROFILE=IAM_PROFILE_CASSANDRA
export CASSANDRA_EC2_INSTANCE_NAME="cassandra-node.$ENV.$REGION"
export CASSANDRA_DNS_NAME="node0.$ENV.$REGION.cloudurable.com."

export HOSTED_ZONE_ID="abc1234"
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 "$SUBNET" --instance-type m4.large --iam-instance-profile "Name=$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
bin/login-ec2-cassandra.sh Log into new EC2 instance using ssh

```bash
#!/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

```bash
#!/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
```
AWS CLI: Putting it all together

Interactive session showing everything running

```bash
$ 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 estab
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

[centos@ip-172-31-5-57 ~]$ systemctl status cassandra
● cassandra.service - Cassandra Service
    Loaded: loaded (/etc/systemd/system/cassandra.service; enabled; vendor)
    Active: active (running) since Wed 2017-03-01 02:15:10 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 metricds
Unit metricds.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)
    Active: active (running) since Wed 2017-03-01 02:15:10 UTC; 14min ago
    Main PID: 5243 (metricsd)
```
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.
Cassandra / Kafka Support in EC2/AWS

Step 2: Run VPC CloudFormation

```bash
$ ./bin/run-vpc-cloudformation.sh
upload: cloud-formation/vpc.json to s3://cloudurable-cloudformer-templates/vpc.json
{
}
```

![CloudFormation Stack Details](image)

- **Stack name**: dev-vpc-cassandra
- **Status**: CREATE_IN_PROGRESS
### Step 2: Watch CloudFormation logs

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Status</th>
<th>Type</th>
<th>Logical ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-03-17</td>
<td>16:49:33 UTC-0700</td>
<td>CREATE_COMPLETE</td>
<td>AWS::EC2::SubnetNetworkAclAssociation</td>
<td>subnetPrivate2AclAssociation</td>
</tr>
<tr>
<td></td>
<td>16:49:33 UTC-0700</td>
<td>CREATE_COMPLETE</td>
<td>AWS::EC2::SubnetRouteTableAssociation</td>
<td>subnetPrivate2RouteTableAssociation</td>
</tr>
<tr>
<td></td>
<td>16:49:32 UTC-0700</td>
<td>CREATE_COMPLETE</td>
<td>AWS::EC2::SubnetRouteTableAssociation</td>
<td>subnetPrivate1RouteTableAssociation</td>
</tr>
<tr>
<td></td>
<td>16:49:32 UTC-0700</td>
<td>CREATE_COMPLETE</td>
<td>AWS::EC2::SubnetRouteTableAssociation</td>
<td>subnetRouteTableAssociationPublic</td>
</tr>
<tr>
<td></td>
<td>16:49:32 UTC-0700</td>
<td>CREATE_COMPLETE</td>
<td>AWS::EC2::SubnetNetworkAclAssociation</td>
<td>subnetPrivate1AclAssociation</td>
</tr>
<tr>
<td></td>
<td>16:49:32 UTC-0700</td>
<td>CREATE_COMPLETE</td>
<td>AWS::EC2::SubnetNetworkAclAssociation</td>
<td>subnetAclAssociationPublic</td>
</tr>
<tr>
<td></td>
<td>16:49:18 UTC-0700</td>
<td>CREATE_COMPLETE</td>
<td>AWS::EC2::SubnetNetworkAclAssociation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16:49:17 UTC-0700</td>
<td>CREATE_COMPLETE</td>
<td>AWS::EC2::SubnetNetworkAclAssociation</td>
<td></td>
</tr>
</tbody>
</table>

 filtrable active

<table>
<thead>
<tr>
<th>Stack Name</th>
<th>Created Time</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev-vpc-cassandra</td>
<td>2017-03-17 16:48:27 UTC-0700</td>
<td>CREATE_COMPLETE</td>
</tr>
</tbody>
</table>
Step 2: Modify Env to match output of CloudWatch

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subnetPrivate1Out</td>
<td>subnet-dfc17696</td>
<td>Subnet Private Id</td>
</tr>
<tr>
<td>subnetPrivate2Out</td>
<td>subnet-10e07d77</td>
<td>Subnet Private Id</td>
</tr>
<tr>
<td>subnetPublicOut</td>
<td>subnet-dac17693</td>
<td>Subnet Public Id</td>
</tr>
<tr>
<td>securityGroupCassandraNodesOut</td>
<td>sg-01525a79</td>
<td>Cassandra Database Node secu...</td>
</tr>
<tr>
<td>securityGroupBastionOut</td>
<td>sg-00525a78</td>
<td>Security Group Bastion for mana...</td>
</tr>
</tbody>
</table>

```bash
#!/bin/bash
set -e
export SUBNET_PUBLIC=subnet-dac17693
export SUBNET_PRIVATE1=subnet-dfc17696
export SUBNET_PRIVATE2=subnet-10e07d77
export BASTION_SECURITY_GROUP=sg-00525a78
export BASTION_AMI=ami-a0a729c0
export CASSANDRA_AMI=ami-a0a729c0
export CASSANDRA_SECURITY_GROUP=sg-01525a79
```
Step 3: Launch the servers

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

# Create Bastion
bin/create-ec2-instance-bastion.sh

# Create Cassandra seed nodes in AWS AZ a and AWS AZ b.
bin/create-ec2-instance-cassandra.sh 10.0.1.10 a
bin/create-ec2-instance-cassandra.sh 10.0.1.10 b

# Create two more Cassandra database servers in AZ a and b.
bin/create-ec2-instance-cassandra.sh 10.0.1.11 a
bin/create-ec2-instance-cassandra.sh 10.0.1.11 b
```
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

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


/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
```
EC2 Instance UserData and MetaData

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

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

This example gets the list of available public keys.

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

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

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

**STEP 4 MAKE SURE IT WORKS!**

<table>
<thead>
<tr>
<th>Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns (effective)</th>
<th>Host ID</th>
<th>Rack</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.2.10</td>
<td>163.15 KiB</td>
<td>32</td>
<td>50.7%</td>
<td>e5993967-26e8-485b</td>
<td>2b</td>
</tr>
<tr>
<td>10.0.1.10</td>
<td>112.04 KiB</td>
<td>32</td>
<td>52.2%</td>
<td>47fbd9e8-0bcf-468f</td>
<td>2a</td>
</tr>
<tr>
<td>10.0.1.11</td>
<td>107.31 KiB</td>
<td>32</td>
<td>47.1%</td>
<td>ac744cc8-4d91-47fe</td>
<td>2a</td>
</tr>
<tr>
<td>10.0.2.11</td>
<td>221.87 KiB</td>
<td>32</td>
<td>50.1%</td>
<td>5dff0b8e-7cae-4e50</td>
<td>2b</td>
</tr>
</tbody>
</table>
SSH config

```
Host *,us-west-2.compute.amazonaws.com
  ForwardAgent yes
  IdentityFile ~/.ssh/test_rsa
  User ansible

Host bastion
  Hostname bastion.dev.us-west-2.cloudurable.com
  ForwardAgent yes
  IdentityFile ~/.ssh/test_rsa
  User ansible

Host cassandra.node0
  Hostname 10.0.1.10
  ForwardAgent yes
  IdentityFile ~/.ssh/test_rsa
  ProxyCommand ssh bastion -W %h:%p
  User ansible

Host 10.0.2.*
  ForwardAgent yes
  IdentityFile ~/.ssh/test_rsa
  ProxyCommand ssh bastion -W %h:%p
  User ansible
  ControlMaster auto
  ControlPath ~/.ssh/ansible-%r@%h:%p
  ControlPersist 5m

Host 10.0.1.*
  ForwardAgent yes
  IdentityFile ~/.ssh/test_rsa
  ProxyCommand ssh bastion -W %h:%p
  User ansible
  ControlMaster auto
  ControlPath ~/.ssh/ansible-%r@%h:%p
  ControlPersist 5m
```
Cassandra YAML (generated)

```yaml
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"
```

```yaml
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"
```
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
Ec2MultiRegionSnitch 1 region

VPC

AWS Region: us-west-2

Public Subnet 1
for Cassandra Cluster
Seed CN1 CN2 CN3 CN4

Public Subnet 2
for Cassandra Cluster
Seed CN10 CN12 CN13 CN14

Public Subnet 3
for Cassandra Cluster
Seed CN20 CN22 CN23 CN24

AWS Internet Gateway

AWS Region: us-west-2
AZ: us-west-2a
AZ: us-west-2b
AZ: us-west-2c
Ec2MultiRegionSnitch 2 regions

USA Virginia

VPC  AWS Region: us-east-1

- Public Subnet 1 for Cassandra Cluster
  - AZ: us-east-1a
  - Seed CN1
  - CN2 CN3 CN4

- Public Subnet 2 for Cassandra Cluster
  - AZ: us-east-1b
  - Seed CN10
  - CN12 CN13 CN14

- Public Subnet 3 for Cassandra Cluster
  - AZ: us-east-1c
  - Seed CN20
  - CN22 CN23 CN24

AWS Internet Gateway

USA Oregon

VPC  AWS Region: us-west-2

- Public Subnet 1 for Cassandra Cluster
  - AZ: us-west-2a
  - Seed CN1
  - CN2 CN3 CN4

- Public Subnet 2 for Cassandra Cluster
  - AZ: us-west-2b
  - Seed CN10
  - CN12 CN13 CN14

- Public Subnet 3 for Cassandra Cluster
  - AZ: us-west-2c
  - Seed CN20
  - CN22 CN23 CN24

AWS Internet Gateway
Cassandra / Kafka Support in EC2/AWS

**Ec2MultiRegionSnitch 4 regions**

**USA West Coast Oregon**

- **VPC**
- **AWS Region: us-west-2**
  - Public Subnet 1 for Cassandra Cluster
    - AZ: us-west-2a
    - Seed: CN1
  - Public Subnet 2 for Cassandra Cluster
    - AZ: us-west-2b
    - Seed: CN12, CN13, CN14
  - Public Subnet 3 for Cassandra Cluster
    - AZ: us-west-2c
    - Seed: CN20, CN22, CN23, CN24

**AWS Internet Gateway**

**USA East Coast Ohio**

- **VPC**
- **AWS Region: us-east-2**
  - Public Subnet 1 for Cassandra Cluster
    - AZ: us-east-2a
    - Seed: CN1
  - Public Subnet 2 for Cassandra Cluster
    - AZ: us-east-2b
    - Seed: CN12, CN13, CN14
  - Public Subnet 3 for Cassandra Cluster
    - AZ: us-east-2c
    - Seed: CN20, CN22, CN23, CN24

**AWS Internet Gateway**

**Brazil – South America**

- **VPC**
- **AWS Region: sa-east-1**
  - Public Subnet 1 for Cassandra Cluster
    - AZ: sa-east-1a
    - Seed: CN1
  - Public Subnet 2 for Cassandra Cluster
    - AZ: sa-east-1b
    - Seed: CN12, CN13, CN14
  - Public Subnet 3 for Cassandra Cluster
    - AZ: sa-east-1c
    - Seed: CN20, CN22, CN23, CN24

**AWS Internet Gateway**

**Ireland - Europe**

- **VPC**
- **AWS Region: eu-west-1**
  - Public Subnet 1 for Cassandra Cluster
    - AZ: eu-west-1a
    - Seed: CN1
  - Public Subnet 2 for Cassandra Cluster
    - AZ: eu-west-1b
    - Seed: CN12, CN13, CN14
  - Public Subnet 3 for Cassandra Cluster
    - AZ: eu-west-1c
    - Seed: CN20, CN22, CN23, CN24

**AWS Internet Gateway**
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!

- 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.
Why SSL? Just in case

- 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).
Cassandra / Kafka Support in EC2/AWS

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"
Create the SSL Cassandra Key

DBA/DevOps Cassandra Task: Create the Cassandra cluster key

```bash
keytool -genkey -keyalg RSA -alias "${CLUSTER_NAME}_CLUSTER" -keystore "$KEY_STORE" -storepass "$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
```
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
```
Import public key for the Cassandra cluster key into the trust store so nodes can identify each other

```bash
# Import the identity of the cluster public cluster key into the trust store so that nodes can identify each other.
keytool -import -v -trustcacerts -alias "${CLUSTER_NAME}_CLUSTER" -file "${CLUSTER_PUBLIC_CERT}" -keystore "${TRUST_STORE}"
-storepass "$PASSWORD" -keypass "$PASSWORD" -noprompt
```
Creating client pem files

```
keytool -importkeystore -srccalias "${CLUSTER_NAME}_CLIENT" -srckeystore "$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
```
SSL Files Generated

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

```bash
$ 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 csqlsh
- **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 csqlsh
Provision Cassandra to Use Keys

```bash
#!/bin/bash

DESTINATION_DIRECTORY=/opt/cassandra/conf/certs
cd ~
SOURCE_DIRECTORY="$PWD/resources$DESTINATION_DIRECTORY"

echo $SOURCE_DIRECTORY

if [ -d "$SOURCE_DIRECTORY" ]; then
    echo "$SOURCE_DIRECTORY was found making directory $DESTINATION_DIRECTORY"
    mkdir -p "$DESTINATION_DIRECTORY"
    echo "$SOURCE_DIRECTORY was found"
    cp "${SOURCE_DIRECTORY}/* "$DESTINATION_DIRECTORY"
fi

if [ ! -d "$SOURCE_DIRECTORY" ]; then
    echo "UNABLE TO INSTALL CERTS AS THEY WERE NOT FOUND"
fi
```
Configure Cassandra to use the keys

```
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

```bash
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

```yaml
classic_regions:
  - hosts: all-nodes
gather_facts: no
become: true
remote_user: ansible
vars:
  cluster_name: test
  bind_ip: "{{aws_metal}}/local-ipv4"
  broadcast_ip: "{{aws_metal}}/public-ipv4"

tasks:
  - name: "Copy template"
copy:
    src: ../resources/opt/cassandra/conf/cassandra-yaml.template
dest: /opt/cassandra/conf/cassandra-yaml.template
owner: cassandra
group: cassandra
mode: "u=rw,g=r,o=r"
  - 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"
  - name: Restart Cassandra
    command: "/bin/systemctl restart cassandra"
```
Using cassandra-cloud from Ansible

vars:
cluster_name: test
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"
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**

```bash
$ ansible seed1 -a "setopt/cassandra/bin/nodetool status"
```

#### 10.1.1.10 | SUCCESS | rc=0 >>

<table>
<thead>
<tr>
<th>Datacenter: us-east-2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Status=Up/Down</th>
<th>State=Normal/Leaving/Joining/Moving</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Address</strong></td>
<td><strong>Load</strong></td>
</tr>
<tr>
<td>UN 52.14.159.203</td>
<td>198.57 KiB</td>
</tr>
<tr>
<td>UN 52.14.156.148</td>
<td>241.97 KiB</td>
</tr>
<tr>
<td>UN 52.14.108.53</td>
<td>239.15 KiB</td>
</tr>
<tr>
<td>UN 52.14.139.245</td>
<td>195.85 KiB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Datacenter: us-west-2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Status=Up/Down</th>
<th>State=Normal/Leaving/Joining/Moving</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Address</strong></td>
<td><strong>Load</strong></td>
</tr>
<tr>
<td>UN 54.202.137.143</td>
<td>194.06 KiB</td>
</tr>
<tr>
<td>UN 54.202.57.109</td>
<td>234.12 KiB</td>
</tr>
<tr>
<td>UN 54.187.138.240</td>
<td>220.24 KiB</td>
</tr>
<tr>
<td>UN 54.149.126.26</td>
<td>194.18 KiB</td>
</tr>
</tbody>
</table>
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 GossippingPropertyFileSnitch
- The CIDR addresses for the VPNs cannot overlap.
  - Care must be taken in setting up multiple private VPCs that can communicate cross region
GossippingPropertyFileSnitch via
**VPN via VPC Peer**

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

**AWS Region: us-west-2**
- VPN Subnet 1
  - AZ: us-west-2a
  - NAT Gateway
- VPN Subnet 2
  - AZ: us-west-2b
  - NAT Gateway
  - Bastion Host
- VPN Subnet 3
  - AZ: us-west-2a
  - NAT Gateway

**Company VPC For VPN**
**VPC Peering**
**Virtual Private Gateway**
Cassandra / Kafka Support in EC2/AWS

GossippingPropertyFileSnitch

4 regions / DCs
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)
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

- Training
- Consulting
- Setting up Cassandra in AWS/EC2
- AWS CloudFormations
- Subscription Support around Cassandra running in AWS/EC2
  - AWS CloudWatch monitoring
  - AWS CloudWatch logging
SIMPLIFY CASSANDRA/KAKFA DEPLOYMENTS

SUBSCRIPTION SUPPORT FOR CASSANDRA DATABASE/AWS

SUBSCRIPTION SUPPORT FOR KAKFA/AWS

STREAMLINE DEVOPS AND PRODUCTION SUPPORT

READ MORE...

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

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.

March 11, 2017
Setting up client and cluster SSL transport for a Cassandra cluster. This article is a Cassandra tutorial on Cassandra setup for SSL and CQL clients, as well as installing Cassandra with SSL configured on a series of Linux servers. Cassandra allows you to secure the client transport (CQL) as well as the cluster transport (storage transport). SSL/TLS have some overhead. This is especially true in the JVM world, which is not as performant for handling SSL/TLS unless you are using Netty/OpenSSL integration.

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.