em User Manual

Table of Contents

Introduction

em (Environment Manager) is a set of utilities for creating and managing environments in Amazon EC2.

Installation

Installation Pre-Requisites

Java SDK

em requires access to Java SDK, version 1.7 or newer.

Download and install the latest JDK 1.7 following instructions specific to your system from Oracle web site:
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Once JDK has been installed on your system, export JAVA_HOME in your environment and add `${JAVA_HOME}/bin` to your PATH.

Amazon EC2 Command Line Interface Tools

em requires access to Amazon EC2 command line interface tools, version 1.7.3.0 or newer. Amazon EC2 command line interface tools can be downloaded from Amazon web site: http://aws.amazon.com/developertools/351.

Installation instructions are available here:
http://docs.aws.amazon.com/AWSEC2/latest/CommandLineReference/set-up-ec2-cli-linux.html

A short version of the installation instructions, tested on Mac OS, is available here:
https://kb.novaordis.com/index.php/Amazon_EC2_CLI_Installation

After installation, Amazon EC2 command line interface tools must be made available to em. Set the EC2_HOME environment variable to point to the installation directory and adjust the PATH environment variable as follows:

```
PATH=${PATH}:${EC2_HOME}/bin
```

After reloading your environment, you can check whether the installation was performed correctly by executing:

```
ec2-version
```

The command should return 1.7.3.0 or higher.

Set the Amazon EC2 API Access Keys

Each Amazon EC2 IAM user has a set of API access keys. These keys are needed when the user attempts to make programmatic calls to AWS or EC2, using Amazon EC2 CLI tools. The user can create, modify, view and rotate these access keys.

Provision the access keys for your account. Once provisioned, set them in the environment as follows:

```
export AWS_ACCESS_KEY=your-aws-access-key-id
export AWS_SECRET_KEY=your-aws-secret-key
```

Test whether the access key installation worked by executing:

```
ec2-describe-regions
```

The command should output something similar to:

```
nombp1:doc ovidiu$ ec2-describe-regions
REGION      eu-central-1      ec2.eu-central-1.amazonaws.com
REGION      sa-east-1         ec2.sa-east-1.amazonaws.com
REGION      ap-northeast-1    ec2.ap-northeast-1.amazonaws.com
REGION      eu-west-1         ec2.eu-west-1.amazonaws.com
REGION      us-east-1         ec2.us-east-1.amazonaws.com
REGION      us-west-1         ec2.us-west-1.amazonaws.com
REGION      us-west-2         ec2.us-west-2.amazonaws.com
REGION      ap-southeast-2    ec2.ap-southeast-2.amazonaws.com
REGION      ap-southeast-1    ec2.ap-southeast-1.amazonaws.com
```

Set your Amazon EC2 Region

Pick the appropriate Amazon EC2 region from the list returned by the ec2-describe-regions command and set the following environment variable:

```
export EC2_URL=https://<service_endpoint>
```

where the <service_endpoint> value should come from the third column of the ec2-describe-regions command output.
Example:

```
export EC2_URL=https://ec2.us-west-2.amazonaws.com
```

Instance Access Key Pairs

Amazon AWS uses public-key cryptography to secure the login to instances. The instance has no password - you use a key pair to access your instance securely. The key pairs are provisioned either via the web interface or with Amazon CLI tools. During the provisioning process, the keys pairs are named.

When the instance is created, you need to specify the name of the key pair to use to protect access to it: Amazon will install the public certificate in `~/.ssh/authorized_keys` when creating the instance, and then you need to provide the private key of the pair to your ssh client when logging into the instance.

em uses at least two key pairs: one key pair for access during instance provisioning and one or more key pairs for routine access.

Instance Provisioning Key Pair

You need to create and register with the Amazon EC2 management facilities a provisioning key pair.

This key pair will be used for access every time a new instance is created and configured. Once the configuration (initial overlay) phase is over, the key is removed and you are free to install as many regular access keys as you wish.

We recommend to name of the installation provisioning key pair "em-provisioning-key-pair", but this is not required.

Details on how to create key pairs are available here:
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html

Instance Access Key Pairs

You must install at least one instance access key pair.

IMPORTANT: If you don't register at least one instance access key pair while creating the instance with em, the instance will be inaccessible for login, because em removes temporary provisioning access credentials at the end of the instance provisioning process. Is not required to register your regular instance access keys with Amazon EC2 management facilities. em will install and manage them.

Security Groups

General theory on security groups. Details on creating security groups needed by the environment. See "Create an NFS Server Security Group".

Subnets

General theory on subnets. Details on creating subnets needed by the environment. See "Building the NFS Server".

em Installation

Download the Installation Archive

Download the latest release from https://github.com/NovaOrdis/em/releases

You will get a zip file that contains everything em needs in order to build and interact with Amazon EC2 environments.

Extract the Installation Archive

Extract the content of the em installation zip into the directory conventionally used for external utilities. /opt or /usr/local are common choices. For the remainder of this document we will assume that the external utility directory is /opt.

The extraction process will create an “em-<version>” top-level directory.

Assuming that the top level directory is em-2.5, link to it with a generic link “em”, as follows:

```
cd /opt
ln -s ./em-2.5 ./em
```

Setup the Environment Variables

Set the EM_HOME environment variable to point to the generic “em” link. This way, you will be able to upgrade by simply unzipping a new version and re-linking, without any environment modification.

Also adjust the PATH environment variable as follows:

```
export EM_HOME=/opt/em
export PATH=${PATH}:${EM_HOME}/bin:~/.em/bin
```

~/.em/bin is the directory where em creates “direct login” links, where executing a remote environment instance name as a command logs the user into that instance.

Install the Instance Provisioning Private Key File

The Amazon EC2 instance provisioning key creation process generates a private key file in PEM format that is automatically downloaded by the browser when the key pair is created. For more details, see "Instance Provisioning Key Pair".

You will need to make this key accessible to em.

Conventionally, the private key file generated by Amazon is stored under a different name (em-provisioning-private-key.pem) in the ~/.ssh directory. The permissions should be adjusted as follows:

```
mv <browser_download_dir>/em-provisioning-key-pair.pem \
   ~/.ssh/em-provisioning-private-key.pem

chmod go-rwx ~/.ssh/em-provisioning-private-key.pem
```

For more details about how em uses the access keys, see "Instance Access Key Pairs" section.

Configure the NFS File Server Internal IP

This is an optional step that is only required if you intend to use the newly installed em to manage an already existing Amazon EC2 environment. If you plan to create a completely new environment, you may skip this step.

Write the internal IP address of the existing environment's NFS server into the em local configuration file ${HOME}/.em/em.conf, as follows:

```
nfs_server_internal_ip=<nfs-server-internal-ip-address>
```

You can get the address by running em status and applying heuristics to figure out which instance is the NFS server. If the instances were named following the conventions offered by this manual, the NFS server's name should start with "nfs".

Test the Installation

Reload your environment to make sure the path is updated, and then execute:

```
em version
```

and then:

```
em status
```

If the environment was previously used, you will get the list of instances already created in the environment:

```
em status
name state   id          public-ip private-ip
f01  stopped i-db0ab82d            172.31.25.44
b01  stopped i-55397ea3            172.31.16.215
b02  stopped i-fe014408            172.31.30.67
```

If no instance were previously created with em, the command should still succeed and produce something similar with:

```
em status
no instances
```

Building a Basic Environment

Basic Environment Overview

The NFS File Server

A basic environment consists in an NFS fileserver, which will serve as environment configuration keeper and shared file keeper, and other instances.

At the time of this writing, any environment *needs* a file server. In the future, we may add support for HTTP-only based environments.

The fileserver does not need significant resources. We routinely use reasonably large environments – tens of instances – served by a t2.micro NFS server, provided that the client instances prefer "local" storage to the shared file space.

The shared file space should only be used for installation kits and configuration, and it should be mostly read by the client instances.

The NFS file server should be kept around (i.e. not terminated) for the life of the environment. It is the only non-expendable – and this just for the useful life of the environment – instance of the environment. Provided that key state on its file system has been backed up, the NFS file server can be terminated, the restored, though we don't recommend this approach.

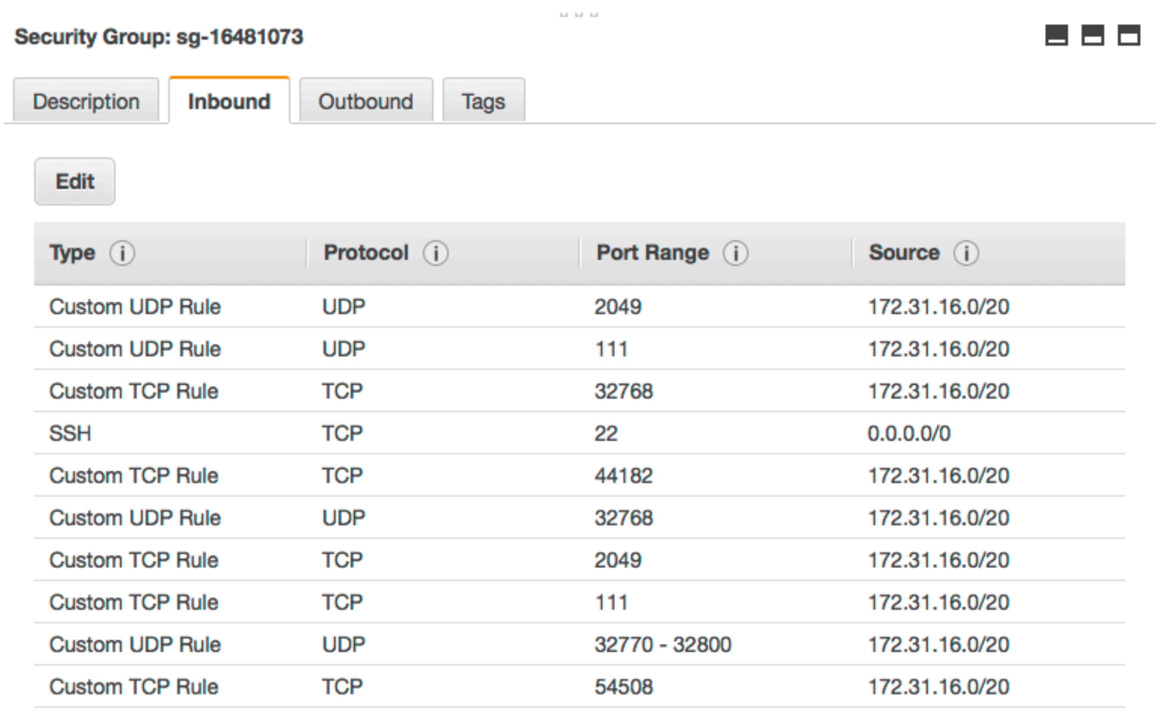
Other Instances

An environment can have an arbitrary number of expendable instances, which can be created and then terminated arbitrarily.

The typical use for "ephemeral environments" is large instance-count load tests. An environment comprising hundreds of instances can be created, and then load can be generated and applied within the environment, data collected and finally, all instances discarded.

Provided that Amazon EC2 bills for storage space even if the instance is stopped, and instances need at least some local storage, this will result in significant bill savings. If you don't care about this part, environments can be kept around in stopped state – or even in running state – for as long as you wish.

Create an NFS Server Security Group


Security Group: sg-16481073

Type	Protocol	Port Range	Source
Custom UDP Rule	UDP	2049	172.31.16.0/20
Custom UDP Rule	UDP	111	172.31.16.0/20
Custom TCP Rule	TCP	32768	172.31.16.0/20
SSH	TCP	22	0.0.0.0/0
Custom TCP Rule	TCP	44182	172.31.16.0/20
Custom UDP Rule	UDP	32768	172.31.16.0/20
Custom TCP Rule	TCP	2049	172.31.16.0/20
Custom TCP Rule	TCP	111	172.31.16.0/20
Custom UDP Rule	UDP	32770 - 32800	172.31.16.0/20
Custom TCP Rule	TCP	54508	172.31.16.0/20

Building the NFS Server

Conventionally, the instances of an environment are named using letters and digits that give a hint on their function in the environment. Following this convention, the only NFS file server of an environment could be named “nfs01”:

```
em create --group <nfs-server-security-group> nfs01
```

IMPORTANT: The security group that allows NFS clients to connect to the NFS server must be created in advance. The group ID must then be specified on the command line following the “--group" option when creating the NFS server instance, otherwise the NFS server will be inaccessible. For more details on Amazon EC2 security groups, see “Security Groups”.

The NFS server security group to use with Nova Ordis test cloud is “sg-16481073”.

To discover more options available to you when running the “create” command, execute:

```
em -h create
```

Run with --verbose and determine the ami_type, availability-zone, subnet and block-device-mapping. Adjust values accordingly.

A few seconds after creation, the NFS server non-configured instance should be available in the “em status” query:

```
nombp1:~ ovidiu$ em status
name  state   id         public-ip    private-ip
...
nfs01 running i-1086b9e6 52.25.189.47 172.31.16.246
...
```

At this point, the instance is not yet accessible for log in by the regular users, because it was configured with a key that can be only used for installation. The instance will become accessible after completing the overlay process:

```
em sync

em overlay nfs-server nfs01
```

IMPORTANT: The nfs-overlay does not start the NFS server (temporary technical difficulties: the attempt to start the NFS server during the installation failed. The section is commented out in linux.shlib# setup-nfs-server. Figure out what happened and restore it). Until this is fixed, it is important to stop and then start the newly created NFS instance. This is also a good test to see if the NFS starts correctly at boot.

```
em stop nfs01

em start nfs01
```

When this step completes successfully, the NFS server should be available for login:

```
em sync

nfs01

ec2-user@nfs01>
```

All logins are executed by default as ec2-user.

NFS Server Termination Protection

em is not yet capable of specifying termination protection on instance creation. Since the NFS server is critical for the environment, you should enable termination protection from the web interface by right-clicking on the instance, and then -> Instance Settings -> Change Termination Protection -> Enable.

Building a Basic Instance

TODO (lessons trying to use em after a relatively long time)

1. If the provisioning key has been regenerated, make sure to install the provisioning public key on the NFS server.
2. Create a ~/.em/em.conf and add the following:

The environment's NFS server must be up and running when a new instance is being built.

Remember that you need to correlate the load that the instance will be running with the instance type. The default instance type is "t2.micro". Also, if the instance is supposed to run any services that should be accessible from outside it, you may want to use a different security group, as the default one does not allow inbound connections.

Use the --ami_id, --group, --instance-type, --storage-size and --subnet "create" command options to qualify the instance.

```
em create test01
```

You may also consider using the --dry-run flag to simulate the instance creation process without actually creating anything.

```
em sync

em overlay basic test01
```

Successful basic configuration output:

```
em overlay basic test01
applying overlay basic to test01 ...
The authenticity of host '52.25.237.21 (52.25.237.21)' can't be
established.
RSA key fingerprint is b6:dc:bf:3b:dd:af:cb:28:9e:86:4c:30:23:06:ec:e8.
Are you sure you want to continue connecting (yes/no)? yes
hostname successfully set to test01
installing package zip ...
zip successfully installed
installing package unzip ...
unzip successfully installed
cursor configured in /root/.bashrc
alias h='history' configured in /root/.bashrc
alias r='sudo su -' configured in /root/.bashrc
alias nfs='cd /nfs' configured in /root/.bashrc
environment configured for ec2-api-tools
cursor configured in /home/ec2-user/.bashrc
alias h='history' configured in /home/ec2-user/.bashrc
alias r='sudo su -' configured in /home/ec2-user/.bashrc
alias nfs='cd /nfs' configured in /home/ec2-user/.bashrc
environment configured for ec2-api-tools
removed requiretty from /etc/sudoers
The authenticity of host '172.31.17.11 (172.31.17.11)' can't be
established.
ECDSA key fingerprint is
f3:6e:2d:fa:ca:34:6b:ff:1d:d3:2e:0a:1d:d7:1b:b7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.31.17.11' (ECDSA) to the list of known
hosts.
NFS server identity successfully configured
installing package nfs-utils ...
created NFS mount point /nfs
NFS client successfully configured
installed ovidiu@nombp1 key into ec2-user's authorized_keys file
Connection to 52.25.237.21 closed.
```

Starting a Basic Instance

```
em start test01
```

Logging into a Basic Instance

```
em sync

test01

ec2-user@test01>
```

Stopping a Basic Instance

```
em stop test01
```

Terminating a Basic Instance

```
em terminate test01
```

Note that you don’t need to stop an instance to terminate it. An instance can be terminated directly from a running state.

Building a Java-enabled Instance

A java-enabled instance can be built from scratch, or an already existing “basic” instance can be upgraded to “java-enabled”. In both cases, the procedure is the same. You will need to apply the “java” overlay (the “java” overlay depends on the “basic” overlay, so all configuration applied there is also applied to a “java” instance).

When creating the instance, make sure to allocate resources appropriate to the future load (memory, CPUs) by choosing the appropriate instance type.

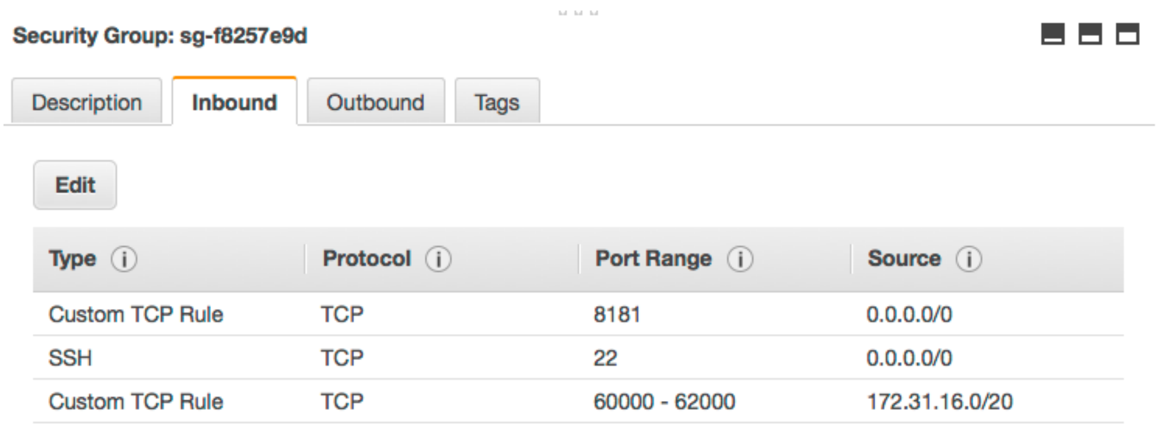
Also, if the instance is supposed to expose services to other instances, you may want to change the security group, as the security group a basic instance is built with does not allow inbound connections.

The “java” overlay needs a Java SDK archive in the environment’s repository. The name of the template is specified in the environment’s configuration file /nfs/environment/environment.conf as “java_template”. The corresponding file must exist under /nfs/environment/repository.

```
em overlay java test01
```

Building an AMQ Cluster

Create the AMQ Cluster Security Group


Security Group: sg-f8257e9d

Type	Protocol	Port Range	Source
Custom TCP Rule	TCP	8181	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
Custom TCP Rule	TCP	60000 - 62000	172.31.16.0/20

Build the AMQ Instances

Clustered JBoss AMQ brokers can be built by applying the “amq-broker” overlay. Each EC2 instance should run a single broker instance, but the broker instances will be connected via network bridges that are automatically setup by the overlay.

When creating the instance, make sure to allocate resources (memory, CPUs) appropriate to the load the instance is expected to run. Start by choosing the appropriate instance type.

IMPORTANT: The security group that allows AMQ clients to connect to the AMQ server must be created in advance. The group ID must then be specified on the command line following the “--group" option when creating the AMQ server instance, otherwise the AMQ server will be inaccessible. For more details on Amazon EC2 security groups, see “Security Groups”. The AMQ server security group to use with Nova Ordis test cloud is “sg-f8257e9d”.

IMPORTANT: The “amq-broker” overlay needs an AMQ template in the environment’s repository. The name of the template is specified in the environment’s configuration file /nfs/environment/environment.conf as “amq_template”. The corresponding file must exist under /nfs/environment/repository.

Note that because the AMQ instances need Java, all considerations described in the "Building a Java-enabled Instance" section apply (we need a Java SDK, etc.).

Create all AMQ cluster instances at the same time, before applying the overlays. This will allow the overlay to correctly resolve all instance names.

```
em create --instance-type c4.2xlarge \
--group sg-f8257e9d --storage-size 15 b01

em create --instance-type c4.2xlarge \
--group sg-f8257e9d --storage-size 15 b02
```

Before applying the "amq-broker" overlay, update the broker configuration (memory, sizes, and especially the cluster membership).

How do I portably do that, currently the only way is to modify the overlay.

```
em sync

em overlay amq-broker b01

em overlay amq-broker b02

em stop b01 b02

em start b01 b02
```

The "amq-broker" overlay configures the instance to start the AMQ broker at boot .

Testing the Installation

Start all instances and make sure the bridges connect to each other.

Building a JBoss EAP Domain

JBoss EAP domains can be built by with the “eap-node” and “eap-dc” overlays: the domain controllers are built first with “eap-dc”. Once the domain controllers are up and running, the subordinated host controllers and nodes are built with “eap-node”.

Note that at least one domain controller for the environment must be up and running when building the subordinated host controllers, because em interacts with the domain controller when building and configuring the host controllers.

The following commands show how to build a domain consisting of one domain controller and two subordinated host controllers, each host controller controlling an arbitrary number of nodes.

The domain controller runs on the host “dc1” and the host controllers runs on the hosts “n1” and “n2”.

Doman Controller

The environment’s NFS server must be up and running.

```
em create \
   --ami_id ami-775e4f16 \
   --group sg-3b64b65c \
   --storage-size 10 \
   dc1

em sync

em overlay eap-dc --server-groups=sg1:full,sg2:full-ha dc1

em stop dc1

em start dc1 # since each EAP instance is configured to start at boot
             # this insures the Domain Controller is up and running
             # during the installation of the subordinate host
             # controllers
```

Subordinate Host Controller

The environment's NFS server and the domain controller must be up and running.

Empty "topology" string is acceptable; it means the host.xml <servers> section will be cleared.

```
em create \
   --ami_id ami-775e4f16 \
   --group sg-3b64b65c \
   --storage-size 10 \
   h1

em sync

em overlay eap-node \
   --domain-controller=dc1 \
   --topology=sg1:s1,s2%sg2:s3,s4 \
   h1

em stop h1

em start h1
```

TODO

- CLI invocation into the domain controller use hardcoded admin/admin123 and port 9999. Need to expose the user as argument (or use subordinate host controller server identity, which is probably better)

Running Arbitrary Shell Commands Across the Environment

```
em run p001 p002 p003 -- uptime
```

Miscellaneous

In-Line Documentation

```
em -h|--help <command>
```

Verbose Execution

Verbose execution is usually helpful if something goes wrong and you need more details on what happens while the command is executed:

```
em -v|--verbose <command>
```

Extending em

Overlays

Overlays must be developed in such a way that they are idempotent.