**Scalable Architecture DR CoN: Docker, Registrator, Consul, Consul Template and Nginx**

*Foreword: This is a brief document based on a post to create a scalable Docker environment. The original article can be found at:* [*https://maori.geek.nz/scalable-architecture-dr-con-docker-registrator-consul-consul-template-and-nginx-8da2820d02f9*](https://maori.geek.nz/scalable-architecture-dr-con-docker-registrator-consul-consul-template-and-nginx-8da2820d02f9)*. There are just some minor updates for running this in a Windows setting. There is a current outstanding issue with WinNAT that makes it so that working with local containers a little difficult. Accessing the containers via localhost:port as you can on other operating systems is not supported on Windows. Only from an external remote machine can it be accessed via the hostip:port.*

Docker is great fun when you start building things by plugging useful containers together. Recently I have been playing with [Consul](https://www.consul.io/) and trying to plug things together to make a truly horizontally scalable web application architecture. Consul is a **Service Discovery and Configuration** application, made by [HashiCorp](https://hashicorp.com/) the people who brought us [Vagrant](http://www.maori.geek.nz/post/vagrant_with_docker_how_to_set_up_postgres_elasticsearch_and_redis_on_mac_os_x).

Previously I experimented using Consul by using SRV records ([described here](http://www.maori.geek.nz/post/docker_web_services_with_consul)) to create a scalable architecture, but I found this approach a little complicated, and I am all about simple. Then I found [Consul Template](https://hashicorp.com/blog/introducing-consul-template.html) which links to Consul to update configurations and restart application when services come up or go down.

In this post I will describe how to use Docker to plug together Consul, Consul Template, Registrator and Nginx into a truly scalable architecture that I am calling **DR CoN**. Once all plugged together, DR CoN lets you add and remove services from the architecture without having to rewrite **any** configuration or restart **any** services, and everything just works!

**Docker**

Docker is an API wrapper around containerization technologies found in Linux and Windows (10 and Server 2016). While there is an installer which allows an easy install, the way the Docker daemon is ran with the GUI that restricts which configurations can be used, such as the *hosts* property. Instead, it can be installed via PowerShell commands to get around the limitation. This is briefly described below:

* In Control Panel > Programs and Features > Turn Windows features on or off > Check “Containers” and “Hyper-V”, then hit “OK”. It may prompt you to restart.
* In a PowerShell prompt download the executable by running:

Invoke-WebRequest "https://get.docker.com/builds/Windows/x86\_64/docker-17.03.0-ce.zip" -OutFile "$env:TEMP\docker.zip" -UseBasicParsing

* Extract the package:

Expand-Archive -Path "$env:TEMP\docker.zip" -DestinationPath $env:ProgramFiles

* Add the Docker directory to the path environment variable:

$env:path += ";$env:ProgramFiles\Docker"

$existingMachinePath = [Environment]::GetEnvironmentVariable("Path",[System.EnvironmentVariableTarget]::Machine)

[Environment]::SetEnvironmentVariable("Path", $existingMachinePath + ";$env:ProgramFiles\Docker", [EnvironmentVariableTarget]::Machine)

* To install Docker as a Windows service, run the following:

dockerd --register-service

* To make the host machine accessible by the containers, the Docker daemon has to run with the *hosts* specified in the daemon.json. Find your internal virtual switch by running *ipconfig*. For this example, I will be using 172.16.0.x as my IP range, with 172.16.0.1 as the virtual switch IP. Open C:\ProgramData\docker\config\daemon.json. Create it if it does not exist. It should be modified to look something like the following:

{"hosts": ["tcp://0.0.0.0:2375", "npipe://"], "fixed-cidr": "172.16.0.0/24"}

* Once installed, the service can be started:

Start-Service Docker

* You can test Docker is correctly installed by opening the command line and using:

docker ps

docker -H [*virtual switch IP*] ps

If running the *ps* command against the virtual switch IP gives you an error, you may need reconfigure virtual switch:

* First stop the Docker service:

Stop-Service docker

* Then remove Docker’s container network and the WinNAT configuration:

Get-ContainerNetwork | Remove-ContainerNetwork -force

Get-NetNat | Remove-NetNat

* Recreate the container network. 172.16.0.0/24 will be used as the subnet and 172.16.0.1 as the gateway for this example:

New-ContainerNetwork –Name nat –Mode NAT –SubnetPrefix 172.16.0.0/24 -GatewayAddress 172.16.0.1

* Restart the Docker service:

Start-Service docker

* Test that the Docker service is accessible with 172.16.0.1 as the host:

docker -H 172.16.0.1 ps

### Run a very simple Web Service with Docker

To test the Dr CoN architecture we will need a service (in consul terminology, it is just an application Consul knows about). For this, let us create a simple service. Create a file called Dockerfile with the contents:

FROM microsoft/aspnet

SHELL ["powershell", "-Command"]

WORKDIR C:/inetpub/wwwroot

ENTRYPOINT $env:computername > index.html; \

& C:/Servicemonitor.exe w3svc

In the same directory as this file execute:

docker build -t aspnet/simple .

This will build the docker container and call it aspnet/simple, which can be run with:

docker run -d -p 8000:80 –-name simple aspnet/simple

Due to outstanding issues with WinNAT, we are unable to access localhost:8000 to get to our page. Instead we’ll have to find out its IP:

docker inspect simple --format="{{.NetworkSettings.Networks.nat.IPAddress}}"

Once the IP is known, it can be used to access the “hello world” page created in the container by simply entering the IP into the browser, or running curl [*container ip*].

**Consul**

Consul is best described as a service registry that has a DNS and a HTTP API. It also has many other features like health checking services, clustering across multiple machines and acting as a key-value store. To run Consul in a Docker container execute:

docker run -d --name consul -p 8500:8500 -p 8600:53/udp spring2/consul agent -ui -server -bootstrap -advertise [*host machine ip*] -log-level debug -data-dir c:\data -client 0.0.0.0 -dns-port 53

Again, you can get the IP address of a container by running:

docker inspect consul --format="{{.NetworkSettings.Networks.nat.IPAddress}}"

If you browse to [*consul container ip*]:8500 there is a dashboard to see the services that are registered in Consul.

To register a service in Consul’s web API we can use Invoke-RestMethod:

Invoke-RestMethod -Method PUT -Uri http://*[consul container ip]*:8500/v1/agent/service/register -Body '{"ID":"simple1","Address":"*[simple container ip]*","Port":80,"Name":"simple"}'

The container [srv-router](https://github.com/vlipco/srv-router) can be used with Consul and nginx to route incoming calls to the correct services, as [described here](http://www.maori.geek.nz/post/docker_web_services_with_consul). However, there is an easier way than that to use nginx to route to services.

Then we can query Consuls DNS API for the service using nslookup:

nslookup simple.service.consul [*consul container ip*]

> nslookup simple.service.consul 172.16.2.123

Server: UnKnown

Address: 172.16.2.123

Name: simple.service.consul

Address: 172.16.1.51

Hold on, **there is a problem**, where is the port of the service? Unfortunately, DNS A records do not return the port of a service, to get that we must check SRV records:

nslookup -q=srv simple.service.consul [*consul container ip*]

> nslookup -q=srv simple.service.consul 172.16.2.123

Server: UnKnown

Address: 172.16.2.123

simple.service.consul SRV service location:

priority = 1

weight = 1

port = 80

svr hostname = ac100133.addr.dc1.consul

ac100133.addr.dc1.consul internet address = 172.16.1.51

SRV records are difficult to use because many technologies do not support them.

The container [srv-router](https://github.com/vlipco/srv-router) can be used with Consul and nginx to route incoming calls to the correct services, as [described here](http://www.maori.geek.nz/post/docker_web_services_with_consul). However, there is an easier way than that to use nginx to route to services.

**Registrator**

Registrator takes environment variables defined when a Docker container is started and automatically registers it with Consul. For example:

docker run -d -–name registrator -e DOCKER\_HOST=[*virtual switch IP*]:2375 spring2/registrator -ip [*host machine ip*] -internal -tags windows consul://[*consul* *container ip*]:8500

Starting a service with:

docker run -d -e "SERVICE\_NAME=simple" -p 8000:80 aspnet/simple

Will automatically add the service to Consul, and stopping it will remove it. This is the first part to plugin to DR CoN as it will mean no more having to manually register services with Consul. Note that the -internal switch is used; this is so that the containers’ IP addresses and ports are used, rather than the host and mapped ports. The -internal switch would not be necessary in a multi-node environment, where the services are on a different host. Rather, you will want to use the -ip switch, with the IP address of the Docker host node instead.

**Consul Template**

[Consul Template](https://hashicorp.com/blog/introducing-consul-template.html) uses Consul to update files and execute commands when it detects the services in Consul have changed.

For example, it can rewrite an nginx.conf file to include all the routing information of the services then reload the nginx configuration to load-balance many similar services or provide a single end-point to multiple services.

**A Windows Container Dockerfile version of Graham’s original post can be located here:** <https://github.com/spring2/dr-con/>

**You will want to clone the repo, as it contains files that will be moved into the image during the build.**

This Docker container will run both Consul Template and nginx, and when the services change it will rewrite the nginx app.conf file, then reload nginx.

This image can be built with:

docker build -t drcon .

and run with:

docker run -d --name drcon -e "CONSUL=[*consul container ip:port*]" -e "SERVICE=simple" -p 80:80 drcon

SERVICE environment variable is used to select which services to include from Consul. So, this DR CoN container will now load balance across all services named simple. Note that this run statement is using port 80. If you have IIS enabled, it is likely the Default Website in IIS is using port 80. You may disable the IIS service (w3svc) or run the container with a different port mapped.

**All Together**

Let’s now plug everything together!

Run Consul

docker run -d --name consul -p 8500:8500 -p 8600:53/udp spring2/consul agent -ui -server -bootstrap -advertise [*host machine ip*] -log-level debug -data-dir c:\data -client 0.0.0.0 -dns-port 53

Run Registrator

docker run -d –-name registrator -e DOCKER\_HOST=[*virtual switch ip*]:2375 spring2/registrator -internal -tags windows consul://[*consul container ip*]:8500

Run DR CoN

docker run -d --name drcon -e "CONSUL=[*consul container ip*]:8500" -e "SERVICE=simple" -p 80:80 drcon

Calling the service:

curl [*drcon container ip*]

curl : 504 Gateway Time-out

Now start a service named simple

docker run -d -e "SERVICE\_NAME=simple" -p 8000:80 aspnet/simple

This will cause:

* Registrator to register the service with Consul
* Consul Template to rewrite the nginx.conf then reload the configuration

Now curl [*drcon container ip*] will be routed successfully to the service.

If we then start another simple service on a different port with:

docker run -d -e "SERVICE\_NAME=simple" -p 8001:80 aspnet/simple

Requests will now be load balances across the two services.

A fun thing to do is to run while($true) {curl [*drcon container ip*]; start-sleep -s 1; } while killing and starting simple services and see that this all happens so fast no requests get dropped.

**Conclusion**

Architectures like DR CoN are much easier to describe, distribute and implement using Docker and are impossible without good tools like Consul. Plugging things together and playing with Docker's ever more powerful tools fun and useful. Now I can create a horizontally scalable architecture and have everything just work.