Using EBS with Auto Scaling Groups

How to use the immense power of AWS
Auto-Scaling Groups for a stateful Docker application.

Background

Background

In a service-oriented world where requests can come from anywhere at any time, keeping a system constantly up and available is essential to its success.

When running at scale, failures happen. This is just a fact of life for modern, distributed systems.

The focus should not be on trying to prevent those failures, unless you want to start a hard-disk company. Instead, we should endeavour to automatically react to those failures - restoring service quickly and with minimal impact.

Background

ASGs (Auto Scaling Groups) can help by automatically monitoring the load and health of your instances. If a node fails, it will be replaced automatically so you don't get woken up in the middle of the night with a PagerDuty alert.

This post explores how to use AWS auto-scaling groups for stateful apps because special care needs to be taken when using EBS volumes.

A Use Case

Our application allows users to post data to our API and we use Cassandra to both save and analyse the data.

We decide to employ one of the killer features of Cassandra - the ability to scale horizontally.

A Use Case

We settle on having three nodes in our Cassandra ring. As well as providing high availability in the event of a node failure, this will also mean we distribute read queries across more CPUs and increase the total disk capacity across the cluster.

We **could** spin up three EC2 nodes and install Cassandra using Terraform or Ansible. However, for the reasons mentioned above, we want the Cassandra cluster to auto-heal if a node fails and so we decide to use an Auto-Scaling Group.

Auto-Scaling Groups

Auto-Scaling Groups

There are a few steps to creating an ASG:

- Create AMI
  - creating the base image to launch instances from
- Launch Configuration
  - configure what instance size and keyname
- Auto-Scaling Group
  - manage multiple instances in the group

Let's walk through this setup:

Auto-Scaling Groups

Create AMI

We bake an AMI based on Ubuntu Xenial 16.04 with Docker CE 17.03 installed so we can run Cassandra inside a container.

```
$ aws ec2 create-image \
    --instance-id ${BUILDER_INSTANCE_ID} \
    --name myami
```

portworx

Auto-Scaling Groups

Launch Configuration

Then we create a launch configuration which uses our AMI and instance type `t2.large` for our group.

```
$ aws autoscaling create-launch-configuration \
    --launch-configuration-name asg_demo_config \
    --image-id myami \
    --instance-type t2.large \
    --key-name my-key \
    --block-device-mappings
"[{\"DeviceName\":\"/dev/sda1\",\"Ebs\":{\"SnapshotId\":\"snap-3decf207\"}},{\"DeviceName\":\"/dev/sdf\",\"Ebs\":{\"SnapshotId\":\"snap-eed6ac86\"}}]"
```

Notice the `--block-device-mappings` field - this describes how our launch configuration will create and attach a new EBS drive to each instance.

Auto-Scaling Groups

Auto-Scaling Group

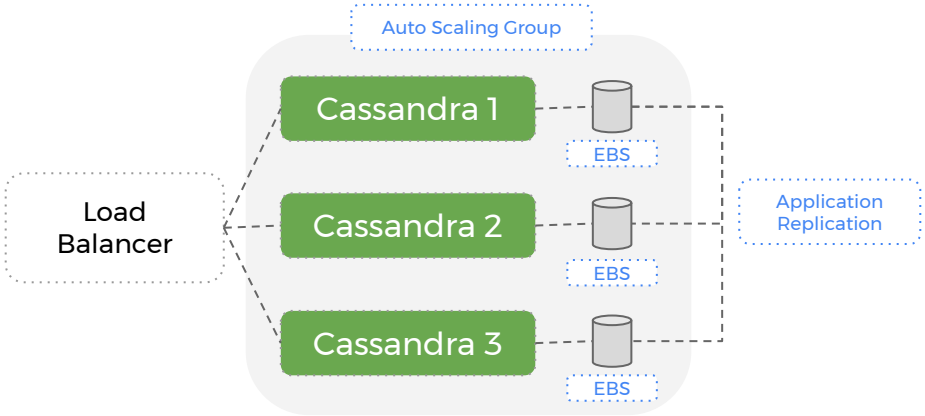
Next, we create the Auto-Scaling Group and point at the Launch Configuration we just made. This ASG now manages the number of instances in our Cassandra cluster. We create a Load-Balancer and point it at the ASG which lets us send traffic to any of the instances in that group.

```
$ aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name asg_demo \
    --launch-configuration-name asg_demo_config \
    --min-size 3 \
    --max-size 3 \
    --desired-capacity 3 \
    --availability-zones eu-west-2
```

portworx

Auto Scaling Groups

Let's see what this looks like:



portworx

A Problem

A Problem

When running tests with this setup, we realise a fundamental flaw in our system:

If a node fails, it is automatically replaced with a new EC2 instance and EBS volume (great), but this volume doesn't have any data. Cassandra will populate this empty volume using a replica but this can take a significant amount of time - which hurts our performance until complete.

The problem is that within an Auto-Scaling Group - AWS treats an EC2 instance and its associated EBS volume as a **single, atomic unit**.

portworx

A Problem

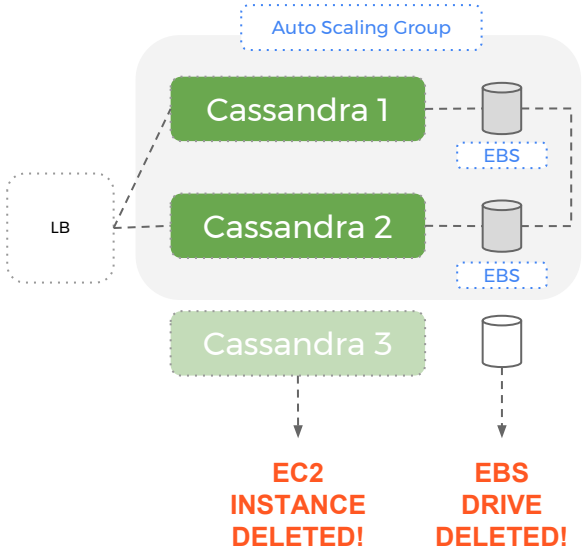
This means that if the EC2 instance is terminated, the EBS drive is deleted - along with the dataset Cassandra it was using.

A new EBS drive will be created but Cassandra will have to send **all** the data over the network to rebuild the dataset on that node.

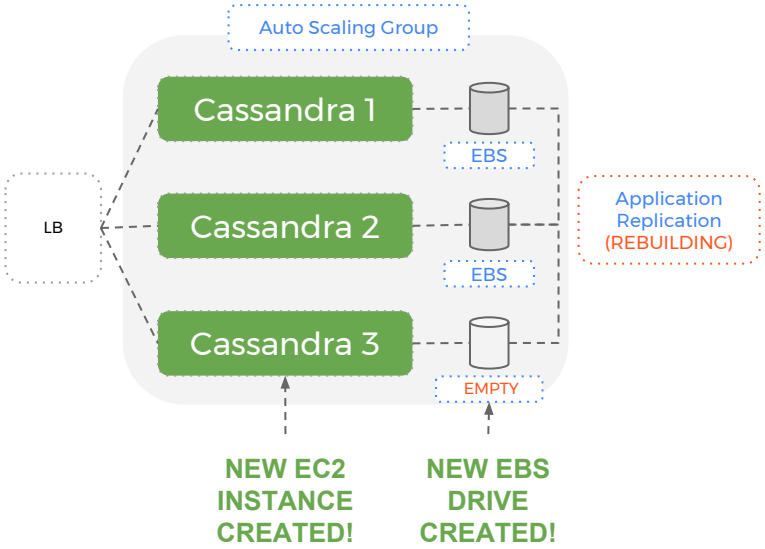
This can take a **long time** if the dataset is large. What if we could just reuse the EBS drive that was attached to the old node? Then most of our dataset is already there when the new node starts up.

We realise that we need to **de-couple compute and storage**.

Node Failure



Cluster Repair



portworx

Portworx:
The Solution

Portworx: The Solution

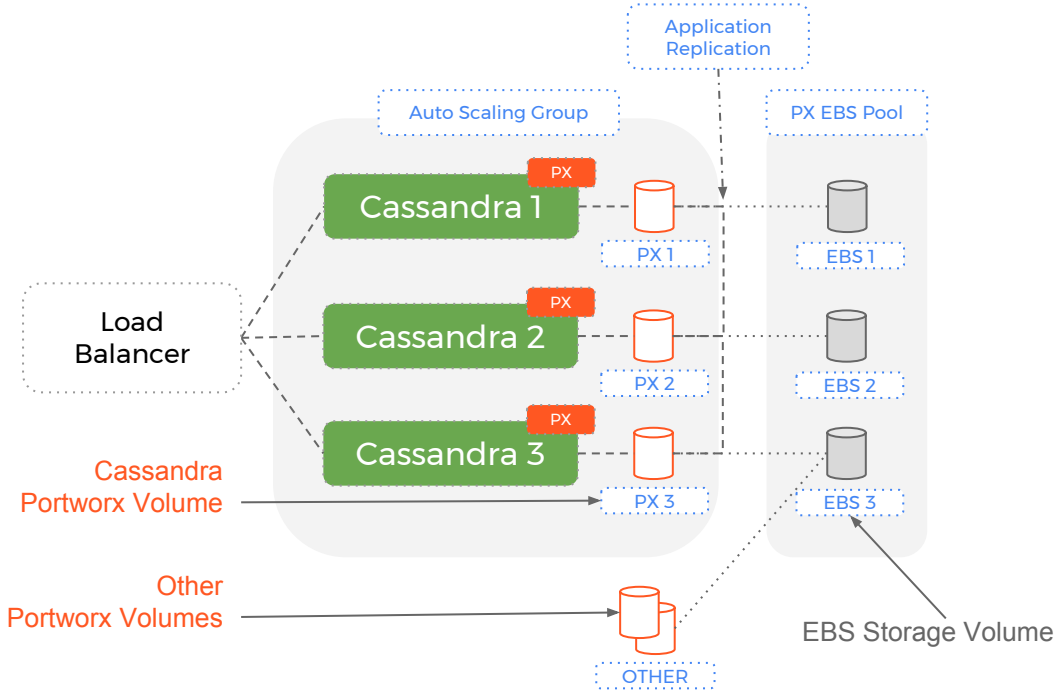
Using Portworx to add a **data services layer** - we can have a level of separation with Auto-Scaling Groups managing EC2 instances (**compute**) and Portworx managing EBS volumes (**storage**).

The key aspect of this solution is that when the Auto Scaling Group terminates an EC2 instance - the EBS volume is **NOT** removed. More importantly, the same EBS volume that was attached to an instance previously, is **re-used** for the next instance.

Let's see what this looks like:

portworx

Portworx EBS Pool



portworx

Portworx: The Solution

This means our design now works because:

- data written by an instance that is terminated is not lost
- Cassandra containers re-use volumes and so already have most of their data
- Rebuilding shards takes significantly less time because only the writes that happened in the last few minutes need to be copied

The reason this works is because Portworx is a **data services layer** that manages your underlying storage (EBS) and leaves the Auto-Scaling Group to manage only the compute (EC2).

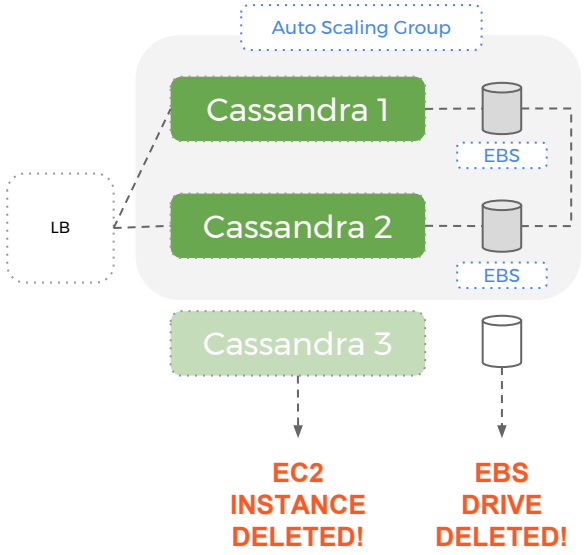
Let's compare how this works in a failure scenario:

portworx

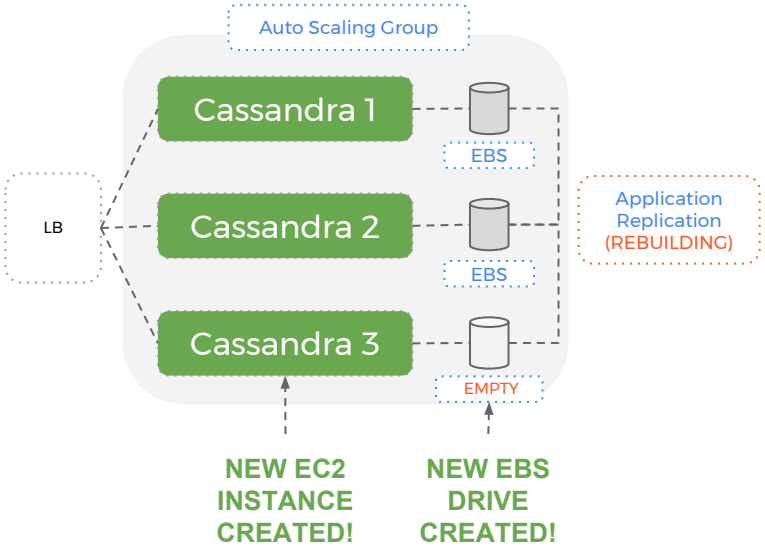
Failover with pure ASGs

1. A single node fails in a 3 node Cassandra ring
2. The ASG creates a new EC2 instance **and** a new EBS volume to attach to it
3. Cassandra starts on the new node and discovers an empty volume and so starts to rebuild from the replica
4. Once the rebuild is complete (some time later) - the cluster is back and healthy

Node Failure



Cluster Repair



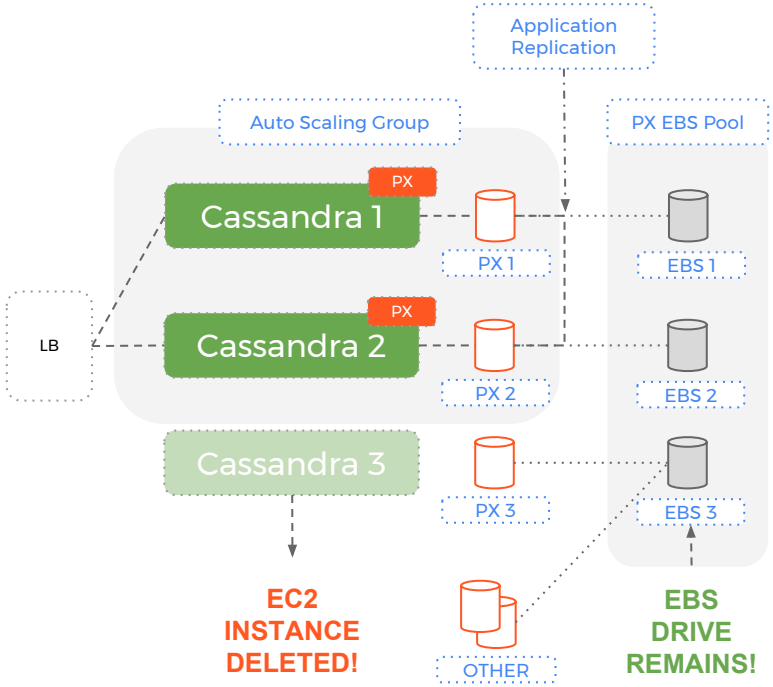
portworx

Auto scaling ASGs plus Portworx

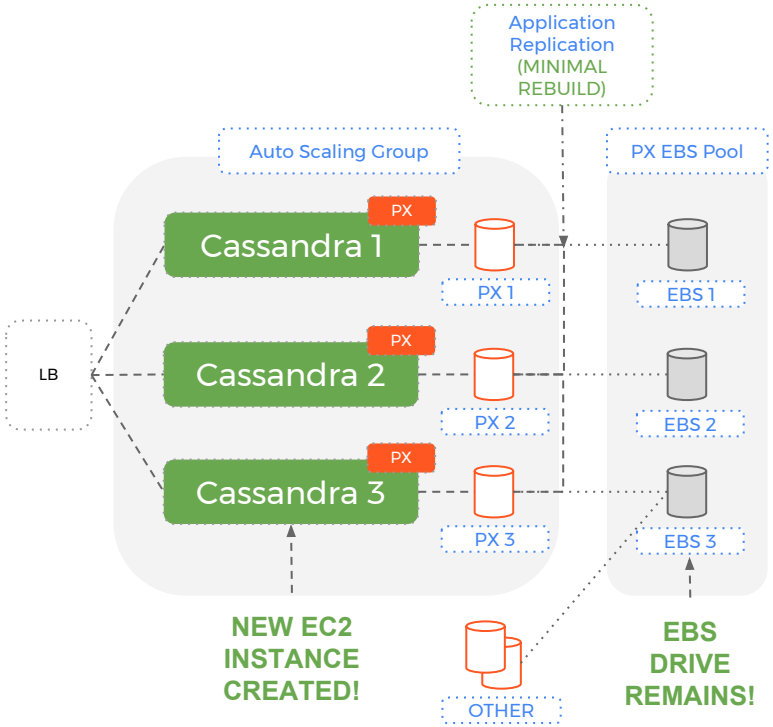
1. A single node fails in a 3 node Cassandra ring
2. The ASG creates a **only** a new EC2 instance - the old EBS volume is **not** deleted
3. Cassandra starts on the new node and discovers a mostly full volume - it starts re-building to catch up with any writes that happened in the last few moments
4. Once the rebuild is complete (significantly less time later) - the cluster is back and healthy

portworx

Node Failure



Cluster Repair



portworx

Comparison

An EBS volume could contain hundreds of Gigabytes. Being able to reuse that existing EBS drive - with dataset intact, means Cassandra takes an order of magnitude less time to rebuild.

This only works because Portworx can **de-couple compute from storage**.

portworx

How it works

Portworx has a clusterid and can use one of three methods to connect to the AWS api:

- Using AWS access credentials
- Using Cloud Init
- Using Instance Privileges

Portworx is now able to create new EBS volumes on demand. As it creates these EBS volumes, it will tag them with identifying values so at any time, it can enumerate the available **pool** of EBS volumes available to an Auto-Scaling Group.

portworx

How it works

When a new instance is added to the group - Portworx does the following:

- check the **pool** to see if there are are candidate EBS volumes that can be used
- if no - then create one using the specification of volumes already in the pool
- in both cases - the EBS volume is associated with the new instance

portworx

How it works

Using this setup - if we have a healthy 3 node Cassandra cluster and one of our nodes dies - whilst the Auto-Scaling Group will **replace** the compute instance, Portworx will **reuse** the storage volume.

portworx

Conclusion

Conclusion

By **de-coupling compute from storage,** we get the immense power of AWS Auto-Scaling Groups to manage compute without worrying that your data will disappear or that your cluster will take hours to actually scale.

To try this out - check out our documentation on AWS Auto Scaling Groups.

portworx

Using EBS with Auto Scaling Groups

Visit the Portworx website to find out more!