Container Setup

Updated 5/02/2018

Overview

MapLarge offers container support on Linux via Docker images, which can be hosted stand-alone or within an orchestration provider such as Kubernetes. Containers simplify on-premise and cloud based solution delivery while retaining a high degree of flexibility while offering efficient management of computing resources.

Obtaining Images

Currently, MapLarge offers docker images using private repositories on Docker Hub (https://hub.docker.com/). In order to access the repository, a Docker Hub ID will need to be created and MapLarge must grant the user access rights. Email a MapLarge representative with the Docker Hub ID and access rights will be granted.

Installation Procedures

Installation procedures vary slightly based on the target operating system. At this time, MapLarge offers Linux Docker images. Windows support is coming soon.

Linux

This document uses the Amazon Linux AMI using an *t2.xlarge* instance type, as an example; however the process is very similar on other distributions (https://docs.docker.com/install/).

Install Docker if Needed

```
# install docker
# this example is for RHEL-based distributions.
# on debian-based distros, run "sudo apt-get install docker.io" instead
sudo yum install docker

# allow users in the "docker" group to access the docker socket file
# note: some distributions (CentOS) name this group "dockerroot" instead
echo '{ "group": "docker" }' | sudo tee /etc/docker/daemon.json

# start docker service
sudo service docker start

# add the non-root user to docker group
# note: some distributions (CentOS) name this group "dockerroot" instead
sudo usermod -aG docker ec2-user

# log out of user, and log back in to pick up group memberships

# test to verify that the docker daemon is running
```

```
# and user has access. Should return an empty list.
docker ps
```

Install MapLarge Server

```
# create credentials on https://hub.docker.com & create ID
# now, log in with the credentials
docker login
<enter credentials>

# get the maplarge server package
docker pull maplarge/server:v2

# run the server
docker run \
        -p 80:80 \
        -p 443:443 \
        -v maplarge_volume:/opt/maplarge/server/App_Data \
        maplarge/server:v2

# test out the server
# note: it generally takes about 30-60 seconds
# to spin up the new MapLarge instance
http://ec2-54-144-119-171.compute-1.amazonaws.com/
```

Customizing the MapLarge Server Image

Using Environment Variables

MapLarge exposes common configuration options using environment variables. They can be specified using the "-e" argument to "docker run".

Variable Name	Default Value	Description
ML_ALL_SSL	false	All AJAX requests will be made using SSL
ML_MASTER_SERVICE_HOST		URL to the MapLarge master node (ServerSync cluster only) This variable is typically set by the orchestration container; However it can be set manually in a docker-only cluster. (http://1.2.3.4)
ML_NO_SSL	true	No AJAX requests will be made using SSL
ML_REPL_ENABLED	false	Set to true if the server is part of a cluster
ML_REPL_GREEDY	False	Sets the consumer mode of a clone

ML_REPL_NICE_NAME		Friendly name of a node in the MapLarge cluster
ML_REPL_PASS		Set the password used by a clone to contact the master
ML_REPL_SELF_URL		Optional URL used by the master to connect back to a clone (http://1.2.3.4)
ML_REPL_USER		Set the username used by a clone to contact the master
ML_ROOT_PASS		Set the root password for the node (will be replaced with another method in the future)
ML_SERVERS_MAIN		Set the URL (typically a load balancer in production) to which the client should make ajax requests.

Using a Dockerfile

In the Docker ecosystem, the typical way to customize an image is to build a new "layer" on top of it. MapLarge recommends that ANY changes outside of the provided environment variables be made using a Dockerfile.

The MapLarge server image is configured to mount a Docker Volume for the App_Data folder. Therefore, it is possible to share configuration (and data!) changes between builds of the MapLarge image, and any customized layers built on top of the default.

Adding custom SSL certificates is one common reason to customize the MapLarge server image. SSL certificates are DNS-name dependent, so they always vary from site to site (and often from host to host as well). This example addresses the SSL use case.

```
# create and change to a project directory
mkdir acme_maplarge
cd acme_maplarge/

# get the certificate and private key in PEM format
cp /tmp/my_private_key.key .
cp /tmp/my_server_cert.crt .

# edit the ML js.js file (contents purple)
vi js.js
ml.allssl=true;
ml.servers.main=[{"domain":"mydomain.net", "prefix":[0,1,2,3,4,5,6,7]}];
```

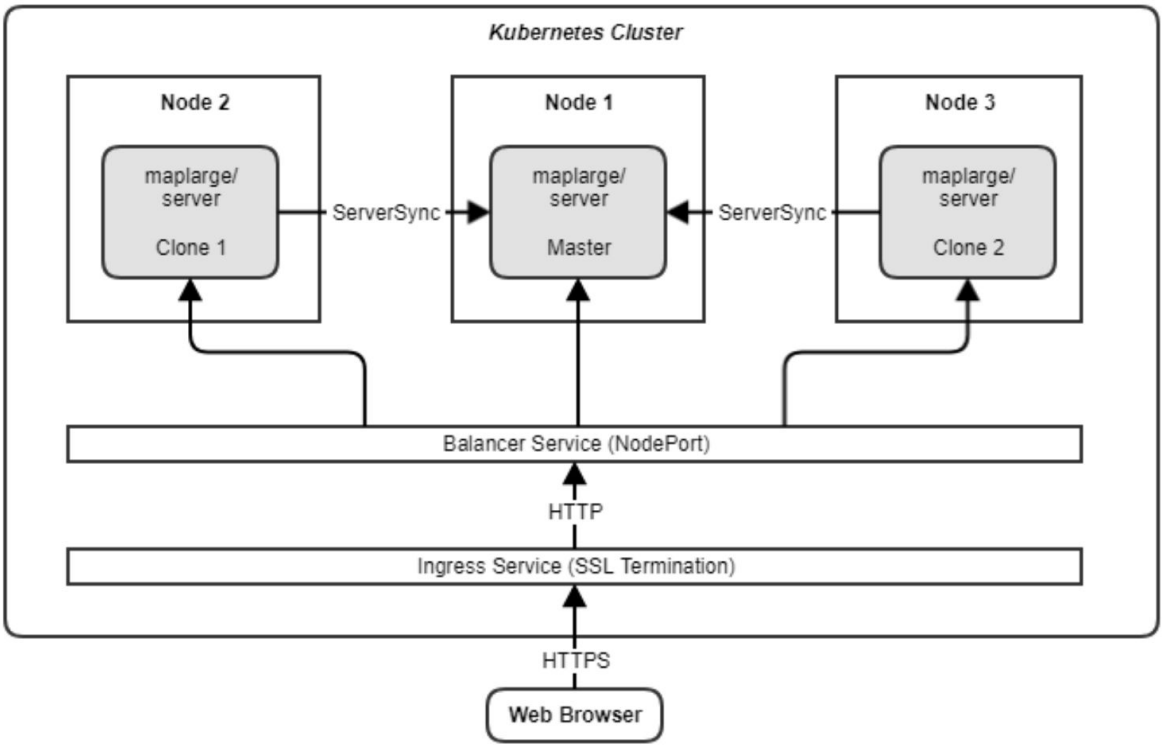
```
# edit the docker file (contents purple)
vi Dockerfile
FROM maplarge/server:v1

COPY my_private_key.key /etc/pki/tls/private/localhost.key
COPY my_server_cert.crt /etc/pki/tls/certs/localhost.crt
COPY js.js /opt/maplarge/server/App_Data/js.js

#build the new image
docker build -t acme/maplarge_ssl:v1 .

# run the new image (inherits exposed ports, etc. from maplarge/server)
docker run \
        -p 443:443 \
        -v maplarge_volume:/opt/maplarge/server/App_Data \
        acme/maplarge_ssl:v1
```

Docker Notes

- MapLarge is using Docker Volumes for storing the App_Data hierarchy. By default, volumes are stored under the **/var/lib/docker/volumes** directory using the "local" driver. In a cloud deployment, it may make sense to use another driver type which utilizes a cloud key / value store such as Azure's blob storage.
- There is a know issue with the "overlay" filesystem type used by Docker and the XFS file system. If XFS is not formatted using ftype=1, overlay will not work properly. The work around is to format with ftype=1, or use another filesystem type such as ext4. This applies to the partition containing the **/var/lib/docker** directory.

Kubernetes Setup

It is possible to configure a MapLarge cluster in Kubernetes using the docker image described in previous sections. Currently, this configuration is tested on Google Cloud. The setup can be performed either in Google cloud console, or in a local console which has been configured with "gcloud" and "kubectl". Instructions for configuring the shell can be found here:
https://cloud.google.com/kubernetes-engine/docs/quickstart



Prerequisites

Before getting started, create a Google Cloud project with appropriate quota sizes. MapLarge is a in-memory database, so nodes in the kubernetes cluster should be sized accordingly (at least 64-256GB per node in production is recommended, depending on the the workload). This translates to using n1-standard-16 up to n1-standard-96 type instances, or alternatively the n1-highmem-8 to n1-highmem-64 instances, if using standard pricing. It is also possible to build custom sized nodes in Google Cloud. Please see this link for pricing details in your region:
https://cloud.google.com/compute/pricing

Please note - if trying out the MapLarge Kubernetes installation using a free trial or new Google Cloud account, there are quota limitations. Quotas can be expanded by submitting a request to Google.

Recommended minimum quota for a test project:

Global Backend Services: > 10
Regional CPUs: > 80
Regional Disk GB: > 8192 (8TB)
Regional In-Use Addresses: 10

Access to the GitHub repository maplarge-kubernetes-demo is also necessary for this example. Your MapLarge support representative can grant access to your GitHub ID.

Cluster Creation

In the project main menu, select **Kubernetes Engine → Kubernetes Clusters → Create Cluster**.

Here are example settings:

Setting	**Value**	**Remarks**
Name	ml-demo-cluster	
Zone	us-central1-a	Choose regions and zones based on locality and availability preferences
Cluster Version	1.8.8-gke.0 (default)	
Machine Type	n1-highmem-8	8 vcpu, 52GB memory
Node Image	Container-Optimized OS (cos)	
Size	3	Can be adjusted later unlike machine type, or can use auto scaling
Advanced / Auto Scaling	On	If you would like to automatically add nodes to the cluster based on resource requests from Kubernetes, turn this on.
Advanced / Minimum Size	3	
Advanced / Maximum Size	5	

Click **Create**.

MapLarge Setup

Click the **Activate Google Cloud Shell** button at in the top right of the console.

Get credentials for the cluster just created:

```
# replace ml-demo-cluster with the name of the cluster created above
# replace us-central1-a with the name of the zone you selected during cluster
creation
# replace maplarge-kubernetes-demo with the name of the project
gcloud container clusters get-credentials ml-demo-cluster --zone us-central1-a
--project maplarge-kubernetes-demo
```

Clone the MapLarge example GitHub repo for Kubernetes setup:

```
git clone
cd maplarge-kubernetes-demo
```

Create a secret with your Docker credentials (which should have access to the maplarge Docker repos, see above):

```
kubectl create secret docker-registry ml-docker-secret
--docker-server=https://index.docker.io/v1/ --docker-username=johndoe7
--docker-password=pass@word1 --docker-email=john.doe@acme.com
```

Choose a DNS name for your installation. For example, **acme.maplarge.com**. Edit master-ss.yaml and clone-ss.yaml to use this DNS name in the ML_SERVERS_MAIN environment variable:

```
- name: ML_SERVERS_MAIN
  value: '[{"domain":"maplarge.acme.com","prefix":[0,1,2,3,4,5,6,7]}]'
```

Create the master stateful set:

```
kubectl create -f master-ss.yaml
```

Create the clone stateful set:

```
# note:  This is optional.  On a smaller cluster,
#you can do a single node install with just the master.
kubectl create -f clone-ss.yaml
```

Create a static IP address:

```
gcloud compute addresses create ml-static-ip --global
```

Find out the IP address:

```
gcloud compute addresses list
NAME                                          REGION  ADDRESS          STATUS
k8s-fw-default-ml-ingress--bd85044dfb23c5d5           35.186.235.194   IN_USE
ml-static-ip                                         35.190.1.162     RESERVED
```

Configure the DNS server to resolve the hostname chosen above, and 8 subdomain prefixes to the IP created above. For example, if the IP is 10.10.10.10, the DNS configuration might be:

```
10.10.10.10 maplarge.acme.com <A Record>
```

```
10.10.10.10 0maplarge.acme.com <CNAME>
10.10.10.10 1maplarge.acme.com <CNAME>
10.10.10.10 2maplarge.acme.com <CNAME>
10.10.10.10 3maplarge.acme.com <CNAME>
10.10.10.10 4maplarge.acme.com <CNAME>
10.10.10.10 5maplarge.acme.com <CNAME>
10.10.10.10 6maplarge.acme.com <CNAME>
10.10.10.10 7maplarge.acme.com <CNAME>
```

Set up the balancing service and ingress:

```
kubectl create -f ml-service.yaml
kubectl create -f ml-ingress.yaml
```

It will take a few minutes for the ingress service to be ready. Access the MapLarge cluster via the configured DNS name should now be available.

TLS Setup

This example demonstrates TLS termination at the ingress point, which is the recommended configuration. It is also possible to configure TLS on each container instance, and dispense with the ingress controller (i.e. the ml-service would be publicly accessible and responsible for load balancing, while the individual ML instances would be responsible for TLS. This configuration is slightly harder to maintain, but would avoid the extra ingress component.

First, obtain the key and certificate in PEM format.

Next, generate a secret containing the certificate details:

```
kubectl create secret tls ml-tls-secret --key /path/to/tls.key --cert
/path/to/tls.crt
```

Edit the ingress YAML file to use the secret (uncomment the “tls:” and “- secretName:” lines):

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ml-ingress
  annotations:
    kubernetes.io/ingress.global-static-ip-name: ml-static-ip
spec:
  tls:
  - secretName: ml-tls-secret
  backend:
    serviceName: ml-service
    servicePort: 80
```

Edit the master-ss.yaml **and** clone-ss.yaml to enable SSL in the MapLarge configuration, setting ML_NO_SSL to false and ML_ALL_SSL to true:

```
env:
 - name: ML_REPL_ENABLED
   value: "true"
 - name: ML_REPL_MODE
   value: "master"
 - name: ML_NO_SSL
   value: "true"
 - name: ML_ALL_SSL
   value: "false"
```

Re-deploy the cluster using the provided shell scripts
./destroy.sh
./create.sh

After a few minutes, the ingress should be ready and access to the MapLarge instance over TLS should be available.

- Note 1: It is not necessary to delete the ingress after every change. The “kubectl apply” command can also be used to update its properties.
- Note 2: When you run ./destroy.sh, or use kubectl to remove the StatefulSets, the underlying storage is NOT destroyed. In order to remove the underlying storage, use “kubectl delete pvc …”
- Note 3: For changes to StatefulSets, it is not necessary to delete and recreate the ml-service or ml-ingress. Just delete and recreate the stateful set.

Scaling the Cluster

The MapLarge cluster can be scaled by adding or removing clone containers from the ml-clone-ss scaleset. This is done using the following command:

```
kubectl scale statefulsets ml-clone-ss --replicas=N
```

Where N is the desired number of clones. If auto-scaling is enabled for the Kubernetes cluster, Google Cloud will add and remove physical nodes (up to the configured min / max) in order to satisfy the resource requirements of a scaling request.

In general, a good practice is to configure resource requests for the MapLarge containers such that a single container ends up being mapped to a single node (VM). This can be accomplished by setting the memory resource request for the container to greater than half the available RAM in the cluster node configuration. Please note that is is not a requirement; however it does more closely match the standard non-containerized

MapLarge deployment topology and makes sizing comparisons easier to a “bare metal” MapLarge cluster. It also makes accidental mis-configuration and resource starvation less likely.

Upgrade Clusters MapLarge Software

In order to deploy a new version of MapLarge API Server software, the cluster is destroyed and re-created. MapLarge provides the *destroy.sh* and *create.sh* scripts which automate the process. When run, the cluster is re-created using the latest Docker image.

Re-deploy the cluster using the provided shell scripts:

./destroy.sh
./create.sh

After a few minutes, the ingress should be ready and access to the MapLarge instance over TLS should be available.

Notes:

- It is not necessary to delete the ingress after every change. The “kubectl apply” command can also be used to update its properties.
- When you run ./destroy.sh, or use kubectl to remove the StatefulSets, the underlying storage is NOT destroyed. In order to remove the underlying storage, use “kubectl delete pvc …”
- For changes to StatefulSets, it is not necessary to delete and recreate the ml-service or ml-ingress. Just delete and recreate the stateful set.