文書番号: LB20190717-001-002

AWS 上で DRBD9 を用いた災害対策(DR)

初版

サイオステクノロジー株式会社

目次

1. 概要

LINBITクラスタスタック・サポートが提供するPacemakerバイナリー(以下Pacemaker)では、Amazon Web Service（以降 AWS）上でのクラスターの作成をサポートしています。 AWS上のPacemakerはAWS用に開発されたリソースエージェントを利用することで、 保護対象のアプリケーションに対して、あらゆるVirtual Private Cloudの内部および 外部からサービスノードにアクセスできる環境を構築することが可能です。

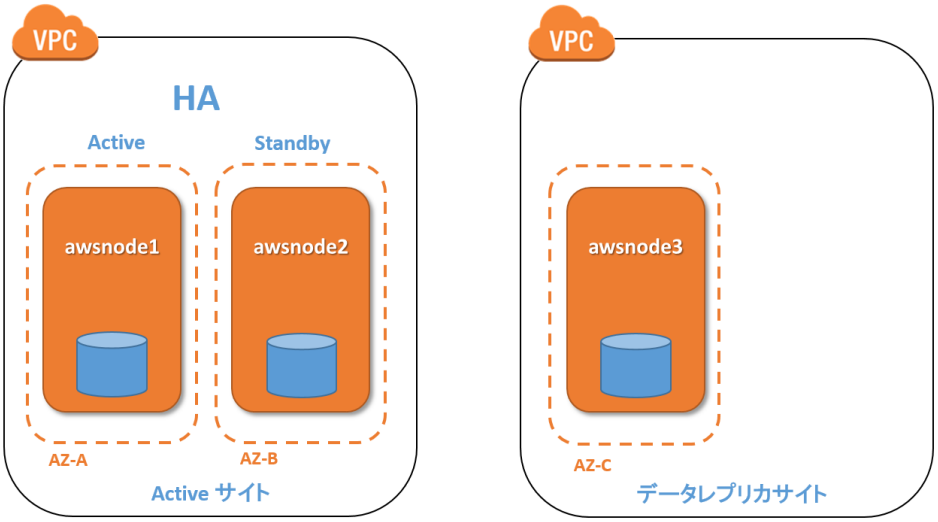
本ガイドでは、PacemaerのAWS専用リソースエージェントを利用した構築の例として、 以下の構成を作成する手順をご案内します。

1. 3ノード構成データレプリケーション
    a. リージョンが異なる東京とシンガポールの2拠点
    b. 東京にアベイラビリティーゾーン（AZ）が異なる2ノード
    c. シンガポールに1ノード
2. 4ノード構成スタッキングクラスタ
    a. リージョンが異なる東京とシンガポールの2拠点
    b. リージョン内の2拠点はアベイラビリティーゾーン（AZ）は同じ
    c. 拠点内はアクティブ・スタンバイのHA構成
    d. 拠点間の切り替えはPacemaker Ticketによる手動切り替え
3. 4ノード構成フラット接続クラスタ
    a. リージョンが異なる東京とシンガポールの2拠点
    b. リージョン内の2拠点はアベイラビリティーゾーン（AZ）が異なる
    c. 拠点内はアクティブ・スタンバイのHA構成
    d. 拠点間の切り替えはPacemaker Ticketによる手動切り替え

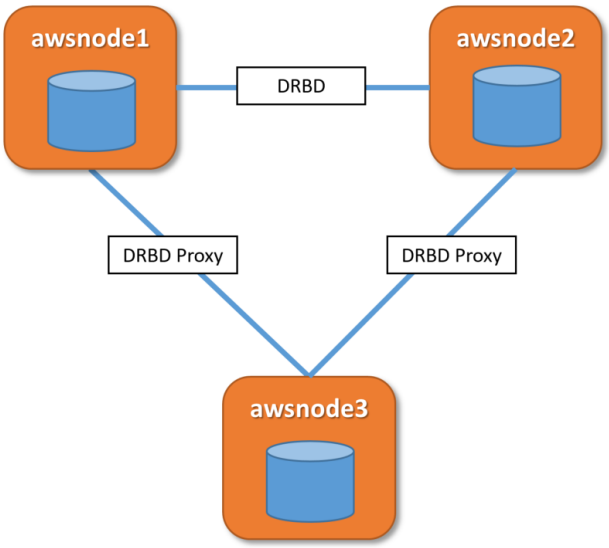
1.1. 各構成の説明

1.1.1. 3ノード構成データレプリケーション

リージョンが異なる東京とシンガポールの2拠点間で東京をメインサイト、シンガポールをDRサイトとし、拠点間でデータの李プリケーションを行います。東京にアベイラビリティーゾーン（AZ）が異なる2ノードを配置し、合計3ノード でシステムを構成します。
東京のアベイラビリティーゾーン（AZ）間ではサービスを HA 化します。

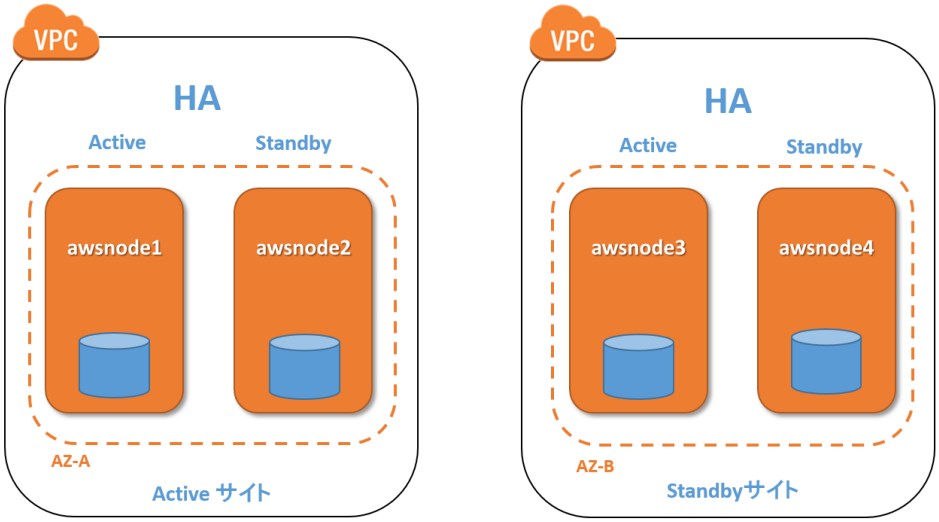


データのレプリケーションにはDRBDを使用して、リージョン間の通信は２つのWAN回線でDRBD Proxyを使って行います。

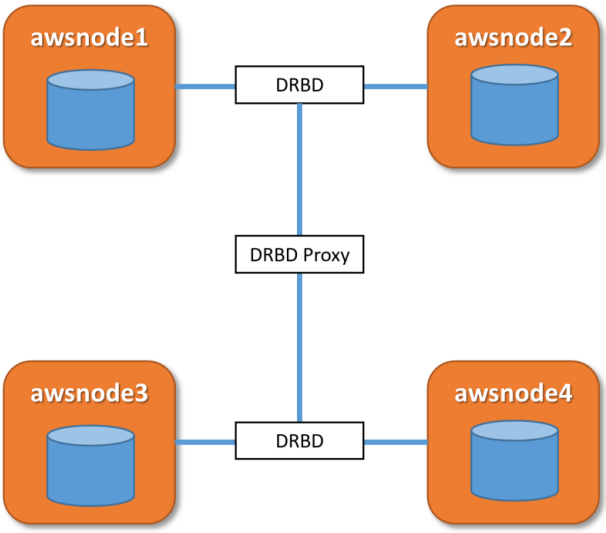


1.1.2. 4ノード構成スタッキングクラスタ

リージョンが異なる東京とシンガポールの2拠点間で東京をメインサイト、シンガポールをDRサイトとし、拠点間でバックアップを行います。東京にアベイラビリティーゾーン（AZ）が同一の2ノードを配置し、加えてシンガポールにもアベイラビリティーゾーン（AZ）が同一2ノードを配置し、合計４ノードでシステムを構成します。

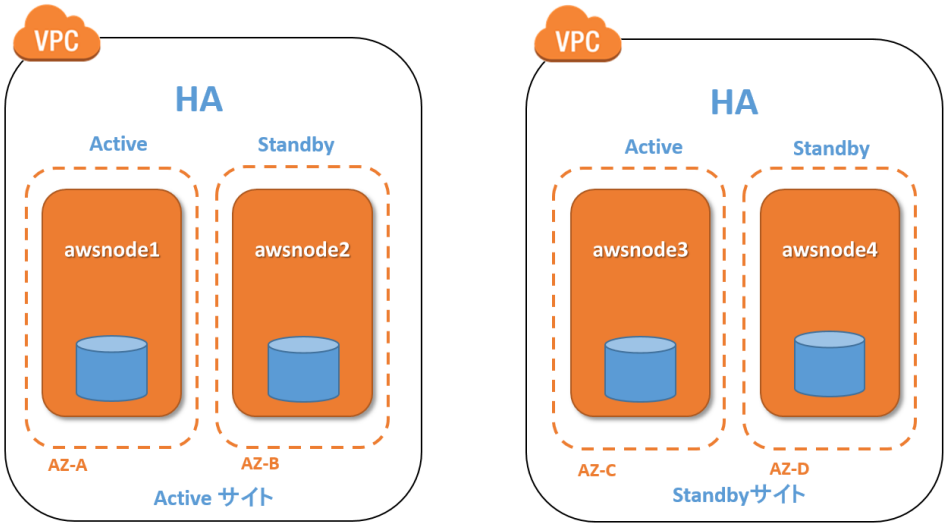


リージョンが異なる東京とシンガポールの２拠点間でサービスを HA 化します。拠点間のHAの制御は Pacemaker のチケット機能を利用します。
東京のアベイラビリティーゾーン（AZ）間ではサービスを HA 化します。
シンガポールのアベイラビリティーゾーン（AZ）間ではサービスを HA 化します。
データのレプリケーションにはDRBDを使用して、リージョン間の通信は1つのWAN回線でDRBD Proxyを使って行います。

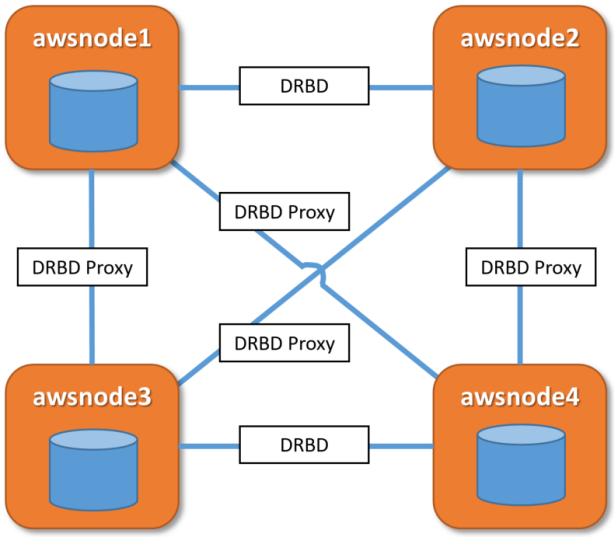


1.1.3. 4ノード構成フラット接続クラスタ

リージョンが異なる東京とシンガポールの2拠点間で東京をメインサイト、シンガポールをDRサイトとし、拠点間でバックアップを行います。東京にアベイラビリティーゾーン（AZ）が異なる2ノードを配置し、加えてシンガポールにもアベイラビリティーゾーン（AZ）が異なる2ノードを配置し、合計４ノードでシステムを構成します。



リージョンが異なる東京とシンガポールの２拠点間でサービスを HA 化します。拠点間のHAの制御は Pacemaker のチケット機能を利用します。
東京のアベイラビリティーゾーン（AZ）間ではサービスを HA 化します。
シンガポールのアベイラビリティーゾーン（AZ）間ではサービスを HA 化します。
データのレプリケーションにはDRBDを使用して、リージョン間の通信は４つのWAN回線でDRBD Proxyを使って行います。



1.2. 各構成の次のステップ

「AWS の設定」、「OS の設定」はどの構成でも必要な章になります。その後は各構成の章にお進みください。

1.2.1. 3ノード構成データレプリケーション

- 2. AWS の設定 (共通設定)
- 3. OS の設定 (共通設定)
- 4. 3ノード構成データレプリケーション

1.2.2. 4ノード構成スタッキングクラスタ

- 2. AWS の設定 (共通設定)
- 3. OS の設定 (共通設定)
- 5. 4ノード構成スタッキングクラスタ

1.2.3. 4ノード構成フラット接続クラスタ

- 2. AWS の設定 (共通設定)
- 3. OS の設定 (共通設定)
- 6. 4ノード構成フラット接続クラスタ

2. AWS の設定

2.1. VPC の作成

東京、シンガポールでそれぞれ VPC を作成します。Route53 を使用するため DNSホスト名、DNS 解決も有効にしておきます。

VPC 名	リージョン	IPv4 CIDR
vpc-tokyo	ap-northeast-1	10.0.0.0/16
vpc-singapore	ap-southeast-1	10.1.0.0/16

2.2. インターネットゲートウェイの作成

東京、シンガポールでそれぞれインターネットゲートウェイを作成し、VPC にアタッチします。

IGW 名	リージョン	アッタチ先 VPC
igw-tokyo	ap-northeast-1	vpc-tokyo
igw-singapore	ap-southeast-1	vpc-singapore

2.3. サブネットの作成

東京、シンガポールでそれぞれサブネットを作成します。サブネットはアベイラビリティゾーンごとに必要になりますので、東京で２つ、シンガポールで２つ作成します。なお、「4ノード構成スタッキングクラスタ」で使用する場合、現時点ではそれぞれのサイトで同じアベイラビリティゾーンで使用しますので、東京、シンガポールでそれぞれ１つになります。

ネームタグ	サブネット	AZ	VPC
tokyo-subnet1	10.0.1.0/24	ap-northeast-1a	vpc-tokyo
tokyo-subnet2	10.0.2.0/24	ap-northeast-1c	vpc-tokyo
singapore-subnet1	10.1.1.0/24	ap-southeast-1a	vpc-singapore
singapore-subnet2	10.1.2.0/24	ap-southeast-1c	vpc-singapore

2.4. ピアリング接続の作成

東京とシンガポールの VPC 間でプライベートアドレスを用いて通信するため、ピアリング接続を作成します。東京で ピアリング接続の作成したあとにシンガポールでこの接続を承諾します。

Peering 接続名	Tokyo VPC 名	Singapore VPC名
pcx-tokyo	vpc-tokyo	vpc-singapore

2.5. ルートテーブルの設定

東京とシンガポールそれぞれのサイトでルートテーブルの設定を行います。

東京 - 関連付けられるサブネット: 10.0.1.0/24, 10.0.2.0/24

接続先	ターゲット
0.0.0.0/0	インターネットゲートウェイ
10.1.0.0/16	ピアリング接続

シンガポール - 関連付けられるサブネット: 10.1.1.0/24, 10.1.2.0/24

接続先	ターゲット
0.0.0.0/0	インターネットゲートウェイ
10.0.0.0/16	ピアリング接続

2.6. セキュリティーグループの設定

東京とシンガポールそれぞれのサイトでセキュリティーグループの設定を行います。以下にインバウンドのルールを示します。LINSTOR のポートは LINSTOR を使う場合のみ必要です。ソースの範囲は必要に応じて変更してください。

タイプ	プロトコル	ポート範囲	ソース	目的
カスタムUDPルール	UDP	5405	10.0.0.0/16, 10.1.0.0/16	corosync
カスタムTCPルール	TCP	7788-7790	10.0.0.0/16, 10.1.0.0/16	DRBD
カスタムTCPルール	TCP	3366	10.0.0.0/16, 10.1.0.0/16	linstor-satellite
カスタムTCPルール	TCP	3370	10.0.0.0/16, 10.1.0.0/16	linstor-rest-api
カスタムTCPルール	TCP	3376	10.0.0.0/16, 10.1.0.0/16	linstor-controller
カスタムTCPルール	TCP	7000-7010	10.0.0.0/16, 10.1.0.0/16	LINSTOR resource
カスタムTCPルール	TCP	3306	0.0.0.0/0	MySQL
すべてのICMP-IPv4	すべて	該当なし	0.0.0.0/0	ping 等のICMPサービス

2.7. Route53 の設定

Route53 レコードを書き換えることにより、ホスト名が常にアクティブなサーバのIPアドレスになるように変更することができます。また、別のVPCからも利用可能になります。以下の設定と aws-vpc-route53 resource agent 使用することで、mysql.test.local で常にアクティブなノードにアクセスできます。

Domain Name	Type	関連付けられる VPC ID
test.local	Private Hosted Zone	vpc-tokyo, vpc-singapore

Name	Type	Value
mysql	A-IPv4 address	10.0.0.1

VPC の DNSホスト名、DNS 解決が有効になっているのも確認します。

2.8. インスタンスの設定

東京とシンガポールそれぞれのサイトでインスタンスの作成を行います。

項目	設定値
マシンイメージ-AMI	Hat Enterprise Linux (RHEL) 7.2 (HVM) または CentOS Linux 7 x86_64 HVM EBS ENA 1901_01
タイプ	t2.micro 以上
ネットワーク	サイトの VPC
ストレージの追加	2G 以上
セキュリティグループ	作成したグループ

2.8.1. 東京 - ap-northeast-1

3ノード構成データレプリケーション

ホスト名	AZ	サブネット
awsnode1	ap-northeast-1a	10.0.1.0/24
awsnode2	ap-northeast-1c	10.0.2.0/24

4ノード構成スタッキングクラスタ

ホスト名	AZ	サブネット
awsnode1	ap-northeast-1a	10.0.1.0/24

4ノード構成フラット接続クラスタ

ホスト名	AZ	サブネット
awsnode1	ap-northeast-1a	10.0.1.0/24
awsnode2	ap-northeast-1c	10.0.2.0/24

2.8.2. シンガポール - ap-southeast-1

3ノード構成データレプリケーション

ホスト名	AZ	サブネット
awsnode3	ap-southeast-1a	10.1.1.0/24

4ノード構成スタッキングクラスタ

ホスト名	AZ	サブネット
awsnode3	ap-southeast-1a	10.1.1.0/24

4ノード構成フラット接続クラスタ

ホスト名	AZ	サブネット
awsnode3	ap-southeast-1a	10.1.1.0/24
awsnode4	ap-southeast-1c	10.1.2.0/24

2.9. 制限事項

2.9.1. Secondady NIC

DRBD と corosync の設定では NIC を複数持つことを推奨していますが、ec2 インスタンスに Secondady NIC をもたせると AWS 用の一部の resource agent が動作しないことが確認されています。現時点では Secondady NIC を持たない構成で使用してください。

2.9.2. 4ノード構成スタッキングクラスタ

4ノード構成スタッキングクラスタで、リージョン内の2拠点のアベイラビリティーゾーン（AZ）が異なる構成は subnet が変わってしまい、現時点では正しく動作しません。4ノード構成スタッキングクラスタを使用する場合は、リージョン内の2拠点は同じアベイラビリティーゾーン（AZ）で構成してください。

3. OS の設定

3.1. OS の更新、追加パッケージ

インスタンス作成後、ログインし yum update で最新にしておきます。

```
# yum update
```

また、DRBD は LINBIT のレポジトリから使用するとし、「サポートガイド」に従ってインストールします。なお、DRBD は 9.0, pacemaker は 1.1.20 を使用してください。

さらに、構成により追加パッケージのインストールが必要になります。以下に従ってすべてのノードにインストールしてください。

構成	LINSTOR	MySQL community	mariadb
3ノード構成データレプリケーション	必要	不要	不要
4ノード構成スタッキングクラスタ	不要	必要	不要
4ノード構成フラット接続クラスタ	不要	不要	必要

LINSTOR をインストールする場合

```
# yum install linstor-controller linstor-satellite linstor-client
```

mysql-community-server をインストールする場合

```
# yum localinstall http://dev.mysql.com/get/mysql57-community-release-el7-8.noarch.rpm
# yum install mysql-community-server
# systemctl disable mysqld
```

mariadb をインストールする場合

```
# yum install mariadb-server
# systemctl disable mysqld
```

3.2. ホスト名の設定

リブート時にホスト名が変わらないように固定にしておきます。以下に awsnode1 の設定例を示します。

```
# hostnamectl set-hostname --static awsnode1
# echo "preserve_hostname: true" >> /etc/cloud/cloud.cfg
```

3.3. aws コマンド

pacemaker を AWS で使用する場合、AWS 用の resource agent が aws コマンドを使用します。よって、これが正しく動作するようにあらかじめ設定しておきます。また、aws コマンドで設定するユーザは EC2, VPC, Route53 にフルアクセスできる権限をもつようにしておきます。さらにアクセスキーを作成しておきます。

```
# curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"
# python get-pip.py
# pip install awscli
# aws configure
# aws configure --profile cluster       # text
# aws ec2 describe-instances
```

profile は default と cluster の 2 つ作成します。output format は default では json , cluster profile では text にします。また、東京サイトの awsnode1, awsnode2 では region ap-northeast-1 , シンガポールサイトの awsnode3, awsnode4 では region ap-southeast-1 になります。

	東京 - default	**東京 - cluster**	**シンガポール - default**	**シンガポール - cluster**
Access Key ID	XXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXX
Secret Access Key	XXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXX
region name	ap-northeast-1	ap-northeast-1	ap-southeast-1	ap-southeast-1
output format	json	text	json	text

3.4. AWS 用の resource agents

Resource Agent	**用途**
aws-vpc-move-ip	ルートテーブルのターゲットを書き換えることによりVPC内の特定IPアドレスが常にアクティブなサーバに届くよう変更する
awseip	Elastic IP アドレスを登録
awsvip	セカンダリプライベートアドレスを登録、同じサブネット内で使用
aws-vpc-route53	Route53レコードを書き換えることにより、ホスト名が常にアクティブなサーバのIPアドレスになるように変更する。別のVPCからも利用可能。

現時点で aws-vpc-route53 は src.rpm (resource-agents-4.2.0.linbit-1+20190116+6a6f48180a59.el7.7.src.rpm) には入っているのですが、デフォルトでは exclude になっています。/usr/lib/ocf/resource.d/heartbeat/aws-vpc-route53 がインストールされていない場合は linbit のレポジトリの src.rpm から resource-agents.spec を書き換えてリビルドする方法でもよいですが、github のソースから直接取得することもできます。

```
# diff -u resource-agents.spec.org resource-agents.spec
--- resource-agents.spec.org    2019-01-26 00:11:11.000000000 +0900
+++ resource-agents.spec    2019-06-22 05:10:31.143657882 +0900
@@ -768,7 +768,6 @@
 %exclude /usr/lib/ocf/resource.d/heartbeat/Xen
 %exclude /usr/lib/ocf/resource.d/heartbeat/anything
 %exclude /usr/lib/ocf/resource.d/heartbeat/asterisk
-%exclude /usr/lib/ocf/resource.d/heartbeat/aws-vpc-route53
 %exclude /usr/lib/ocf/resource.d/heartbeat/dnsupdate
 %exclude /usr/lib/ocf/resource.d/heartbeat/eDir88
 %exclude /usr/lib/ocf/resource.d/heartbeat/fio
```

aws-vpc-route53 は bash のスクリプトですので、ここではビルドせずに該当部分を書き換えてそのまま使用します（リビルドする場合は ./autogen.sh && ./configure && make）。

```
# wget https://github.com/ClusterLabs/resource-agents/archive/v4.2.0.tar.gz
# tar xzf v4.2.0.tar.gz
# sed -e 's;@BASH_SHELL@;/bin/bash;' resource-agents-4.2.0/heartbeat/aws-vpc-route53.in >
/usr/lib/ocf/resource.d/heartbeat/aws-vpc-route53
# chmod 755 /usr/lib/ocf/resource.d/heartbeat/aws-vpc-route53
```

また、このバージョンの aws-vpc-route53 は ec2metadata に依存していますので、これも取得します。CentOS の場合は cloud-utils に入っているのでこれをインストールします。

```
# yum install -y cloud-utils
```

RHEL の場合は github のソースから直接取得します。

```
# wget https://launchpad.net/cloud-utils/trunk/0.27/+download/cloud-utils-0.27.tar.gz
# tar xzf cloud-utils-0.27.tar.gz
# cp cloud-utils-0.27/bin/ec2metadata /usr/bin/ec2metadata
# chmod 755 /usr/bin/ec2metadata
```

4. 3ノード構成データレプリケーション

ここではリージョンが異なる東京とシンガポールの2拠点間で東京をメインサイト、シンガポールをDRサイトとし、拠点間でデータのレプリケーションを行う。さらに東京にアベイラビリティーゾーン（AZ）が異なる2ノードを配置し、合計３ノードでシステムを構成し、３重に冗長化されたデータ領域を使用します。

4.1. /etc/hosts

東京の２ノードを awsnode1, awsnode2, シンガポールの１ノードを awsnode3 とします。以下のIPアドレス、ホスト名を仮定してこれ以降説明します。

```
# cat /etc/hosts
:
10.0.1.248  awsnode1
10.0.2.85   awsnode2
10.1.1.39   awsnode3
10.2.0.10   linstor
```

4.2. DRBDディスク領域の設定

DRBD 用に確保した領域を初期化します。ここでは /dev/xvdb を使用し、しそのうち 1G バイトを linstor database 用、残りを作業領域とします。すべてのノードで以下を実行します。

物理デバイス	**/dev/xvdb**
linstor database 下位デバイス	/dev/drbdpool/linstor_db
linstor drbdpool 下位デバイス	/dev/drbdpool/disk1

```
# pvcreate /dev/xvdb
# vgcreate drbdpool /dev/xvdb
# lvcreate --thin -L 1G drbdpool/linstor_db
# lvcreate --thin -l 100%FREE drbdpool/disk1
```

4.3. linstor database 領域の設定

/var/lib/linstor の linstor database 領域をDRBDを使用して冗長化しておきます。linstor-controller が起動される前に必要なので、ここでは res ファイルは手動で作成します。また、３ノードの特性を活かし、DRBD のクォーラムを使用します。以下の res ファイルを各ノードで設定します。

```
# cat /etc/drbd.d/linstor_db.res
resource linstor_db {
  options
  {
    quorum majority;
    on-no-quorum io-error;
  }

  volume 0
  {
    disk        /dev/drbdpool/linstor_db;
    meta-disk   internal;
    device      minor 0;
  }

  on awsnode1 {
    node-id 0;
  }

  on awsnode2 {
    node-id 1;
  }

  on awsnode3 {
    node-id 2;
  }

  connection
  {
    net
    {
        protocol C;
    }
    host awsnode1 address ipv4 10.0.1.248:7788;
    host awsnode2 address ipv4 10.0.2.85:7788;
  }

  connection
  {
    net
    {
        protocol A;
    }
    host awsnode1 address ipv4 10.0.1.248:7789;
    host awsnode3 address ipv4 10.1.1.39:7789;
  }
```

```
  connection
  {
    net
    {
        protocol A;
    }
    host awsnode2 address ipv4 10.0.2.85:7790;
    host awsnode3 address ipv4 10.1.1.39:7790;
  }
}
```

それぞれのノードで初期化します。

```
# drbdadm create-md linstor_db
# drbdadm up linstor_db
# drbdadm status
```

awsnode1 でファイルシステムを作成します。

```
# drbdadm --force primary linstor_db
# drbdadm secondary linstor_db
# mkfs.xfs /dev/drbd0
```

すべてのノードで linstor-satellite, drbd を有効にし linstor-satellite はスタートします。

```
# systemctl enable --now linstor-satellite
# systemctl enable drbd
```

4.4. corosync の設定

linstor-controller サービスは東京の AZ 間のノードで pacemaker を用いて HA 化しますので、まず cororosync を設定します。

awsnode1, awsnode2 で以下を設定します。

```
# cat /etc/corosync/corosync.conf
totem {
        version: 2

        crypto_cipher: none
        crypto_hash: none
        cluster_name: cluster_test

        transport: udpu
        rrp_mode: passive
}

logging {
        fileline: off
        to_stderr: no
        to_logfile: yes
        logfile: /var/log/corosync.log
        to_syslog: yes
        debug: off
        timestamp: on
        logger_subsys {
                subsys: QUORUM
                debug: off
        }
}

quorum {
        provider: corosync_votequorum
        expected_votes: 2
        two_node: 1
}

nodelist {
        node {
                ring0_addr: 10.0.1.248
                nodeid: 1
        }
        node {
                ring0_addr: 10.0.2.85
                nodeid: 2
        }
}
```

awsnode1, awsnode2 で pacemaker を起動します。

```
# systemctl start pacemaker
```

awsnode2 で pacemaker を監視します。

```
# crm_mon -rf
```

4.5. linstor controller vip の登録

aws-vpc-move-ip で linstor vip の設定値を変更するには、あらかじめ VPC のルートテーブルに 10.2.0.10/32 のルートをどちらかのネットワークインターフェースに登録しておく必要があります。以下を awsnode1 で実行します。

```
# echo $(curl -s http://169.254.169.254/latest/meta-data/instance-id)
i-XXXXXXXXXXXXXXXXX
# aws ec2 create-route --route-table-id rtb-XXXXXXXXXXXXXXXXX --destination-cidr-block
10.2.0.10/32 --instance-id i-XXXXXXXXXXXXXXXXX
```

i-XXXXXXXXXXXXXXXXX は echo で取得したインスタンスID, rtb-XXXXXXXXXXXXXXXXX はルートテーブルIDを指定します。

ここで awsnode1, awsnode2 のインスタンスは送信元、送信先チェックが外れている必要があります。

4.6. linstor controller の pacemaker 設定

linstor controller を pacemaker に設定します。crm configure edit で以下のように設定します。

```
# crm configure edit
:
primitive p_linstor_cntl systemd:linstor-controller \
    op start interval=0 timeout=100s \
    op stop interval=0 timeout=100s \
    op monitor interval=100s timeout=100s

primitive p_linstor_fs Filesystem \
    params device="/dev/drbd0" directory="/var/lib/linstor" fstype=xfs options=noatime \
    op start interval=0 timeout=60 \
    op stop interval=0 timeout=60 \
    op monitor interval=20 timeout=40

primitive p_linstor_vip aws-vpc-move-ip \
    params ip=10.2.0.10 routing_table=rtb-XXXXXXXXXXXXXXXXX interface=eth0 monapi=true \
    op monitor timeout=30s interval=60s depth=0

group rg_linstor_cntl p_linstor_vip p_linstor_fs p_linstor_cntl

location l1 rg_linstor_cntl 100: awsnode1

property cib-bootstrap-options: \
    :
    stonith-enabled=false \
    no-quorum-policy=ignore \
    :
rsc_defaults rsc-options: \
    resource-stickiness=200
```

最後に awsnode1, awsnode2 で linstor-controller のアドレスを設定します。

```
# cat /etc/linstor/linstor-client.conf
[global]
controllers=10.2.0.10
```

以上で linstor-controller の設定完了です。drbdadm, linstor コマンドが正しく動作するか確認します。

```
# drbdadm status
# linstor node list
```

4.7. linstor の設定

awsnode1 上の linstor で各ノードを登録、storage-pool を作成します。

```
# linstor node create --node-type Combined awsnode1 10.0.1.248
# linstor node create --node-type Combined awsnode2 10.0.2.85
# linstor node create --node-type Satellite awsnode3 10.1.1.39
# linstor node list
+-------------------------------------------------------+
| Node      | NodeType  | Addresses               | State  |
|-------------------------------------------------------|
| awsnode1  | COMBINED  | 10.0.1.172:3366 (PLAIN) | Online |
| awsnode2  | COMBINED  | 10.0.2.70:3366 (PLAIN)  | Online |
| awsnode3  | SATELLITE | 10.1.1.249:3366 (PLAIN) | Online |
+-------------------------------------------------------+

# linstor storage-pool create lvmthin awsnode1 drbdpool drbdpool/disk1
# linstor storage-pool create lvmthin awsnode2 drbdpool drbdpool/disk1
# linstor storage-pool create lvmthin awsnode3 drbdpool drbdpool/disk1
# linstor storage-pool list
+-------------------------------------------------------------------------------------------------
----------------------+
| StoragePool | Node     | Driver   | PoolName       | FreeCapacity | TotalCapacity |
SupportsSnapshots | State |
|-------------------------------------------------------------------------------------------------
----------------------|
| drbdpool    | awsnode1 | LVM_THIN | drbdpool/disk1 |     3.88 GiB |      3.98 GiB |
true              | Ok    |
| drbdpool    | awsnode2 | LVM_THIN | drbdpool/disk1 |     3.87 GiB |      3.98 GiB |
true              | Ok    |
| drbdpool    | awsnode3 | LVM_THIN | drbdpool/disk1 |     3.87 GiB |      3.98 GiB |
true              | Ok    |
+-------------------------------------------------------------------------------------------------
----------------------+
```

4.8. data1 領域の設定

data1 で使用する領域 2G を 3 ノードに確保します。

```
# linstor resource-definition create r_data1
# linstor resource-definition set-property r_data1 FileSystem/Type xfs
# linstor volume-definition create r_data1 2G
# linstor resource-definition drbd-options --quorum majority r_data1
# linstor resource-definition drbd-options --on-no-quorum io-error r_data1
# linstor resource create r_data1 --storage-pool drbdpool --auto-place 3
# linstor resource-connection drbd-options awsnode1 awsnode3 r_data1 --protocol A
# linstor resource-connection drbd-options awsnode2 awsnode3 r_data1 --protocol A
# linstor resource list
+-----------------------------------------------------+
| ResourceName | Node      | Port | Usage  |    State |
|-----------------------------------------------------|
| r_data1      | awsnode1  | 7000 | Unused | UpToDate |
| r_data1      | awsnode2  | 7000 | Unused | UpToDate |
| r_data1      | awsnode3  | 7000 | Unused | UpToDate |
+-----------------------------------------------------+
```

data1 は頻繁に更新があるので、ここには drbdproxy を使用します。proxy のメモリは 100MB, 50MB で pull-ahead に移行します。ここで注意点として memlimit の単位は byte, congestion-fill の単位は sector(512byte) になっています。

```
# linstor drbd-proxy enable awsnode1 awsnode3 r_data1
# linstor drbd-proxy enable awsnode2 awsnode3 r_data1
# linstor drbd-proxy options r_data1 --memlimit 100000000
# linstor drbd-proxy compression zlib r_data1 --level 9
# linstor resource-connection drbd-options awsnode1 awsnode3 r_data1 --on-congestion
pull-ahead
# linstor resource-connection drbd-options awsnode2 awsnode3 r_data1 --on-congestion
pull-ahead
# linstor resource-connection drbd-options awsnode1 awsnode3 r_data1 --congestion-fill
102400
# linstor resource-connection drbd-options awsnode2 awsnode3 r_data1 --congestion-fill
102400
```

4.9. data1 への書き込み

東京サイトの各ノードで data1 領域をマウントして使用します。

```
# mkdir /data1
# mount /dev/drbd1000 /data1
```

使い終わったら umount します。

4.10. DRBD クライアント

新たに東京サイトにノードを追加することで、これらのノードからも DRBD 領域にアクセスすることができます。例えば、awsclient という 10.0.2.176 のアドレスを持つノードを追加する場合は以下のようになります。

```
# linstor node create --node-type Satellite awsclient 10.0.2.176
# linstor resource create awsclient r_data1 --diskless
```

awsclient からは他の awsnode と同様に

```
# mkdir /data1
# mount /dev/drbd1000 /data1
```

でアクセスできます。

4.11. pacemaker サービスのスタート、ストップ

4.11.1. スタート

awsnode1, awsnode2 で pacemaker サービスを起動します。

```
# systemctl start pacemaker
```

4.11.2. ストップ

awsnode1, awsnode2 のうち現在 slave のほうから pacemaker を止めます。

```
# systemctl stop pacemaker
```

4.12. DRBD proxy のパフォーマンス効果

DRBD proxy はデータをバッファリングし、圧縮して送るので DR 環境で特に威力を発揮します。さらに ahead モードというのがあり、バッファが一杯になる前に一時的に同期を止め、比較的空いた時間に同期を再開するという機能もあります。この機能の効果は絶大で特に、DRBD 領域に一時的に大きなデータがおかれたときなど、この機能がないとバッファが満たされアプリケーションが止まってしまいます。

以下の例は DRBD proxy ありと DRBD の protocol A のみのときの比較です。data1 が proxy あり、data2 が protocol A のみです。protocol A のみと比較するのこのケースでは 9 倍近く差がでています。

```
[root@awsnode1 ~]# time dd if=/dev/urandom of=/data1/test bs=1M count=512 oflag=direct
512+0 records in
512+0 records out
536870912 bytes (537 MB) copied, 9.86747 s, 54.4 MB/s

real    0m9.869s
user    0m0.001s
sys     0m3.210s

[root@awsnode1 ~]# time dd if=/dev/urandom of=/data2/test bs=1M count=512 oflag=direct
512+0 records in
512+0 records out
536870912 bytes (537 MB) copied, 87.0439 s, 6.2 MB/s

real    1m27.048s
user    0m0.003s
sys     0m3.301s
```

DRBD proxy では dd で 512MByte のデータを流し込むと、proxy のメモリを使い切る前に ahead モードに移り、一時的にレプリケーションを止めます。これによりすばやく終了し、アプリケーションが止まってしまうケースを防ぐことができます。その後、比較的に空いた時間に同期が再開するという仕組みです。バッファリングと同期を止めることにより大きなデータもなるべく平準化して送ることが可能になります。

```
[root@awsnode1 ~]# drbd-proxy-ctl -c "show memusage"
r_data1-awsnode3-awsnode1 47 of 99 MiB used, 0 persistent
  normal   [######*************.....................] 49637888/99999232 bytes
  prio     [....................................] 1024/4193792 bytes


[root@awsnode1 ~]# cat
/sys/kernel/debug/drbd/resources/r_data1/connections/awsnode3/0/proc_drbd
1000: cs:Ahead ro:Primary/Secondary ds:UpToDate/Outdated A r-----
    ns:1629204 nr:0 dw:2100244 dr:1388457 al:70 bm:0 lo:0 pe:[0;0] ua:0 ap:[0;0] ep:1
wo:2 oos:471040
        resync: used:0/61 hits:17875 misses:10 starving:0 locked:0 changed:5
        act_log: used:0/1237 hits:32683 misses:15231 starving:0 locked:0 changed:171
        blocked on activity log: 0/0/0

[root@awsnode1 ~]# cat
/sys/kernel/debug/drbd/resources/r_data1/connections/awsnode3/0/proc_drbd
1000: cs:SyncSource ro:Primary/Secondary ds:UpToDate/Inconsistent A r-----
    ns:1668780 nr:0 dw:2100308 dr:1427969 al:70 bm:0 lo:0 pe:[0;7] ua:0 ap:[0;0] ep:1
wo:2 oos:431796
        [>..................] sync'ed:  9.5% (431796/471040)K
        finish: 0:01:27 speed: 4,904 (6,592 -- 4,904) K/sec
          0% sector pos: 0/4201520
        resync: used:0/61 hits:1504 misses:2 starving:0 locked:0 changed:1
        act_log: used:0/1237 hits:32684 misses:15231 starving:0 locked:0 changed:171
        blocked on activity log: 0/0/0
```

5. 4ノード構成スタッキングクラスタ

ここではリージョンが異なる東京とシンガポールの2拠点間で東京をメインサイト、シンガポールをDRサイトとし、拠点内は同一アベイラビリティーゾーンでアクティブ・スタンバイのHA構成でサービスを提供する。メインサイトに障害が発生したときは、DRサイトでサービスを継続する。拠点間の切り替えはPacemaker Ticketによる手動切り替えとする。

5.1. /etc/hosts

東京の 2 ノードを awsnode1, awsnode2, シンガポールの 2 ノードを awsnode3, awsnode4 とします。
以下のIPアドレス、ホスト名を仮定してこれ以降説明します。

```
# cat /etc/hosts
:
10.0.1.38   awsnode1
10.0.1.169  awsnode2
10.1.1.33   awsnode3
10.1.1.59   awsnode4
```

5.2. 下位 DRBD の設定

下位の DRBD の設定を行います。以下の r0.res, r1.res をすべてのノードで設定します。ここでは /dev/drbdpool/disk1 を下位デバイスとして使用します。

```
# cat /etc/drbd.d/r0.res
resource r0 {
  options
  {
    auto-promote no;
  }

  net {
    protocol C;
  }

  on awsnode1 {
    device    /dev/drbd0;
    disk      /dev/drbdpool/disk1;
    address   10.0.1.38:7788;
    meta-disk internal;
  }

  on awsnode2 {
    device    /dev/drbd0;
    disk      /dev/drbdpool/disk1;
    address   10.0.1.169:7788;
    meta-disk internal;
  }
}

# cat /etc/drbd.d/r1.res
resource r1 {
  options
  {
    auto-promote no;
  }

  net {
    protocol C;
  }

  on awsnode3 {
    device    /dev/drbd0;
    disk      /dev/drbdpool/disk1;
    address   10.1.1.33:7788;
    meta-disk internal;
  }

  on awsnode4 {
    device    /dev/drbd0;
    disk      /dev/drbdpool/disk1;
    address   10.1.1.59:7788;
    meta-disk internal;
  }
}
```

awsnode1, awsnode2 で r0 を初期化します。

```
# drbdadm create-md r0
```

awsnode1 でプライマリにし、初期同期を行っておきます。

```
# drbdadm --force primary r0
# drbdadm secondary r0
# drbdadm down r0
```

r1 に関しても awsnode3, awsnode4 で行います。

```
# drbdadm create-md r1
```

awsnode3 でプライマリにし、初期同期を行っておきます。

```
# drbdadm --force primary r0
# drbdadm secondary r0
# drbdadm down r0
```

5.3. corosync の設定

corosync は東京の awsnode1, awsnode2 で設定、シンガポールの awsnode3, awsnode4 で設定します。違いは ring0_addr: の IP address の箇所だけになりますので、ここでは 東京の awsnode1, awsnode2 で設定例を示します。シンガポールの awsnode3, awsnode4 ではそれぞれの IP address に読み替えて設定してください。

```
# cat /etc/corosync/corosync.conf
totem {
        version: 2

        crypto_cipher: none
        crypto_hash: none
        cluster_name: cluster_test

        transport: udpu
        rrp_mode: passive
}

logging {
        fileline: off
        to_stderr: no
        to_logfile: yes
        logfile: /var/log/corosync.log
        to_syslog: yes
        debug: off
        timestamp: on
        logger_subsys {
                subsys: QUORUM
                debug: off
        }
}

quorum {
        provider: corosync_votequorum
        expected_votes: 2
        two_node: 1
}

nodelist {
        node {
                ring0_addr: 10.0.1.248
                nodeid: 1
        }
        node {
                ring0_addr: 10.0.2.85
                nodeid: 2
        }
}
```

設定しましたら、すべてのノードで pacemaker service を起動します。pacemaker を起動すると corosync は自動的に起動されます。

```
# systemctl start pacemaker
```

5.4. 上位 DRBD の設定

上位の DRBD の設定を行います。以下の rU.res をすべてのノードで設定します。

```
# cat /etc/drbd.d/rU.res
resource rU {
  options
  {
    auto-promote no;
  }

  proxy {
    memlimit 100M;
    plugin {
      zlib level 9;
    }
  }

  net {
    protocol A;
    on-congestion pull-ahead;
    congestion-fill 50M;
  }

  stacked-on-top-of r0 {
    device    /dev/drbd10;
    address   127.0.0.1:7789;
      proxy on awsnode1 awsnode2 {
      inside 127.0.0.2:7789;
      outside 10.0.1.10:7789;
    }
  }

  stacked-on-top-of r1 {
    device /dev/drbd10;
    address 127.0.0.1:7789;
    proxy on awsnode3 awsnode4 {
      inside 127.0.0.2:7789;
      outside 10.1.1.10:7789;
    }
  }
}
```

vip を使用して上位の DRBD 設定を行いますので、先に pacemaker で下位の DRBD, vip を設定しておきます。pacemaker は東京、シンガポールの各サイトでそれぞれ設定します。

awsnode1 で以下を設定します。ここで東京サイトの vip は 10.0.1.10 とします。

```
# crm configure edit
:
primitive p_awsvip awsvip \
    params secondary_private_ip=10.0.1.10 \
    meta allow-migrate=true \
    op monitor timeout=30s interval=20s depth=0
primitive p_drbd_r0 ocf:linbit:drbd \
    params drbd_resource=r0 \
    op start interval=0 timeout=240 \
    op stop interval=0 timeout=120
primitive p_vip IPaddr2 \
    params ip=10.0.1.10 cidr_netmask=24 nic=eth0 \
    op start interval=0 timeout=20 \
    op stop interval=0 timeout=20 \
    op monitor interval=10 timeout=20
group rg_vip p_vip p_awsvip
ms ms_drbd_r0 p_drbd_r0 \
    meta master-max=1 master-node-max=1 clone-max=2 clone-node-max=1 notify=true
colocation c1 inf: rg_vip ms_drbd_r0:Master

property cib-bootstrap-options: \
    :
    stonith-enabled=false \
    no-quorum-policy=ignore \
    :
rsc_defaults rsc-options: \
    resource-stickiness=200
```

awsnode3 で以下を設定します。ここでシンガポールサイトの vip は 10.1.1.10 とします。

```
# crm configure edit
:
primitive p_awsvip awsvip \
    params secondary_private_ip=10.1.1.10 \
    meta allow-migrate=true \
    op monitor timeout=30s interval=20s depth=0
primitive p_drbd_r1 ocf:linbit:drbd \
    params drbd_resource=r1 \
    op start interval=0 timeout=240 \
    op stop interval=0 timeout=120
primitive p_vip IPaddr2 \
    params ip=10.1.1.10 cidr_netmask=24 nic=eth0 \
    op start interval=0 timeout=20 \
    op stop interval=0 timeout=20 \
    op monitor interval=10 timeout=20
group rg_vip p_vip p_awsvip
ms ms_drbd_r1 p_drbd_r1 \
    meta master-max=1 master-node-max=1 clone-max=2 clone-node-max=1 notify=true
colocation c1 inf: rg_vip ms_drbd_r1:Master
property cib-bootstrap-options: \
    :
    stonith-enabled=false \
    no-quorum-policy=ignore \
    :
rsc_defaults rsc-options: \
    resource-stickiness=200
```

東京サイトの r0 のマスターノードで以下を実行します。

```
# crm_mon -rf1|grep Masters:
     Masters: [ awsnode1 ]
```

この場合は awsnode1 がマスターなので awsnode1 で以下を実行し、上位 DRBD デバイスの初期化を行います。

```
# drbdadm -S create-md rU
# drbdadm -S up rU
# drbdadm -S --force primary rU
# mkfs.xfs /dev/drbd10
```

同様にして、シンガポールサイトの r1 のマスターノードで以下を実行します。

```
# crm_mon -rf1|grep Masters:
     Masters: [ awsnode3 ]
```

awsnode3 で

```
# drbdadm -S create-md rU
```

5.5. mysql の pacemaker 設定

mysql を pacemaker に設定します。東京サイトのマスターノードで最初に /var/lib/mysql 領域に mysql 初期データを書き込みます。

```
# mount /dev/drbd10 /var/lib/mysql
# systemctl start mysqld
# grep 'temporary password' /var/log/mysqld.log
# mysql -u root -p
> SET PASSWORD='XXXXXXX';
> grant all privileges on *.* to test_sios@"%" identified by 'XXXXXXX' with grant option;
> select user,host from mysql.user;
# systemctl stop mysqld
# umount /var/lib/mysql
```

次に各サイトのマスターノード configure edit で設定します。

```
# crm configre edit
:
primitive p_drbd_rU ocf:linbit:drbd \
    params drbd_resource=rU \
    op start interval=0 timeout=240 \
    op stop interval=0 timeout=120

primitive p_drbdproxy_rU lsb:/etc/init.d/drbdproxy \
    op start interval=0 timeout=60

ms ms_drbd_r10 p_drbd_rU \
    meta master-max=1 master-node-max=1 clone-max=1 clone-node-max=1 \
    notify=true

colocation c2 inf: ms_drbd_r10 p_drbdproxy_rU rg_vip

# On r0 nodes, use the followng.
# order o1 inf: ms_drbd_r0:promote rg_vip:start p_drbdproxy_rU:start \
#   ms_drbd_r10:start

# On r1 nodes, use the following.
# order o1 inf: ms_drbd_r1:promote rg_vip:start p_drbdproxy_rU:start \
#   ms_drbd_r10:start

rsc_ticket res_drbd_req-activeS activeS: \
    ms_drbd_r10:Master loss-policy=demote
```

なお、東京サイトの場合は、On r0 nodes の次の２行のコメントを削除

```
# On r0 nodes, use the followng.
order o1 inf: ms_drbd_r0:promote rg_vip:start p_drbdproxy_rU:start \
    ms_drbd_r10:start
```

シンガポールサイトの場合は、On r1 nodes の次の２行のコメントを削除してください。

```
# On r1 nodes, use the following.
order o1 inf: ms_drbd_r1:promote rg_vip:start p_drbdproxy_rU:start \
    ms_drbd_r10:start
```

ここまで両サイトで上位 DRBD がスレイブ状態で接続されています。ticket が与えられないとマスターになれない状態です。各サイトのマスターで引き続き以下を設定します。

```
# crm configre edit
:
primitive p_filesystem Filesystem \
        params device="/dev/drbd10" directory="/var/lib/mysql" fstype=xfs \
        options=noatime \
        op start interval=0 timeout=60 \
        op stop interval=0 timeout=60 \
        op monitor interval=20 timeout=40

primitive p_mysql mysql \
        params binary="/usr/sbin/mysqld" \
        op start interval=0 timeout=120 \
        op stop interval=0 timeout=120 \
        op monitor interval=20 timeout=30

primitive p_mysql_vpc_route53 aws-vpc-route53 \
    params hostedzoneid=ZXXXXXXXXXXXXXXXXXXXX ttl=10 \
    fullname=mysql.test.local. profile=cluster \
    op start interval=0 timeout=180 \
    op stop interval=0 timeout=180 \
    op monitor interval=300 timeout=180

group rg_service p_filesystem p_mysql p_mysql_vpc_route53

# on both nodes,
order o2 inf: ms_drbd_r10:promote rg_service:start
colocation c3 inf: rg_service ms_drbd_r10:Master
```

最後にすべてのノードで drbdproxy の自動起動は止めておきます。

```
# chkconfig drbdproxy off
```

以上で設定完了です。

5.6. サイトの切り替え

サイトの切り替えは crm_ticket で行います。

東京サイトで mysql サービスを使用する場合は、東京サイトのノードで以下を実行します。

```
# crm_ticket --ticket activeS --grant --force
```

サイトの切り替えは、東京サイトで以下を実行し、

```
crm_ticket --ticket activeS --revoke --force
```

シンガポールサイトで以下を実行します。

```
# crm_ticket --ticket activeS --grant --force
```

5.7. mysql クライアントからの接続

AWS 内の同じ hostedzoneid 内のクライアントから fullname (この例では mysql.test.local) で mysql サーバにアクセスできます。サイトが切り替わっても同じ方法でアクセスできます。

```
$ mysql -u test_sios -p'XXXXXXXXX' -h mysql.test.local -P 3306
```

5.8. pacemaker サービスのスタート、ストップ

5.8.1. スタート

すべてのノードで pacemaker サービスを起動します。

```
# systemctl start pacemaker
```

5.8.2. ストップ

現在 ticket のあるサイトで以下を実行します。

```
# crm_ticket --ticket activeS --revoke --force
```

すべてのノードで pacemaker サービスを終了します。

```
# systemctl stop pacemaker
```

6. 4ノード構成フラット接続クラスタ

ここではリージョンが異なる東京とシンガポールの2拠点間で東京をメインサイト、シンガポールをDRサイトとし、拠点内は異なるアベイラビリティーゾーン(AZ)でアクティブ・スタンバイのHA構成でサービスを提供する。メインサイトに障害が発生したときは、DRサイトでサービスを継続する。拠点間の切り替えはPacemaker Ticketによる手動切り替えとします。

6.1. /etc/hosts

東京の２ノードを awsnode1, awsnode2, シンガポールの２ノードを awsnode3,awsnode4 とします。
以下のIPアドレス、ホスト名を仮定してこれ以降説明します。

```
# cat /etc/hosts
:
10.0.1.188  awsnode1
10.0.2.181  awsnode2
10.1.1.6    awsnode3
10.1.2.251  awsnode4
```

6.2. DRBDディスク領域の設定

DRBD の設定を行います。以下の mariadb.res をすべてのノードで設定します。ここでは /dev/drbdpool/mariadb を下位デバイスとして使用します。

```
# cat /etc/drbd.d/mariadb.res
resource "mariadb"
{

    net
    {
        cram-hmac-alg     sha1;
        shared-secret     "+Qm+6dAVy3XyIX4icACf";
    }

    on awsnode1
    {
        volume 0
        {
            disk        /dev/drbdpool/mariadb;
            disk
            {
                discard-zeroes-if-aligned yes;
                rs-discard-granularity 65536;
            }
            meta-disk   internal;
            device      minor 1000;
        }
        node-id    0;
    }
```

```
    on awsnode2
    {
        volume 0
        {
            disk        /dev/drbdpool/mariadb;
            disk
            {
                discard-zeroes-if-aligned yes;
                rs-discard-granularity 65536;
            }
            meta-disk   internal;
            device      minor 1000;
        }
        node-id    1;
    }

    on awsnode3
    {
        volume 0
        {
            disk        /dev/drbdpool/mariadb;
            disk
            {
                discard-zeroes-if-aligned yes;
                rs-discard-granularity 65536;
            }
            meta-disk   internal;
            device      minor 1000;
        }
        node-id    2;
    }

    on awsnode4
    {
        volume 0
        {
            disk        /dev/drbdpool/mariadb;
            disk
            {
                discard-zeroes-if-aligned yes;
                rs-discard-granularity 65536;
            }
            meta-disk   internal;
            device      minor 1000;
        }
        node-id    3;
    }

    connection
    {
        host awsnode1 address ipv4 10.0.1.188:7000;
        host awsnode2 address ipv4 10.0.2.181:7000;
    }
```

```
    connection
    {

        net
        {
            protocol A;
        }
        host awsnode1 address 127.0.0.1:7001 via proxy on awsnode1
        {
            inside 127.0.0.2:7001;
            outside ipv4 10.0.1.188:7001;
        }
        host awsnode3 address 127.0.0.1:7001 via proxy on awsnode3
        {
            inside 127.0.0.2:7001;
            outside ipv4 10.1.1.6:7001;
        }
    }

    connection
    {

        net
        {
            protocol A;
        }
        host awsnode1 address 127.0.0.1:7003 via proxy on awsnode1
        {
            inside 127.0.0.2:7003;
            outside ipv4 10.0.1.188:7003;
        }
        host awsnode4 address 127.0.0.1:7003 via proxy on awsnode4
        {
            inside 127.0.0.2:7003;
            outside ipv4 10.1.2.251:7003;
        }
    }

    connection
    {
        host awsnode4 address ipv4 10.1.2.251:7000;
        host awsnode3 address ipv4 10.1.1.6:7000;
    }

    proxy
    {
        memlimit 20000000;
    }
}
```

awsnode1, awsnode2, awsnode3, awsnode4 でデータリソースmariadbを初期化します。
次コマンドをすべてのノードで実行します。

```
# drbdadm create-md mariadb
# drbdadm up mariadb
```

データリソース mariadb を awsnode1 でプライマリにし、初期同期を行っておきます。
次のコマンドを awsnode1 で実行します。

```
# drbdadm --force primary r0
# drbdadm secondary r0
```

データリソース mariadb を xfs ファイルシステムで初期化しておきます。
次のコマンドを awsnode1 で実行します。

```
# mkfs.xfs /dev/drbd1000
```

6.3. mysql の pacemaker 設定

mysql を pacemaker に設定します。東京サイトのマスターノードで最初に /var/lib/mysql 領域に
mysql 初期データを書き込みます。
次のコマンドを awsnode1 で実行します。

```
# mount /dev/drbd1000 /var/lib/mysql
# systemctl start mysqld
# grep 'temporary password' /var/log/mysqld.log
# mysql -u root -p
> SET PASSWORD='XXXXXXX';
> grant all privileges on *.* to test_sios@"%" identified by 'XXXXXXX' with grant option;
> select user,host from mysql.user;
# systemctl stop mysqld
# umount /var/lib/mysql
```

6.4. Corosync設定

corosync は東京の awsnode1, awsnode2 で設定、シンガポールの awsnode3, awsnode4 で設定します。違いは ring0_addr: の IP address の箇所だけになりますので、ここでは 東京の awsnode1, awsnode2 で設定例を示します。シンガポールの awsnode3, awsnode4 ではそれぞれの IP address に読み替えて設定してください。

```
[root@awsnode1 ~]# cat /etc/corosync/corosync.conf
totem {
        version: 2

        crypto_cipher: none
        crypto_hash: none
        cluster_name: cluster_test

        transport: udpu
        rrp_mode: passive
}
logging {
        fileline: off
        to_stderr: no
        to_logfile: yes
        logfile: /var/log/corosync.log
        to_syslog: yes
        debug: off
        timestamp: on
        logger_subsys {
                subsys: QUORUM
                debug: off
        }
}
quorum {
        provider: corosync_votequorum
        expected_votes: 2
        two_node: 1
}
nodelist {
        node {
                ring0_addr: 10.0.1.188
                nodeid: 1
        }
        node {
                ring0_addr: 10.0.2.181
                nodeid: 2
        }
}
```

設定しましたら、すべてのノードで pacemaker service を起動します。pacemaker を起動すると corosync は自動的に起動されます。

```
# systemctl start pacemaker
```

6.5. Pacemaker設定

東京の awsnode1, シンガポールの awsnode3 で Pacemaker の設定を入力します。

```
# crm configure edit
:
primitive p_drbd_mariadb ocf:linbit:drbd \
        params drbd_resource=mariadb \
        op start interval=0 timeout=240 \
        op stop interval=0 timeout=120
primitive p_res_mariadb mysql \
        params binary="/usr/libexec/mysqld" \
        op start interval=0 timeout=120 \
        op stop interval=0 timeout=120 \
        op monitor interval=20 timeout=30
primitive p_res_mariadb_fs Filesystem \
        params device="/dev/drbd1000" directory="/var/lib/mysql" \
    fstype=xfs options=noatime \
        op start interval=0 timeout=60 \
        op stop interval=0 timeout=60 \
        op monitor interval=20 timeout=40 \
        meta target-role=Started
primitive p_mysql_vpc_route53 aws-vpc-route53 ¥
    params hostedzoneid=ZXXXXXXXXXXXXXXXXXXX ttl=10 \
    fullname=mysql.test.local. profile=cluster \
    op start interval=0 timeout=180 \
    op stop interval=0 timeout=180 \
    op monitor interval=300 timeout=180
group rg_mariadb p_res_mariadb_fs p_res_mariadb p_mysql_vpc_route53
ms ms_drbd_r_mariadb p_drbd_mariadb \
        meta master-max=1 master-node-max=1 clone-max=2 clone-node-max=1 notify=true
colocation c1 inf: rg_mariadb ms_drbd_r_mariadb:Master
order o1 inf: ms_drbd_r_mariadb:promote rg_mariadb:start
rsc_ticket res_drbd_req-activeS activeS: ms_drbd_r_mariadb:Master loss-policy=demote
property cib-bootstrap-options: \
:
        stonith-enabled=false \
        no-quorum-policy=ignore \
:
rsc_defaults rsc-options: \
        resource-stickiness=200
```

6.6. サイトの切り替え

サイトの切り替えは crm_ticket コマンドで行います。

東京サイトで mysql サービスを使用する場合は、東京サイトのノードで以下を実行します。

```
# crm_ticket --ticket activeS --grant --force
```

動作確認はcrm_monコマンドで行います。

```
[root@awsnode1 ~]# crm_mon -rD1 --ticket

Online: [ awsnode1 awsnode2 ]

Full list of resources:

 Resource Group: rg_mariadb
     p_res_mariadb_fs   (ocf::heartbeat:Filesystem):    Started awsnode1
     p_res_mariadb      (ocf::heartbeat:mysql): Started awsnode1
 Master/Slave Set: ms_drbd_r_mariadb [p_drbd_mariadb]
     Masters: [ awsnode1 ]
     Slaves: [ awsnode2 ]

Tickets:
* activeS:      granted last-granted='Tue Jun 25 12:42:37 2019'
```

mariadbが 東京のawsnode1で実行されていることが判ります。

サイトの切り替えは、東京サイトで以下を実行し、

```
crm_ticket --ticket activeS --revoke --force
```

シンガポールサイトで以下を実行します。

```
# crm_ticket --ticket activeS --grant --force
```

6.7. mysql クライアントからの接続

AWS 内の同じ hostedzoneid 内のクライアントから fullname (この例では mysql.test.local) で mysql サーバにアクセスできます。サイトが切り替わっても同じ方法でアクセスできます。

```
$ mysql -u test_sios -p'XXXXXXXXX' -h mysql.test.local -P 3306
```

6.8. pacemaker サービスのスタート、ストップ

6.8.1. スタート

すべてのノードで pacemaker サービスを起動します。

```
# systemctl start pacemaker
```

6.8.2. ストップ

現在 ticket のあるサイトで以下を実行します。

```
# crm_ticket --ticket activeS --revoke --force
```

すべてのノードで pacemaker サービスを終了します。

```
# systemctl stop pacemaker
```

7. 更新履歴

日付	版	内容
2019-07-01	1.0	初版