MapR Demonstration: an Edge-to-Cloud Data Fabric

featuring Cross-Site Replication of a Microservices Data Pipeline

Includes networking configurations for multiple deployment environments:
Edge-to-Sandbox, Sandbox-to-Cloud, and Multi-Cloud.

Jonathan Janos
MapR Federal

Introduction: Evolution of the Demo

This demonstration was originally built as a booth demo for a technical tradeshow, showcasing MapR's suitability as a platform for systems built using a microservices-based architecture. The front end shows imagery data being pulled down from NASA and subsequently processed by multiple stages of a data processing pipeline. The different stages of the pipeline are interconnected using pub/sub messaging (MapR-ES). The files are stored in MapR-XD and the imagery metadata is stored in MapR-DB, showing the benefits of an integrated data platform. The agility afforded by the microservices approach is illustrated within the demo by quickly deploying a new service to classify the images (using AWS Rekognition) and then deploying an improved version of that service, which runs in parallel to the first.

Feedback from the conference was that it would have been great to combine that capability with MapR Edge. So, three Intel NUCs were procured and packaged inside a ruggedized Pelican Case, hosting a

MapR Edge cluster. A second version of the front end was created for the Edge, representing a soldier's view in the field, and cross-cluster replication became the central aspect of the demo: streams replication, file mirroring, and resiliency in the context of unreliable networks.

That quickly led to "Edge-to-Cloud" and multi-cloud discussions and messaging, and expansions of the demo to include these capabilities. Since the two versions of the demo, now referred to as the "HQ Edition" and the "Edge Edition", could be deployed anywhere, what remained to be done was a non-trivial exercise in networking. That body of knowledge has been captured here to support future efforts. With this segment complete, the Edge cluster is no longer a requirement: you can set up the two MapR clusters anywhere you wish.

Currently, the demo supports 3 clusters, running concurrently:

- An "HQ" cluster, running a data processing pipeline and pushing new information out to the Tactical Edge.
- A MapR Edge cluster.
- A cluster running in the cloud, which (according to the demo storyline) is aggregating all information requests gathered from all users across the world, for the purpose of using this data to build recommendation engines (not currently implemented).

Demo Script I - Deploying a Global Data Fabric

Note: You can watch the accompanying video for the demo storyline and delivery. The video is available at the following locations:

- https://mapr.com/resources/videos/demos/
- https://www.youtube.com/user/maprtech/videos

Intro

The goal of this demo is to show how quickly and easily you can set up a worldwide data fabric with MapR. Additionally, it shows the kinds of applications that can build on top of MapR, and what kinds of missions and scenarios can be supported.

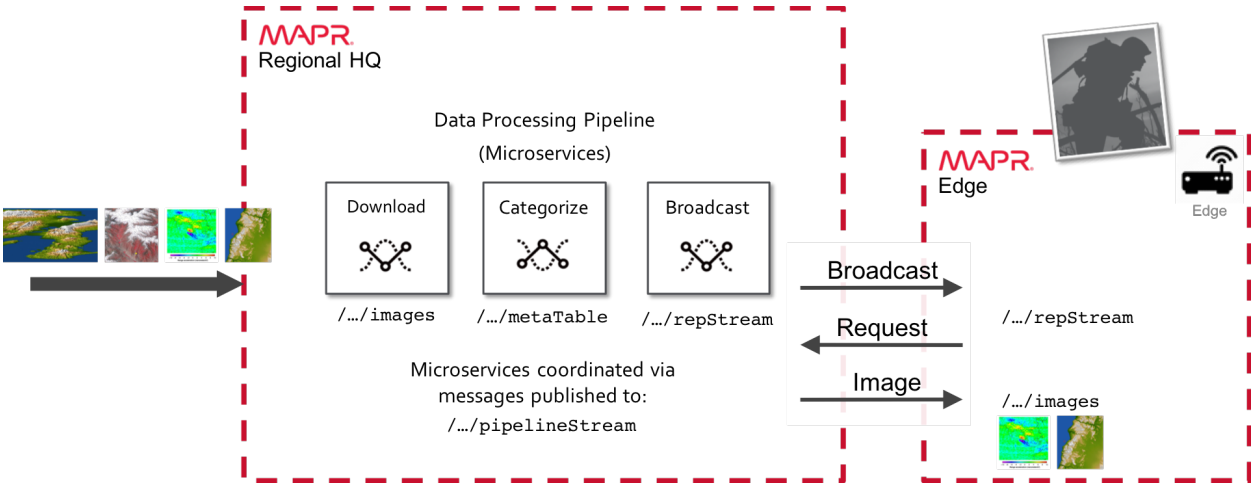
A 3-node MapR cluster has been packaged up into a portable unit that can be carried out into the field. This is what we are referring to when we talk about MapR Edge. At the outset, this system is receiving no data.

A second MapR cluster represents the "HQ Cluster", or the "Data Center Cluster" (this is a MapR Sandbox running on a laptop). At the outset, this system has live data feeds coming into it, which are

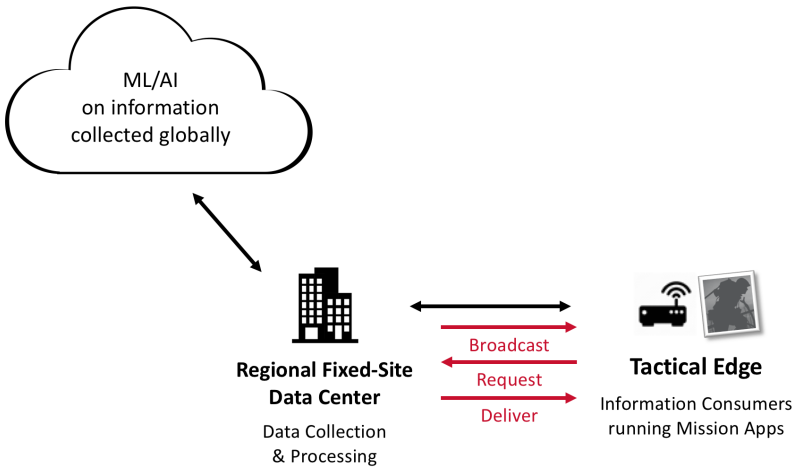
being processed by multiple microservices (sidebar: MapR is a great platform for microservices-based architectures). Although data is coming into the cluster, no data is being replicated anywhere external. That is, replication has not been established.

Although it's certainly possible in this demonstration to replicate all data from HQ to the Edge, the scenario being demonstrated is intended to demonstrate an approach for operating within limited bandwidth environments. In this scenario, as the data comes into the HQ cluster, the only thing replicated to the field is an announcement that a new information asset is available. The user at the Edge receives these announcements and indicates which assets they would like to download. That request is replicated back to the HQ cluster, which then ships the full asset to the Edge. This, of course, is just one of many possible approaches to the challenge of reducing the amount of data that must travel over limited bandwidth. A logical improvement would be the introduction of a rule set, governing which assets are initially broadcast (based on relevance), or even sent immediately without requiring any explicit request, as well as advanced analytics that insert more intelligence into the broadcast service.

The overall process flow is pictured below.



This has now been updated with a third cluster running in the cloud, to function as a data science cluster, aggregating information gathered from all over the world. So the high-level picture now looks as follows (you can watch this complete scenario in the video):



Initial Setup

Initially, both clusters should be up and running. There is a web-based application for each, providing a view of the data entering the system. The sandbox running on your laptop requires internet connectivity, to pull down imagery. The Edge cluster does not.

At the Edge

On the Edge, the "Edge Dashboard" app shows only that it is actively waiting for requests. No data is entering the system.

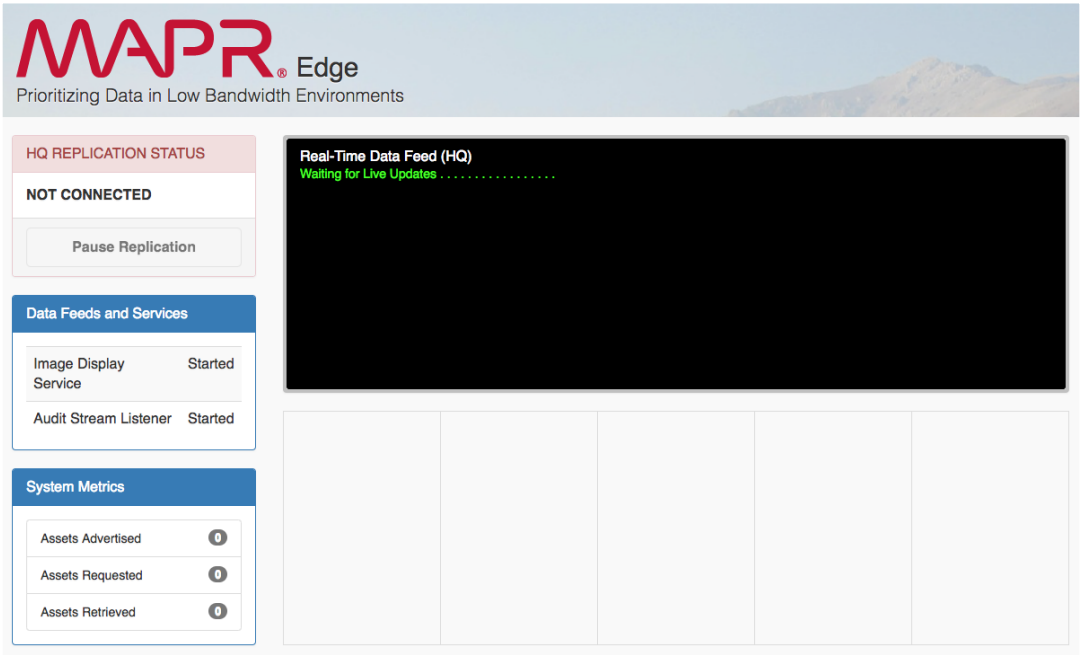


Figure 1 - Edge Dashboard, Initial Screen

Note that there are two services that have been started up automatically:

- The Audit Stream Listener
- The Image Display Service.

A third service has not yet been started: the Upstream Communication Service. This service simply listens for announcements on the ASSET_BROADCAST stream and sends them to the dashboard for display. This service can't be started yet, however, because the stream itself doesn't yet exist (it will be set up automatically by MapR when replication is configured). So instead, the Audit Stream Listener monitors the audit stream (another MapR feature worthy of discussion), listening for the replication to be established, at which points it launches the Upstream Communication Service. The Image Display Service, which is launched immediately on startup, simply monitors a directory for the arrival of new imagery. This won't find anything until the Mirror Volume is created (part of the demo).

At HQ

Meanwhile, on the HQ Cluster, the "Pipeline" app shows imagery data being ingested and processed by a data pipeline, implemented as a collection of microservices.

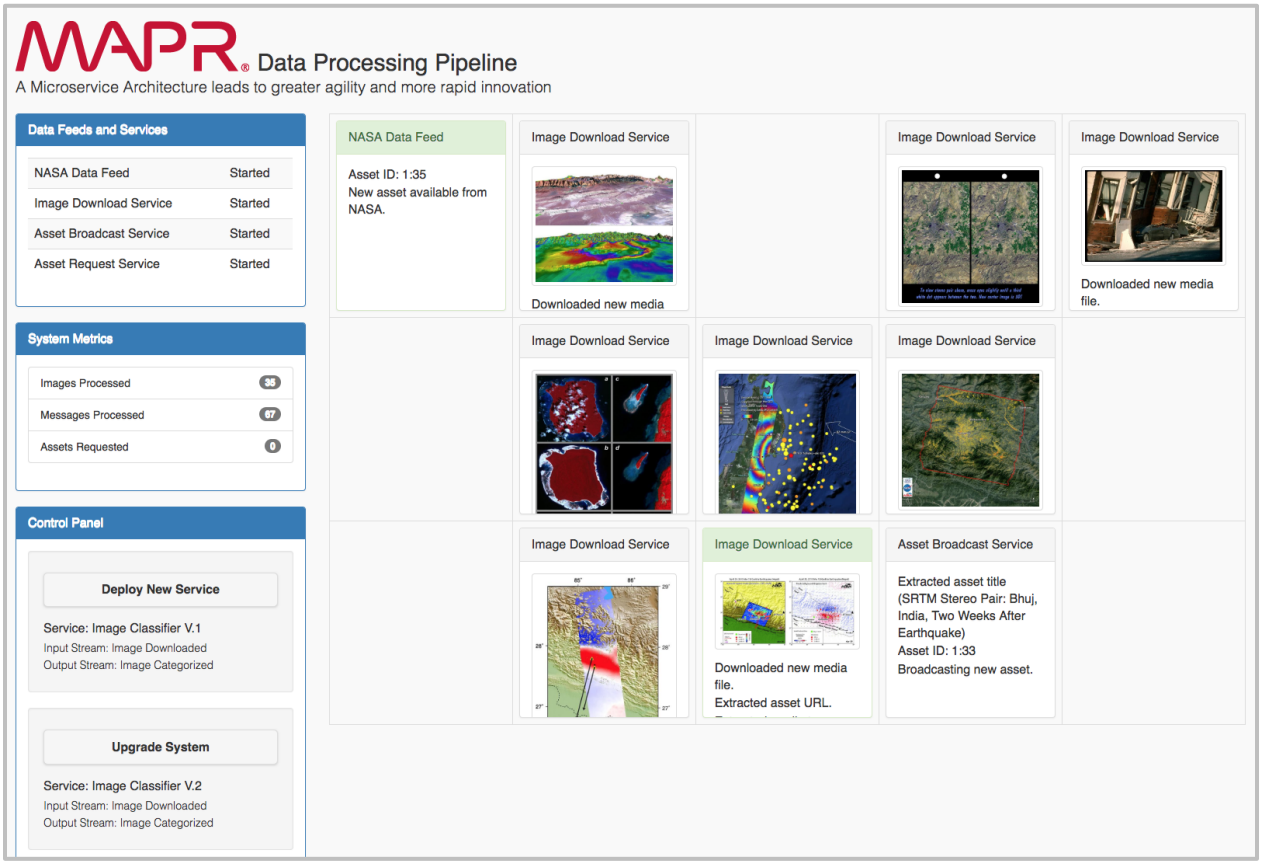


Figure 2 - HQ Dashboard, Initial Screen

The Pipeline App automatically starts with 3 services running:

- The NASA Data Feed
- The Image Download Service
- The Asset Broadcast Service

- The Asset Request Service

The NASA Data Feed simulates a real-time feed by parsing a JSON data file[1], creating a database record for each entry, and publishing the event (i.e. the arrival of the asset) to a stream. The Image Download Service listens for this event, retrieves the image URL from the database record, and downloads the image to MapR-FS.

The Asset Broadcast Service is new, and does the following:

- Fetches the asset metadata from the database.
- Extracts the image title.
- Publishes the event to the ASSET_BROADCAST stream, which will be replicated to the Edge. The stream to be replicated is: /apps/pipeline/data/replicatedStream:ASSET_BROADCAST.

Although the ABS is launched automatically and is pushing data to the ASSET_BROADCAST stream, that stream is not yet being replicated anywhere. As a result, the data never leaves the cluster, and nothing shows up at the Edge. Similarly, the Asset Request Service is actively listening for requests on that same stream, but since nothing is replicated, no requests are coming in. Note on the left of the Dashboard, under "System Metrics", the number of Assets Requested is listed as zero.

In the Cloud

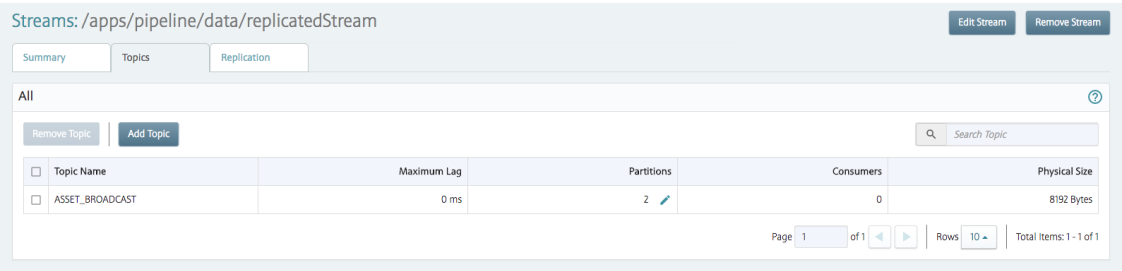
A third cluster can be deployed in the cloud, completing the Edge-to-Cloud topology. The cloud deployment is much simpler, consisting of a single stream (/apps/pipeline/data/cloudStream). The HQ cluster receives data requests from the Edge, and forwards those requests to the cloud, where they will be used (conceptually) for building advanced models and recommendation engines. A simple listener service catches those requests and writes them to an output file.

Demonstrate Stream Replication

Use the MCS to demonstrate the ease with which active/active, or bi-directional, stream replication can be established. Perform the following steps on the HQ cluster, via the MCS:

- From the navigation menu, select Data > Streams.
- Enter the path to the ASSET_BROADCAST stream, and click *Go*:

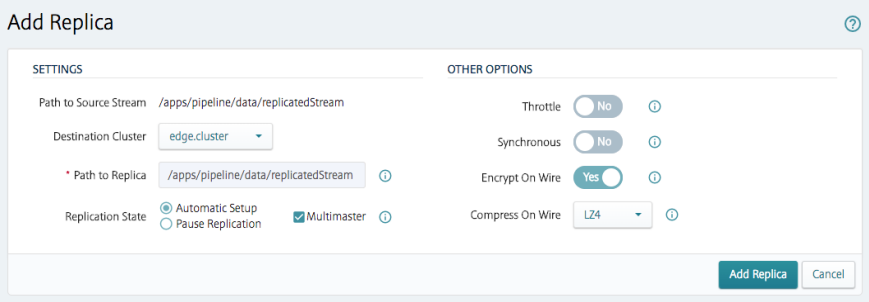
  /apps/pipeline/data/replicatedStream
- If you click the Topics tab, you will see the ASSET_BROADCAST topic:



- Select the Replication tab, and click *Add Replica*. Specify the following:

[1] The JSON file was created by utilizing a NASA API, which accepts some search query terms and generates a list of search hits, which is returned as a JSON file.

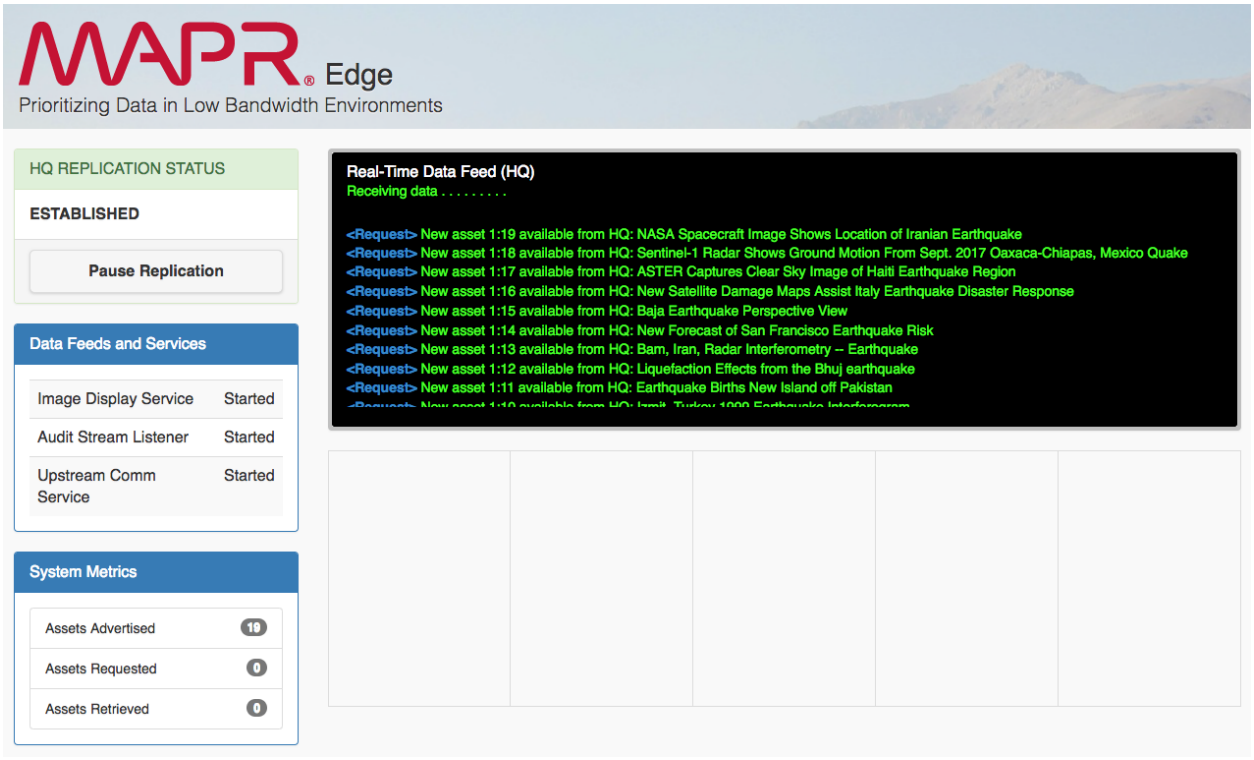
- Path to Replica: /apps/pipeline/data/replicatedStream
- Automatic Setup
- Multimaster (checked)
- Defaults elsewhere



The stream should be created successfully. Note also the Errors columns – I've seen it be "created successfully", where the Errors column indicates that it hasn't. This is usually because the streams file already exists on the remote (Edge) cluster. This can happen when the replica is removed via the MCS (the file remains intact on the remote MapR cluster).

At this point, the Edge Dashboard should show the following:

- In "Data Feeds and Services", the Audit Stream Listener should show "Replication Established".
- In "Data Feeds and Services", the Upstream Comm Service should show "Started".
- The Real-Time Data Feed (HQ) panel should display a constantly updating list of new assets that are now available for download.



Click on any of the request links. With each click, the count of "Assets Requested" should increment by one. Each request should be shipped back to the HQ cluster, demonstrating bi-directional replication. On the HQ Dashboard, the number of Assets Requested should keep in synch with the number of assets requested at the Edge (indicating message receipt).

This could also be demonstrated from the command line. Managing Stream Replication is covered here in the docs. To configure this same stream replication via the command line (or use the included script, */microservices-dashboard/scripts/hq/create-edge-replica.sh*):

```
maprcli stream replica autosetup
  -path /mapr/dc1.enterprise.org/apps/pipeline/data/replicatedStream
  -replica /mapr/edge1.enterprise.org/apps/pipeline/data/replicatedStream
  -multimaster true

# To verify:
maprcli stream replica list -path /apps/pipeline/data/replicatedStream -json
```

Demonstrate File Mirroring

Once requests have been received at HQ, the next step is to deliver those assets to the Edge.

Asset metadata, stored in the database, is not transferred in this demo. It can be explained that active/active table replication is configured in the exact same fashion as stream replication.

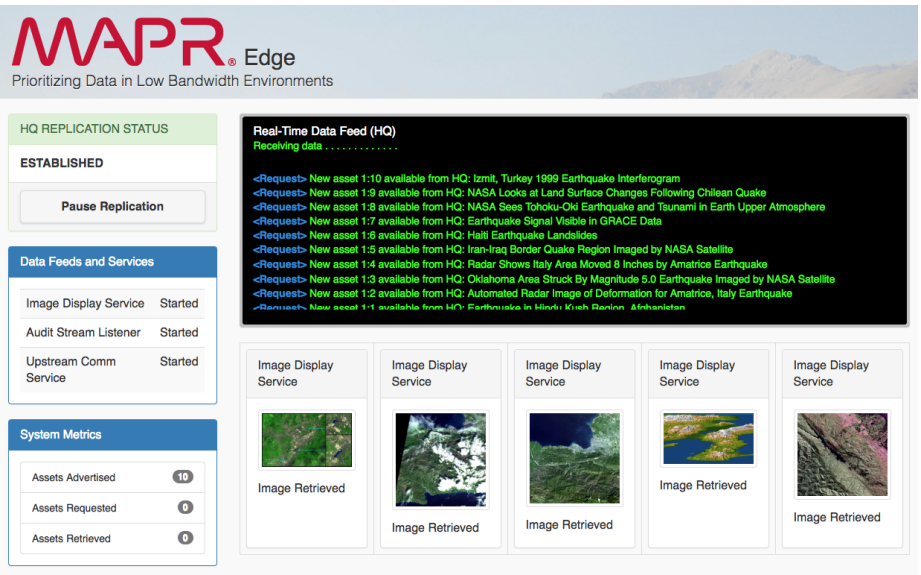
To move the images requires Mirroring. (To Do: vet whether smaller images can be transferred via Streams). The *files-missionX* volume should have been created on the HQ cluster by the install script. Verify that some files have been placed there by the Asset Request Service - if there aren't any files, then you won't see anything being mirrored. Then, perform the following steps using the MCS on the ***Edge Cluster***:[2]:

- Navigate to Data > Mirrors.
- Click *Create Volume*. Specify the following, click *Create Volume* again:
    - Volume Type: Mirror Volume.
    - Volume Name: files-missionX
    - Source Cluster Name: demo.mapr.com
    - Source Volume Name: files-missionX
    - Mount Path: /apps/pipeline/data/files-missionX
    - Topology: /data
    - Audit: Yes
- The volume should be created successfully. Click *Select Action*, and from the drop-down list select *Start Mirroring*.

Go back to the Edge Dashboard. After a brief delay, the images should show up within a grid in the bottom-right corner of the dashboard.

---

[2] Although this is extremely simple to set up, there's still an opportunity here to screw this up during a live demo. To avoid this risk, you can opt to use the *createMirror.sh* script on the Edge cluster, which creates the Mirror volume and initiates the mirror. An effective approach is to walk through the steps in the MCS, then explain that the MCS is built upon CLI and REST API equivalents. The script then becomes a demonstration of that design.



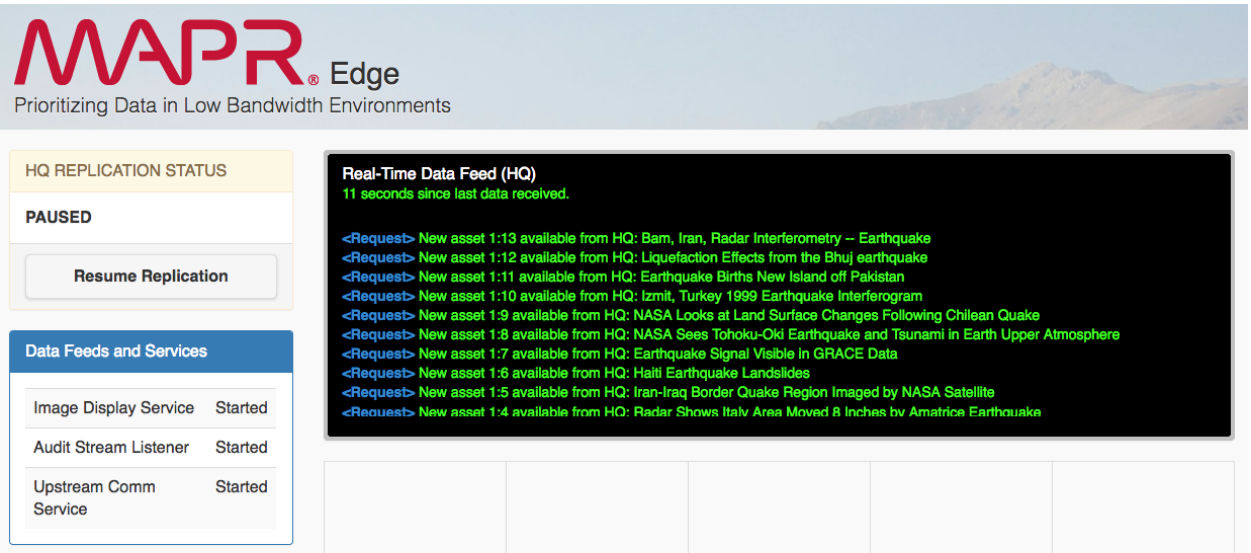
Note that these will disappear after a time (20 seconds by default), so don't miss it! This can be adjusted using the time sliders in the dashboard Control Panel ("Keep Image Tiles After Completion"). Since currently this image collection is fixed, you can't just request more assets to see it again. You can't even remove the mirror volume and recreate it, because the Image Display Service remembers which images have been displayed and won't display the same image twice. You would have to add new images to the volume and manually initiate a mirror operation (or wait for the next scheduled operation).

As with stream replication, mirrors can also be demonstrated via the command line. Example maprcli commands can be found here in the docs.

Pausing Replication

Replication can be paused to demonstrate two points:

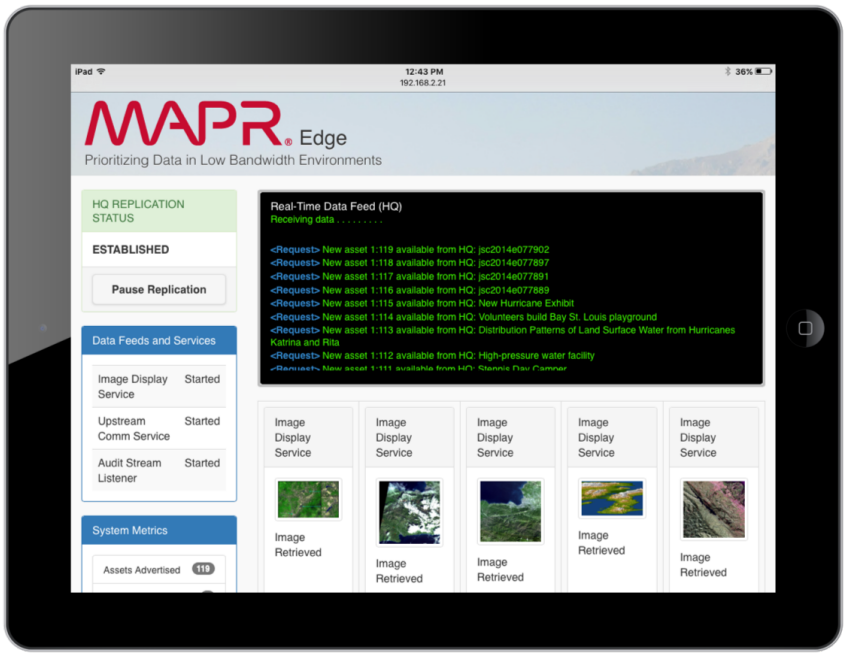
1. Replication can be paused on an individual stream basis. In limited bandwidth environments, this might be done to ensure that the available bandwidth is allocated to higher priority streams.
2. Administration tasks are accessible via REST API, allowing them to be embedded in custom application interfaces.

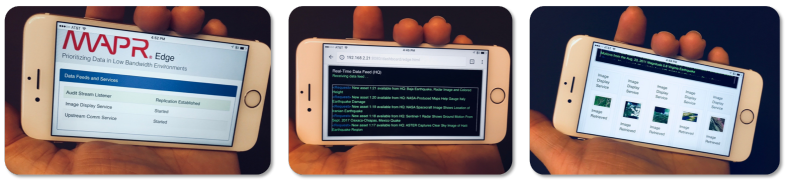


Mobile Devices

The Edge Dashboard can also be demonstrated via a mobile device, which is very useful for demonstrating intermittent connections. With the mobile device connected to the Edge Dashboard, disconnected the Edge Cluster from the HQ cluster (i.e. physically disconnect the cable) and see that updates to the mobile device cease. Reconnect the cable, and observe that they resume. Note, however, that whatever client you used to *launch* the Dashboard app (i.e. using the *runDashboard.sh* script) cannot lose connectivity!

The Edge cluster is connected using a wireless router. A mobile device can join the wireless network (EDGENET-5G or EDGENET) and then access the Edge Dashboard using any browser app. Note that you must specify the IP address for edge1 – iPads and such do not have a local /etc/hosts file to modify, and a DNS server was not installed on the Edge cluster.





Data Science in the Cloud

Requests for information can be forwarded from the HQ cluster to the Cloud cluster. The command to do so is included in a script, */home/mapr/microservices-dashboard/scripts/hq/create-stream-replica.sh*:

```
maprcli stream replica autosetup
  -path /mapr/dc1.enterprise.org/apps/pipeline/data/cloudStream
  -replica /mapr/edge1.enterprise.org/apps/pipeline/data/cloudStream
  -useexistingreplica true
```

Note the use of the *useexistingreplica* parameter (not available from the MCS), indicating that cloudStream should already exist in the cloud cluster. With the replica already present, a listener can subscribe to the stream and startup and immediately begin writing output to the data science file. The new messages can therefore be shown as they arrive using a simple 'tail -f' command on the output file.

Command Line Demo Suggestions

Depending on the audience, there are several things that can be shown from the command line:

Shows converged data – files, tables, and streams – as managed by MapR:

```
ls /apps/pipeline/data
```

Shows the first record of image metadata. You can cut and paste this data into a web service like jsonprettyprint.com:

```
mapr dbshell
maprdb mapr:> find /apps/pipeline/data/imagesTable --limit 1
```

Pulls the specified record back (make sure it exists):

```
maprdb mapr:> findbyid /apps/pipeline/data/imagesTable --id 1:200
```

Examine a stream:

```
mapr streamanalyzer -path /apps/pipeline/data/pipelineStream
```

Cleanup

In tradeshow-type settings, it's helpful to have a demo reset function that allows the demo to be re-run for a new audience. From the Edge Dashboard, click the "Reset Demo" button to undo all the replication that was configured during the demo. Status information is returned within a standard dashboard tile. The Reset buttons implements a combination of both client-side and server-side actions, including the following:

- Removes the mirror volume (on the Edge Cluster).
- Removes the replicated stream on the Edge cluster, and removes it as a downstream replica and an upstream source of the same stream on the HQ cluster.
- Switches the replication status to "Not Connected".
- Resets all related user-interface elements on the Edge Dashboard.

Consult the code (DemoHelper.java) to see the full list of actions. These actions are implemented as a combination of MapR REST API calls and Java commands.

If you're using a third cluster in the cloud, note that the Demo Reset function does *not* undo replication between the HQ and cloud clusters. You can either continue giving demonstrations with the cloud replication permanently in place, or you can stop and restart the demo at each location (i.e. quit and re-run the *runDashboard.sh* script).

Demo Script II - Microservices on MapR

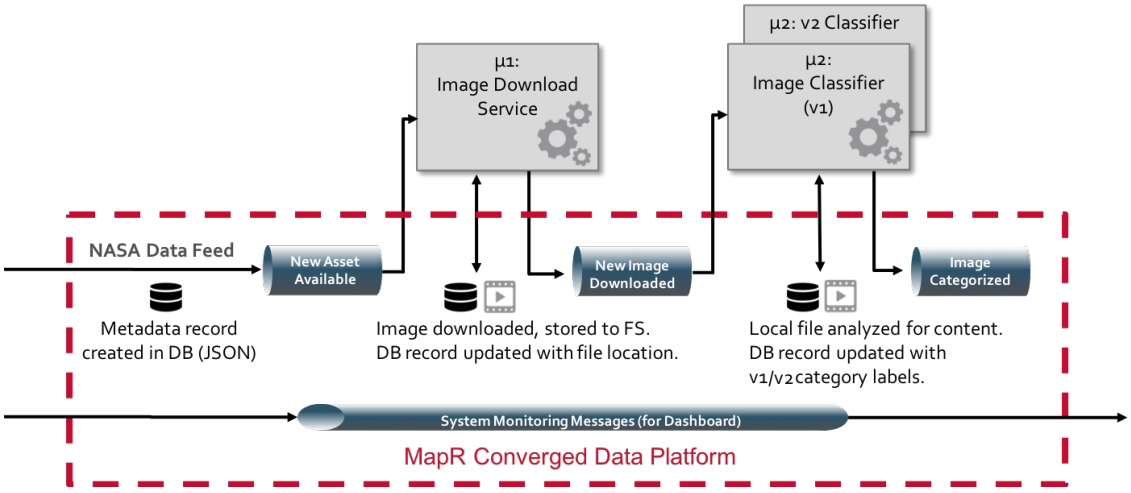
Before the Edge cluster was built, this demo began as a way to illustrate the ease and agility with which data pipelines can be constructed and enhanced using a microservices-based architecture. New and improved services can be quickly added to the pipeline without any alteration to existing processes.

This portion of the demo leverages the AWS Rekognition service. A valid AWS access key and secret key, authorized to use the Rekognition service, needs to be specified in the config.properties file. Internet connectivity is always required for this portion of the demo.

To deploy a new service, click the "Deploy New Service" button in the HQ Dashboard. This introduces a new classification service (implemented using AWS Rekognition) into the pipeline, which fires whenever a new image is downloaded. To show how you can quickly iterate and deploy new system enhancements without any risk to existing production pipelines, click the "Upgrade System" button. This will deploy a second version of the classifier service, which was enhanced to return a collection of image labels, rather than just a single label. Both versions of the service can be run in parallel – MapR Streams ensures that new data is fed to both versions.

To revert back to the original state, click the "Decommission Service" buttons.

This demo flow is illustrated below:



Network Design

Since MapR works the same in every environment, demonstrating cross-site replication really comes down to an exercise in networking. Which may sound trivial, but actually can be quite time consuming if you're not a networking ace. It is not MapR's role to design a customer's networking infrastructure, and given the many concerns around security, performance characteristics, and cost, *customers must be strongly advised to conduct their own due diligence and develop their own networking solutions*. But for demonstration purposes, there exists a reasonable expectation from customers and prospects that MapR can come to the table with a capability that works. The following section consolidates a lot of networking information into one place, allowing you to deploy MapR clusters anywhere you need to and establish replication between them. The scenarios that have been verified to work include Edge-to-Sandbox, Sandbox-to-Cloud, and Cloud-to-Cloud (across AWS and Azure).

Edge-to-Sandbox

This original configuration requires two clusters, one representing the "HQ" view, or the "data center" view, and the other representing an Edge location. The "HQ" cluster is a MapR sandbox, running on a MacBook. The Edge cluster was built using 3 Intel NUCs, packaged inside a Pelican case. The first step in setting up replication across two clusters is the network configuration – each cluster has to be able to see the other. Additionally, the sandbox requires internet access, in order to download imagery data as part of the demo.

These requirements can be met using the following setup:

- A single-node MapR Sandbox is running on a MacBook.
- The Sandbox comes pre-configured with a NAT adapter and port forwarding enabled. This enables you to access the Sandbox from your MacBook (e.g. http://localhost:8443), and the Sandbox can access the internet using the MacBook's internet connection.
- With just NAT, however, it is not possible for the Edge cluster to see the MapR virtual machine. The solution is to add a second *Bridged* adapter to the sandbox. This bridged adapter is configured to use an IP address on the same subnet as the Edge nodes.
- The Edge nodes are configured with static IP addresses on this common subnet.
- The Edge cluster is connected by an Ethernet cable to a Thunderbolt Ethernet adapter on the MacBook. This port must also be configured with a static IP address on the same subnet.

With this setup, you can use *ping* or *ssh* from the Sandbox all the way through to each of the Edge nodes, and vice versa. One challenge is that the Edge nodes cannot access the internet, which is only a problem for the installation of MapR software (internet access from the Edge cluster is not required for the demo itself). To achieve this, the NUCs were initially configured to connect to the local WiFi network while MapR was installed. After installation, this WiFi connection was dropped (see the section on "Building an Edge Cluster" for the full details).

The network diagram is pictured below. Refer to the setup instructions for both the sandbox and the Edge cluster for full details on how to configure this networking.

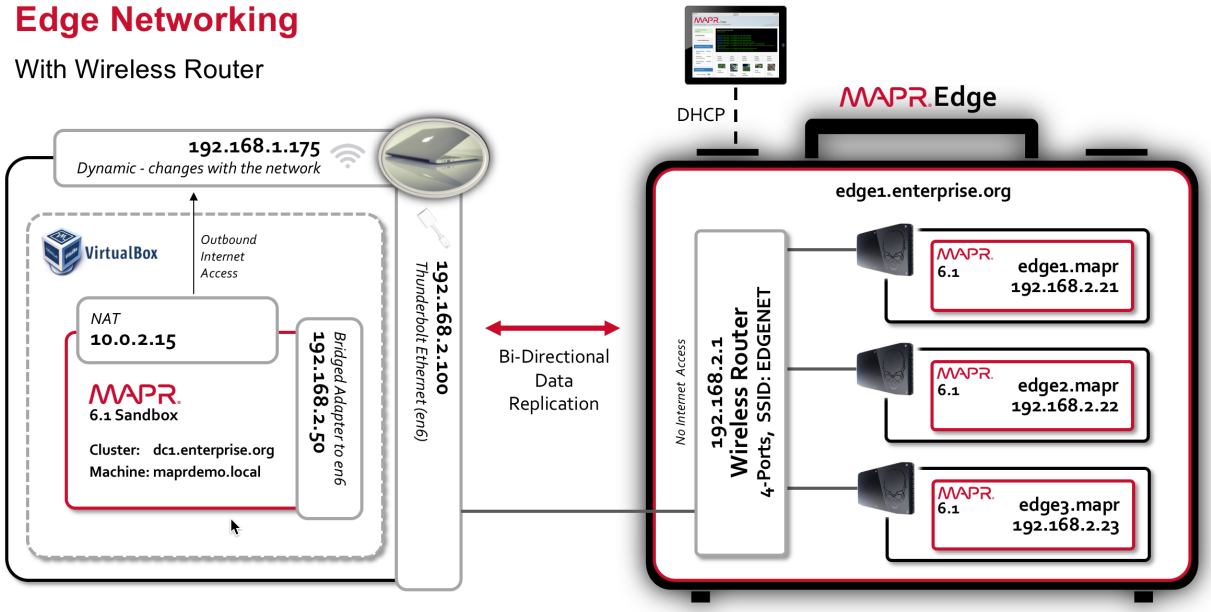


Figure 3 - Edge Networking

Sandbox-to-Cloud

After demonstrating an Edge system replicating data to your MacBook (representing a "fixed site", or an on-premise data center), the concept of a complete Cloud-to-Edge data fabric comes up pretty quickly. And since MapR functions identically across all of these deployment environments (sort of the whole point), this once again becomes a networking exercise.

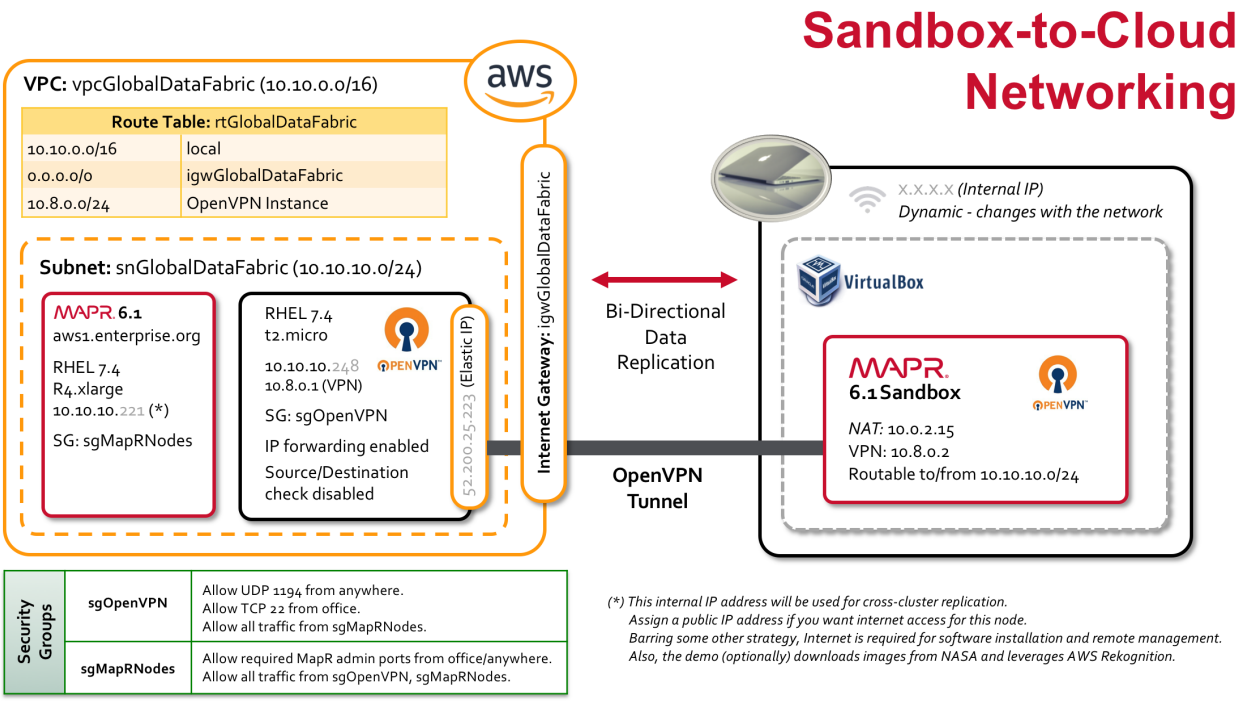
One approach that works is to configure an OpenVPN server on an Amazon EC2 instance and an OpenVPN client *on the MapR Sandbox*. That's worth emphasizing: it's not installed on your MacBook, it's installed on the sandbox. When you do that, the sandbox will be able to communicate with the *internal IP addresses* in your Amazon VPC, and your MapR cluster in AWS will be able to reach the VPN subnet configured on your sandbox.

Resources

This procedures was based largely on this tutorial, "How to Set Up and Configure an OpenVPN Server on CentOS 7". For firewall settings and route tables, it's necessary to switch to an AWS-specific guide, such as the one found here ("Running a Free VPN Server on AWS"). OpenVPN also has a home page and a How To page.

Network Diagram

The environment that will be configured is illustrated below.



AWS Setup

1. Create a new VPC:
   - **Name:** vpcGlobalDataFabric
   - **IPv4 CIDR Block:** 10.10.0.0/16

2. Create a new Internet Gateway (Name: "igwGlobalDataFabric") and attach it to the VPC.

3. Create a new Subnet:
   - **Name:** snGlobalDataFabric
   - **VPC:** vpcGlobalDataFabric
   - **AZ:** <your AZ>
   - **IPv4 CIDR Block:** 10.10.10.0/24

4. Edit the default Route Table for the VPC. When you create a VPC, a route table is created automatically with a single entry mapping the VPC CIDR to local. Change the name of this route table from <nothing> to "rtGlobalDataFabric". Then, add an additional route to the Internet through the Internet Gateway (**Destination:** 0.0.0.0/0, **Target:** igwGlobalDataFabric)

5. Create the necessary Security Groups. There's a lot of flexibility here, depending on what you want to do. It's generally a good idea to have two security groups, one for the OpenVPN server and another for your MapR nodes. Those Security Groups might look like this:

sgOpenVPN:

- Name: sgOpenVPN
- Description: Access to OpenVPN Server
- VPC: vpcGlobalDataFabric
- Inbound Rules:

- TCP 22 (ssh): Source = <your office IP>
- UDP 1194: Source Anywhere (or limit to specific IP addresses).
  This is the only port required for the VPN connection.
- All Traffic, Source: sgMapRNodes (note this group has to first be created)

sgMapRNodes:

- Name: sgMapRNodes
- Description: Access to MapR Nodes
- VPC: vpcGlobalDataFabric
- Inbound Rules:
  - All Traffic, Source: sgMapRNodes
  - All Traffic, Source: sgOpenVPN (note this group has to first be created)
  - Any other ports you want to access (e.g. 22, 9443, 8443, etc.). This assumes your MapR nodes have public IP addresses and can be connected to directly. This is not necessary - replication will be configured using the internal IP addresses and will go through the VPN server. But certain tasks are much easier with an internet connection (installing MapR, for example).

6. Launch an instance for the OpenVPN server:

- **Image:** I used RHEL 7.4.
- **Instance Type:** t2.micro.
- **Network:** vpcGlobalDataFabric
- **Subnet:** snGlobalDataFabric
- **Auto-Assign Public IP:** Disable (the EIP will be attached later)
- **Security Group:** sgOpenVPN

7. Create an Elastic IP and attach it to the OpenVPN server. This allows the external IP address to remain constant across server shutdowns. Otherwise, you'll have to update your OpenVPN client configuration file with a new IP address every time you run the demo (assuming you shut down the VPN server when it's not in use).

Installing and Configuring the OpenVPN Server

Connect to your server via SSH, using the EIP address associated with your instance. Install OpenVPN using the following sequence of commands:

```
# Install EPEL (OpenVPN isn't available in the default CentOS repositories)
sudo yum install wget -y

# Doesn't work: sudo yum install epel-release -y
# Instead, do this:
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

sudo yum localinstall epel-release-latest-7.noarch.rpm -y

# Install OpenVPN
sudo yum install openvpn -y

# Install Easy RSA
# latest version of easy-rsa-2 on this page:
```

```
# https://github.com/OpenVPN/easy-rsa-old/releases

wget https://github.com/OpenVPN/easy-rsa-old/archive/2.3.3.tar.gz
tar xvf 2.3.3.tar.gz
sudo mkdir /etc/openvpn/easy-rsa
sudo cp -rf easy-rsa-old-2.3.3/easy-rsa/2.0/* /etc/openvpn/easy-rsa
```

To configure the OpenVPN server, start by copying the included sample configuration file:

```
sudo cp /usr/share/doc/openvpn-2.4.6/sample/sample-config-
files/server.conf /etc/openvpn
```

Modify that file wherever appropriate. There's lots of flexibility in terms of how you configure the server. Here are the steps I took, including those where I veered from the tutorial and my rationale for doing so. You may need to tailor these settings based on your objectives.

Leave Commented: `push "redirect-gateway def1 bypass-dhcp"`

This configures all clients to redirect all their IP traffic to go through the VPN. I didn't see any need to do this. My sandbox can continue to use my ISP connection for DNS lookups, downloading images from NASA, calling the AWS Rekognition service, etc. The only traffic I need to route through the VPN is traffic specifically bound for the MapR cluster in my AWS VPC.

Leave commented: `push "dhcp-option DNS 8.8.8.8"; push "dhcp-option DNS 8.8.4.4"`

This isn't relevant, as I did not enable the "redirect-gateway" option above.

Uncomment: `user nobody` and `group nobody`

Uncomment: `topology subnet`

The following line indicates what address the server and clients will take:

`server 10.8.0.0 255.255.255.0`

*By default, the server will take **10.8.0.1** for itself,*
*and the first client (i.e. the MapR Sandbox) will get **10.8.0.2**.*

Add Line: `remote-cert-eku "TLS Web Client Authentication"`

Comment Line: `tls-auth ta.key 0`

Add Line: `tls-crypt myvpn.tlsauth`

Logging proved instrumental in troubleshooting connection issues.

*I found it convenient to put both log files (*openvpn.log *and* openvpn-status.log*) in a log directory I created (/etc/openvpn/logs)*

Uncomment: `log openvpn.log`

Change: `verb 6`

Also generate the static encryption key that you specified in the conf file above:

```
sudo openvpn --genkey --secret /etc/openvpn/myvpn.tlsauth
```

Generating Keys and Certificates

From here: The PKI (public key infrastructure) consists of a separate certificate (also known as a public key) and private key for the server and each client, and a master Certificate Authority (CA) certificate and key which is used to sign each of the server and client certificates.

The following steps generate the necessary keys, certificates, and other artifacts. First:

```
sudo mkdir /etc/openvpn/easy-rsa/keys
```

Modify the file */etc/openvpn/easy-rsa/vars* as follows (substitute the correct internal hostname):

```
#DO NOT RUN THESE COMMANDS! MODIFY vars FILE!
export KEY_CN=ip-10-10-10-248.ec2.internal
export KEY_NAME="server"
export KEY_OU="Community"
```

Then, as *root* (*clean-all* requires sudo, which means you have to source the vars as sudo also):

```
cd /etc/openvpn/easy-rsa
source ./vars
./clean-all

# Note: Having sourced the variables, you can accept the defaults in
# in build-ca and build-key-server even when they look strange.
# In particular, in build-key-server, Common Name defaults to
# "server", which ends up being fine.
# Leave blank: Challenge Password, and optional company name
# y to sign the certificate, y to commit

./build-ca

./build-key-server server

# Generate Diffie-Hellman key exchange file.
./build-dh

# Copy server keys and certs from keys directory into openvpn.
cd /etc/openvpn/easy-rsa/keys
cp dh2048.pem ca.crt server.crt server.key /etc/openvpn
```

Each client will also need a certificate in order for the OpenVPN server to authenticate it. These are created on the server and then copied to the clients. Generate separate keys and certs for each client you intend to connect to your VPN.

```
cd /etc/openvpn/easy-rsa

# Once again, you can accept all defaults here.
# The Common Name will be set to "sandbox".
# Name still defaults to "server" (which works, but I'm not sure...)
./build-key sandbox

# If you're looking ahead to a multi-cloud environment,
# this is a convenient time to build a key for the Azure gateway.
./build-key azureGateway
```

Lastly:

```
cp /etc/openvpn/easy-rsa/openssl-1.0.0.cnf /etc/openvpn/easy-rsa/openssl.cnf
```

Starting the OpenVPN Server

Start OpenVPN Server as follows ("@server" will get mapped to the server.conf file):

```
systemctl start openvpn@server
```

If nothing comes back, it's probably running. Check with "systemctl status openvpn@server". Then, run "ifconfig" and verify a "tun0" interface with an IP address of 10.8.0.1 (which was specified in the config file).

If there are any issues, check the logs and try again. Log file locations are configured in your server.conf file - active connections will be listed in *openvpn-status.log* and startup/connection messages will show in *openvpn.log*. *"verb 6"* is a good log level for debugging.

Once you've got it working, enable it to start automatically at boot:

```
sudo systemctl -f enable openvpn@server
```

The list of enabled services can be verified using this command:

```
sudo systemctl list-unit-files | grep enabled
```

Installing, Configuring, and Running the OpenVPN Client on the MapR Sandbox

As OpenVPN's How-To document explains, "the OpenVPN executable should be installed on both server and client machines, since the single executable provides both client and server functions". So, connect to the sandbox as root and install OpenVPN. (As before, this may require EPEL).

```
sudo yum install -y openvpn
```

Make two directories to hold the keys and log files:

```
sudo mkdir /etc/openvpn/keys
sudo mkdir /etc/openvpn/logs
```

Locate the following files on the server and replicate them on the sandbox. You can copy these files or use cut-and-paste. Either way gets cumbersome, so I built some utility scripts and placed them in the demo project's openVPN directory. I place all of these files in */etc/openvpn/keys*:

```
/etc/openvpn/easy-rsa/keys/ca.crt
/etc/openvpn/easy-rsa/keys/sandbox.crt
/etc/openvpn/easy-rsa/keys/sandbox.key
/etc/openvpn/myvpn.tlsauth
```

Just a note: There's a suggestion on this page by q23 to package the keys up into the ovpn file. I'm not sure if that would be any easier. Here's the full comment:

> *A suggestion: .ovpn files support inline certificates and keys. Instead of having to mess around with multiple files and multiple paths, you can just copy and paste everything from -----BEGIN {CERTIFICATE,RSA PRIVATE KEY}----- to -----END {CERTIFICATE,RSA PRIVATE KEY}----- in between tags for each one: <ca> for the CA public key, <cert> for the server or client's public key, <key> for the server or client's private key, and <tls-auth> for the static key if you're using it. That way, you can have it all packaged up nicely in one .ovpn file instead of having 3-4 files. Makes it a lot easier to use the OpenVPN for Android client, too.*

Warning: Verify that your sandbox's system date is roughly equal to the timestamp of the server where the certificates were generated, or you may get errors ("Your certificate is not yet valid"). You can set the sysdate on the Sandbox as follows:

```
date -s "16 JAN 2019 18:35:00" --utc
```

Create the client configuration file (see below for file naming tips) containing the following lines:

```
client
tls-client
ca /etc/openvpn/keys/ca.crt
cert /etc/openvpn/keys/sandbox.crt
key /etc/openvpn/keys/sandbox.key
tls-crypt /etc/openvpn/keys/myvpn.tlsauth
remote-cert-eku "TLS Web Server Authentication"
proto udp
remote 52.44.120.117 1194 udp
dev tun
topology subnet
pull
user nobody
group nobody
log logs/openvpn.log
verb 6
```

Here are some critical lessons-learned about that file:

- The first line has to read "client". The online tutorial I used seems to indicate it should "reflect the name you gave the client in your key and certificate". But "client" is the valid token that the parser is expecting.
- Make sure the file locations and names are all correct.
- On the client, the remote-cert-eku needs to read "TLS Web *Server* Authentication". The tutorial mistakenly says, "TLS Web *Client* Authentication".
- "Verb 6" generates log messages, which are useful for troubleshooting. Once everything's working, you can throttle back the logging or eliminate this line altogether.

You have to decide what you're going to call the file and how you're going to start the VPN client. If you call it "sandbox.ovpn", then you can initiate the connection as follows:

```
sudo openvpn --config /path/to/sandbox.ovpn
```

That leaves the connection in the foreground, however, forcing you to open another terminal window to do your work. A better option is to run it as a service. To do this, name the file "sandbox.conf" and put it in the */etc/openvpn* directory (the file extension must be ".conf" for it to be picked up by the OpenVPN daemon, as described here). Then, you can start the VPN connection as follows:

```
sudo systemctl start openvpn@sandbox
```

Once again, you can run "systemctl status openvpn@sandbox" and *ifconfig* to verify that you obtained a new "tun0" IP address. That IP address, determined by your server-side OVPN settings, defaults to 10.8.0.2.

You may want to configure the VPN connection to start automatically:

```
sudo systemctl -f enable openvpn@sandbox
```

Finalizing the Cross-Cluster Route

At this point, you should be able to establish a VPN connection from your MapR sandbox to the OpenVPN server deployed in AWS. From the sandbox you should be able to ping the OpenVPN server (on 10.8.0.1) and vice versa (on 10.8.0.2). The next step is to launch one or more additional AWS instances to serve as your AWS MapR cluster. Assuming you've done this, there are several additional

steps you need to take to ensure that your sandbox can ping the MapR nodes (using their internal IP addresses) and vice versa.

1. Within AWS, verify connectivity between your OpenVPN server and your MapR nodes. This validates your Security Group settings. It's best practice to use telnet rather than ping, because firewalls can be configured to allow ICMP traffic (such as ping) while denying regular TCP traffic (such as that required for MapR replication). You probably have to install telnet (yum install telnet). Then test:

   ```
   telnet <ipaddr> 22
   ```

2. Enable IP Forwarding on the OpenVPN server:

   ```
   sysctl net.ipv4.ip_forward=1
   ```

   Make this change permanent by adding the line to */etc/sysctl.conf* (as root):

   ```
   echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
   ```

3. Disable Source/Destination Check on the OpenVPN Server. With the server instance selected in the AWS console, navigate to Actions > Networking > Change Source/Dest Check > "Yes, Disable". Otherwise, network traffic will not be able to flow through your VPN server.

4. Modify the OpenVPN server config to publish a route for the AWS subnet to the VPN clients. Open the */etc/openvpn/server.conf* file and locate the "push routes" section. Add the following line (where 10.10.10.0 corresponds to the snGlobalDataFabric subnet):

   ```
   push "route 10.10.10.0 255.255.255.0"
   ```

   Then, restart the OpenVPN server:

   ```
   systemctl restart openvpn@server
   ```

5. Add a route to the rtGlobalDataFabric route table so that the MapR cluster nodes know to reach the Sandbox through the OpenVPN server. Add a route with a *Destination* 10.8.0.0/24 and a *Target* of the OpenVPN instance.

6. On the sandbox, restart the VPN client:

   ```
   systemctl restart openvpn@sandbox
   ```

Now test the configuration. From the sandbox, you should be able to ping the MapR nodes in AWS using their internal IPs, and from the AWS nodes you should be able to ping the sandbox at 10.8.0.2.

At this point, provided that you've used *telnet* to test connectivity instead of *ping*, you're able to install MapR and establish cross-cluster replication. Since this demo replicates from the Sandbox to the Cloud (one way), these steps are sufficient. If you wanted to replicate from the Cloud to the Sandbox, you could do so on 10.8.0.2 (the OpenVPN tunnel IP of the Sandbox). That means that MapR on the Sandbox has to be started after the OpenVPN client has established the connection (for MapR to see that IP interface), and also assumes that the Sandbox always gets assigned 10.8.0.2 (it's possible to ensure this, but this has not been configured, so there's an element of good luck here to assume that it's always going to be 10.8.0.2). It's probably cleaner to configure replication using the Sandbox's permanent NAT address (10.0.2.15) or the Sandbox's second network adapter (running 198.168.2.x, the same network as the Edge cluster). If you take that latter approach, you can have the option of replicating from the Edge Cluster straight through to the Cloud cluster, and vice-versa (because the Sandbox and the Edge Cluster are on the same subnet, by virtue of the Sandbox's second adapter). To close the loop on this (literally), see the next section, "Edge-to-Cloud".

Edge-to-Cloud

Continuing from the previous section, Edge-to-Cloud replication isn't a part of this demo scenario (which has the Sandbox acting as an intermediary). Still, we're so close, we may as well see what it would take. The previous steps get us to the present state:

- From the Sandbox, you can ping the OpenVPN server in AWS, using either its internal AWS IP address (10.10.10.x) or its OpenVPN tunnel address (10.8.0.1).
- Also from the Sandbox, you can ping the MapR node on its internal IP address (10.10.10.z).
- From either of the AWS instances, you can ping the MapR Sandbox using 10.8.0.2.
- All those connections also check out using *telnet*, which is a better test (crucial, in fact).
- Neither of the AWS instances can ping the sandbox via its NAT address (10.0.2.15) or its address on the Edge Cluster subnet (192.168.2.50), which also means they can't reach the Edge cluster nodes.

To fix this situation, I found this web page extremely helpful. Adapting it to our scenario, the required steps are as follows:

1. On the AWS OpenVPN server, add the following lines to the *server.conf* file:

   ```
   client-config-dir /etc/openvpn/ccd
   route 192.168.2.0 255.255.255.0
   push "route 192.168.2.0 255.255.255.0"
   ```

   According to that web page, the route entries adjust the local (AWS server) routing table, telling it to route those networks over the VPN, while the push routes are added on the clients connecting, telling *them* to route those networks over the vpn.

2. Also on the AWS OpenVPN server, create the directory */etc/openvpn/ccd* and in it create a file called *sandbox* (to match the CN of the client). This file should contain just a single line as follows:

   ```
   iroute 192.168.2.0 255.255.255.0
   ```

3. In AWS, restart the OpenVPN server:

   ```
   systemctl restart openvpn@server
   ```

4. On the sandbox, restart the VPN client:

   ```
   systemctl restart openvpn@sandbox
   ```

5. In the AWS console, add a route to the rtGlobalDataFabric route table so that the MapR cluster nodes know to reach the Edge subnet through the OpenVPN server. Add a route with a *Destination* 192.168.2.0/24 and a *Target* of the OpenVPN instance.

Now test it. From the MapR nodes in AWS, you should now be able to connect to the Sandbox on 192.168.2.50 (using *ping* and *telnet*). This enables you to establish two-way replication between the Sandbox and the AWS cluster, using the 192.168.2.50 address. The same procedure should work on 10.0.2.15.

You *cannot* yet reach the Edge nodes from the AWS (or vice versa), until you do the following:

1. Enable IP Forwarding on the Sandbox.

   ```
   sysctl net.ipv4.ip_forward=1
   ```

   Make this change permanent by adding the following line to */etc/sysctl.conf*. As root:

```
echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
```

2. *On each node* in the Edge cluster *(I think mirroring requires all nodes to know about this route, whereas table/streams replication… ?)*, add a route to AWS subnet (this is ephemeral):

```
ip route add 10.10.10.0/24 via 192.168.2.50
# To verify (use either):
route -n
ip r
```

   To make that change permanent (on RHEL/CentOS), create a new file called */etc/sysconfig/network-scripts/route-eno1* (to match the interface) and add the following single line:

```
10.10.10.0/24 via 192.168.2.50
```

   Then restart the network and verify the route is still there:

```
systemctl restart network.service
```

3. Test the connectivity. If you can ping from the Edge cluster nodes to the AWS nodes, but not the reverse, then implement the following iptables rule (on the Sandbox), as explained here. I admit I'm a little bit fuzzy about what this is doing, but it seems to worked. In subsequent iterations, I haven't needed this, so I'm not sure what was different.

```
iptables -t nat -A POSTROUTING -d 192.168.2.0/24 -j MASQUERADE
```

You should now have complete bi-directional connectivity from the Edge cluster to the Sandbox to the AWS cluster! In one case - and it can be hard to keep all this straight

Multi-Cloud

You can extend this demonstration to highlight MapR's support for multi-cloud strategies and architectures. Either edition (HQ/Edge) can be deployed to either cloud, but since Rekognition is an AWS-based service I tend to deploy the HQ version to AWS.

There are (at least) two ways to achieve cross-cloud connectivity. The first is using StrongSwan VPN and an Azure-managed VPN service, as described in a blog post by Nick Amato. I stopped using this approach for several reasons:

1. It's very specific to Azure.
2. Costs continually accrue. The Azure VPN Gateway costs $4.56/day, and you can't just shut it down and restart it when you need it. I had to delete the Gateway, as well as the Connection attached to it.
3. An alternative approach based on OpenVPN[3] can be used in every scenario (including the Edge), and it works everywhere in a uniform way. You can also shut it down when it's not in use, minimizing cost.

Nevertheless, the work having been done, you can find both approaches documented here.

It starts getting confusing having all these instances deployed in different environments. To make it easier for audiences to follow, a feature was added to show the environment in the dashboard banner.

[3] OpenVPN is an open-source SSL VPN solution and uses SSL/TLS to create site-to-site tunnels instead of IPsec.

The following banner, for example, indicates an "Edge" cluster hosted in Azure. Set the banner logo using the dashboard Control Panel:

If using terminal windows, take advantage of the various default color schemes - there are options for orange and blue which map nicely to AWS and Azure.

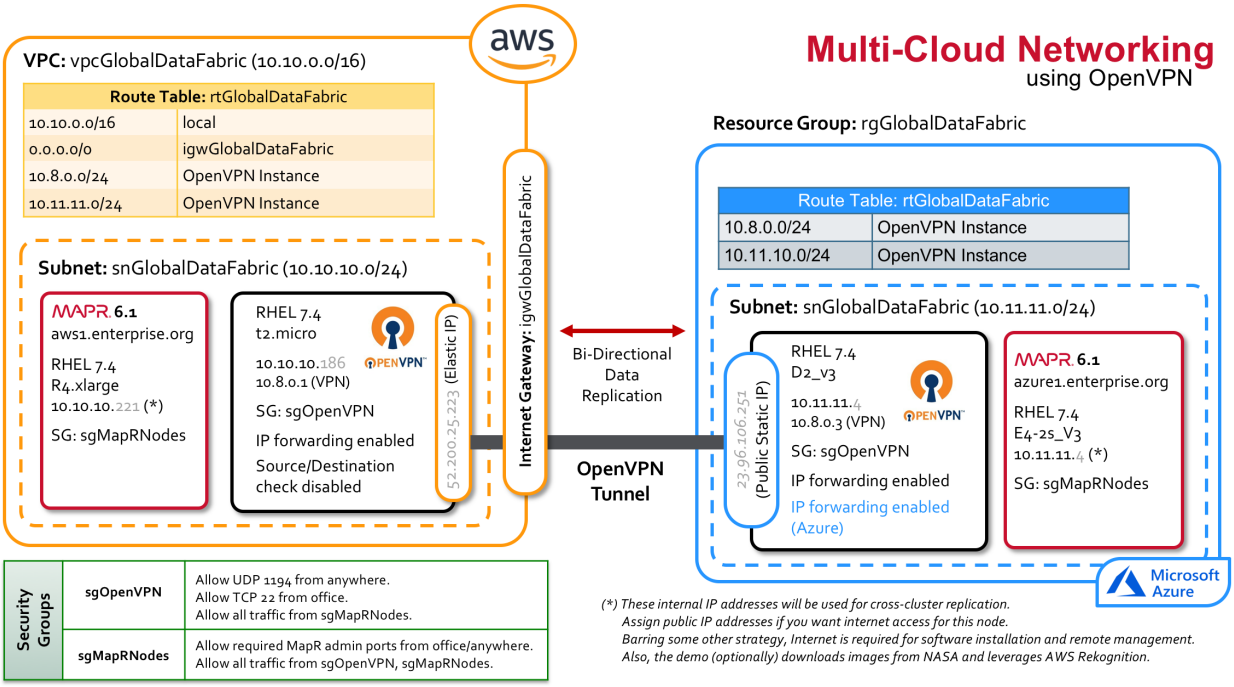
Multi-Cloud Connectivity Via OpenVPN

This builds on the OpenVPN lessons learned from the Sandbox-to-Cloud and Edge-to-Cloud networking described earlier. In that exercise, we installed an OpenVPN server in AWS, configured the AWS-based routing, and connected an OpenVPN client hosted on a MapR Sandbox. This section assumes that those pieces are already working (the sandbox isn't required, but you should verify that clients can successfully connect to your OpenVPN server).

To connect AWS to Azure, we only have to install the OpenVPN client in Azure, configure the Azure routing, and then tweak the routing back in AWS to make the subnets aware of each other. In Azure, the OpenVPN client will be acting as a Linux VPN gateway. There's a good description of this to be found here ("Site-to-Site Routing Explained in Detail") which emphasizes that it's a *client* configuration on the remote end (not another server, as you might think), and that additional VPN clients can log on to the OpenVPN Access Server and gain access to any of the devices *in these two networks*".

Network Diagram

The following diagram illustrates the environment that we will be configuring.



Azure Setup

Get started in **Azure by creating the resources to host the OpenVPN client that will act as the gateway:**

- Create a **Resource Group**: "rgGlobalDataFabric"
- Create a **Virtual Network**: "vnGlobalDataFabric", 10.11.0.0/16 (address space), "snGlobalDataFabric" (subnet), 10.11.11.0/24 (address range for the subnet)
- Create a **Virtual Machine** to host the Open VPN client, which will act as a gateway server.
    - ResourceGroup: rgGlobalDataFabric
    - Name: vmOpenVPN
    - Image: I used Red Hat Enterprise Linux 7.4
    - Size: I used the default Standard D2 v3 (2 vcpus, 8GB memory)
    - Authentication Type: SSH public key
    - Username: azure-user
    - SSH public key: <you have to provide this>
    - Public inbound ports: None (Network Security Groups will be configured later)
    - OS Disk Type: HDD
    - Review & Create.
    - Cost: 0.0960 USD/hr

A Network Security Group ("vmOpenVPN-nsg") was automatically created for this VM. Add an inbound security role to that group to allow SSH from your IP address (TCP port 22).

A Public IP address ("vmOpenVPN-ip") was automatically assigned to this VM. Reconfigure this IP address to be *static*, so it won't change when you shut down the instance: select the VM, click the Public IP address, click Static, and click Save. An IP address will be assigned at this time. Azure allows you 5 static IP addresses for free.

SSH into the new VM, and install and configure the client, as described previously ("Installing, Configuring and Running the OpenVPN Client"). In summary:

```
sudo yum install wget -y
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-
7.noarch.rpm
sudo yum localinstall epel-release-latest-7.noarch.rpm -y
sudo yum install openvpn -y
sudo mkdir /etc/openvpn/keys
```

Copy the following files into the */etc/openvpn/keys* from the AWS Open VPN server. You'll have to create the *azureGateway.crt* and *.key* files if you haven't previously.

```
/etc/openvpn/easy-rsa/keys/ca.crt
/etc/openvpn/easy-rsa/keys/azureGateway.crt
/etc/openvpn/easy-rsa/keys/azureGateway.key
/etc/openvpn/myvpn.tlsauth
```

Create the client configuration file (*/etc/openvpn/azureGateway.conf*) as follows:

```
client
tls-client
ca /etc/openvpn/keys/ca.crt
cert /etc/openvpn/keys/azureGateway.crt
key /etc/openvpn/keys/azureGateway.key
tls-crypt /etc/openvpn/keys/myvpn.tlsauth
remote-cert-eku "TLS Web Server Authentication"
proto udp
remote <AWS-OpenVPN-ElasticIP> 1194 udp
dev tun
topology subnet
pull
user nobody
group nobody
log logs/openvpn.log
verb 6
```

Now initiate the connection, and if it's successful, you can enable the service to start automatically:

```
sudo systemctl start openvpn@azureGateway
sudo systemctl -f enable openvpn@azureGateway
```

Finalizing the Cross-Cluster Route

At this point, you should be able to establish a VPN connection from your OpenVPN gateway client in Azure to your OpenVPN server in AWS. Also, because of steps taken previously, in "Finalizing the Cross-Cluster Route" for the Sandbox-to-Cloud scenario, the Azure VPN client gateway box should be able to ping any machine in the AWS subnet using their internal IP addresses.

The next step is to launch one or more additional Azure instances to serve as your Azure MapR cluster. As the networking currently stands, the AWS MapR nodes will not be able to contact any MapR nodes in the Azure subnet, nor will those Azure MapR nodes be able to contact the AWS MapR nodes.

To fully enable routing between the two cloud environments:

1. Within Azure, verify connectivity between your OpenVPN gateway client and your MapR nodes. This validates your Security Group settings. It's best practice to use telnet rather than ping, because firewalls can be configured to allow ICMP traffic (such as ping) while denying regular

TCP traffic (such as that required for MapR replication). You probably have to install telnet (yum install telnet). Then test:

```
telnet <ipaddr> 22
```

2. Enable IP Forwarding on the Azure OpenVPN gateway client:

   ```
   sysctl net.ipv4.ip_forward=1
   ```

   Make this change permanent by adding the following line to */etc/sysctl.conf*: As root:

   ```
   echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
   ```

   The current value of this setting can be verified at any time using the following command:

   ```
   sysctl net.ipv4.ip_forward
   ```

3. Azure also has an IP forwarding setting which must be enabled on that server's *network interface*. Navigate to that network interface (on the OpenVPN gateway client) and find its *IP Configurations*. Set IP Forwarding to *Enabled*, remembering to save the new setting.

4. Create a new custom route table so that the MapR cluster nodes in Azure know to reach the AWS subnet through the OVPN server. In the Azure console, search for Route Tables, and select it.

   - Name = rtGlobalDataFabric
   - Resource Group = rgGlobalDataFabric

   Then, add two routes:

   - Route Name: "awsSubnet", Address Prefix = "10.10.10.0/24" (the AWS subnet), Next Hop Type = Virtual Appliance, Next Hop Address = 10.11.11.4 (the Internal IP address of the OpenVPN client gateway)
   - Route Name: "vpnSubnet", Address prefix = "10.8.0.0/16", Next Hop Type = Virtual Appliance, Next Hop Address = 10.11.11.4

   Finally, associate the route to the snGlobalDataFabric subnet. Navigate through Virtual networks to the vnGlobalDataFabric, select Subnets, select the snGlobalDataFabric subnet, and change Route table to be rtGlobalDataFabric. Click Save.

At this point, the Azure MapR nodes can now ping 10.8.0.3 (the Azure VPN gateway client), but still cannot ping the AWS VPN server at 10.8.0.1. Looking at the OpenVPN log files on both AWS and Azure, it's clear the packet is received by the AWS server, at which point it is dropped. The remedy is to modify the OpenVPN server config (in AWS) so that it can find the subnet behind the Azure gateway. We did this before, to allow the AWS subnet to find the Edge cluster. The same steps apply here:

- In the *server.conf* file of the AWS OpenVPN server, there should already be a *route* and *push* entry for the Edge subnet. Add a second set that points the Azure subnet:

  ```
  route 10.11.11.0 255.255.255.0
  push "route 10.11.11.0 255.255.255.0"
  ```

- In that same file, the line containing "client-config-dir /etc/openvpn/ccd" should already be uncommented. Create a second file called "/etc/openvpn/ccd/azureGateway" with a single line in it: "iroute 10.11.11.0 255.255.255.0". The name of the file has to match the CN you used when you generated the client certificate.
- Restart OpenVPN in AWS and then Azure. You should now be able to ping 10.8.0.1 (the OpenVPN server in AWS) from the MapR node in AWS.

- In the AWS console, add a route to the rtGlobalDataFabric route table so the MapR nodes in AWS can find the MapR nodes in Azure. (Destination = 10.11.11.0/24, Target = OpenVPN server). You should now be able to ping any node in Azure from any node in AWS, and vice versa, using only internal IPs.
- Switching from *ping* to *telnet*, you'll discover that while *ping* is successful, *telnet* produces a "no route to host" error. So it's pretty unlikely at this point that the MapR clusters will be able to talk to each other (and this is why it's so important to use *telnet*). The problem is that the RHEL nodes in Azure are running *firewalld*, which is blocking the traffic. In AWS, firewalld isn't running by default - the firewall is the function of the AWS Security Groups. But in Azure, you go through all the trouble of configuring the Network Security Group only to discover that the Linux firewalld service is still functioning as a second firewall. You can see this by running "sudo systemtctl status firewalld". On AWS this will indicate that the service could not be found, whereas Azure will show it running (this is using RHEL 7.4 and 7.6 in both environments - it may vary with other Linux flavors/versions). "sudo iptables -S" in Azure shows a bunch of rules in place. I don't really know why the RHEL instances in Azure have *firewalld* running in addition to the Security Groups. Shutting it down completely ("systemctl stop firewalld") on the OpenVPN gateway client solves the connectivity issue, but you should proceed with caution. I'd like to hear back from somebody with Azure experience whether it's safe to do this (given the presence of the Security Groups). Otherwise, this page gives you an idea of what the firewall would need to look like.

The last missing piece (if you want it) is that you cannot yet connect from a MapR node in Azure (on 10.10.11.x) to the Edge node (192.168.2.[21-23]), or vice versa. Even though all the VPN clients (10.8.0.x) *can* connect to each other. What's missing are the routes. On the Edge nodes, add a route to the Azure subnet as follows:

```
ip route add 10.11.11.0/24 via 192.168.2.50

# Verify:
route -n
```

And in Azure, add a new route to the rtGlobalDataFabric route table:

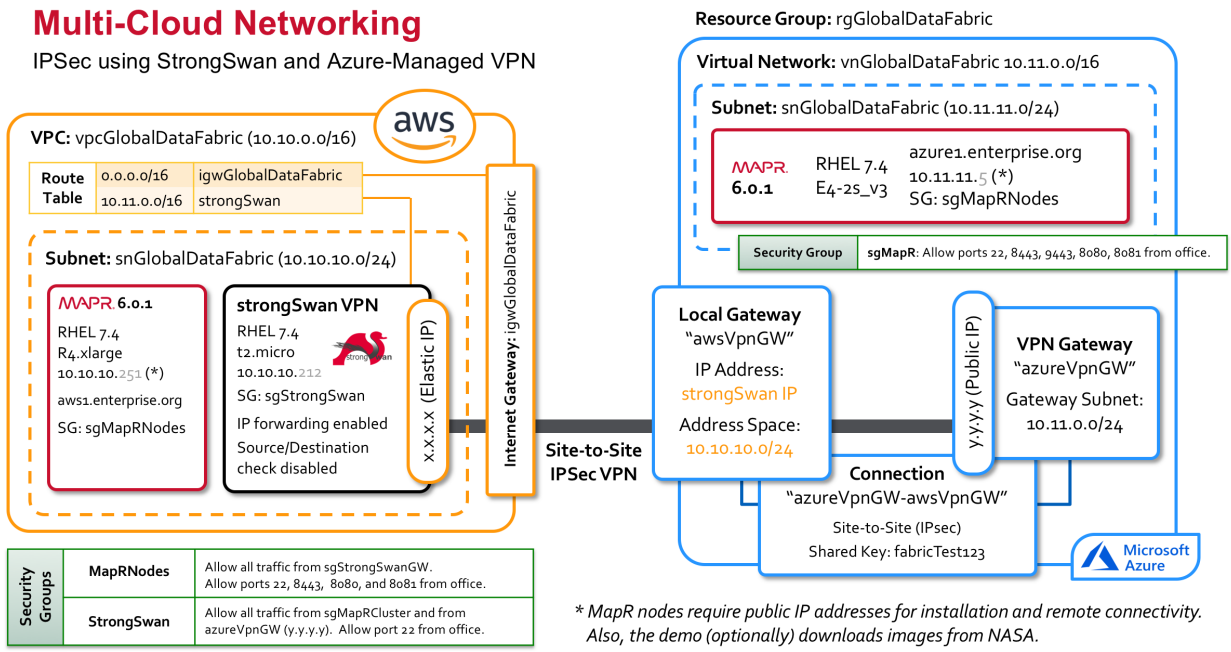
- Name = edgeSubnet
- Address Prefix = 192.168.2.0/24
- Next Hop Type = Virtual Appliance
- Next Hop Address = <internal IP of the Azure OpenVPN client gateway>

I've also seen it mentioned that this requires an additional setting in the AWS *server.conf* file. Here, for example, but also in the comments of the *server.conf* file, it says you need the "client-to-client" directive to allow clients to see each other. But I haven't observed this to be the case.

This was a pretty lengthy process, but it provides the options to cover a lot of different scenarios. At the end, you could have 4 MapR clusters all talking to each other: an Edge cluster, a MapR Sandbox, and cloud clusters operating in AWS and Azure.

Multi-Cloud Connectivity via StrongSwan, IPSec, and Azure-Managed VPN Services

In this alternative approach to OpenVPN, connectivity between the two environments is established using IP forwarding across an IPSec tunnel, per the configuration steps outlined in Nick Amato's outstanding blog on the topic (found here). The network configuration is illustrated below.



Here are the step-by-step instructions. An introduction to strongSwan can be found here. Note that Nick's blog only goes so far as to establish connectivity across clouds. This is fine if you're setting up a single-node MapR cluster, collocated on the same instance as the strongSwan VPN gateway software. If you want a multi-node MapR cluster, or if you install MapR on a separate node from strongSwan, there are additional steps at the end that must be implemented.

In **Azure:**

- Create a **Resource Group**: "rgGlobalDataFabric"
- Create a **Virtual Network**: "vnGlobalDataFabric", 10.11.0.0/16 (address space), "snGlobalDataFabric" (subnet), 10.11.11.0/24 (address range for the subnet)
- Create a **Public IP Address**: "AzureGatewayIP".
- Create a **Virtual Network Gateway**: "AzureVpnGW", vnGlobalDataFabric (virtual network), 10.11.0.0/24 (Gateway subnet address range), AzureGatewayIP.
- Deploy the **Virtual Machine(s)** to host the MapR cluster. I used an E4-2s_V3 instance size, which provide 32 GB RAM and 2 vCPU's. A free Azure trial only includes 4 vCPU's max, so it's important to minimize VCPU consumption in that scenario. I added an extra disk to be the MapR disk.
- Install MapR.

In **AWS**:

- Create a **VPC**. There are multiple ways to do this. To mirror Nick's instructions, ignore the "Launch VPC Wizard" button and navigate instead to the VPC Dashboard. Then, select "Your VPCs", and click "Create VPC". Call it "vpcGlobalDataFabric" (name tag), and give it the 10.10.0.0/16 CIDR block.
- Create a **Subnet**: "snGlobalDataFabric", us-east-1a (AZ), 10.10.10.0/24 (CIDR Block).
- Create an **Internet Gateway**: "igwGlobalDataFabric". Attach it to the VPC.

- Launch an **EC2 Instance** to host the strongSwan VPN software (or "to maintain the VPN", in Nick's language). Specify *disable* for the public IP, because we explicitly create an Elastic IP and associate it with the instance in a subsequent step.
- Create an **Elastic IP** address and associate it the instance just created. 3.216.60.36
- Launch additional EC2 instances to serve as the nodes of the MapR cluster. These nodes must also have public IP addresses, to enable remote access as well as to install the MapR software.
- Create the necessary **Security Groups.** The strongSwan instance must allow all traffic coming from the Azure VPN Gateway, all traffic from the local (AWS) MapR nodes, and ssh from your local machine. The MapR nodes require all traffic coming from strongSwan, as well as ports 22, 8443, 9443, 8080 (Jetty), and 8081 (Vert.x) from your local machine.
- Create a **Default Route** to the Internet for the subnet via the Internet Gateway created previously. Select the VPC, navigate to the Route Table, navigate to the Routes tab, and add a route to 0.0.0.0/0, specifying the Internet Gateway as the Target. This enables remote connectivity, MapR software installation, and access to the NASA data sets.

Back in **Azure:**

- Add a **Local Network Gateway**: "awsVpnGW" (Name), 137.117.97.48 (IP address), 10.10.10.0/24 (the AWS subnet). Note that "local" here means our "local network", i.e. *not* Azure, i.e. our AWS network. "A local network gateway represents the hardware or software VPN device in your local network. Use this with a *connection* to set up a site-to-site VPN connection between an Azure virtual network and your local network."
- Add a **Connection**. Site-to-Site (Connection Type), azureVpnGW (Virtual Network Gateway), awsVpnGW (Local network gateway), azureVpnGW-awsVpnGW (Connection name), fabricTest123 (Shared Key).

Back in **AWS**:

- Install strongSwan:

```
yum install gcc gmp-devel wget bzip2
wget https://download.strongswan.org/strongswan-5.6.0.tar.bz2
bzip2 -d strongswan-5.6.0.tar.bz2
tar xvf strongswan-5.6.0.tar
cd strongswan-5.6.0
./configure && make && sudo make install
```

- Configure strongSwan. Edit */usr/local/etc/ipsec.conf*, adding the following section. Be sure to tab-indent everything below "conn azure" – you get parsing errors if you don't:

```
conn azure
	authby=secret
	type=tunnel
	leftsendcert=never
	left=10.10.10.49              # Internal IP of strongSwan instance
	leftsubnet=10.10.10.0/24      # AWS subnet
	right=23.96.39.117            # Public IP of the azureVpnGW
	rightsubnet=10.11.11.0/24     # Azure subnet
	keyexchange=ikev2
	ikelifetime=10800s
	keylife=57m
	keyingtries=1
	rekeymargin=3m
```

```
compress=no
auto=start
```

- Add the secret key to the /usr/local/etc/ipsec.secrets file:

```
10.10.10.49 23.96.39.117 : PSK "fabricTest123"
```

- Start strongSwan:

```
/usr/local/sbin/ipsec start
```

Or:

```
systemctl start strongswan
```

To start strongSwan automatically on reboot:

```
sudo systemctl enable strongswan
```

To view connection logs:

```
sudo tail -f /var/log/messages
```

Now, from the strongSwan box on AWS, try to ping the maprN1 in Azure using its private IP address. From the MapR node in Azure, try pinging the StrongSwan box in AWS using its private IP address. You should be successful in both directions.

Now launch the additional node(s) in AWS for the MapR cluster and install MapR. From these nodes, you won't be able to ping Azure (or vice versa) without performing the following steps:

- Add a **second route** to the Route Table, creating a route to 10.11.0.0/16 (the Azure subnet) through the strongSwan instance.
- **Disable Source/Destination Check** on the strongSwan instance. That important piece comes from here: "the instance will act as a router, therefore will need to have source/destination check disabled (the instance will not be the source or the destination of the traffic between the Azure VM and the EC2 instance from the private subnet)." To find this option, select the EC2 instance and navigate to Action > Network > "Change Source/Dest. Check".
- **Enable IP forwarding** in the strongSwan instance. This is documented here. "In order to forward traffic to hosts behind the gateway, the following option has to be enabled on Linux gateways:"

```
sysctl net.ipv4.ip_forward=1
sysctl net.ipv6.conf.all.forwarding=1
```

To enable this permanently, add the following line to */etc/sysctl.conf*:

```
net.ipv4.ip_forward = 1
```

You should now be able to ping from any instance in Azure to any instance in AWS.

Building an Edge Cluster

Hardware

This three-node MapR Edge cluster was built using Intel NUCs ("Next Unit of Computing"). NUCs are bare-bones out of the box, so 32GB RAM and a 500GB SSD had to be added to each. The NUCs were

connected together using a wireless router[4] and cables. Network cables, surge protectors, and extension cords were added to make this demo Edge cluster completely self-contained. Everything was packaged into a ruggedized Pelican case for easy transportability.

Once an OS is installed and network connectivity is established, you can connect to the NUCs from your laptop via ssh. But to set them up initially, you need an external USB keyboard, a USB mouse, and a monitor. These items can depend on the hardware you have already in your office. I already had a USB keyboard and a mouse, and I was able to use my Apple display monitor using a Thunderbolt3-to-Thunderbolt2 adapter (the NUCs have Thunderbolt3 ports). To connect the Edge cluster to my MacBook, I used a Thunderbolt2-to-Ethernet adapter.

Here are some useful links for the Intel NUC Kit NUC6i7KYK:

- Technical Product Specification
- This page has Recommended Accessories. UPS?
- Integration Guide. Contains port info. USB 3.0, CIR (Consumer Infrared), Ethernet, Mini DisplayPort, Thunderbolt 3, HDMI.
- User Guide. Nothing useful.
- Getting Started, which points to three things:
    - Follow the User Guide (nothing useful there)
    - Update the BIOS.
    - Install an OS. Installing Linux Lite. Needs a USB device with an image.
- Support Site
- BIOS & Drivers (these are almost exclusively for Windows OS)

Bill of Materials

The following table shows the complete Bill of Materials for the Edge Cluster used in this demo.

Item	Qty	Description and Model Number	Unit Cost	Link
NUC	3	Intel NUC Kit NUC6i7KYK	$579	link
32GB RAM	3	Crucial 32GB Kit (16GBx2) DDR4 2133 MT/s (PC4-17000) DR x8 SODIMM 260-Pin Memory - CT2K16G4SFD8213	$367	link
500 GB SSD	3	Crucial MX300 525GB 3D NAND SATA M.2 (2280) Internal SSD - CT525MX300SSD4	$150	link
Router	1	Linksys E2500 (N600) 4-Port Wireless Router	$65	
Monitor Adapter	1	Thunderbolt 3 to Thunderbolt 2 Adapter	$50	link

[4] You could also use a Hub, but a wireless router allows you to connect to the Edge cluster from a mobile device such as an iPad or iPhone.

Ethernet Adapter	1	Thunderbolt to Gigabit Ethernet Adapter	$30	link
Cables	4	3' Cat-5e Network Cables	$10 (total)	
Surge Protector	1	6 Outlet Surge Protector with 4' Cord	$17	
Extension Cord	1	9' Extension Cord	$13	
Pelican Case	1	Pelican 1560 with Foam, Desert Tan	$185	link
Keyboard	1	External USB Keyboard		
Mouse	1	External USB Mouse		
Total Cost			~$3600	

Table 1 – Bill of Materials for Assembling the Complete Self-Contained Edge Cluster

Configuring the Router

I originally used a 5-port Ethernet Hub for the Edge cluster. That works, but you can't connect to it from a mobile device, which makes a nice demonstration. Even using a second laptop, it's nice to connect wirelessly. So I swapped the Hub out for a wireless router, which creates its own Wi-Fi network. The NUCs are still connected to the router using ethernet cables and static IP addresses, and replication still happens over an ethernet cable[5]. The only thing connecting wirelessly are mobile Edge Dashboard clients.

I couldn't use the Linksys installation CD, which requires that I hook up the router to the Internet. That's not the goal here – this cluster will never be hooked up to the Internet. So instead, I hooked up an ethernet cable from the Thunderbolt Ethernet port on my MacBook to Port1 on the Linksys Router. The router's default address is 192.168.1.1. This requires temporarily changing the IP address of the Thunderbolt Ethernet port from 192.168.**2**.100 to 192.168.**1**.100, putting it on the same subnet as the router. You then point a browser to http://192.168.1.1.

Login using the default admin/admin username and password. The router's basic setup screen allows you to change the IP address. Before doing anything else, change the IP address to be 192.168.**2**.1 and click *Save Settings*, which reboots the router. Now, reconfigure the Thunderbolt Ethernet port *back* to 192.168.**1**.1, and connect to the router at http://192.168.**2**.1.

Then, change the following basic settings:

[5] The Linksys E2500 (N600) router is equipped with 4 Fast Ethernet (10/100 Mbps) ports.

- Internet Connection Type: Leaving this at DHCP. This router will never be connected to the internet, so I'm not sure this matters. But if you switch it to "Static IP", you're able (required?) to specify an Internet IP Address, Subnet Mask, Default Gateway, DNS.
- Router Name: MapREdgeRouter
- DHCP Server: Enabled
- Start IP Address: 192.168.2.200
- Max Number of Users: 5
- IP Address Range: 192.168.2.200 to 204
- Client Lease Time: 0 (one day)
- Static DNS (blank for now).
- Time Zone: default – GMT-08:00 Pacific Time (USA & Canada)
- Save Settings

Wireless Settings:

- Manual
- 5 GHz Wireless Settings:
    - Network Mode: Mixed
    - SSID: EDGENET-5G
    - Channel Width: Auto (20 MHz or 40 MHz)
    - Channel: Auto
    - SSID Broadcast: Enabled
- 2.4 GHz Wireless Settings:
    - Network Mode: Mixed
    - SSID: EDGENET
    - Channel Width: Auto
    - Channel: Auto
    - SSID Broadcast: Enabled
- Save Settings
- Wireless Security (use this for both networks).
    - Security Mode: WPA2 Personal
    - Passphrase: SecurityIsN0t@J0k3
- Guest Access: Not Allowed
- Wireless MAC Filter: Disabled
- Administration:
    - Router Password: mapredge
    - Local Management Access
        - Access via: Enabling HTTPS and disabling HTTP caused problems. Reverted to default (HTTP only, not HTTPS). "Local Management" suggests that the cable has to be plugged in here.
    - Remote Management: Disabled

To Test, connect your MacBook Wi-Fi to the newly established EDGENET-5G network. You should be prompted for a password and successfully connect. From System Preferences > Network, the Wi-Fi status should report that "Wi-Fi is connected to EDGENET-5G and has the IP address 192.168.2.204". Remember that Internet will *not* be available on this network.

Installing CentOS

Note: Ubuntu appears to be the preferred Linux variant for Intel NUCs. Only Windows is officially supported (here) by Intel, but the Product Compatibility Tool does include Ubuntu on its list of "non-Windows operating system versions that are reported as compatible by Intel NUC product owners". That appears to be as good as it gets. Ubuntu also has NUC instructions on their website, something you can't find for CentOS. Nevertheless, I went with CentOS because that's what I'm familiar with and that's what my customers use (and I didn't encounter any issues).

Pre-Requisites

To install the OS, you'll need an external wired USB keyboard, an external monitor, and a USB stick. A USB mouse also helps. For a monitor, I used a Thunderbolt 3 to Thunderbolt 2 Adapter to connect the NUC to my external Apple display. And since you'll be setting hostnames and IP addresses, decide upon these settings and put them in a network diagram.

Creating a Bootable USB

To install CentOS (or any OS) on a NUC, you need to download an ISO, burn it to a USB, then boot from that USB and install the OS.

The CentOS Download page has three choices: 1) DVD ISO, 2) Everything ISO, 3) Minimal ISO. At 7GB, the *Everything ISO* is comfortable smaller than my 32GB USB stick, so I used that. Everything here is the latest version, which ended up being 7.4. They make it strangely difficult to find an older release.

To create a bootable USB device, see here. I got some additional help on specific *diskutil* commands from here. Note that */dev/disk4* was identified as the USB drive using the output from the *diskutil* command. The process didn't go smoothly, but eventually I had my USB stick.

```
sudo su
diskutil list
diskutil unmountDisk /dev/disk4
dd if=CentOS-7-x86_64-Everything-1708.iso of=/dev/disk4
```

Install from a Bootable USB

To boot from the USB, connect the device and power on the NUC.

- Press F10 to enter the Boot Menu.
  - *Press hard here – if it's too light it boots in some different way.*
  - *I seem to have the best luck by rapidly pushing the F10 button several times.*
- Select the USB Boot Drive
- Select Install CentOS 7
- On the Installation Summary Screen, visit the various topics

Date & Time: There's a warning on the bottom of this screen that says, "You need to set up networking first if you want to use NTP". Revisit this screen after setting up networking.

Installation Source: Accept Default (Local Media).

Software Selection: Accept Default (Minimal Install).

Installation Destination: This is where you set up the partitions:

- A single Local Standard Disk – 489 GB – should have been identified in your NUC as /sda.
- Under Partitioning, select "I will configure partitioning", and click "Done".
- The Manual Partitioning screen should list just one category for "New CentOS 7 Installation". Delete any existing partitions and mount points that might have been created previously.
- Underneath the link to automatically create partitions is a drop-down selector. Select "**Standard Partition**" as the partitioning scheme. Then click the link to "**create them automatically**".
- **Delete the /home partition**. This 400+GB space will be reserved for MapR. That leaves the following: /boot (sda1, 1024MiB, xfs), swap (sda2, 15.69 GiB, swap), and / (sda3, 50GiB, xfs). If CentOS instead created 4 partitions, it's doing something weird, like not booting off the USB stick – reboot, and this time press that F10 key a few times rapidly to enter the Boot Menu.
- Click Done. Accept the scary-looking changes with confidence.

Network & Hostname: The installer recognizes the Ethernet (eno1) and Wireless (wlp3s0) NICs. Configure each as follows:

Ethernet (eno1):

- Click *Configure*.
- Go to the IPv4 Settings tab.
- Change Method from DHCP to **Manual**.
- Add a static address (IP address = 192.168.2.**x**, Netmask=255.255.255.0) to match your network diagram.
- Check the box that says "Require IPv4 addressing for this connection to complete".
- Click Save.
- Turn it on (top right corner). You should see the configuration displayed.

Wireless (wlp3s0):

- Choose your WiFi network from the dropdown list. Wait while the adapter connects and retrieves an IP address (e.g. 192.168.1.x).

Click Done to finish the networking section. Network & Hostname should say "Connected: <wireless-network>, eno1.

Date & Time (Redux): Revisiting this section, the warning had changed to something like you have no NTP servers. On the top right, I toggled the Network Time off and then on, or you can leave this screen and come back. At some point, NTP kicks in. Clicking on the tool icon in the top right, the installer displays 4 available NTP servers, configured to use by default ([0-4].centos.pool.ntp.org). Click Done.

Select "Begin Installation".

While it's installing, set a **Root Password**. Don't create any users.

The installation takes only 2 minutes to complete. When you reboot, leave the USB stick in – you need to boot off of it one more time to add an additional disk partition for MapR.

Adding the MapR Partition

The CentOS installer doesn't allow you to create an unmounted partition, so I used *parted* to add a new partition for MapR to use[6]. As stated here, "when creating a new partition on a device, said device must not be in use. For a device to not be in use, none of the partitions on the device can be mounted, and any swap space on the device must not be enabled. The easiest way to achieve this is to boot your system in rescue mode. When prompted to mount the file system, select *skip*." Accordingly, boot the NUC in rescue mode, using the same USB drive. Once again, beware: the NUCs don't always seem to want to boot off the USB drive. If it doesn't, you won't be able to modify the partitions on your hard drives, so try again. The steps that should work are these:

- Press F10 for Boot Menu.
- Select the USB Boot Drive
- Select Troubleshooting
- Select Rescue a CentOS system
- Select 3 – Skip to shell.

From the shell, create the partition for MapR:

```
parted /dev/sda
print
```

Use the output of the print command to specify the start point of the new partition – 71.6 GB in this case, but it might vary:

```
mkpart primary xfs 71.6GB 100%
print
quit
```

Now, shutdown the machine, pop the USB, and fire it back up. Verify proper setup using the *lsblk* command, which should now show something like this:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 489.1G  0 disk
├─sda1   8:1    0     1G  0 part /boot
├─sda2   8:2    0  15.7G  0 part [SWAP]
├─sda3   8:3    0    50G  0 part /
└─sda4   8:4    0 422.4G  0 part
```

Network Configuration

The primary decision to make with regards to connecting the NUCs is whether to use wireless or wired connections. The NUCs do have built-in Wireless-AC 8260 (see the spec page). Network speeds on the router will vary by model. I ended up going with wired connections from the NUCs to a wireless router, reserving the Wi-Fi for mobile clients only. My intent is to bring this briefcase cluster to demonstrations and conferences, and I didn't want to worry about weak signals, signal interference, dropped connections, etc. Either way, avoid DHCP and manually assign a set of static IP addresses instead – you don't want the IP addresses of the MapR nodes to change unexpectedly.

The *eno1* network interface is not enabled by default – you can verify this using the "*ip a*" command. Open the file */etc/sysconfig/network-scripts/ifcfg-eno1* and set **ONBOOT=yes**. Then run *systemctl*

[6] An easier approach (untested) may be to create the partition using the installer, and then unmount it for MapR.

restart network. At this point, recheck "*ip a*" and test the connection from your laptop. Upon successful connection, can ditch the wired USB keyboard and external monitor and switch to using a terminal window from your laptop (much easier).

Update the hostname, which the installer left set (?) to *localhost.localdomain*:[7]

- *hostnamectl set-hostname edgeX.mapr*

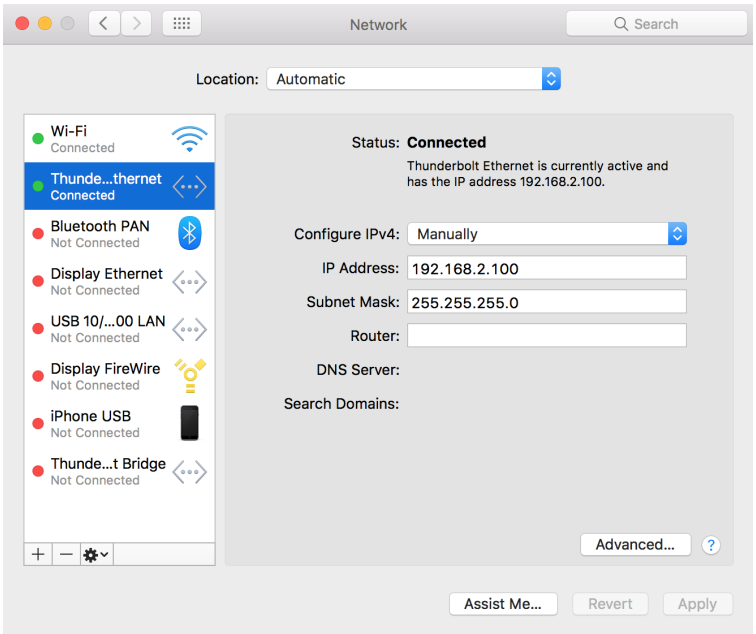
Add the following lines to the /etc/hosts file (these are the hardwired IP addresses):

```
192.168.2.21    edge1.mapr       edge1
192.168.2.22    edge2.mapr       edge2
192.168.2.23    edge3.mapr       edge3
192.168.2.100   macbook
192.168.2.50    maprdemo.local   maprdemo
```

Connecting a Laptop

Once you have IP configured on a NUC, it's much more convenient to work off a laptop:

- Using a Thunderbolt2-to-Ethernet adapter, connect your laptop to the switch.
- Configure the new Ethernet port with a static IP address on the same subnet as the NUC hardware:
    - Open the Network settings, in System Preferences.
    - Select the Thunderbolt Ethernet service on the left. Note that this will not appear in the list until you have attached the adapter or configured one previously.
    - Set "Configure IPv4" to "Manually", define an IP address, and set the subnet mask to 255.255.255.0. The configuration screen should look similar to the following:



- Modify your /etc/hosts file to include the (three) IP addresses of the NUCs, as well as an IP address of the Thunderbolt-to-Ethernet adapter. For example:

```
#Edge Demo
```

[7] An overview of *hostnamectl* can be found here.

```
192.168.2.21    edge1.mapr      edge1
192.168.2.22    edge2.mapr      edge2
192.168.2.23    edge3.mapr      edge3
192.168.2.100   macbook
```

- Test the connectivity:
    - From the MacBook, connect to the Sandbox VM and each of the NUCs using SSH.
    - From the Sandbox VM, ssh to each of the NUCs.
    - From each of the NUCs, ssh to the Sandbox VM.

Installing MapR

Using the Installer

After you've installed and configured the OS on all three NUCs, you're ready to install MapR.

MapR installation is easiest when there's an Internet connection. The NUC's wireless card is useful for this purpose, as it can connect to your home network[8]. Instructions for configuring Wi-Fi were included in the CentOS installation instructions. If configuring after the fact, see "Setting up Wireless" in the Lessons Learned.

Installation is quick and painless using the GUI Installer. When installing the Installer (*mapr-setup.sh*), note that **the MapR UID/GID in the 6.0 Sandbox is 2000/2000**. Specify the same UID/GID when the installer prompts you to create the MapR user. **This is important for replication, where the UID/GIDs must be the same in both clusters.**

Use the following settings in your installation:

- MapR Version: 6.0.1
- Edition: Converged Enterprise Edition[9]
- License Option: Install Trial License
- MEP Version: 5.0.0
- Enable MapR Secure Cluster: **No**
- Auto-Provisioning Template: MapR-XD (i.e. MapR core only, no additional services)
- Monitoring – Metrics: No
- Monitoring – Logs: No
- Cluster Name: **edge1.enterprise.org**
- Nodes: **edge1.mapr**, **edge2.mapr**, **edge3.mapr**
- Disks: **/dev/sda4**
- Login Method: SSH – Password

If the Installer issues any warnings about DNS inconsistencies, it may be due to your internet provider's DNS servers. Refer to the Lessons Learned section for some troubleshooting tips.

For future reference, the node layout chosen by the Installer looks like this:

---

[8] Another option is to enable Internet Sharing on the Mac, sharing the internet through the Thunderbolt Ethernet connector.

[9] The basis for this demo is cross-cluster replication, which requires an enterprise license to function.

Node Layout

Node Id	Node Count	Services
edge[1-2].mapr	2	librdkafka, CLDB, Streams Java Client, NFS, Apiserver, Zookeeper, Core Services, Webserver, File Server
edge3.mapr	1	librdkafka, CLDB, Streams Java Client, NFS, Zookeeper, Core Services, File Server

Verify Licensing

The basis for this demo is cross-cluster replication, which requires an enterprise license to function. Use the MCS to verify that your sandbox has a valid Enterprise Edition license.

Install the Replication Gateways

The **replication gateways** have to be installed manually. The instructions are here, and then Adding a Role is here, and finally the *configure.sh* reference is here. In summary, run the following on each NUC[10]:

```
yum install mapr-gateway
/opt/mapr/server/configure.sh -R -N edge1.enterprise.org
systemctl restart mapr-warden
```

Configure Audit Streaming

Audit streaming is used to automatically detect, at the Edge cluster, the establishment of stream replication from the HQ cluster. The primary goal of the demo is to convey how simple MapR makes it to build a data fabric with cross-cluster connectivity. To get that point across, it seemed best to use the MCS (although *maprcli* commands might also work for a targeted audience). It also seemed most intuitive for audiences to see a blank screen suddenly start to fill with data once replication was established. The Edge dashboard application, however, cannot subscribe to a stream that does not yet exist (the stream object – i.e. the stream replica – is created automatically by MapR as part of the autosetup process). The solution was to listen on the audit stream for the creation of this replica, after which it was trivial to subscribe to that newly-created stream and immediately start pushing data to the dashboard. Not only is this pretty cool, but it's also a chance to highlight another MapR capability: streaming audit data. (*maprcli*, it should be noted, provides an option to use a pre-existing stream replica object if one already exists. This could also have solved the problem, but you can't use the MCS.)

Streaming Audit Logs is here in the docs, where it describes the format of the stream to listen on, as well as a sample consumer application. First, though, you have to enable audit streaming in the Edge cluster (note this feature was introduced in 6.0.1, and therefore isn't present in the 6.0.0 Sandbox). Then you have to enable auditing of specific data access operations on the specific volume and directory that you wish to audit. The dashboard application (i.e. the Audit Listener Service) is coded to listen for the DB_REPLICAADD operation.

[10] Only one gateway is required, but multiple gateways is the recommended approach, and potentially useful for demonstration purposes.

The exact commands used for the demo have been incorporated into the *install-demo.sh* script on the Edge cluster, so you don't have to worry about setting it up. But for future reference, the commands used are as follows:

```
# Enable audit streaming on the Edge cluster.
# This creates the stream:
#  /mapr/edge1.enterprise.org/var/mapr/auditstream/auditlogstream
# Previously, the directory exists but the stream file does not.
maprcli config save -values '{"mfs.enable.audit.as.stream":"1"}'

# Enable auditing of filesystem and table operations
maprcli audit data -enabled true -retention 1

# Enable auditing on the mapr.apps volume
maprcli volume audit -name mapr.apps -enabled true -dataauditops
   +create,+delete,+tablecreate,-setattr,-chown,-chperm,-chgrp,-getxattr,-listxattr,-setxattr,-
   removexattr,-read,-write,-mkdir,-readdir,-rmdir,-createsym,-lookup,-rename,-createdev,-
   truncate,-tablecfcreate,-tablecfdelete,-tablecfmodify,-tablecfScan,-tableget,-tableput,-
   tablescan,-tableinfo,-tablemodify,-getperm,-getpathforfid,-hardlink

# Verify:
maprcli volume info -name mapr.apps -json

# Enable auditing on the directory where the replica is to be created
hadoop mfs -setaudit on /apps/pipeline/data

# Verify. Here, an "A" in the second column indicates it is being audited.
hadoop mfs -ls /apps/pipeline
```

When audit streaming is set up correctly, creation of the /apps/pipeline/data/replicatedStream replica will be detected. On the Edge Dashboard, HQ Replication will show a status of ESTABLISHED, and the Data Feeds and Services window will indicate that the *Upstream Comm Service* has started.

Disabling Wi-Fi

The intent of the demo is to demonstrate connectivity to the Sandbox cluster via wired Ethernet. Wi-Fi was enabled as a convenience to facilitate installation. Now, Wi-Fi should be disabled:

- Ensure that you've downloaded everything external that you need on the Edge cluster. This includes the **MapR Gateway**.
- Change ONBOOT to ***no*** in the Wi-Fi configuration file on each NUC – e.g. */etc/sysconfig/network-scripts/ifcfg-<network>*
- Reboot the machine.
- Verify via *maprcli* that MapR is not trying to use the Wi-Fi interface:
  `maprcli node list -columns hn,svc`

Failing to take this step will cause problems, as MapR will still try to communicate with other nodes over this interface.

You can also disable Wi-Fi by deleting the interface using the *nmtui* utility, as described here.

Monitor Notes

VNC might be an Ubuntu thing? A VNC client lets you connect to a desktop that's been shared on another computer. Excellent VNC clients are available for every major Linux distribution and other operating system. Here.

Setting Up Wireless

I needed an Internet connection on my NUCs to install MapR, and wireless was a way to get that without disrupting my static IP configuration. I discovered that Wireless can be configured quickly and painlessly using the CentOS installer. But if you need to modify the wireless configuration after that point, here are some helpful lessons learned.

The NUC Technical Product Specification shows the following wireless capabilities:

- Intel® Dual Band Wireless-AC 8260
- 802.11ac, Dual Band, Wi-Fi + Bluetooth 4.2
- Supports Intel® Wireless Display 6.0 (WiDi)

Initial research on how to configure wireless on CentOS was somewhat alarming. You have to install additional drivers or firmware in order to activate wireless interfaces. Browsing and searching for a driver on the Intel web site (http://intel.com/wireless) leads only to Windows versions. The Linux Support for Intel Wireless Adapters web page has firmware for use with the Linux kernel.

CentOS instructions point you to NetworkManager to use your wireless device, which I haven't used before. The following command shows NetworkManager is already running on the NUC:

```
service NetworkManager status
systemctl status NetworkManager
```

If it isn't running, here's how to enable it. But it's primarily a GUI tool, and I didn't install the GUI for CentOS. Here's a page on installing the GNOME Desktop, launching it, and configuring it to start in graphical mode when you boot the OS.

But, ***here's the easy way***. First, quickly identify the Ethernet cards on the machine using any of the following:

```
nmcli d
ifconfig
ip link show
```

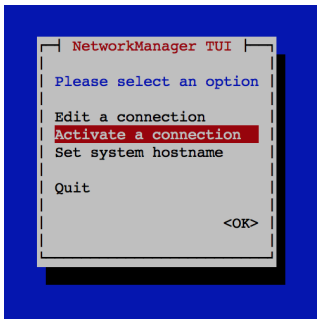
Sample output from a NUC1 identifies the wireless device as wlp3s0:

```
DEVICE  TYPE      STATE         CONNECTION
eno1    ethernet  connected     System eno1
wlp3s0  wifi      disconnected  --
lo      loopback  unmanaged     --
```

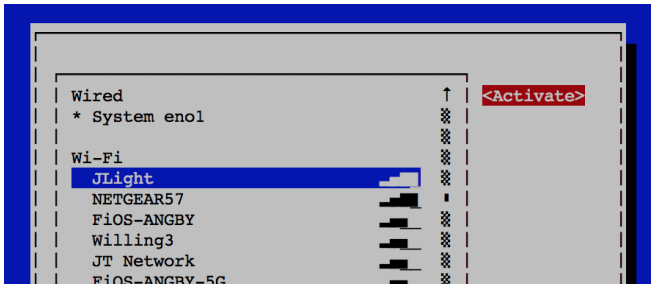
Then, this page has a way to use a "GUI" Network Manager without the GUI:

```
nmtui
```

I went to "Activate a Connection":



It lists the available WiFi networks. I chose my network and hit *Activate*:



Enter the encryption key, and it connects! Verify the IP settings using *ifconfig* or *"ip a"*. It also looks like using the NetworkManager in this fashion did not disrupt my manual setup of the wired Ethernet NIC.

Sandbox Setup

Base Version

The starting point for this configuration is the VirtualBox edition of the MapR 6.0.1 Sandbox for Hadoop. This sandbox is running CentOS 7.3[11]. **Note:** the 6.0.1 Sandbox does not include an Enterprise license, which is required to demonstrate replication.

Network Configuration

Adding a Network Adapter

This sandbox comes pre-configured with a single NAT network adapter, allowing connectivity from the host machine only via Port Forwarding. For example:

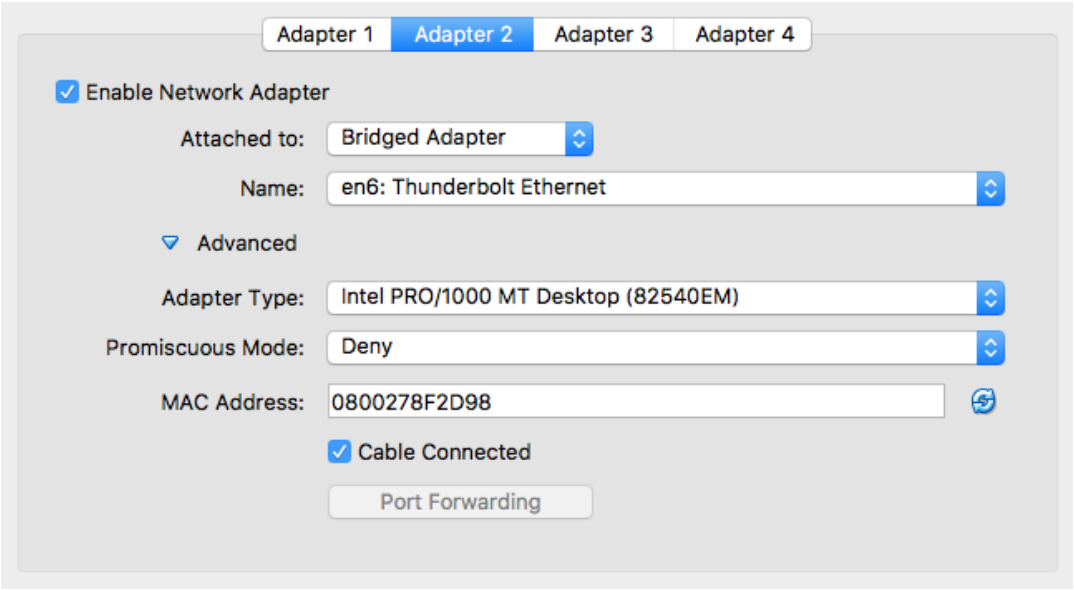
```
ssh -p 2222 mapr@localhost
```

Add a second, bridged adapter, connected to the Thunderbolt Ethernet device – "en6: Thunderbolt Ethernet" on my MacBook. This will be configured to use a static IP address within the bridged network on which the Edge cluster resides. Within the VirtualBox network settings for this VM (and with the VM shut down) Adapter 2 should look like this:

- **Enable** Network Adapter (checked)
- **Attached to:** Bridged Adapter
- **Name:** en6 – Thunderbolt Ethernet

[11] Verified using: `rpm --query centos-release`

Accept the defaults for Adapter Type, Promiscuous Mode (Deny), and MAC address, and leave the Cable Connected option checked. Take note of the MAC Address, which is used later to configure the device within CentOS.



With the second adapter configured *and the Thunderbolt Ethernet adapter connected*, the sandbox image can be started. It's simplest to continue using the NAT adapter to *ssh* into the virtual machine (i.e. *ssh -p 2222 root@maprdemo*), because this connection survives when the cable is disconnected.

Configuring the New Adapter

Shut down MapR:

```
systemctl stop mapr-warden
systemctl stop mapr-zookeeper
```

Within the */etc/sysconfig/network-scripts* directory is a file called *ifcfg-enp0s3*, corresponding to the NAT adapter. Run *ifconfig* to verify that the new bridged adapter is called *enp0s8*, and it currently has no IP address. Copy the *ifcfg-enp0s3* to a new file called *ifcfg-enp0s**8*** and replace the contents of that new file with the following. HWADDR comes from the MAC address provided by VirtualBox for the new adapter (you have to add the colons). *192.168.2.50* must be an available address within the same subnet as the Edge cluster.

```
NAME="enp0s8"
DEVICE="enp0s8"
ONBOOT="yes"
BOOTPROTO="static"
IPADDR=192.168.2.50
NETMASK=255.255.255.0
HWADDR=08:00:27:18:D6:38
```

Add the following lines to */etc/hosts* to provide visibility to the Edge NUCs:

```
127.0.0.1       localhost
192.168.2.21    edge1.mapr      edge1
192.168.2.22    edge2.mapr      edge2
192.168.2.23    edge3.mapr      edge3
```

```
192.168.2.100   macbook
192.168.2.50    maprdemo.local  maprdemo
```

Restart the network and test the new settings. With both a NAT adapter and a Bridge adapter, you should be able to ping everything: the local machine, the MacBook Thunderbolt Ethernet Adapter, the Edge nodes, and the outside world.

```
systemctl restart network
ping 192.168.2.50
ping maprdemo
ping macbook
ping edge1 (edge2, edge3)
ping edge1.mapr (edge2.mapr, edge3)
ping google.com
```

From your Mac, verify connectivity through the new adapter using the static IP address instead of localhost:

```
ssh root@maprdemo.local
```

From an edge node, verify you can *ping* and *ssh* to the sandbox, using hostname only (*maprdemo.local*).

Reboot the virtual machine and verify that the networking changes persist.

Verify that MapR is still operational on the sandbox, using *maprcli*, *hadoop fs*, nfs, and the MCS.

Verify Licensing

The basis for this demo is cross-cluster replication, which requires an enterprise license to function. Use the MCS to verify that your sandbox has a valid Enterprise Edition license.

Install the Replication Gateway

As with the Edge Cluster, the **replication gateway** has to be manually added to the sandbox. Refer to "Install the Replication Gateways" in the Edge Cluster section (using *dc1.enterprise.org* as the cluster name for the configure statement).

It's unclear whether you need to run *configure.sh* after installing a gateway.

Configuring Cross-Cluster Communication

Now that both the Edge cluster and the Sandbox cluster are up and running, cross-cluster communication can be established.

Setting up Cross-Mirroring Between Unsecure Clusters

The easiest way to verify cross-cluster data replication is to use mirroring, which doesn't require the installation or configuration of any additional components (i.e. gateways). Successfully mirroring data across sites would validate network connectivity and licensing.

Setting up Cross-Mirroring Between Clusters is documented here in the MapR docs. Example mirror creation commands can be found here. On each cluster, simply add an entry for the remote cluster to *mapr-clusters.conf* file, then restart NFS and the APIserver. The two files look like this:

/opt/mapr/conf/mapr-clusters.conf (SANDBOX):

```
dc1.enterprise.org secure=false maprdemo:7222
edge1.enterprise.org secure=false edge1.mapr:7222 edge2.mapr:7222 edge3.mapr:7222
```

/opt/mapr/conf/mapr-clusters.conf (EDGE1, EDGE2, and EDGE3):

```
edge1.enterprise.org secure=false edge1.mapr:7222 edge2.mapr:7222 edge3.mapr:7222
dc1.enterprise.org secure=false maprdemo:7222
```

To restart the services (do this on each cluster):

```
maprcli node services -name apiserver -action restart -nodes <maprdemo | edge1,edge2>
maprcli node services -name nfs -action restart -nodes <maprdemo | edge1,edge2,edge3>
```

To verify:

- On the Edge cluster, as the MapR user, create a test source volume and a test file:

  ```
  maprcli volume create -name mirrorVol1 -path /mirrorVol1
  echo "This is a test" > /mapr/edge1.enterprise.org/mirrorVol1/testfile
  ```

- Back on the Sandbox, create a mirror volume:

  ```
  maprcli volume create -name mirror1FromEdge
       -source mirrorVol1@edge1.enterprise.org
       -path /mirror1FromEdge -type mirror
  ```

- Initiate the mirror:

  ```
  maprcli volume mirror start -name mirror1FromEdge
  ```

- Verify that files move over successfully.
- This can also be done in the reverse direction.

To remove the volumes (source and mirror) after a successful test:

```
maprcli volume remove -name
```

Setting Up Table and Stream Replication

Prerequisite: The MapR Gateway should have been installed on the Sandbox and all Edge nodes. This was a previous manual step.

Configuring the gateways is a simple maprcli command to tell the source cluster about the destination cluster's CLDB and gateway nodes. And it turns out that if you've updated the *mapr-clusters.conf* file, this step isn't even necessary if the gateways are on the CLDB nodes. To find out, run the following command on the Sandbox:

```
maprcli cluster gateway resolve -dstcluster edge1.enterprise.org
```

If *edge1.enterprise.org* is resolved, then it's probably not necessary to do anything else. The output should also indicate the source, likely to be *mapr-cluster.conf*, which agrees with the Managing Gateways page in the docs, where it says that MapR-DB and ES can find gateways using either DNS records, lists created via the *cluster gateway set* command, or the *mapr-clusters.conf* file.

Nevertheless, the instructions for configuring the gateways say to issue the following command:

```
# On the Sandbox:
maprcli cluster gateway set -dstcluster edge1.enterprise.org -gateways "edge1 edge2 edge3"

# On the Edge cluster:
maprcli cluster gateway set -dstcluster dc1.enterprise.org -gateways maprdemo
```

To verify:

```
maprcli cluster gateway list
```

Note that this command might return "No gateways configured", but as long as "maprcli cluster gateway resolve" returns the correct information, replication should work.

To delete the gateway entry:

```
maprcli cluster gateway delete -dstcluster edge1.enterprise.org
```

For more info, Configuring Gateways for Table and Stream Replication is here. Managing Gateways contains all the gateway administration commands. Configuring Gateways for Table and Stream Replication is here.

Installing the Front-End Apps

Although you could use this Edge cluster setup for any number of demonstrations, this specific demonstration is based on the Microservices Data Pipeline demo, which was originally developed to illustrate MapR as a platform for microservices-based architectures. That demo was refactored into three "editions". The first, representing the view of things at HQ, is to be run on the Sandbox. The second, representing the view of things at the Edge, is to be run on the Edge cluster. A third, representing a data science cluster collecting data from multiple sites, is run in the cloud. All editions are packaged into a single tar file, *microservices-dashboard-demo.tar*.

To install the application:

- Copy the tar file to the appropriate environment.
- Extract the tar file to the */home/mapr* directory. This is hardcoded in at least one place – the Jetty configuration files – so at this point it's not advisable to choose a different path.
- Run the *installDemo.sh* script, specifying the appropriate edition for your environment:
  `installDemo [hq|edge|cloud]`

Running the Apps

The apps can be run using the runDashboard.sh script, specifying the appropriate edition for your environment:

```
runDashboard.sh [hq|edge|cloud]
```

The app will let you know when to connect from a web browser. Note that the web app establishes a connection to the backend Vert.X server that's running as part of the app. If you restart the backend, be sure to reconnect your web browser or there won't be any communication. The Edge Dashboard will let you know when connectivity to the local server has been lost (the HQ app will not).

- Note: If Vert.x fails to start, it may be that your cluster has HBase Rest running on the same port. Shut down HBase Rest and restart the app.

Both apps should begin begin processing data immediately, resembling the screens shown back in Figure 1 and Figure 2. The data window within the edge dashboard will show nothing until stream replication or volume mirroring is established.

The dashboard URLs are as follows:

http://localhost:8080/dashboard/dashboardHQ.html

http://<edgenode>:8080/dashboard/dashboardEdge.html

The cloud edition does not have an associated dashboard. After running the app, use another window to 'tail -f' the specified output file. This will display messages being replicated from the HQ cluster.

Appendix A: Resources

Mirroring Essentials

Mirroring is here in the docs, and here in GDrive. Managing Mirrors is here.

- Mirroring from a lower version to a higher version is supported, but not the reverse.
- Mirroring restricts itself to 70% of available bandwidth. Throttling can be disabled to use all available bandwidth.
- In the case of a network partition, the mirror operation periodically retries the connection. Once the network is restored, the mirroring operation continues.
- Nodes in cluster A cannot have the same IP as nodes in cluster B.
- All servers in cluster A must be able to reach all servers in cluster B, and vice versa.
- The MAPR_USER uid/guid must be the same in both clusters.

Gateways and Stream Replication

Gateways are setup by installing the gateway on the destination cluster and specifying the gateway node(s) on the source cluster.

- Administering Gateways is here.
- Configuring Gateways for Table and Stream Replication is here.
- Managing Gateways contains all the gateway administration commands (starting and stopping the service, troubleshooting tips, etc.).

Networking

- https://askubuntu.com/questions/309461/how-to-disable-ipv6-permanently
- https://docs.oracle.com/cd/E37670_01/E41138/html/ol_about_netconf.html
- https://www.tummy.com/articles/networking-basics-how-arp-works/
- https://superuser.com/questions/1204229/bridge-virtual-interface-into-physical-network

Appendix B: Notes on Building the Dashboard Web UI

The dashboard UI is built using Bootstrap, JavaScript, and Vert.x (which handles messaging between the browser and MapR Streams).

Bootstrap

https://getbootstrap.com

http://getbootstrap.com/getting-started/

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites. It's open source.

This came from Tug. His car demo includes bootstrap.min.css, bootstrap.min.js, and jquery.*.js. That led me to the Bootstrap getting started page:

- This is the most basic form of Bootstrap: precompiled files for quick drop-in usage in nearly any web project.
- Requires JQuery, so that explains why Tug included it. Used for the JavaScript plugins, so if you just use the CSS part of Bootstrap, you don't need JQuery.
- Getting started page has a basic HTML template.

Tug included the Bootstrap css. That requires JQuery. Looking here: http://getbootstrap.com/getting-started/ . Also includes the boostrap.min.js, as described in that page.

I imported all the "resources" into my existing pipeline project. HTML would not display properly until I adjust all the links to the scripts and style sheets (changed all locations from "/etc" to "./etc". Not sure if this is correct.

Use https://www.w3schools.com/bootstrap to get started.

Vert.x

This demo uses Vert.x to push data from MapR Streams to the browser (i.e. the Dashboard) and vice versa. It is packaged in the demo tar file and installed using the demo *install* script. The following information is for reference only.

1. The download link is here. Download the full 3.3.3 distribution and unzip it (I unzipped it to the mapr home directory).
2. "/vertx/lib/*" is added to the classpath when launching the dashboard app. Note that Vert.x introduces jar conflicts with MapR (I think it's netty). The workaround I used was to reverse the classpath order on the execution command. For example, looking below I found that the second command works, whereas the first command does not:

   ```
   Does not work:
   java -cp `mapr classpath`:./vertx/lib/*:./ms-pipeline-1.0.jar <classname>

   Works:
   java -cp ./vertx/lib/*:`mapr classpath`:./ms-pipeline-1.0.jar <classname>
   ```

3. Vert.x requires Java 8 (which is now common). The sandbox should already have this, but ensure your dev environment has it, too, if you plan to further develop or compile the project. Java8 introduced Lambda Expresssions – Java's first foray into functional programming. It allows you create a function that doesn't belong to any class, which can be passed around as if it was an object and executed on demand.

Resources:

- http://vertx.io/

- Documentation.
- Download. If you are using Apache Maven, you don't need to download anything, as the required artifacts are retrieved by your dependency manager. But they need to be available in the runtime environment – for that, they have been downloaded and packaged with demo.
- The best description of what Vert.x gets you can be found here: https://mapr.com/blog/monitoring-uber-with-spark-streaming-kafka-and-vertx/
  - A Vert.x Kafka client verticle consumes messages from the MapR Streams topic and publishes the messages on a Vert.x event bus.
  - A Javascript browser client subscribes to the Vert.x event bus using SockJS and displays the Uber trip locations on a Google Heatmap.
- Also, from Carol MacDonald:
  - https://github.com/caroljmcdonald/vertx-kafka
  - https://github.com/caroljmcdonald/mapr-sparkstreaming-vertx-uberheatmap

Some musings regarding the jar conflict: From the Dependency Hierarchy (viewed in Eclipse, looking at the pom.xml file), it shows that vertx-core: 3.3.3 requires netty-common-4.1.5. So which one is MapR using? Community blog says 3.2.2. Says it's the mapr zookeeper jar. That's /opt/mapr/lib/zookeeper-3.4.5-mapr-1604.jar. But when I list the contents it doesn't show anything related to jetty… how did this guy figure out what version of netty it's using? And where is it getting it from? I did find the netty jars on my local machine. In the project properties (Eclipse), look at the Java Build Path (left) and Libraries (right). That shows the actual jars that are included in the build path, including netty 4.1.5. And the jars are all there (/Users/jjanos/.m2/…). So that explains how this is even working – Maven downloaded all the jars, put them in the build path, but they're all invisible. When I did a new Maven Install, the first thing it did was "Rebuild the Workspace" – i.e. download all the required jars. It also lists all the mapr jars I need, which means I should be able to specify this when I run it on the server.

AWS Rekognition

The Image Classifier service is implemented using AWS Rekognition. You simply send this service an image and it returns a set of labels. The service is not free – after some trial period, there is a fee per image processed. If you only run a few images through it for a demo, it amounts to nothing. But if you leave it running an entire day at a conference, the costs will soon add up. Pricing details can be found here. Also note that use of this demo feature and service requires an internet connection, so ensure that your sandbox has this required connectivity.

The AWS SDK has been bundled into the demo app, but you do need to ensure that an AWS account is set up and specified in the app configuration. The following information is for reference only.

Two things are required: the AWS SDK has to be downloaded to both your dev environment and the sandbox, and AWS credentials have to be obtained and passed into the demo app at runtime. Getting Started with Amazon Rekognition outlines 3 steps – Steps 1 and 3 are required for the demo ("Setting Up an AWS account" and "Getting Started Using API").

Creating the Rekognition User

Assuming you already have an AWS account, log in and access the IAM (Identity and Access Management) home page. You may see something that says, "Users: #". Click that link, and you get a list of users associated with your account. Click "Add User" and specify the following:

- User name: something like "rekognitionuser".
- Access Type: select "Programmatic access" and de-select "AWS Management Console access".

Click "Attach existing policies directly" to grant the new user access to Rekognition. Select "AmazonRekognitionFullAccess" from the list of Policy names and click "Review". Then click "Create user". At this point, you will see a Success screen with your new user listed along with their Access key ID and Secret access key. You need both of these values – either cut and paste them or download them as a csv file. These keys must be specified in the application's *config.properties* file, or alternatively passed in as Java system properties ("-Daws.accessKeyId=<key> -Daws.secretKey=<key>" when running the demo app.

Setting Up the SDK

Following Amazon's best practices, this demo was built using the Amazon SDK, not the HTTP Rekognition interface. Amazon provides instructions for downloading and installing the SDK here. This demo was built using Eclipse and Apache Maven – Amazon's instructions for using the SDK with Maven can be found here. The easiest approach was to import the entire SDK, which simply required adding the following dependency to the project's *pom.xml* file:

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk</artifactId>
  <version>1.11.139</version>
</dependency>
```

The SDK also has to be set up on the sandbox. Following Amazon's instructions found here, download the sdk and unzip the file to "*<project-base>*" (so you'll see "<project-base>/aws/lib", etc). Both "/aws/lib/*" and "/aws/third-party/lib/*" need to be in the classpath of the pipeline execution command. The AWS SDK also introduces jar conflicts with MapR, and so the order in which the jar files appear in the classpath is, once again, important.

Other image classifiers include:

- Google
- opencv.org/about.html
- deeplearning4j.org
- http://mxnet.io/tutorials/python/predict_image.html
- http://aiplaybook.a16z.com/docs/guides/vision

Jetty

The Jetty download is included in the demo tar file. The *install* script automatically extracts it to *<project-base>/extracted* and handles all necessary project configuration. The demo *run* script starts it automatically using port 8080. The following information is for reference only.

The download site for Jetty is here, and download instructions are here. To install, just unpack it to a directory. I downloaded the .tgz file for version 9.4.5 and extracted it via "tar –xvf".

Jetty has to be configured to serve static content (this is also done by the install script). This is used by the Dashboard application to serve imagery files as well as the application HTML and js scripts. Configuration files resembling the following are placed in the *<jetty-home>/webapps* directory:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN"
"http://www.eclipse.org/jetty/configure_9_0.dtd">
<Configure class="org.eclipse.jetty.server.handler.ContextHandler">
  <Set name="contextPath">/images</Set>
  <Set name="handler">
    <New class="org.eclipse.jetty.server.handler.ResourceHandler">
      <Set
name="resourceBase">/mapr/dc1.enterprise.org/apps/pipeline/data/nasa/image_files</Set>
      <Set name="directoriesListed">true</Set>
    </New>
  </Set>
</Configure>
```

If jetty was already started, it has to be restarted to register this modification.