DC/OS - AWS Networking Discussion

Base Considerations for Success

Arthur Johnson,
Solutions Architect, Mesosphere
ajohnson@mesosphere.io

Table of Contents

DC/OS on AWS and Networking

Overview

This paper covers some basic details on networking and security configuration for deploying a DC/OS cluster on AWS. It should have the feel of a question/answer session; or, like a fireside chat about how things work, why certain parameters must be so, and what to expect when deploying DC/OS within AWS.

The genesis of this writing is a result of a DC/OS customer deploying a cluster in an AWS environment with no security groups, no default passwords changed, and what happened as a result. After installing their cluster, 3 days later they found it being used as a bitcoin mining lab servicing 100s of containers for mining.

Out of the box, DC/OS cannot control anything below it's own software layer: That is, the underlying infrastructure, including hardware, software, networking, and storage must be properly configured to protect the overall environment. If an operation has no security groups configured, and no firewall, it is not the fault of DC/OS becoming a bitcoin mining engine. In this case, DC/OS is just running the tasks which was commanded to run. By anyone would could get into the environment and take command.

Underlying configurations can range in complexity and robustness:

- Air-gapped environments with networking packet filtering and well defined VLANs/Subnets for various components
- Multiple network interfaces for handling various traffic types
- Complex firewall rules and proxy definitions to obscure the environment behind Internet ports
- OS hardening with packages sanity verified for exploits, ports disabled, user accounts disabled, etc.
- Or, a single subnet, with all components in one flat network, default AMIs deployed, with no passwords changed, no security groups and/or firewalls

Planning and design for infrastructure must be done both carefully and thoroughly. Once in deployment, making changes to the underlying infrastructure has serious impact on the DC/OS cluster, and, in some cases, might not even be feasible. It all depends on the purpose of the environment: what will it be used for, who will use it, how will they use it, what kind of retention requirements exist, what sort of governance, compliance and quality needs must be met? And the list goes on, depending on the enterprise deploying the environment...

Our Question and Answer Session

Customer Asks…

#1 *What is the detailed port/security group configuration for each component in a DC/OS cluster? Like for masters, agents, public agents, etc.?*

We maintain a published list of ports that are used on Masters, Agents, and other components in a DC/OS cluster. This list is published here:

- https://docs.mesosphere.com/1.10/installing/oss/ports/
- This above list includes the ports for the <1.10> version
- This above list also publishes the ports for the OSS variant of DC/OS which may be different than the EE variant
- https://docs.mesosphere.com/1.10/installing/ee/ports/ ← Enterprise version

We consider a vast majority of these ports to be "internal-DC/OS" ports and should never be exposed to a live Internet connection. That is, for inbound traffic. For example, I may not want to expose TCP port 8123 (Mesos-DNS) to the outside world because I don't want to expose the possibility of lookups outside of the cluster. However, if there is a need to provide external connectivity to my applications running in DC/OS, I may expose these ports but it should be calculated-- and with such calculation you may decide that packet filtering is necessary, or that other actions must be taken to further harden the environment.

Cluster nodes should have the following Ethernet configurations, depending on their role(s) within the cluster:

Masters

- `eth0`, public IP address, accessible on `ports 80 and 22` (HTTP and SSH respectively) -- there may be other ports which need to be opened, but this is a good starting point
- `eth1`, private IP address, only addressable and accessible by other nodes in the cluster. This interface should not have any connection to the outside world/Internet.

This is the interface that the masters will communicate to the agents over and need a wide-range of ports available (see the documentation link above)

Private Agents

- `eth0`, private IP address, only addressable and accessible by other nodes in the cluster, and most specifically to the masters. This interface should most definitely not be exposed to the outside world/Internet

Public Agents

- `eth0`, public IP address, with controlled/well defined ports open for ingress and egress access outside of the cluster -- This interface on Private Agents may be best suited to be in a DMZ, or isolated VLAN where it is known that outside traffic will exist
- `eth1`, private IP address, for internal cluster communication, same as the private agents

Now, with these requirements being stated, let's take a look at 3 high-level use-case traffic patterns for an environment...

Use-Case #1:

Step 1: An end-user --> Requests UI access on port 80 (web browser) -->

Step 2: ELB in front of the master nodes receives this request, and forwards to the acting Leader --->

Step 3: TCP port 80 traffic established on the public interface on the master with the Admin Router component, via ELB, and communication can then happen between the client and the master normally.

Step 4: Communication established between the end-user and the UI on port 80 via the Admin Router.

Use-Case #2:

Step 1: End-user --> Requests to see properties of an agent node via the UI--> Port 80 request to ELB→

Step 2: ELB--> Acting Leader (master node)--> makes internal request to agent node details requested for, establishes communication (this happens on the internal only ports of the nodes in the DC/OS cluster.

Step 3: Agent and Master exchange information →

Step 4: The master is then able to present that information back to the UI via Admin Router.

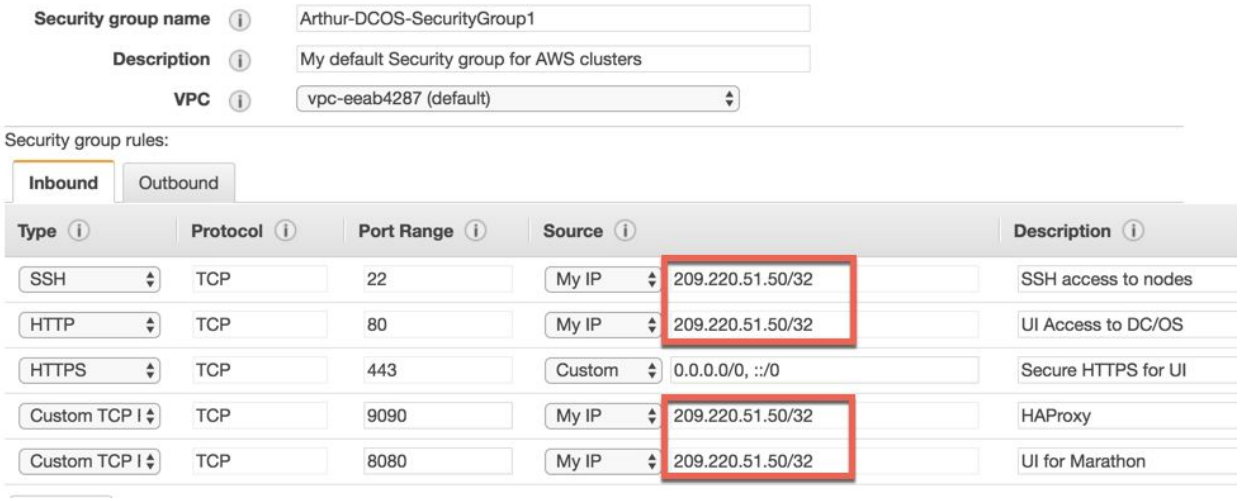
Use-Case #3:

An end-user connects to a publicly facing app on a public node and interacts with an application stack running on DC/OS,

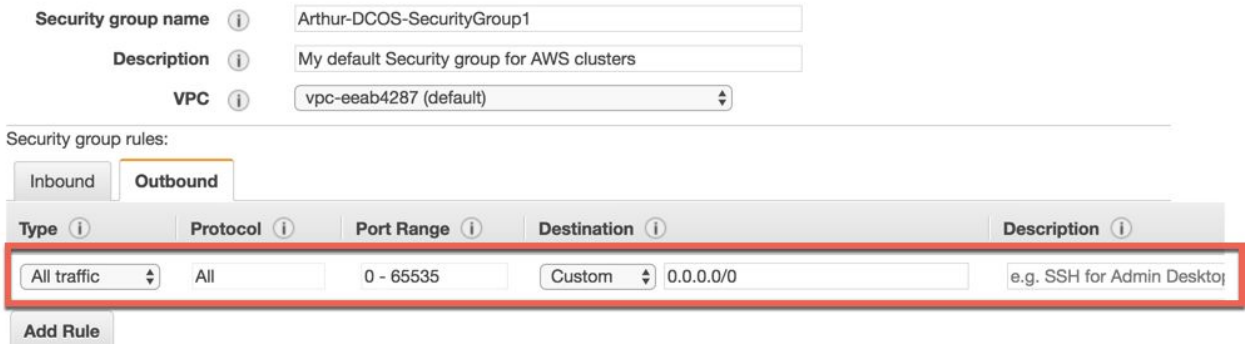
End-user requests to talk to application X, running on a pre-defined port (8888)---> The request lands on a ELB or GLB (Maybe something like HAProxy) which then---> forwards onto the public agent answer on port 8888. Depending on what the application does, it may need to talk to other containers/services running on the cluster and this internal only communication can be facilitated with MLB (Marathon Load Balancer). The apps running on the cluster will have either pre-defined IP ports, or defined at run time and then MLB picks up these ports/addresses as part of service discovery. The private agents never know or see the outside world/Internet.

You would only allow traffic on ports that you define and need to connect for management, application use, etc. Here is my sample security group for DC/OS clusters I create on AWS:

--Please note that for UI, SSH, and Marathon access, I only have those specific ports allowed, and more importantly, it's not open to the world, it's only open to the IP address of my system, on my network. I could open it up to all systems in my network, but for now, it's just my workstation.



My egress rules are: -- Any node that has a public Internet interface is allowed to send traffic out on any port.



#2 How do we make sure port 8080 is not opened (to outside world) from within the DC/OS installation script itself?

Technically the installation script doesn't open this port- this port is used for Marathon and this port and access to Marathon are not even password protected. Therefore access to the port should be very limited (e.g. your subnet block in the office, or IP address of DC/OS admins)

Additionally, you could prevent end-users from accessing Marathon on port 8080 and have them use a URL such as:

`https://master-url/service/marathon/`

#3 How can we set-up port 8080 password with non default, or implement a password for this port?.

As far as I know, there is no mechanism to password protect the Marathon endpoint 8080. You would want to restrict access at the network layer. Please the the above items.

And while we are on the topic of passwords-- another absolute must is to change the default password for the bootstrap user (default login user for DC/OS UI), this is done at installation time and the updated password should be included in your clusters "`config.yaml`". Making a custom different password for bootstrapuser is done by:

```
sudo bash dcos_generate_config.ee.sh --hash-password <superuser_password>
```

This is a must-- this user account must either be removed or the password should absolutely be changed on production systems that are accessible over the Internet. This is

a well know and published user account on DC/OS and anyone with the knowledge or wherewithal to do some research will easily find this password and hence, the key to your DC/OS cluster!

Conclusion

This paper (chat) really covers the beginning basics for configuring your AWS environment to support DC/OS. In addition to these basic items, there are many other considerations to make; such as services permitted on the network, ingress-egress ports, operating environment hardening, security group configuration, firewall rules, etc. Beyond a PoC (Proof of Concept) environment, it is not adequate to simply install AWS instances to run DC/OS on top of. The underlying infrastructure must be completely vetted, verified, tested, and automated. Automation will be a key aspect of recovering from operating environment issues and for restoring the environment in the event it needs to be recreated. Additionally, keep in mind that distributed systems change over time, and, constantly. Configurations that work today may not be practical tomorrow- As the requirements for the business (environment) change, it is likely that the DC/OS cluster and underlying infrastructure will also need to change to meet those requirements.