How to create a web scale infrastructure based on Docker, CoreOS, Vulcand and Mesos. And why Object Storage becomes the de facto data repository.

Sun, Jan 11, 2015

Introduction

Let's first discuss about why I decided to use these software to show how to create a web scale infrastructure.

Why Docker ?

The first question should even be, why Linux containers ?

Linux containers are providing a very low compute and storage overhead compared to virtual machines.

Docker has really simplified the way anyone can leverage Linux containers, but also provide useful features like the Dockerfiles, the Docker Hub and a layered file system (see https://docs.docker.com/terms/layer/).

In my setup, I use a private Docker Registry which is storing the images on ViPR using the Amazon S3 API (as described in my previous post).

More information available at http://www.docker.io

Why CoreOS ?

All the components will run in Docker containers, so you could say that the Operating System isn't really important.

But, CoreOS is providing many advantages:

- Automatically update the entire OS as a single unit, instead of package by package (and can even reboot the system if you don't have any SPOF in your infrastructure)
- Include etcd for Service Discovery, which is also used by Vulcand (and even mesos in this setup)

- Include systemd and fleet, a tool that presents your entire cluster as a single init system. I don't use fleet in this setup, but I use it for other purposes, like starting an elastic search cluster in few seconds)

More information available at http://www.coreos.com

Why Vulcand ?

I see many tutorials about how to deploy containers or virtual machines, but I'm always surprised to see that they rarely cover the load balancing part of the infrastructure.

In my opinion, load balancing is a key component of a web scale infrastructure. Why applying automation everywhere if then your application cannot be reached by your users ?

Vulcand is a reverse proxy for HTTP API management and microservices.

And Vulcand is watching etcd to automatically detect new rules it needs to implement, so you don't need to reload any service. Simply add the right keys in etcd and your service/application becomes available from the outside world.

More information available at http://www.vulcanproxy.com

Why Mesos ?

There are few different options to automate Docker container deployments and I have evaluated 3 options:

- Fleet: This is natively provided by CoreOS and is a good option if you want to start containers and you want to manually specify the port to use on the host side
- Kubernetes: This is probably one of the most promising option, but is still at too early stage
- Mesos: The native container support has just been added, but Mesos is already a robust platform and can be used to deploy other workloads, like Hadoop

Why Object Storage ?

Using the different software above, an application can be deployed, scaled easily and accessed from the outside world in few seconds.

But, what about the data ?

Structured content would probably be stored in a distributed database, like MongoDB, for example

Unstructured content is traditionally stored in either a local file system, a NAS share or in Object Storage.

A local file system doesn't work as a container can be deployed on any node in the cluster.

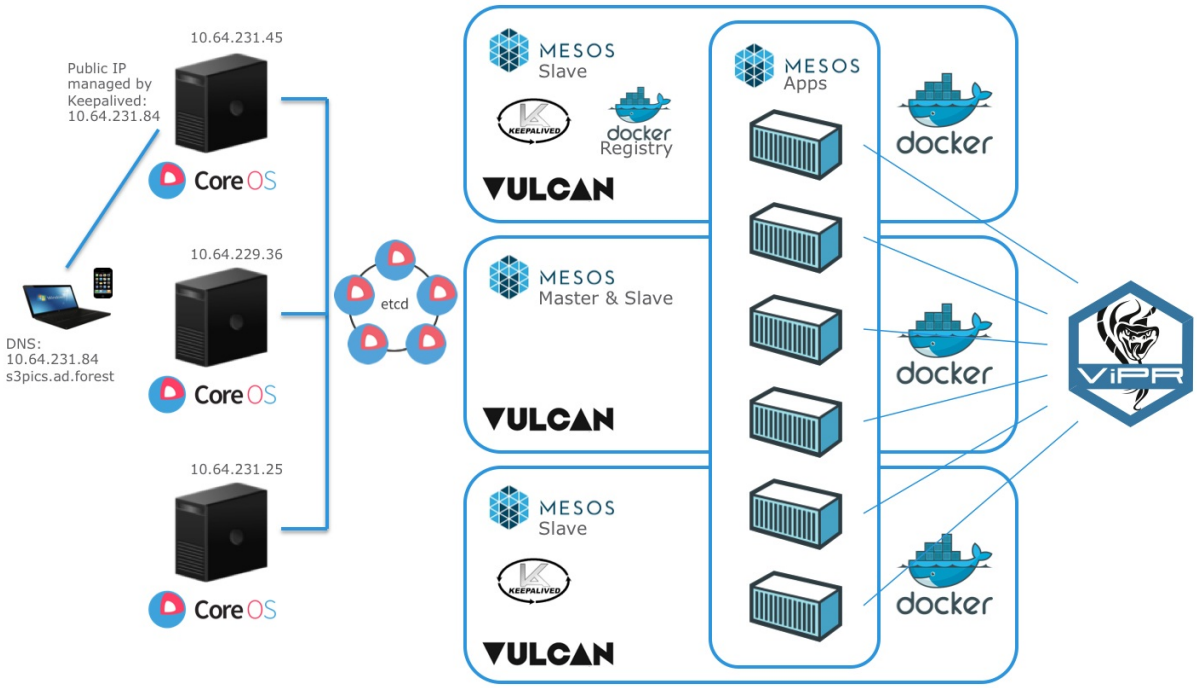
A NAS share could technically work, but would be very complex. For example, the share would have to be mounted on all the hosts, then you would have to specify the volume for each container, run the container in privileged mode, … If the NAS share isn't available when the container is launched, then the application inside the container will have to manage the issue as well

On the other side, Object Storage can be used by any application from any container, is highly available due to the use of load balancers, doesn't require any provisioning and accelerate the development cycle of the applications. Why ? Because a developper doesn't have to think about the way data should

be stored, to manage a directory structure, and so on.

Also, I've developed a web application which shows how an application can manage uploads and downloads without being in the data path. I'll run this application in this setup later.

Big picture



This diagram shows the different components and how I have setup the 3 nodes CoreOS cluster.

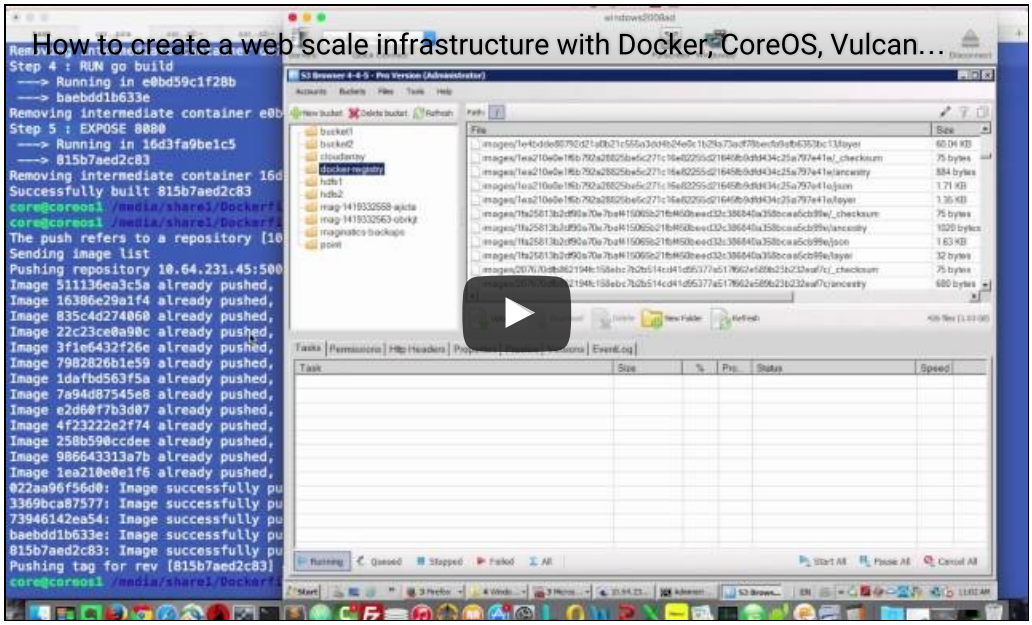
I use Keepalived to make sure the public IP 10.64.231.84 is always up either on the coreos1 or the coreos3 node.

Vulcand is running on each node to balance the load between the users and the web application, but also between the web application and the different ViPR nodes.

A private Docker Registry is running on the coreos1 node and is storing the image layers on ViPR using the Amazon S3 API.

The Mesos master and Marathon are running on the coreos2 node.

Demo



The different steps

Creating a Docker image for my web application

Here is the Dockerfile I use to create the Docker Image:

```
FROM golang
WORKDIR /
RUN git clone https://djannot:xxxx@github.com/djannot/s3pics.git
WORKDIR /s3pics
RUN go build
EXPOSE 8080
```

I build the image using the `--no-cache` parameter to make sure the latest source code is cloned from github.

```
core@coreos1 /media/share1/Dockerfiles/s3pics $ docker build --no-cache .
Sending build context to Docker daemon 2.048 kB
Sending build context to Docker daemon
Step 0 : FROM golang
---> 1ea210e0e1f6
Step 1 : WORKDIR /
---> Running in f6987b175723
---> 022aa96f56d0
Removing intermediate container f6987b175723
Step 2 : RUN git clone https://djannot:xxxx@github.com/djannot/s3pics.git
---> Running in 54d6a32e90ba
Cloning into 's3pics'...
---> 3369bca87577
Removing intermediate container 54d6a32e90ba
Step 3 : WORKDIR /s3pics
---> Running in d875bc08eac9
---> 73946142ea54
Removing intermediate container d875bc08eac9
Step 4 : RUN go build
---> Running in e0bd59c1f28b
---> baebdd1b633e
Removing intermediate container e0bd59c1f28b
Step 5 : EXPOSE 8080
---> Running in 16d3fa9be1c5
---> 815b7aed2c83
Removing intermediate container 16d3fa9be1c5
Successfully built 815b7aed2c83
```

Finally, I push the image to the Docker registry

```
core@coreos1 /media/share1/Dockerfiles/s3pics $ docker push 10.64.231.45:5000/s3pics:2.0
The push refers to a repository [10.64.231.45:5000/s3pics] (len: 1)
Sending image list
Pushing repository 10.64.231.45:5000/s3pics (1 tags)
Image 511136ea3c5a already pushed, skipping
Image 16386e29a1f4 already pushed, skipping
Image 835c4d274060 already pushed, skipping
Image 22c23ce0a90c already pushed, skipping
Image 3f1e6432f26e already pushed, skipping
Image 7982826b1e59 already pushed, skipping
Image 1dafbd563f5a already pushed, skipping
Image 7a94d87545e8 already pushed, skipping
Image e2d60f7b3d07 already pushed, skipping
Image 4f23222e2f74 already pushed, skipping
Image 258b590ccdee already pushed, skipping
Image 986643313a7b already pushed, skipping
Image 1ea210e0e1f6 already pushed, skipping
022aa96f56d0: Image successfully pushed
3369bca87577: Image successfully pushed
73946142ea54: Image successfully pushed
baebdd1b633e: Image successfully pushed
815b7aed2c83: Image successfully pushed
Pushing tag for rev [815b7aed2c83] on {http://10.64.231.45:5000/v1/repositories/s3pics/tags/2.0}
```

I've specified a tag (2.0) to make sure each node of the cluster will pull the latest version from the private Docker Registry

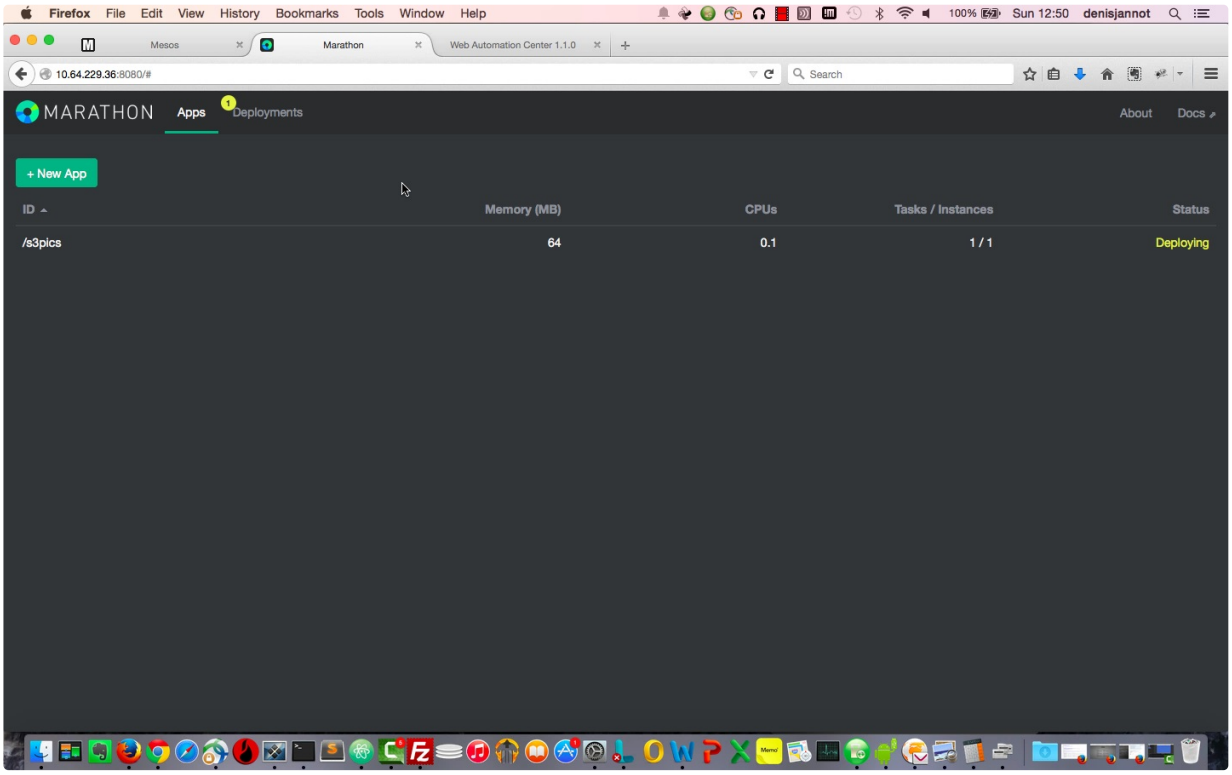
Deploying the Mesos app

Let's now start a new Mesos app using this Docker image:

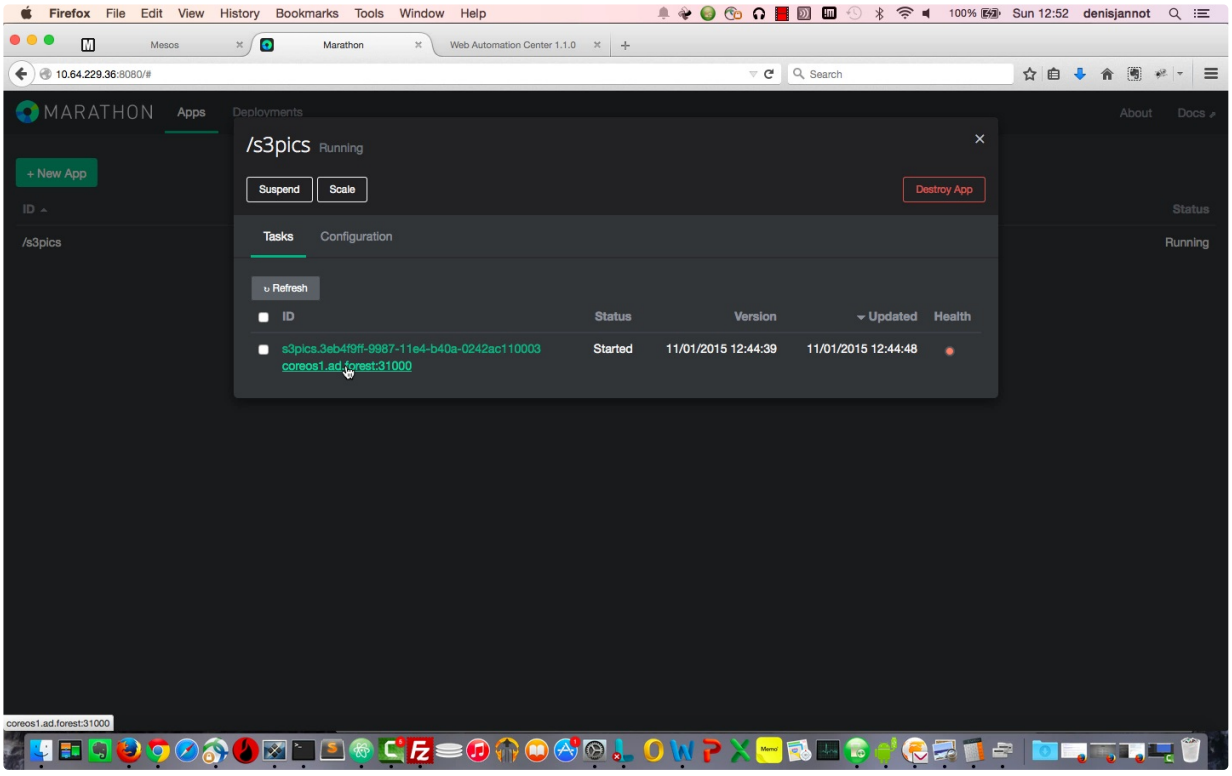
```
POST http://<Mesos Marathon IP>:8080/v2/apps

{
  "id": "s3pics",
  "cmd": "cd /s3pics; ./s3pics -AccessKey=denis@ad.forest -SecretKey=xxxx -
EndPoint=http://denisnamespace.ns.viprds.ad.forest -Namespace=denisnamespace",
  "cpus": 0.1,
  "mem": 64.0,
  "instances": 1,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "10.64.231.45:5000/s3pics:2.0",
      "network": "BRIDGE",
      "portMappings": [
      { "containerPort": 8080, "hostPort": 0, "protocol": "tcp" }
      ]
    }
  },
  "healthChecks": [
  {
    "protocol": "HTTP",
    "portIndex": 0,
    "path": "/",
    "gracePeriodSeconds": 10,
    "intervalSeconds": 20,
    "maxConsecutiveFailures": 3
  }
  ]
}
```

As soon as the Mesos app is started, the Mesos Marathon UI displays the status of the application.

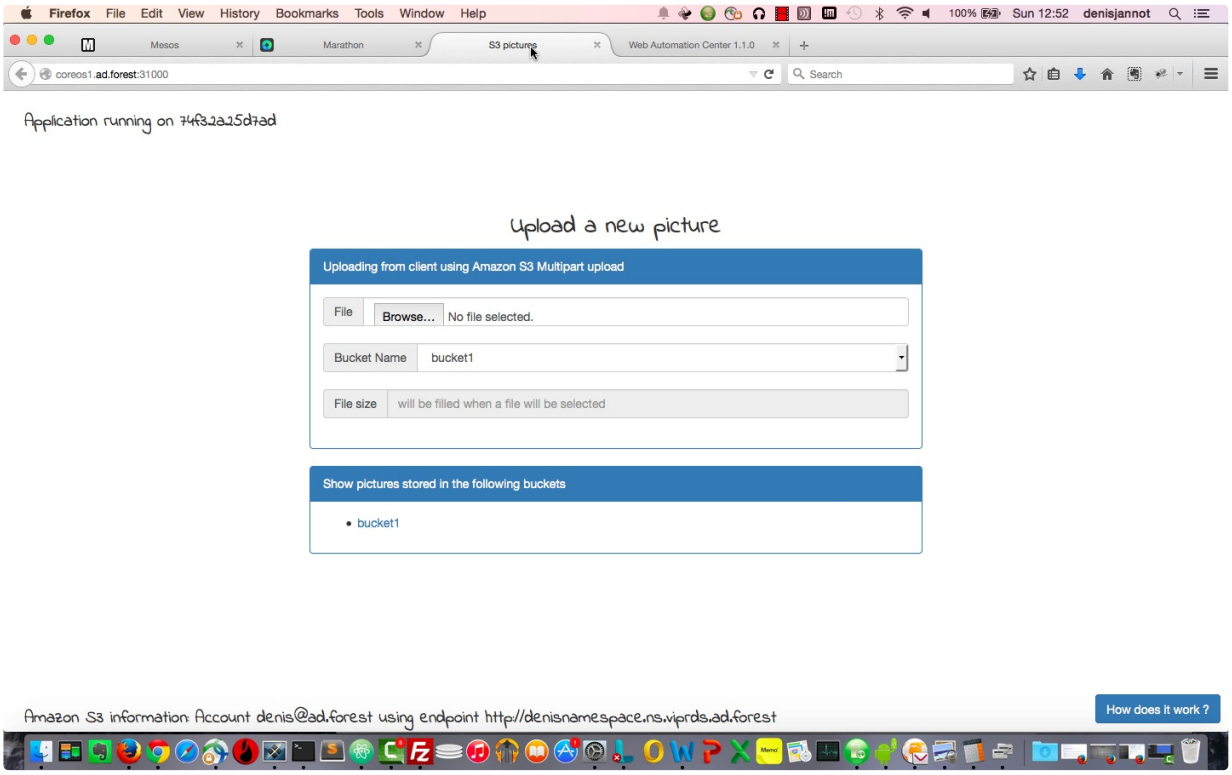


After few seconds, the app is deployed and the Docker host and port are displayed.



Accessing the web application

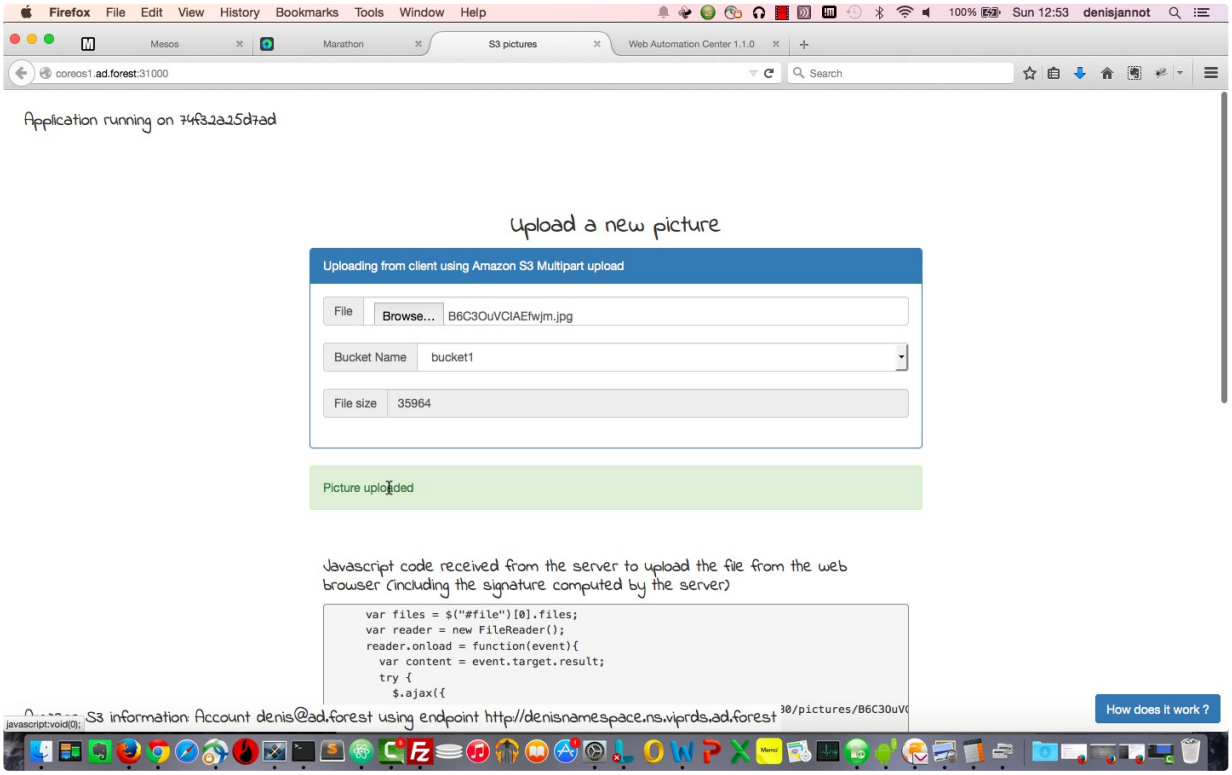
Using the information displayed in the Marathon UI, I can access the web application on `http://coreos1.ad.forest:31000`.



The name of the container where the application is running is displayed on the top left corner.

The Amazon S3 endpoint used to upload and download pictures is displayed on the bottom left corner and shows that ViPR is used to store the data.

We can now upload a picture.



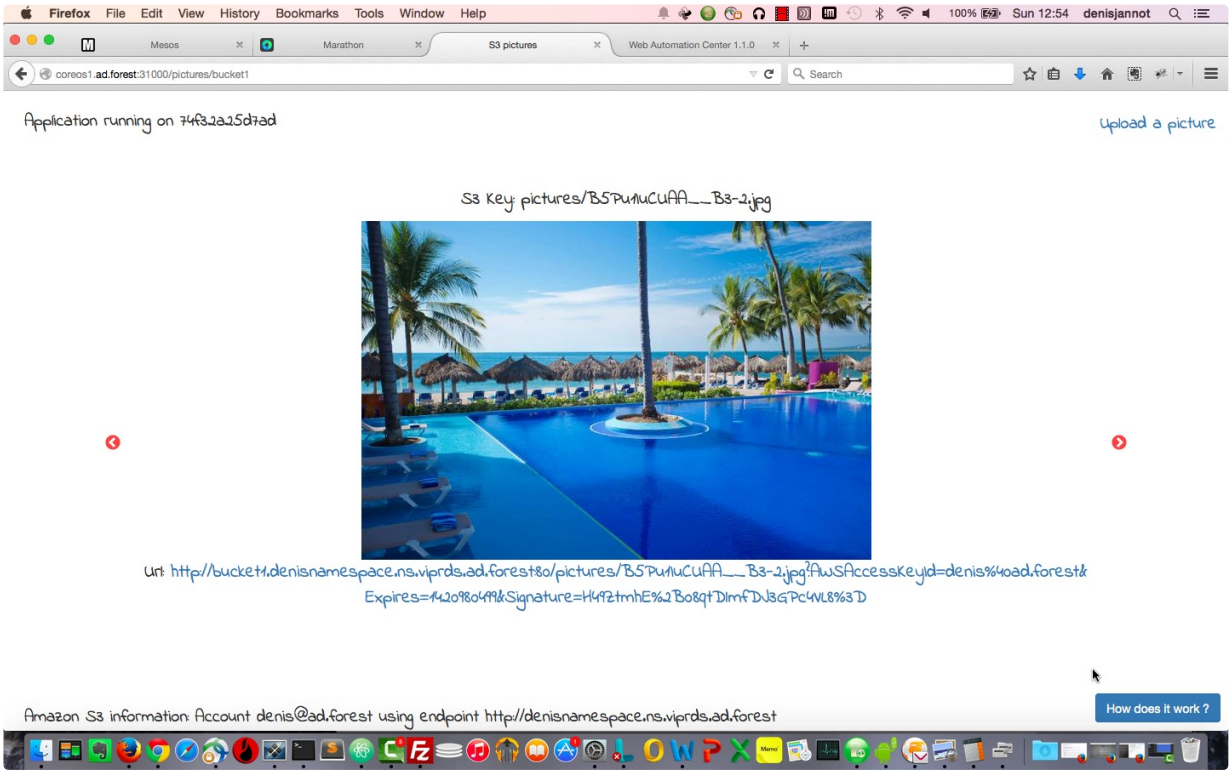
The code below is sent to the internet browser by the web application to allow the browser to upload the picture directly to the Object Storage platform.

```
var files = $("#file")[0].files;
var reader = new FileReader();
reader.onload = function(event){
  var content = event.target.result;
```

```
  try {
   $.ajax({
     url: 'http://bucket1.denisnamespace.ns.viprds.ad.forest:80/pictures/B6C3OuVCIAEfwjm.jpg',
     data: content,
     cache: false,
     processData: false,
     type: 'PUT',
     beforeSend: function (request)
     {
      request.setRequestHeader('Content-Length','35964');
      request.setRequestHeader('Content-Type','binary/octet-stream');
      request.setRequestHeader('x-amz-date','Sun, 11 Jan 2015 12:53:12 UTC');
      request.setRequestHeader('host','bucket1.denisnamespace.ns.viprds.ad.forest');
      request.setRequestHeader('Authorization','AWS denis@ad.forest:iCnahEUOy8/lanI96tQYA3WKQVE=');
     },
     success: function(data, textStatus, request){
      $('#alert-success').html("Picture uploaded").show().delay(5000).fadeOut();
      },
     error: function(data, textStatus, request){
      $('#alert-danger').html("Upload failed").show().delay(5000).fadeOut();
     }
   });
  }
  catch (e) {
   alert(e);
  }
 }
 reader.readAsArrayBuffer(files[0]);
```

The fact that the picture is uploaded directly to the Object Storage platform means that the web application is not in the data path. This allows the application to scale without deploying hundreds of instances.

This web application can also be used to display all the pictures stored in the corresponding Amazon S3 bucket.



The url displayed below each picture shows that the picture is downloaded directly from the Object Storage platform, which again means that the web application is not in the data path.

This is another reason why Object Storage is the de facto standard for web scale applications.

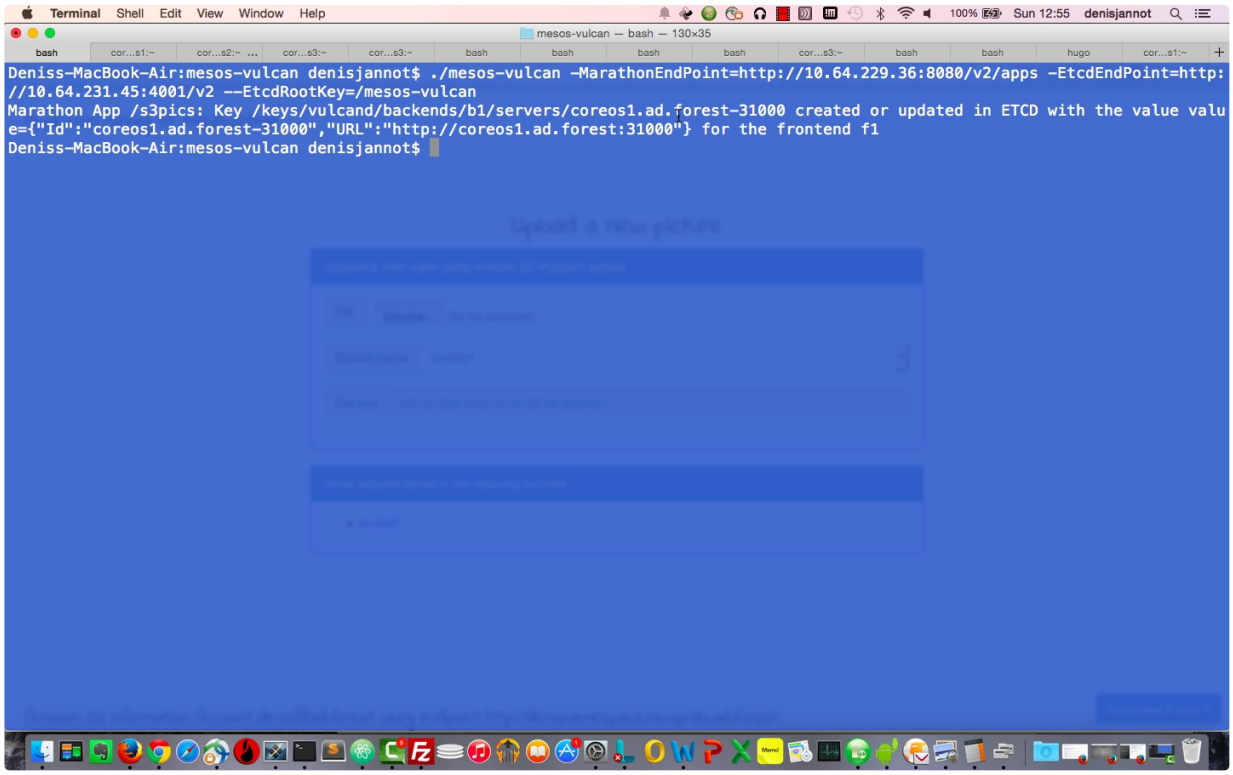
Updating the vulcand proxy

Let's now see how we can access this web application from the outside world.

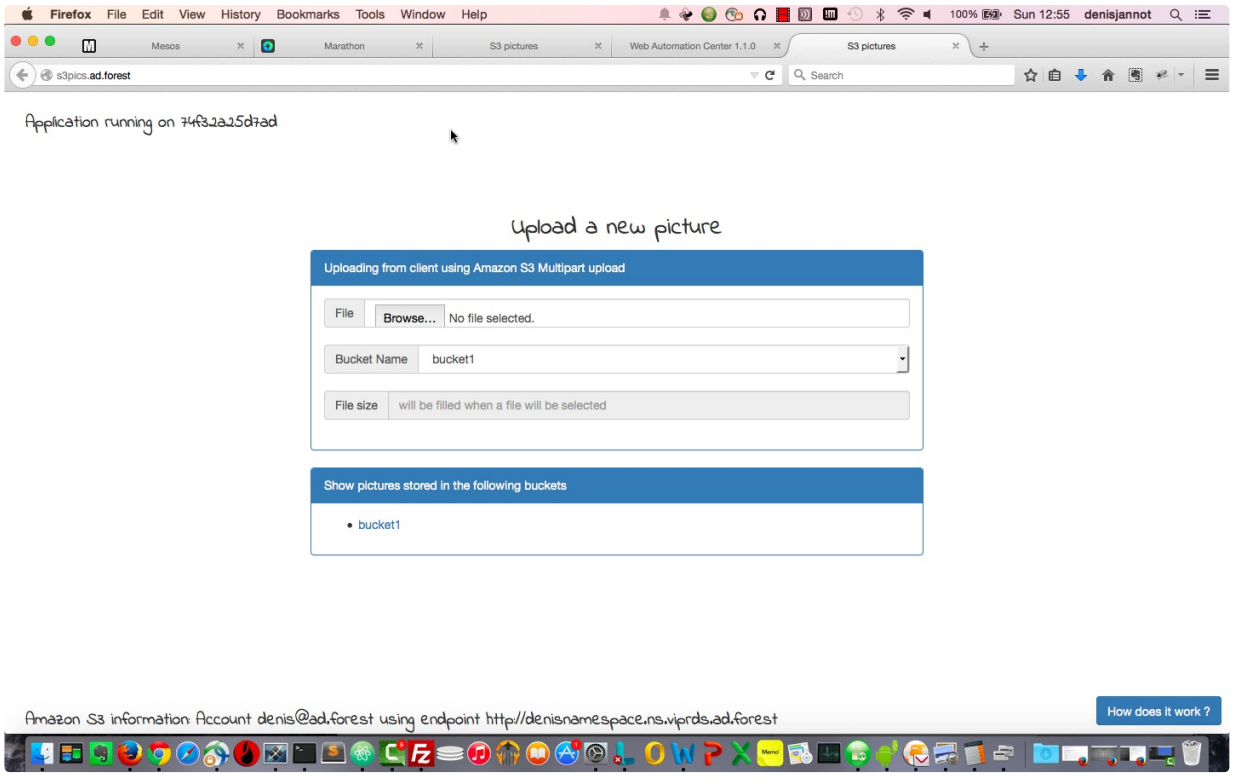
I've developed a small tool in golang which is using both the Marathon API and the etcd API to:

- determine what Mesos applications are running without a corresponding vulcand rule in etcd and to create the missing rules
- determine what vulcand rules exist in etdc for Mesos applications which aren't running anymore to delete them

I now run this tool to create the vulcand rule in etcd for the instance I've just deployed.

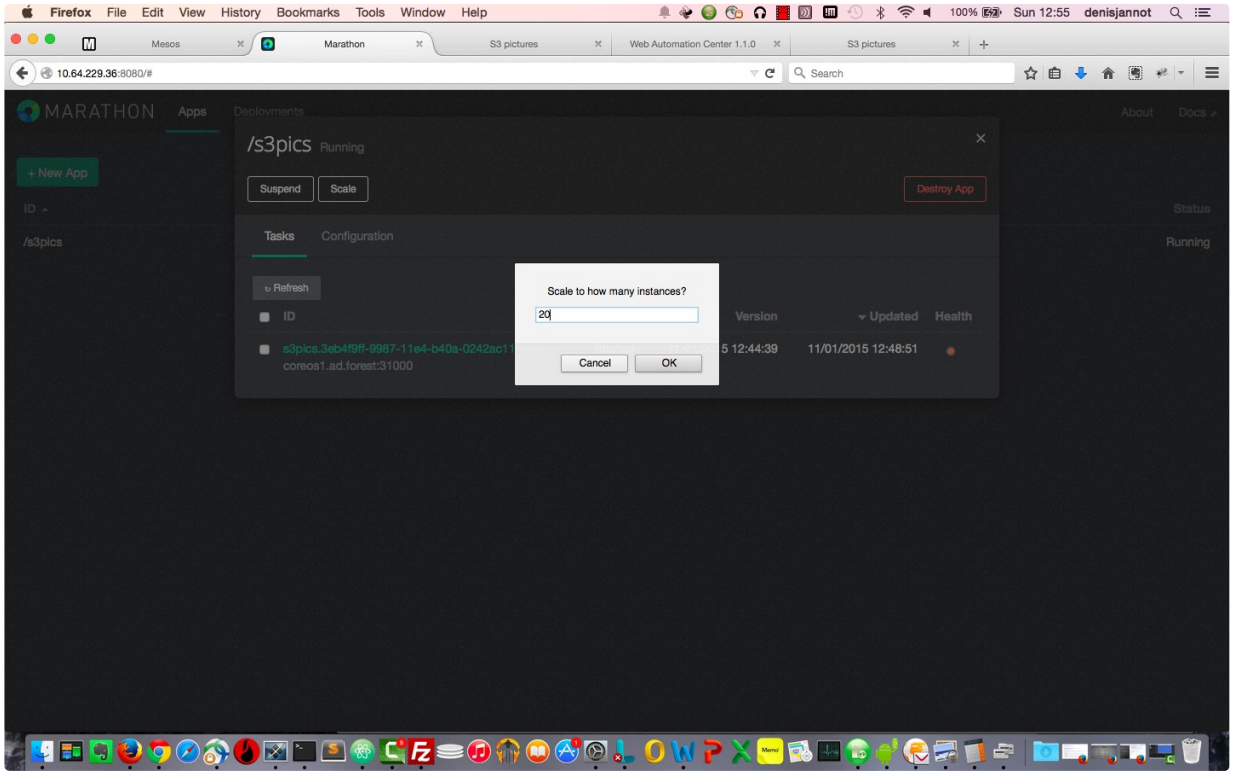


The web application can now be accessed from the outside world (http://s3pics.ad.forest).

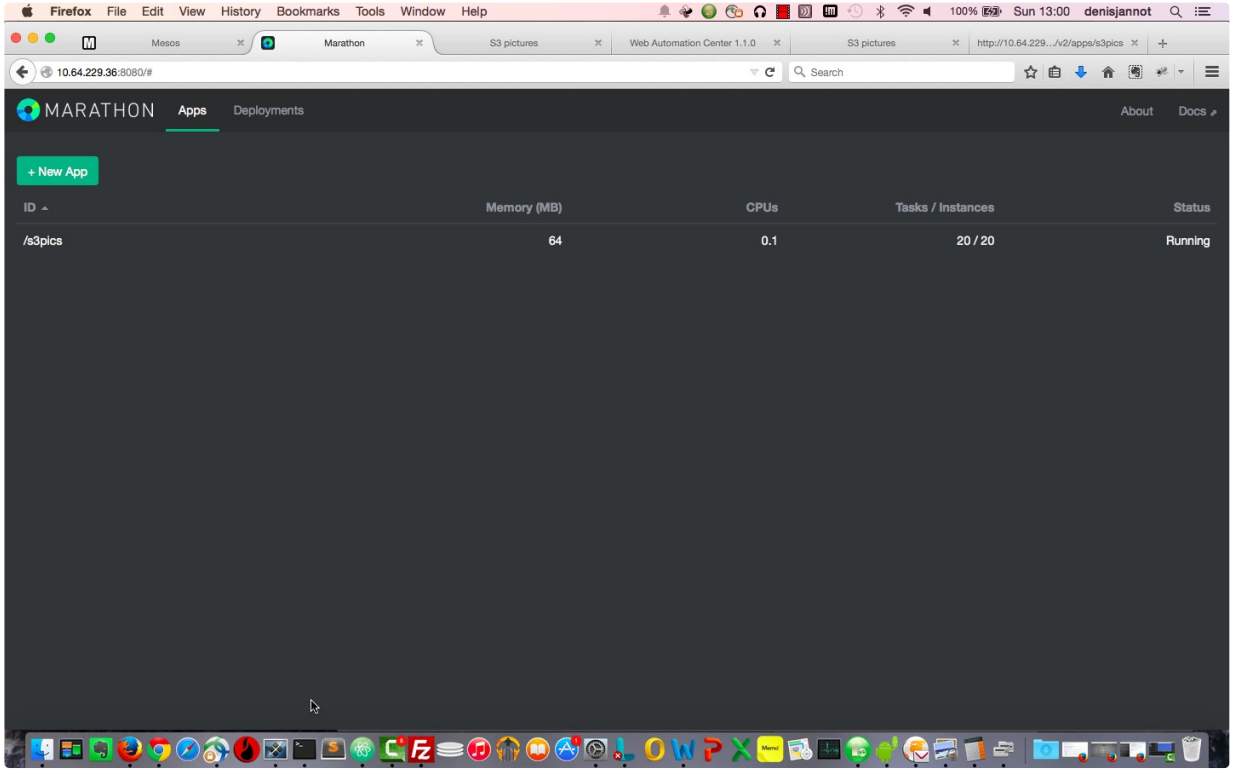


Scaling the Mesos app

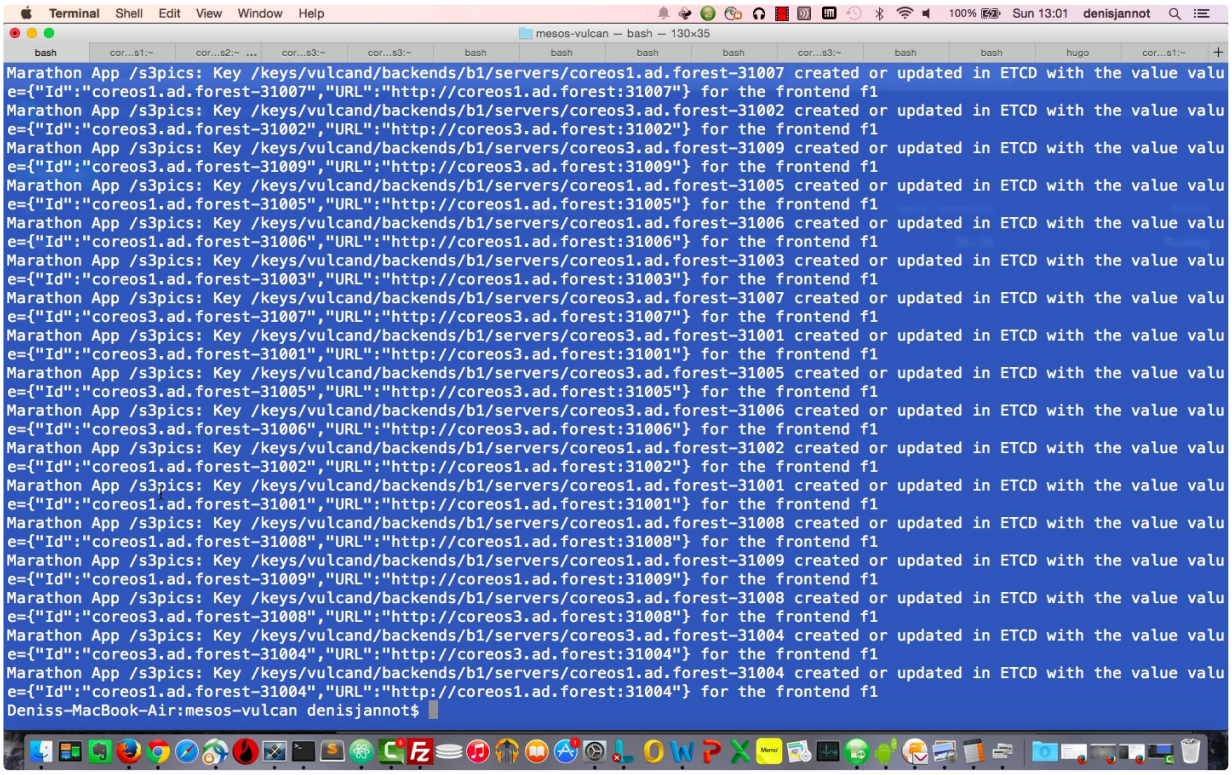
One of the beauty of Mesos is its ability to scale easily the number of instances of an application currently running.



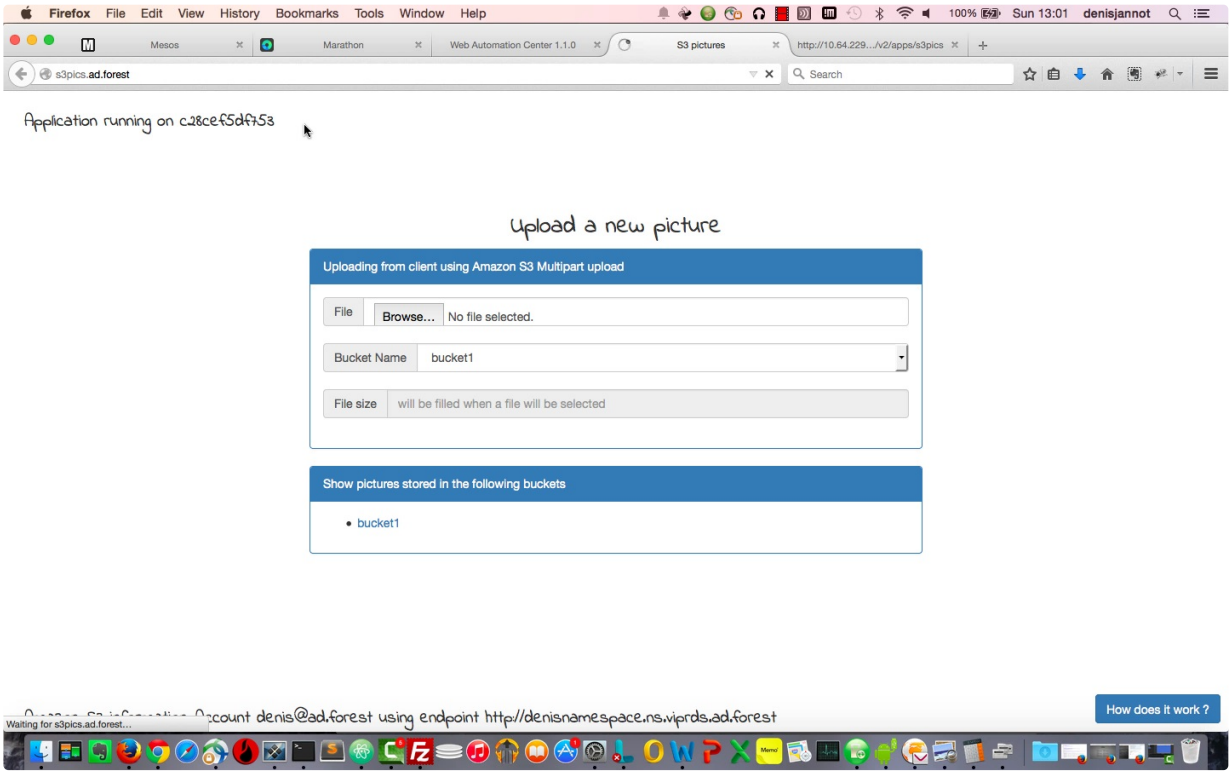
After few seconds, 20 instances are running.

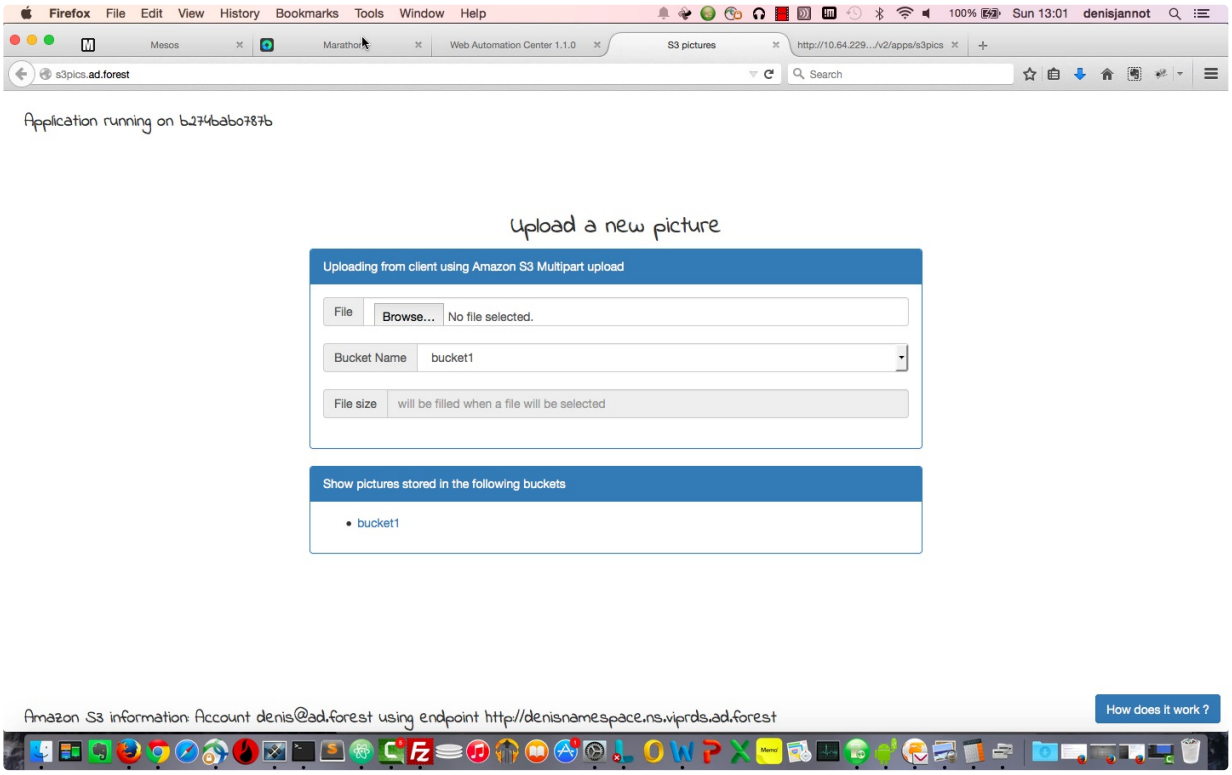


I need to run my tool again to update the vulcand rules.



Now, if I refresh my web browser, I can see that the container name displayed on the top left is changing based on the instance serving the application.





Using the Marathon UI or the API, it's also possible to scale down the number of instances and to run the tool again to update the vulcand rule.

Denis Jannot Mod ➚ Frederic Branczyk • 2 years ago

Yes, I'll open source this tool.
I'll also create another post to show the details about the cluster setup

Reply • Share ›

Frederic Branczyk ➚ Denis Jannot • 2 years ago

Awesome! Thanks in advance.

Reply • Share ›

Denis Jannot Mod ➚ Frederic Branczyk • a year ago

Just shared the code on github:
https://github.com/djannot/mes...

I've also created a new post to explain how it works

Reply • Share ›

Frederic Branczyk ➚ Denis Jannot • a year ago

Awesome stuff dude. Thank!

Reply • Share ›

Michael Hamrah • 2 years ago

I can't tell by this setup if CoreOS is running on a separate set of servers than mesos, or if you're somehow running Mesos on top of CoreOS.

Reply • Share ›

Denis Jannot Mod ➚ Michael Hamrah • 2 years ago

I'm running Mesos on top of CoreOS in Docker containers.

Reply • Share ›

Michael Hamrah ➚ Denis Jannot • 2 years ago

Cool, I'd be interested in seeing the code. Based on the mesos discussion thread it seemed like running a mesos-slave in a docker container had some issues, but I guess if you exposed the DOCKER_HOST env variable correctly it should proxy without issue.

I'm also curious if you contrasted tools like registrator or confd which also provide configuration proxies on top of docker.

1 Reply • Share ›