Hosting & Monitoring solution with: Icinga2 – MySQL – Apache – AWS S2 – AWS

2016-05-18
Crossover

Overall Objective

This procedure explains how to perform a monitoring solution using Icinga to monitor an Apache Web Server and a Mysql Database server and send logs to Amazon S3 dynamically using Bash Scripting.

Functional Requirements

This document will go through the process of setup and configuring the following infrastructure:

1. Download, install and configure Icinga solution on the server.
2. Troubleshoot any system issues to ensure availability of services
3. Install Apache web server and a Mysql Database on different Docker containers
4. Ensure that all logs that are generated by the Apache Web Server and Mysql Database are collected dynamically through a Bash Script
5. Those logs should be automatically sent to Amazon s3 at 7 pm daily
6. Ensure proper backups are optimally taken and sent to Amazon S3 bucket.
7. Write a Chef Recipe (Puppet Manifest or Ansible Playbook) to automate this process

PHONE EMAIL
Adolfo Alsina 3037, Buenos Aires, Capital Federal 1207 +5491159577244 palacosfr@gmail.com

What resources will we use?

This list shows what technologies and resources will be used on the process:

- AWS.amazon.com – EC2 server and S3 storage instances will be used.
- Ubunt – We are going to start of with an Ubuntu image from AWS.
- Amazon EC2 – The complete infrastructure will be hosted on an Amazon Virtual Server.
- Amazon S3 – Backups will be uploaded to the Amazon Storage
- Docker - Each of the monitoring web and database servers will be build as containers.
- Docker-Compose – Some of the testing and automation will be performed with Compose
- MySQL Container – Pre configured and easy to use MySQL instance.
- Puppet – Container construction and configuration will be automated with Puppet.
- Icinga2 – Icinga will server as monitoring for all the Containers.
- YAML – Formatting language.
- BASH – A few bash scripts will be used as well.

Procedure

1) Create and Ubuntu EC2 Instance.
   a) Log on to your AWS console.
   b) At the top left click on the EC2 orange icon.
   c) Click on the Blue “launch instance” button to create a new EC2.
   d) Select Ubuntu
   e) Choose the appropriate scale (t2.micro in my case) and click “Next:Configure Instance Details”
   f) Leave the defaults and click on “Next:Add Storage”
   g) Modify the “Size” of the main disk and add more disks if needed, then click “Next:Tag Instance”
   h) Edite the “Value” and set it to a representative value and then click on “Review and Launch”
   i) Check your setting and “Launch” your instance.
   j) Take note of your “Public DNS address” and continue to the next step.

2) Log on to the server and install Docker.

   a) Log on via SSH using your AWS keys and the new Public DNS Address.
      $ ssh -i < path to pem file >.pem ubuntu@<Public DNS Address>

   b) Install Docker dependencies
      $ sudo apt-get install apt-transport-https ca-certificates

   c) Add docker keys
      $ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D

   d) Check the OS release to add Docker repositories.
      $ lsb_release -a

   e) Add Docker repo (Note: modify <release> accordingly).
      $ sudo echo “deb https://apt.dockerproject.org/repo ubuntu-<release> main” >> /etc/apt/sources.list.d/docker.list && apt-get update

   f) Install Docker
      $ sudo apt-get install docker-engine apparmor linux-image-generic-lts-trusty

   g) Start Docker
      $ sudo service docker start

3) Install Docker Compose

   a) Download the binaries to /usr/local/bin
   ```
   $ sudo curl -L https://github.com/docker/compose/releases/download/1.7.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
   ```

   b) Apply execute permisions
   ```
   $ sudo chmod +x /usr/local/bin/docker-compose
   $ sudo chown root:bin /usr/local/bin/docker-compose
   ```

   c) Test the binary
   ```
   $ docker-compose --version
   ```

4) Install the AWS client for the EC2 to interact with the S3

   a) Use apt to download and install the client.
   ```
   $ sudo apt-get install awscli
   ```

   b) Go to your AWS Dashboard and go to:
   > IAM > Users > "username" > create access key and save your key pair for the next step.

   c) Configure the AWS client keys with the configure GUI.
   ```
   $ aws configure
   ```

5) Create and configure a S3 instance.

   a) From the AWS Dashboard to to (make sure to select US Standard for region):
   > S3 > Create Bucket > Bucket Name

   b) Test connectivity.
   ```
   $ sudo aws s3 cp /var/log/syslog s3://crossover-data
   $ sudo aws s3 ls s3://crossover-data
   ```

6) Create local directory to store containers and configurations.
   ```
   $ mkdir /docker-data
   ```

7) Generate a Composer file to start the environment under </docker-data/crossover.yml>

```
icinga:
  image: icinga/icinga2
  hostname: monitoring
  container_name: monitoring
```

```
  ports:
    - "3080:80"
    - "2022:22"
  volumes:
    - /docker-data/config_files:/var/config_files
  links:
    - httpd
    - mysql
httpd:
  image: httpd:latest
  hostname: webserver
  container_name: webserver
  volumes:
    - /docker-data/config_files:/var/config_files
  ports:
    - "80:80"
  links:
    - mysql
mysql:
  image: mysql:latest
  hostname: dbserver
  container_name: dbserver
  environment:
    - MYSQL_ROOT_PASSWORD=crossover
    - MYSQL_USER=icinga
    - MYSQL_PASSWORD=icinga
    - MYSQL_DATABASE=icinga
  ports:
    - "3306:3306"
```

8) Generate the environment with compose and review.

   a) Let docker-compose generate the environment
      `$ sudo docker-compose -f crossover.yml up -d`

   b) Check Containers.
      `$ docker ps`

   c) Test connectivity to the icinga container with default user|password (icingaadmin|icinga) and then update the password from the web interface.
      http://<AWS DNS Name>:3080/icinga2

9) Repeat and extend the configuration with Puppet

a) Install Puppet.

```
$ sudo apt-get install puppet
```

b) Install Docker modules for Puppet

```
$sudo puppet module install garethr-docker
```

c) Make a Puppet manifest file </docker-data/crossover.pp>.

```
docker::run { 'webserver':
  image       => 'httpd:latest',
  ports       => ['80:80'],
  env             => ['APACHE_LOG_DI=R"/var/log/httpd"',
'APACHE_PID_FILE="/var/run/httpd/httpd.pid"'],
  volumes     => ['/docker-data/www/:/usr/local/apache2/htdocs'],
  hostname    => 'webserver',
  after       => [ 'dbserver' ],
  depends     => [ 'dbserver' ],
}
docker::run { 'dbserver':
  image       => 'mysql:latest',
  env             => ['MYSQL_ROOT_PASSWORD=crossover', 'MYSQL_USER=crossover',
'MYSQL_PASSWORD=crossover', 'MYSQL_DATABASE=crossover'],
  ports       => ['3006:3006'],
  hostname    => 'dbserver',
}
docker::run { 'monitoring':
  image       => 'icinga/icinga2',
  ports       => ['3080:80'],
  hostname    => 'monitoring',
  volumes     => ['/docker-
data/Include/crossover_network.conf:/etc/icinga2/conf.d/crossover_network.conf'],
  links       => ['dbserver:dbserver', 'webserver:webserver'],
  after       => [ 'webserver', 'dbserver' ],
  depends     => [ 'webserver', 'dbserver' ],
}
```

10) Create a list of hosts to be monitored by Icinga2 </docker-data/Include/crossover_network.conf>

a) Create the /docker-data/include directory

```
$ mkdir /docker-data/Include
```

b) Create the < /docker-data/Include/crossover_network.conf> file and assign icigna ownership

```
object Host "webserver" {
address = "webserver"
check_command = "hostalive"
}
object Host "dbserver" {
address = "dbserver"
check_command = "hostalive"
}

$ sudo chown 996:993 crossover.conf
$ sudo chmod 640 crossover.conf
```

11) Change the default web page on the web server.

a) Bring your own webpage and locate it under </docker-data/www>

```
$ aws s3 cp s3://palacosfr/main.tar /docker-data/www
$ cd /docker-data/www && tar -xvf main.tar
```

12) Stop any running containers and restart the environment with the crossover.pp file

a) Stop running containers.

```
$ docker ps
$ docker stop <container>
```

b) Build the environment with Puppet.

13) Perform tests to check the environment.

14) Create scripts to keep logs locally and backup to the S3 Bucket.

a) Create a script to keep the logs of the running Containers on the Puppet master </docker-data/Src/keeplogs.sh>.

```
#!/bin/bash
log_path=""
log_store_path="/docker-data/log"
CURRENT_TIMESTAMP=000000000
```

```
for container in $(docker ps |awk '{print $NF}'|grep -v NAMES)
    do
    if [[ "/var/log/messages" == "$(docker exec $container ls /var/log/messages 2>/dev/null)" ]]
        then
        NEW_TIMESTAMP=$(docker exec $container stat -c %Y /var/log/messages)
        log_path="/var/log/messages"
        else
        NEW_TIMESTAMP=$(docker exec $container stat -c %Y /var/log/dmesg)
        log_path="/var/log/dmesg"
        fi
    if [[ $CURRENT_TIMESTAMP < $NEW_TIMESTAMP ]]
        then
        docker cp $container:$log_path /docker-data/log/$container.log.$(date +%F)
        fi
    done
```

b) Create script to perform daily archives of the logs at 7 pm to the S3 Bucket.

```
#!/bin/bash

log_path=""
log_store_path="/docker-data/log/"
CURRENT_TIMESTAMP=000000000

for log in $(ls -l $log_store_path/*.log.*|awk '{print $9}')
    do
    aws s3 cp $log s3://crossover-data/
    done
```

c) Add both scripts to the Puppet Master cron having keeplogs run every minute and archive every day at 19 hs.

```
$ sudo crontab -e
* * * * * /docker-data/Src/keeplogs.sh
0 19 * * * /docker-data/Src/archivelogs.sh
```

d) Verify logs are being sent to the S3 Bucket.

15) Make full backups of the containers and send them to the S3 Bucket.

a) Generate the </docker-data/Src/backupcontainers.sh>

```
#!/bin/bash
container_store_path="/docker-data/backups"
for repo in $(docker images|grep -v REPO|awk '{ print  $1}')
      do
      image=$(docker images |grep $repo|awk '{print $3}')
      container=$(docker ps |grep $repo|awk '{print $NF}')
      echo "docker save -o $container_store_path/$container.$(date +%F).tar $image"
      echo "aws s3 cp $container_store_path/$container.$(date +%F).tar s3://crossover-data/"
      done
```

b) Add the container to cron to perform daily backups.

```
$ crontab –e
0 20 * * * /docker-data/Src/backupcontainers.sh
```

c) Check your /docker-data/backups and S3 Buchet for existence of the backups.