At Zalando we've created Nakadi (https://github.com/zalando/nakadi), a distributed event bus that implements a RESTful API abstraction on top of Kafka-like queues. It helps to provide an available, durable, and fault tolerant publish/subscribe messaging system for simple microservices communication.

A Kafka cluster is able to grow to a huge amount of data stored on the disks. Hosting Kafka requires support of instance termination (on purpose or just because the "cloud provider" decided to terminate the instance), which in our case introduces a node with no data: the rebalance of the whole cluster has to be accomplished in order to evenly distribute the data among the nodes, taking hours of data copying. Here, we are going to talk about how we avoided rebalance after node termination in a Kafka cluster hosted on AWS.

In the beginning, at least when I joined, our Kafka cluster had the following configuration:

- 5 Kafka brokers: m3.medium 2TB gp2
- Replication factor 3 and min insync replicas 2
- 3 Zookeeper nodes: m3.medium
- Ingest 250GB per day

Nowadays, the cluster is much bigger:

- 15 Kafka brokers: m4.2xlarge 8TB st1
- Replication factor 3 and min insync replicas 2
- 3 Zookeeper nodes: i3.large
- Ingest 5TB per day and egress 30TB per day

Problem

The above setup results in a number of problems that we are looking to solve, such as:

Loss of instance

AWS is able to terminate or restart your instance without notifying you in advance of the fact. Once it has happened, the broker has lost its data, which means it has to copy the log from the other brokers. This is a pain point, because it takes hours to accomplish.

Changing instance type

The load is growing and at some point in time, the decision is to upgrade the AWS instance type to sustain the load. This could be a major issue in the sense of time as well as availability. It correlates with the first issue, but a different scenario of losing broker data.

Upgrading a Docker image

Zalando has to follow certain compliance guidelines, which is provided by using services like Senza and Taupage. In their turn, they have requirements themselves which is to have immutable Docker images that are not replaceable once the instance is running. To overcome this, one has to relaunch the instance, hence coping a lot of data from other Kafka brokers.

Kafka cluster upgrade

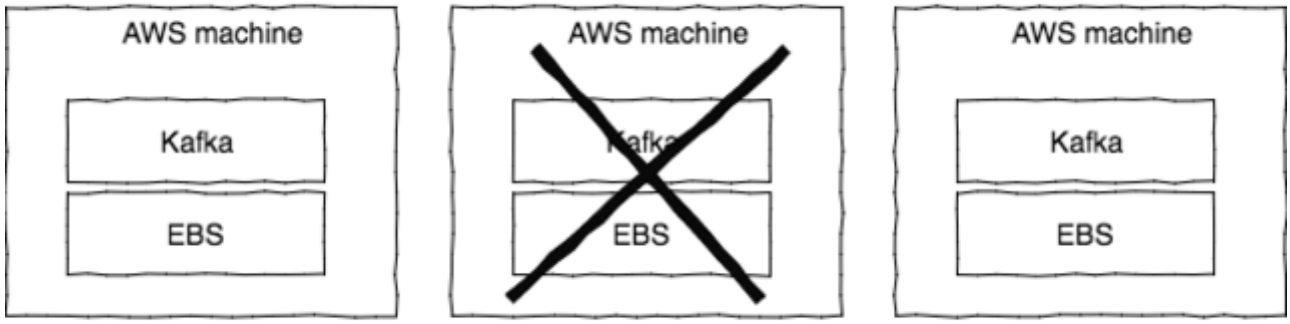
Upgrading your Kafka version (or maybe downgrading it) requires the building of a different image which holds new parameters for downloading a Kafka version. This again requires the termination of the instance involving data copying.

When the cluster is quite small, it is pretty fast to rebalance it, which takes around 10-20 mins. However the bigger the cluster, the longer it takes to rebalance. It has happened that a rebalance of our current cluster takes about 7 hours in the case that one broker is down.

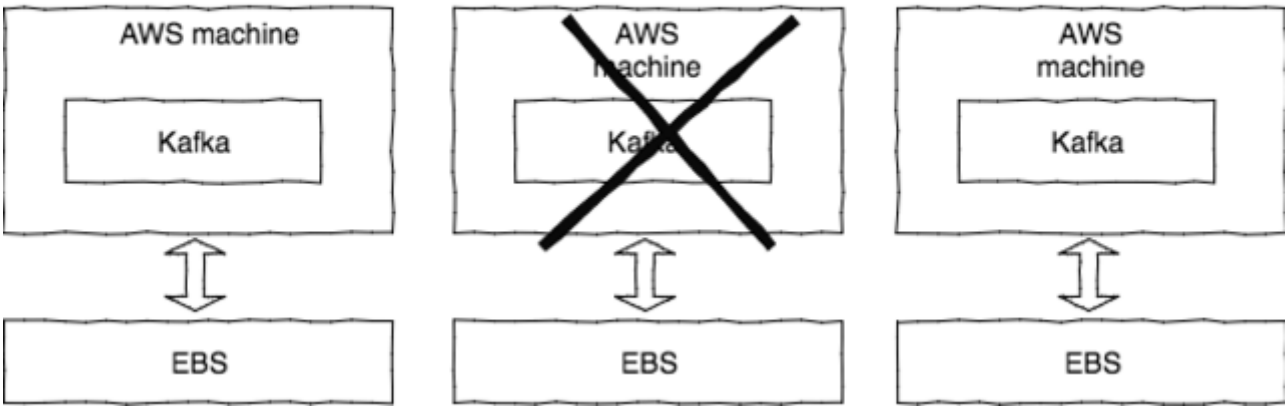
Solution

Our Kafka brokers were already using attached EBS volumes, which is an additional volume, located somewhere in the AWS Data Center. This is connected to the instance via network in order to have durability, availability and more disk space available. The AWS documentation (https://aws.amazon.com/ebs) states: "Amazon Elastic Block Store (Amazon EBS) provides persistent block storage volumes for use with Amazon EC2 instances in the AWS Cloud."

The only tiny issue here is that instance termination would bring the EBS volume down together with the instance, introducing data loss for one of the brokers. The figure below shows how it was organized:



The solution we found was to detach the EBS volume from the machine before terminating the instance and attach it to the new running instance. There is one small detail here: it is better to terminate Kafka gracefully, in order to decrease the startup time. In case you terminated Kafka in a “dirty” way without stopping, it would rebuild the log index from the start, requiring a lot of time and depending on how much data is stored on the broker.



In the diagram above we see that the instance termination does not touch any EBS volume, because it was safely detached from the instance.

Losing a broker without detaching EBS (terminating it together with the instance) introduces under-replicated partitions on other brokers, which holds the same partitions. To recover from that state, the rebalance takes around around 6-7 hours. If during the rebalance other brokers go down, which have the same partitions, it will provoke offline partitions and it will not be possible to publish to them anymore. It is better not to lose any other broker.

Reattachment of EBS volumes is possible to accomplish using the AWS Console by clicking buttons there, but to be honest I have never done it myself. Our team went about automating it with Python scripts and the Boto 3 library (https://boto3.readthedocs.io/en/latest/) from AWS.

The scripts are able to do the following:

- Create a broker with attached EBS volume
- Create a broker and attach an existing EBS volume
- Terminate a broker, detaching the EBS volume beforehand
- Upgrade a Docker image reusing the same EBS volume

Kafka instance control scripts (https://github.com/zalando-nakadi/bubuku/tree/master/instance_control) can be found in our GitHub account, where the usage is described. Basically, these are one line commands which consume configuration (https://github.com/zalando-nakadi/bubuku/blob/master/instance_control/bubuku-1.json) for the cluster in order to not pass in the script parameters. Remember, we use Senza and Taupage, so the scripts are a bit Zalando specific, but can be changed quite quickly with very little effort.

However, it's important to note that the instance could have a Kernel panic or some hardware issues while running the broker. AWS Auto Recovery helps to address this kind of issue. In simple terms, it is a feature of the EC2 instance to be able to recover after network connectivity, hardware or software failure. What does recovery mean here? The instance will be rebooted after failure to preserve a lot of parameters from an impaired instance, among that being EBS volume attachments. This is exactly what we need!

In order to turn it on, just create CloudWatch Alarm for StatusCheckFailed_System and you are all set. The next time the instance has a failure scenario it will be rebooted, preserving the attached EBS volume, which helps to avoid data copying.

Impact

Our team no longer worries about losing a Kafka broker, as it can be recovered in a number of minutes without copying data and wasting money on traffic. It only takes 2 hours to upgrade 15 nodes of a Kafka cluster and it just so happens that it is 42x faster than our previous approach.

In the future, we plan to add this functionality directly to our Kafka supervisor (https://github.com/zalando-nakadi/bubuku), which will allow us to completely automate our Kafka cluster upgrades and failure scenarios.

Have any feedback or questions? Find me on Twitter at @a_dyachkov (https://twitter.com/a_dyachkov).

Andrey Dyachkov
Software Engineer

Zalando (https://jobs.zalando.com/tech/blog/?tags=Zalando&gh_src=4n3gxh1) · Zalando Tech (https://jobs.zalando.com/tech/blog/?tags=Zalando+Tech&gh_src=4n3gxh1) · Microservices (https://jobs.zalando.com/tech/blog/?tags=Microservices&gh_src=4n3gxh1) · Apache Kafka (https://jobs.zalando.com/tech/blog/?tags=Apache+Kafka&gh_src=4n3gxh1) · AWS (https://jobs.zalando.com/tech/blog/?tags=AWS&gh_src=4n3gxh1) · Nakadi (https://jobs.zalando.com/tech/blog/?tags=Nakadi&gh_src=4n3gxh1)
