Hazelcast on AWS: Best Practices for Deployment

This paper is targeted towards cloud architects/developers who gear up an Hazelcast application from a physical environment to a virtualized cloud environment, namely EC2 (Amazon Elastic Compute Cloud). EC2 is a central part of Amazon.com's cloud computing platform, Amazon Web Services(AWS). Focus of this paper is to highlight best practices in running Hazelcast applications on EC2.

EC2 has certain limitations, for example Multicast is not supported. Moreover, the network can fluctuate since you share your data center, physical machine or even CPU with other applications in the cloud that is unknown to you. This, results in a more unpredictable environment than a traditional in-house data center. Hence, it is important to configure Hazelcast in a way that it minimizes any network outage or fluctuation.

This paper:

- Provides an overview on the terminology and high-level architecture of Amazon Web Services (AWS)
- Delivers guidance on a smooth deployment of Hazelcast on EC2
- Explains some of the high availability options on EC2

AWS Terminology

Amazon EC2

Amazon EC2 stands for Elastic Compute Cloud and is the core of Amazon`s cloud-based services. It provides resizable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

Regions and Availability Zones

Amazon EC2 is hosted in multiple locations world-wide each of which are composed of *Regions* and *Availability Zones*. Each *Region* is a separate, independent geographic area and has multiple, isolated locations known as *Availability Zones*. Although these zones are isolated, they are connected through low-latency links.

Using Amazon EC2 enables the resources (instances and data) to be placed in multiple locations. There is no resource replication between regions unless it is specified. This topic is discussed in *WAN Replication for EC2 Regions* section. Note that some AWS resources might not be available in all Regions and Availability Zones, so before deploying any applications, it is required to guarantee that the resources can be created in the desired location. Amazon EC2 resources are either global, tied to a region, or tied to an Availability Zone.

Since Amazon EC2 offers multiple regions, you can start Amazon EC2 instances in any location that meets your requirements. For example, you may want to start instances in Asia Pacific region to be closer to your far east customers. Below is the table listing the regions that provide support for Amazon EC2.

Region Code	Region Name
ap-northeast-1	Asia Pacific (Tokyo) Region
ap-southeast-1	Asia Pacific (Singapore) Region
ap-southeast-2	Asia Pacific (Sydney) Region
eu-west-1	EU (Ireland) Region
sa-east-1	South America (Sao Paulo) Region
us-east-1	US East (Northern Virginia) Region
us-west-1	US West (Northern California) Region
us-west-2	US West (Oregon) Region

Deployment Best Practices for Hazelcast on AWS

Below sections provide best practices and recommendations to deploy Hazelcast on AWS.

Which EC2 Instance Types to Choose

Hazelcast is an in-memory data grid that distributes the data and computation to the nodes that are connected with a network, making it very sensitive to the network. Not all EC2 Instance types are the same in terms of network performance. It is highly recommended to choose instances that have *10 Gigabit* or *High* network performance for Hazelcast deployments. Below table lists the specifications of Amazon EC2 instance types and you can see the instance types that we recommend in red.

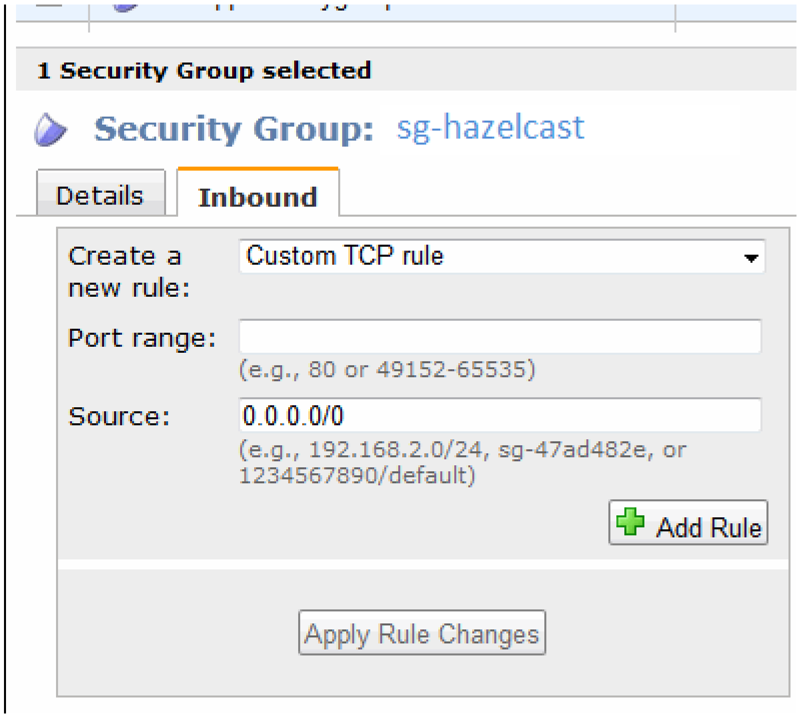
Instance Family	**Instance Type**	**Processor Arch**	**vCPU**	**ECU**	**Memory (GiB)**	**Instance Storage**	**EBS-Optimized Available**	**Network Performance**
General purpose	m3.xlarge	64-bit	4	13	15	2 x 40 SSD	Yes	Moderate
General purpose	**m3.2xlarge**	**64-bit**	**8**	**26**	**30**	**2 x 80 SSD**	**Yes**	**High**
General purpose	m1.small	32-bit or 64-bit	11	1	1.7	1 x 160	-	Low
General purpose	m1.medium	32-bit or 64-bit	1	2	3.75	1 x 410	-	Moderate
General purpose	m1.large	64-bit	2	4	7.5	2 x 420	Yes	Moderate
General purpose	**m1.xlarge**	**64-bit**	**4**	**8**	**15**	**4 x 420**	**Yes**	**High**
Compute optimized	c3.large	64-bit	2	7	3.75	2 x 16 SSD	-	Moderate
Compute optimized	c3.xlarge	64-bit	4	14	7.5	2 x 40 SSD	Yes	Moderate
Compute optimized	**c3.2xlarge**	**64-bit**	**8**	**28**	**15**	**2 x 80 SSD**	**Yes**	**High**
Compute optimized	**c3.4xlarge**	**64-bit**	**16**	**55**	**30**	**2 x 160 SSD**	**Yes**	**High**
Compute optimized	**c3.8xlarge**	**64-bit**	**32**	**108**	**60**	**2 x 320 SSD**	**-**	**10 Gigabit**
Compute optimized	c1.medium	32-bit or 64-bit	2	5	1.7	1 x 350	-	Moderate
Compute optimized	**c1.xlarge**	**64-bit**	**8**	**20**	**7**	**4 x 420**	**Yes**	**High**
Compute optimized	**cc2.8xlarge**	**64-bit**	**32**	**88**	**60.5**	**4 x 840**	**-**	**10 Gigabit**
GPU instances	g2.2xlarge	64-bit	8	26	15	1 x 60 SSD	Yes	High
GPU instances	cg1.4xlarge	64-bit	16	33.5	22.5	2 x 840	-	10 Gigabit
Memory optimized	m2.xlarge	64-bit	2	6.5	17.1	1 x 420	-	Moderate
Memory optimized	m2.2xlarge	64-bit	4	13	34.2	1 x 850	Yes	Moderate

Memory optimized	**m2.4xlarge**	**64-bit**	**8**	**26**	**68.4**	**2 x 840**	**Yes**	**High**
Memory optimized	**cr1.8xlarge**	**64-bit**	**32**	**88**	**244**	**2 x 120 SSD**	**-**	**10 Gigabit**
Storage optimized	i2.xlarge	64-bit	4	14	30.5	1 x 800 SSD	Yes	Moderate
Storage optimized	i2.2xlarge	64-bit	8	27	61	2 x 800 SSD	Yes	High
Storage optimized	i2.4xlarge	64-bit	16	53	122	4 x 800 SSD	Yes	High
Storage optimized	i2.8xlarge	64-bit	32	104	244	8 x 800 SSD	-	10 Gigabit
Storage optimized	hs1.8xlarge	64-bit	16	35	117	24 x 2,0483	-	10 Gigabit
Storage optimized	hi1.4xlarge	64-bit	16	35	60.5	2 x 1,024 SSD2	-	10 Gigabit
Micro instances	t1.micro	32-bit or 64-bit	1	Variable5	0.615	EBS only	-	Very Low

Configuring the Security

By default, Hazelcast uses port 5701. It is recommended to create a Hazelcast specific security group. Then, an inbound rule for port 5701 from *sg-hazelcast* needs to be added to this security group. Follow the below instructions:

- Open the Amazon EC2 console.
- Click **Security Groups** in the left menu.
- Click **Create Security Group** and enter a name (e.g. *sg-hazelcast*) and description for the security group, click **Yes, Create**.
- On **Security Groups** page, select the security group *sg-hazelcast* on the right pane.
- You will see a field below the security group list with the tabs **Details** and **Inbound**. Select **Inbound**. Below screen appears.


1 Security Group selected

Security Group: sg-hazelcast

Create a new rule: Custom TCP rule

Port range:
(e.g., 80 or 49152-65535)

Source: 0.0.0.0/0
(e.g., 192.168.2.0/24, sg-47ad482e, or 1234567890/default)

- Select *Custom TCP rule* in the field **Create a new rule**. Type *5701* into the field **Port range** and *sg-hazelcast* into **Source**.
- Click **Add Rule**.

Configuring for EC2 Auto Discovery

Hazelcast either uses Multicast or TCP/IP for discovery, but EC2 does not support multicast. To configure discovery using TCP/IP, you need the IP addresses upfront and this is not always possible. To solve this problem, Hazelcast supports EC2 auto discovery which is a layer on top of TCP/IP discovery. EC2 auto discovery uses AWS API to get the IP addresses of possible Hazelcast nodes and feeds those IP addresses to TCP/IP discovery. This way the discovery process becomes dynamic and it eliminates a need for knowing the IP addresses upfront. To limit the IP addresses only to Hazelcast related nodes, EC2 discovery supports filtering based on security group and/or tags.

A simple cluster configuration should be as follows:

- If you have the dependency to *hazelcast-all.jar,* there is no need to do anything. If you are not using *hazelcast-all.jar* but depend on *hazel cast-jar* and other module jars that you need, then you should add *hazelcast-cloud.jar* too.
- Disable Multicast and TCP/IP in the *join* section of the cluster configuration XML file.
- Enable AWS and provide the access and secret keys in the *join* section.
- Set the correct region. If it is not set, the default region *us-east-1* will be used.
- As an important recommendation, you can add the security group you specified at EC2 management console (please see Configuring the Security section) to narrow the Hazelcast nodes to be within this security group. This is done by specifying the security group name in the configuration XML file (using the *security-group-name* tag, in the below sample it is given as *sg-hazelcast*).
- You can also add any available tags to narrow only to Hazelcast nodes using the tags *tag-key* and *tag-value*.
- If the cluster that you are trying to form is relatively big (>20 nodes), you may need to increase the connection timeout (specified by the *c onn-timeout-seconds* attribute). Otherwise, it is possible that not all members join the cluster. It may lead to a confusion to set the timeout on TCP/IP although we are using AWS. But, remember that AWS discovery is using TCP/IP discovery underneath.

Sample configuration is shown below.

```
<join>
    <tcp-ip conn-timeout-seconds="20" enabled="false">
    </tcp-ip>
    <multicast enabled="false">
        ......
    </multicast>
    <tcp-ip enabled="false">
        ......
    </tcp-ip>
    <aws enabled="true">
        <access-key>my-access-key</access-key>
        <secret-key>my-secret-key</secret-key>
        <region>us-east-1</region>
        <host-header>ec2.amazonaws.com</host-header>
        <security-group-name>sg-hazelcast</security-group-name>
        <tag-key>type</tag-key>
        <tag-value>hz-nodes</tag-value>
    </aws>
</join>
```

Node Failure Detection: Ping

Hazelcast nodes send heartbeats to each other at every second. This is to determine whether each node is alive or not. If a node does not respond to a heartbeat for 5 minutes, then it is considered as down. Let`s say a problem has occurred in the network (e.g. a node is terminated in EC2 management console, a switch has broken, etc.) and a node, which is not in the garbage collection state (GC), cannot respond to heartbeat messages. How do you know if that node is up or down: the answer is by using ICMP requests (pings).

Now, Amazon EC2 instances do not respond to pings by default. To learn whether a node is up or down, pings should be enabled on EC2 through your security group and also ICMP must be enabled in your Hazelcast configuration.

To enable ICMP on EC2:

- After logging into AWS, from the management console click on "**Security Groups**" and select the related group.
- Click on "**Inbound**" tab and create a new rule with the following attributes:
    - Select "**Custom ICMP Rule**" in the field *Create a new rule*.
    - As the rule type, select **Echo Request**.
    - Into the *Source* field, type **0.0.0.0/0** to allow all nodes to ping your node, or type your own IP addresses.

- Or, you can use the EC2 command line tool and type the below commands:
    - To allow all nodes to ping yours:

      ```
      ec2-authorize default -P icmp -t -1:-1 -s 0.0.0.0/0
      ```

    - To allow specific nodes to ping yours:

      ```
      ec2-authorize default -P icmp -t -1:-1 -s <IP addresses>
      ```

To enable ICMP on Hazelcast:

- In your Hazelcast configuration, set the property *hazelcast.icmp.enabled* to **true (**which is **false** by default):

  ```
  <properties>

  <property name="hazelcast.icmp.enabled">true</property>

  </properties>
  ```

Now, Hazelcast nodes can ping each other and if they get responds from a node that does not answer to heartbeats, that node is considered to be in GC or to have some anomalies like overloaded CPU. Therefore, the nodes will wait for the heartbeat responses, and if there is no response for the heartbeats for 5 minutes, then that node is considered as down.

On the other hand, if the nodes cannot get a response to pings from a node, then that node is considered as down. This way a failure can be detected very fast.

Network Latency

Since data is sent-received very frequently in Hazelcast applications, latency in network becomes a crucial issue. In terms of latency, performance of the AWS cloud is not same for each region and there are vast differences in speed and optimization from region to region. *"We deployed our Hazelcast application on AWS Cloud Virginia and we were noticing lot of problems related to network. Then I moved the whole thing to Oregon and boom! Everything is working like the clock."* says Mr. Abbas Bradan, Managing Partner at eConcepts.

It is a known fact that when you do not pay attention to AWS regions, Hazelcast applications may run tens or hundreds times slower than necessary. So, in this section, let`s look at what can be the workarounds:

- Create a cluster only within a region. We do not recommend to deploy a single cluster that spans across multiple regions.
- If a Hazelcast application is hosted on Amazon EC2 instances in multiple EC2 regions, the latency can be reduced by serving the end users` requests from the EC2 region which has the lowest network latency. Changes in network connectivity and routing result in changes in the latency between hosts on the Internet. Amazon has a web service (*Route 53*) that lets the cloud architects use DNS to route end-user requests to the EC2 region that gives the fastest response. This latency-based routing is based on latency measurements performed over a period of time. Please have a look at http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/HowDoesRoute53Work.html.
- Simply move the deployment to another Region. The tool located at http://www.cloudping.info/ gives instant estimates on the latency from your location. By using it frequently, it can be helpful to determine the regions which have the lowest latency.
- Also, the test provided at http://cloudharmony.com/speedtest allows you not only test the network latency but also the downloading/uploading speeds.

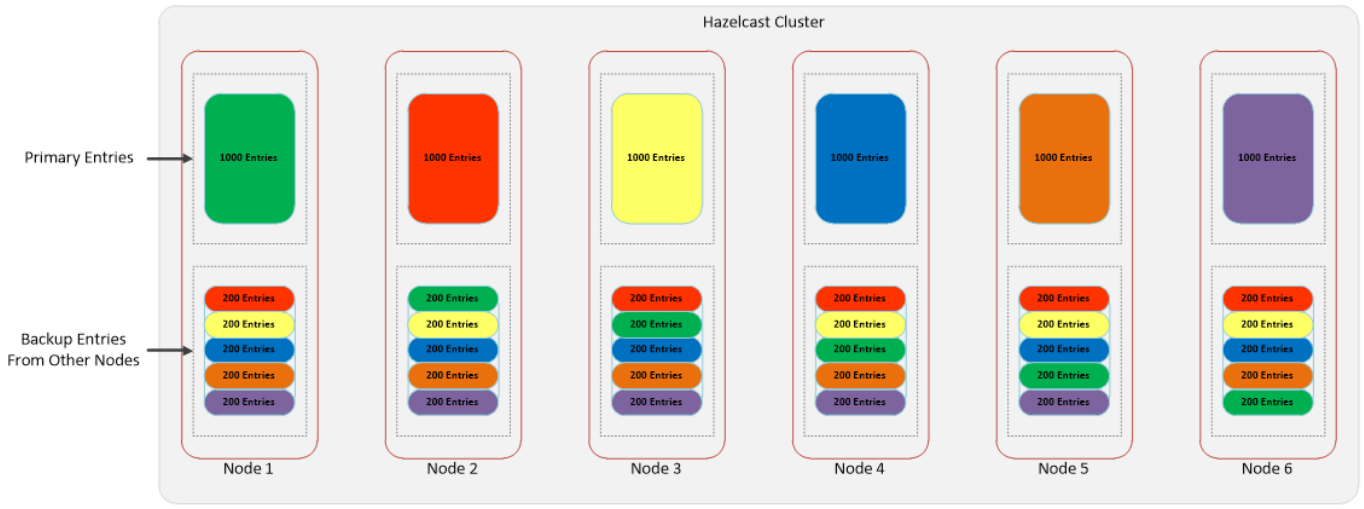
Debugging

When and if needed, Hazelcast can log the events for the instances that exist in a region. To see what has happened or trace the activities while deploying, change the log level in your logging mechanism to **FINEST** or **DEBUG**. After this change, you can also see whether the instances are accepted or rejected, and the reason of rejection for the rejected instances in the generated log. Note that, changing the log level to one of the mentioned levels may affect the performance of the cluster.

High Availability Across Zones

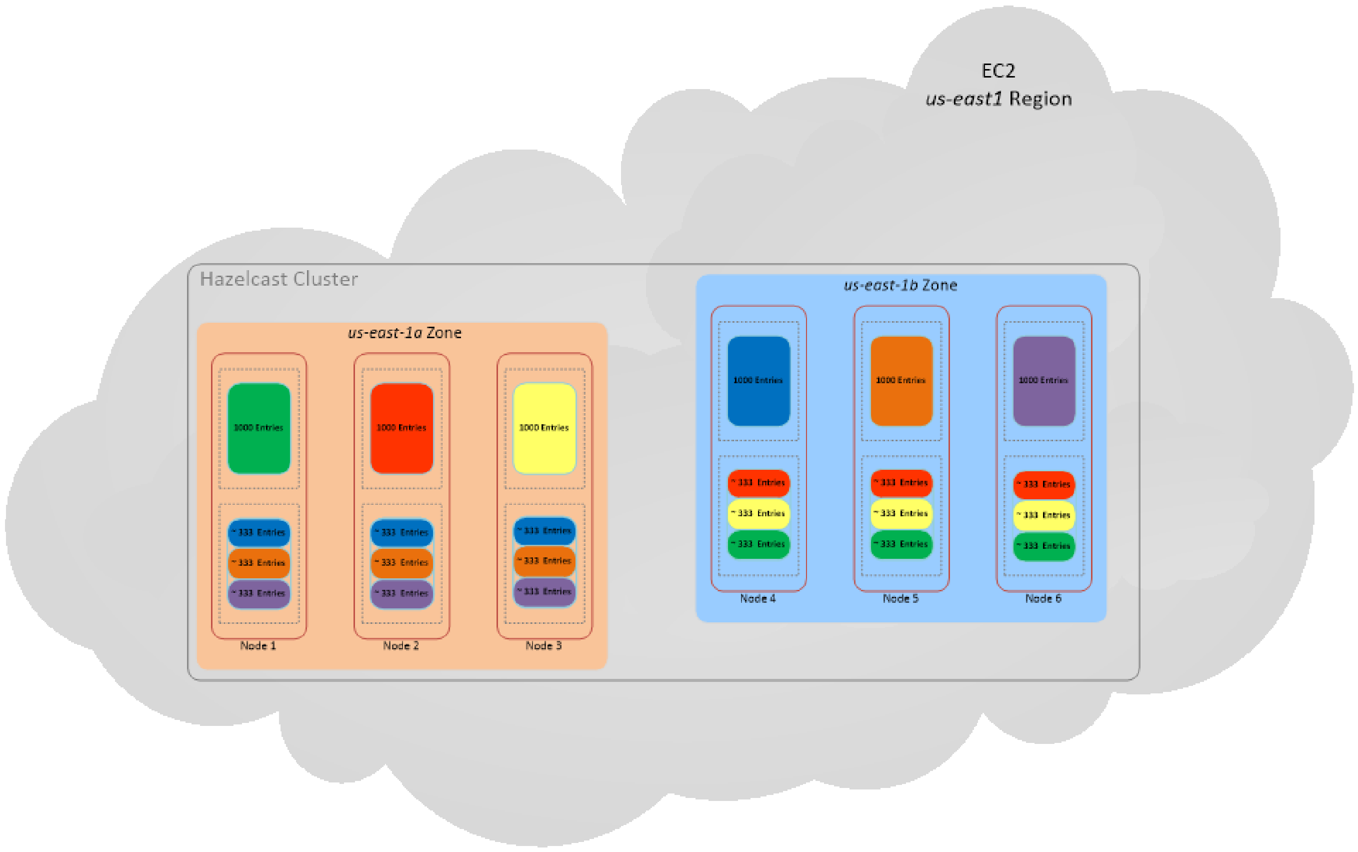
As you know Hazelcast is highly available thanks to the fact that it stores the backup of the data. Let us explain this fact with an example.

Assume that you have 6000 entries to be stored on 6 nodes, and the backup count is 1 (this means that every entry in a distributed map will have 1 backup in the cluster). In this case, each node will be storing around 1000 entries. To be highly available and by default, Hazelcast will backup these 1000 entries on the other 5 nodes by storing (approx.) 200 entries on each of them. So, each node will have 1000 entries as primary and another 1000, which is the combination of 200 entries from each other node, as the backup as shown in below illustration (please show regard to the colors as they are aimed to show the primary entries in each node and where their backups are).



If a node crashes, Hazelcast will automatically recover the data that is lost from the backups. But, if two nodes are lost at the same time in this configuration, then the cluster will lose some data (roughly 400 entries will be lost for this example). Putting this situation in terms of EC2: in an EC2 environment, it is very possible that an entire EC2 zone can go down causing your nodes in that zone to be dead. In order not to lose data in that environment, Hazelcast supports **partition groups** where you create a cluster within the same region that spans multiple zones.

From the same example above, assume that you have a 6-node cluster in *us-east-1,* where three of them is in *us-east-1a* and the rest three is in *us-east-1b*. Now, what you would like is one zone to be the backup of the other zone. Namely, when we store 6000 entries, each node will hold 1000. But, the backup of these 1000 entries will reside in three nodes on the other zone. This way, if one of the zones goes down entirely, Hazelcast will not lose any data even if the backup count is 1. See the illustration below.



Now, to have a zone to include the backup of the other one, you need to create member (node) groups under the *partition-group* tag. For the above example, you will create a group of 3 members for each zone and they will be the backups of each other. To create the member groups, you simply give the IP addresses of the nodes under the *member-group* tag. Below is a sample configuration.

```
<hazelcast>
   <partition-group enabled="true" group-type="CUSTOM">
       <member-group>
           <interface>12.12.12.1</interface>
           <interface>12.12.12.2</interface>
           <interface>12.12.12.3</interface>
       </member-group>
       <member-group>
           <interface>12.12.13.*</interface>
       </member-group>
   </partition-group>
<hazelcast>
```

As you can see in the above sample, wildcards can be used for IP addresses (e.g. *12.12.13.** stands for the IP addresses ranging between *12.12.13.0* and *12.12.13.255*).

Now, you have 2 member groups, first one includes the nodes in, say *us-east-1a* and the second one includes the ones in *us-east-1b*. This way, Hazelcast will group the primary partitions of the nodes in each member group (zone) and distributes the backup of the nodes in one zone to the nodes in the other one (instead of distributing the backups to all other nodes).