DAIR

Infrastructure as a Service: Best Practices

canarie.ca | @canarie_inc

Table of Contents

Introduction

Following best practices will maximize your results with Infrastructure as a Service (IaaS), both from the DAIR infrastructure and with other commercial clouds. The following are some recommended approaches.

Instance Initialization

Instances are as likely to fail on DAIR as any commercial commodity server. Instances should be regarded as disposable and contain no data, applications, or configuration that cannot be easily recreated. If an instance is to fail, one of the benefits of cloud computing is the ability to start another instance within seconds or minutes. To take advantage of this capability you need to give some thought and effort to how you initialize your instances.

Instance initialization is anything that needs to happen to a server before it can become truly useful to you or your end users. Using an example implementation of providing web services, on a new instance the Apache Web Server needs to be installed and configured, and the Python Django web application needs to be checked out of source control and configured. These are the types of tasks that a developer or system administrator will do manually to get an instance into a ready state and, if that instance crashes, they will have to manually do that work all over again. It is to your advantage to automate instance initialization.

Here are three ways you can automate instance initialization.

Scripting

You can write a script that will be run immediately after the instance boots, which will put your instance into a ready state. The most widely supported scripting language is bash but there are images that support Python and Ruby as well.

An example script called prep-app-server.sh might look like:

```
#! /bin/bash
sudo apt-get install apache2 git
sudo sed -i “s/some config/my config/g” /etc/apache2/apache2.conf
git clone git://github.com/mycompany/myproject.git
sed -i “s/dev db/prod db/g” myproject/settings.py
```

To launch an instance with the script running immediately after it boots, you would run the following command:

```
$ nova –-key_name mykey –-user-data prep-app-server.sh –-image “6f9bba21-5689-4e5c-af87-63c54a4ddbfe” –-flavor 2 boot myinstancename
```

The –-user-data parameter specifies the User Data file where the instructions are located to initialize the instance.

The nova command is an OpenStack command line interface tool that needs some introduction. This is a powerful command line tool that offers remote management of your environment.

To install nova on Centos:

```
$ sudo yum install epel-release
$ sudo yum install -y python-pip
$ sudo pip install --upgrade pip
$ sudo pip install python-novaclient
```

You now have the nova command

To install nova on Ubuntu

```
$ sudo apt install python-pip
$ sudo pip install --upgrade pip
$ sudo pip install python-novaclient
```

You now have the nova command

To run your script when starting an instance from the Dashboard, use the “Post-Creation” tab of the Launch Instance window. Simply copy the contents of the script into the “Customization Script” text area in the Launch Instance dialog.

NOTE: Unfortunately, this type of automation is not compatible with Windows instances.

Snapshots

Once you’ve manually got your instance into a ready state you can Snapshot that instance, which can be used to start new instances based on the original. This has benefits and drawbacks. The primary benefit is that the instance you start based on the original should be ready to go as soon as it boots without any manual intervention or scripting (which may include time consuming downloads). The primary drawback is that all of the decisions you made when manually creating the original instance are baked into any new instances based on that instance.

One other factor is the root disk is the only disks data to be included in the snapshot. No data on the ephemeral drive nor volumes is included in the snapshot.

For you, the answer may lie somewhere between scripting and bundling. One suggestion is to install all of the large pieces of software you need on an instance, create a snapshot, and then configure the installed software with a Post-Creation script when launching the instance from the snapshot.

Software Configuration Management Tools

There is a new breed of software that is rapidly gaining mainstream adoption. These tools are dedicated to managing software configuration on both physical and virtual servers alike. There are many contenders in this such as Chef, Puppet, Ansible, SaltStack and Jenkins. This is by no means an exhaustive list. There is much to recommend each tool and the decision between them may come down to personal preference. The use of these tools is far beyond the scope of this document but the gradual adoption of one of these will go a long way to automating your infrastructure.

Security

Security is important in any environment. Much of the security advice that is applicable to running your own physical servers is applicable to running virtual machines in the cloud.

Firewall

One notable exception is the use of Security Groups in DAIR. Security Groups act as a firewall for your VMs. Before even launching your first VM you should go to the Security Groups tab in the Dashboard and review the rules there. These rules specify what kind of traffic can reach your VMs and from where. In particular, editing the CIDR will restrict who can connect to the VM, which is a very powerful security measure. We recommend locking all access down to the source IP of those using the service. You can have each user search google with “What is my IP” to find out the public IP any user is accessing your service with.

SSH Keys

A private and public key authentication framework is supported by DAIR and Highly recommended as it provides a 2-factor authentication to your VM’s if you add a pin.
When you create a new Keypair you are entirely responsible for ensuring the security of the private half of that Keypair. Once it has been downloaded to your local machine, that is the only file of that private key in existence. If you lose it, you will not be able to access any VM launched

with that Keypair. If it is compromised, you must consider any VM launched with that Keypair to be potentially compromised. Safeguard your private keys.

A good checklist for security in any environment follows (source: Twenty Rules for Amazon Cloud Security)

1. Encrypt all network traffic.
2. Use only encrypted file systems for block devices and non-root local devices.
3. Encrypt everything you put in Object Storage using strong encryption.
4. Never allow decryption keys to enter the cloud—unless and only for the duration of an actual decryption activity.
5. Include NO authentication credentials in your Images except a key for decrypting the file system key.
6. Pass in your file system key encrypted at instance start-up.
7. Do not allow password-based authentication for shell access. Ever.
8. Do not require passwords for sudo access.
9. Design your systems so that you do not rely on a particular Image structure for your application to function.
10. Regularly pull full backups out of your cloud and store them securely elsewhere.
11. Run only one service per Instance.
12. Open only the minimum ports necessary to support the services on an Instance.
13. Specify source addresses when setting up your instance; only allow global access for global services like HTTP/HTTPS.
14. Segment out sensitive data from non-sensitive data into separate databases in separate security groups when hosting an application with highly sensitive data.
15. Automate your security embarrassments *(You know you have had them at one time or another. Things like that anonymous FTP site you have to have open for the batch file a client is sending you every night.)*
16. Install a host-based intrusion detection system like OSSEC.
17. Leverage system hardening tools like Bastille Linux.
18. If you suspect a compromise, backup the root file system and block volumes, and shut down the Instance. You can perform forensics on an uncompromised system later.
19. Design things so you can roll out a security patch to an Image and simply relaunch your Instances.
20. Above all else, write secure web applications.

Patching

Ultimately upgrading packages for security is a necessity. How and when you take those security patches depends on your application's tolerance for change. Typically, most security patches are fully backwards compatible so you are free to take them as they become available. However, your application may depend on some nuanced detail in a package that gets changed by a security patch that breaks your application. With knowledge of your application, only you can decide how and when to apply security patches. If you are worried about the impact of a patch snapshot your instance first patch and test and if all goes well delete the snapshot. If there are issues back out.

One approach is to trust your operating system vendor and enable automatic security patches. Here is how to enable automatic security updates for the operating systems used in DAIR:

- Ubuntu
- CentOS
- Windows

A more conservative approach is to just receive notifications that security patches are available. This allows you to plan for the upgrades and potentially try out the upgrades in a development environment before applying them in your test environment.

- Ubuntu
- CentOS
- Windows

By default, all Linux distributions in DAIR will notify you of patches available when you login. Windows is configured with patching services disabled.

Kernels

"You don't have to worry too much about security issues in kernels. Most security issues don't happen there. Far more important is compatibility / stability with your hardware platform.

If there are newer vendor supplied kernels and you are doing maintenance anyway, throwing one up using the command you supplied is a good idea. If you are using Amazon kernels, I would just leave it that way and not worry about it. Only if you have a specific need would I bother with making it use a custom kernel from inside your AMI."

Source: Do I need to manually administer EC2 kernel updates?

Application

Your operating system is only as secure as the applications you are running (or developing) in it. Application security is beyond the scope of this document but the Open Web Application Security Project is an excellent starting point for helping you develop secure web applications.

Volumes

Store all data on Volumes!

Any data, applications, or configuration that cannot be easily recreated must be stored on a Volume. The default storage that an instance launches using its' root disk **(/dev/vda)** and an ephemeral disk **(/dev/vdb)** is transient and does not persist. If an instance is terminated, any data on these disks will be irrevocably lost. If you store the data on a Volume, the data will survive an instance termination and the Volume can be attached to another instance to continue working with the data.

When attaching volumes via the Dashboard or the command line you need to know that it ignores the specified device. The volume will be attached at the first available device name in **/dev/vd[a-z] (typically 'c')**. When a volume is detached the device name will not be reused until the instance is rebooted *(eg. continually reattaching a volume will cause it to move down the chain of available names)*. If an instance is rebooted internally *(i.e. the user does the reboot command in a terminal within the instance)* the attached volumes will get new device names in accordance with the previous points unless they are already in that order.

/etc/fstab

When using the fstab to handle automatically mounting volumes additional care should be taken since the device name a volume receives can change. Therefore entries in the fstab should use either labels or UUIDs.

It's also useful to add nobootwait to the options for an entry so that the instance will not hang during a reboot if the volume has become detached for any reason. As well, errors=remount should prevent any disk corruption in the event of network or server downtime.

These are the steps starting at creation for safely adding a volume to fstab. Step 1 can be skipped for existing volumes.

1. Create your volume, attach it to your instance, create your file system of choice and complete one of the two optional steps below
2. (Optional) Label the file system
    a. This step depends on the file system type; a list of utilities is available here: https://wiki.archlinux.org/index.php/Persistent_block_device_naming - By-label
    b. Can also usually be done during file system creation by specifying -L <label> when using mkfs, see the man page for your file system type

```
$ mkfs <fstype>
```

    c. To find a forgotten label use

```
ls -l /dev/disk/by-label or blkid /dev/XXX
```

3. (Optional) Find the UUID

```
ls -l /dev/disk/by-uuid or blkid /dev/XXX
```

4. When adding the entry to your fstab specify the file system (first column) by label or UUID depending on your choice above
    a. LABEL=<label>
    b. UUID=<UUID>

Example fstab entries:

```
UUID=c4d66b8a-2984-438b-8004-394613d50c30  /mnt      auto    nobootwait    0       0
LABEL=MY_VOLUME          /some/dir       ext3    nobootwait    0       0
```

As an alternative, or if nobootwait did not work and the instance hangs due to not being able to access the volume, you can create a startup and shutdown script to attach and detach the volume. This also ensures that the volumes are correctly detached if you reboot your instance.

Startup: add the following to /etc/rc.local, above the exit 0 line:

/root/attach-volumes.sh

Next, create /root/attach-volumes.sh:

```
#!/bin/bash
source /path/to/your/rc/file

nova volume-attach “server id” “volume id” “/dev/vdc”
sleep 4
mount -L <LABEL /your/mount/point
```

Repeat those three lines for however many volumes you want to mount.

To create the shutdown script, create the file /etc/init.d/detachvolumes

```
#! /bin/bash
### BEGIN INIT INFO
# Provides:          detachvolumes
# Required-Start:
# Required-Stop:
# Default-Start:
# Default-Stop:      0 6
# Short-Description: Detach external volumes at shutdown and reboot
# Description:
### END INIT INFO

PATH=/sbin:/usr/sbin:/bin:/usr/bin
. /lib/init/vars.sh

. /lib/lsb/init-functions

umask 022

do_stop () {
        # umount fs
        umount /your/mount/point
        # then detach external volumes
        source /path/to/your/rc/file
        sleep 1
        nova volume-detach “server id” “volume id”
        sleep 1
}

case "$1" in
  start)
        # No-op
        ;;
  restart|reload|force-reload)
        echo "Error: argument '$1' not supported" >&2
        exit 3
        ;;
```

```
 stop)
        do_stop
        ;;
 *)
        echo "Usage: $0 start|stop" >&2
        exit 3
        ;;
esac
```

Then enable this script to be run during shutdown:

```
$ update-rc.d detachvolumes stop 22 0 6 .
```

Backups

Make regular backups of your Volumes!

Volumes can and will go bad. Make regular backups of your volumes to another DAIR region, other cloud volumes or local volumes using rsync. If you have databases, find out how to perform consistent backups for your particular database software.

Architecture

It is beyond the scope of this document to try to determine what architecture is best for your application. However, here are some resources that can help you do so.

- AWS Architecture Center
- Cloud Application Architecture
- Scaling Experts