Migration of historic systems hosted in the cloud

by

Adrien FIAND

Submitted to

CESI

in partial satisfaction of the

requirements for the degree of

Engineering cycle, Computer Science Specialty

Supervised by

Romain BRUNELOT

September 11, 2021

- Problematic -

Amazon Web Services Cloud (AWS)
Linux Elastic Cloud 2 (EC2)
-
Migrating from Para-Virtualized (PV) instance type
to Hardware-Virtual-Machine (HVM) instance type

- Abstract -

This documents aims to find a solution to the problematic written above. In order to fully comprehend the issue that arise the document will start by giving an introduction to Cloud Technologies, in this introduction an quick opening on the issue and virtualization is done.

The introduction will be followed by a State-Of-The-Art of Virtualization, while going through a bit of history it will mostly raise several technical aspect of the subject. Then a deeper presentation of Cloud Provider and AWS in particular will be done. At that point I will go deeper on the problematic, the associated context and stakes.

Then will begin the development parts, where I list several hypotheses. Based off reflexion I will narrow down to only one hypothesis that seems suitable but require testing. The chosen hypothesis is still very broad and multiple approaches are possible, I will go deep in the principle involved to choose an approach that I will test.

Finally the chosen approach will be tested, in the same environment in case multiple trials are necessary. The procedure used will be illustrated in this document directly, for the technical development appendix A & B are available. The document will end with a the result of the test and a Conclusion on whether the problematic is solved or not.

I. Table of Contents

II. List of Figures

III. Acknowledgments

I would like to extend my special thanks to Blueberry Consultants Ltd, which is the company where I made my 4th year internship. Blueberry is a United Kingdom based group, it is composed of three companies: Blueberry Consultants, Blueberry Software, Blueberry Systems. Each company in the group has its own services offer around software development. Their customer base is spread across the world.

I joined Blueberry Consultants as a Sysadmin Intern. The opportunity they gave me to grow my knowledge in information systems was amazing and sort of unexpected, in a pleasant way of course. The projects and tasks I worked on opened me to the world of SysOps/DevOps, as well as cloud technologies: Blueberry aims to build and put as much as they can in the cloud mainly for resiliency purposes.

Near the end of my internship we started migrating servers from a region to another region in the Amazon Web Services cloud. However a few servers couldn't be migrated straight away because of their instance type. . . Unfortunately I didn't had time to treat this during my internship. But without my experience at Blueberry I wouldn't have thought, nor came across that problematic, which is the perfect candidate for this document.

IV. Summary | Résumé

1. English | Anglais

In this document we raise the problematic of migrating server in the cloud. While this could seem very easy and it is for the most part, we actually raise the rare case where the migration is blocked due to incompatibilities. Here the issue was that the original Linux virtual machine was using a para-virtualization (PV) mechanism, and the only mechanism where the migration needed to land was hardware-assisted mechanism (HVM).

PV is an intrusive mechanism, the system undergoes quite the modification to boot and work. That's those modification that make the system impossible to migrate straight away to HVM. After doing deep research on the virtualization mechanisms and the Linux architecture, a few hypothesis were listed. In all of these apart starting from scratch only one seemed plausible : Swapping the system files between HVM and PV. We prove that this hypothesis was a valid solution as we managed to migrate from the PV to HVM, using it. Finally an opening is done similar cases and the possibility to proceed the same way.

2. French | Français

Dans ce document, nous soulevons la problématique de la migration de serveur dans le cloud. Bien que cela puisse sembler simple et la plupart vrai, nous soulevons les cas rare où la migration est bloquée en raison d'incompatibilités. Ici, le problème est que la machine virtuelle (Linux) d'origine utilisait un mécanisme de para-virtualisation (PV) et le seul mécanisme disponible de l'autre coté de la migration était hardware-assisted (HVM).

PV est un mécanisme intrusif, le système subit de nombreuses modifications afin de démarrer et fonctionner. Ce sont ces modifications qui rendent le système impossible à migrer directement vers HVM. Après avoir fait des recherches approfondies sur les mécanismes de virtualisation et l'architecture de Linux, quelques hypothèses ont été répertoriées. Dans toutes celles-ci, à part repartir de zéro une seule semblait plausible : échanger les fichier système entre PV et HVM. Nous prouvons que cette hypothèse est une solution valide car nous parvenons à migrer de PV vers HVM, avec celle-ci. Enfin une ouverture se fait dans des cas similaires et la possibilité de procéder de la même manière.

V. Introduction

1. Notion of Cloud

Since it's early beginning the Cloud has open a lot of doors to business expansion. of course it has pros and cons like On-Premise infrastructures have its own. The idea behind the Cloud, to describe it simply, is to offer you (a business), the same possibilities as On-Premise but in a better, faster and cheaper way. The concept of Cloud has been around for years but Cloud solutions only started to take off in the last few years. It has been a solid decade of evolution to get to today Cloud Service Provider offer . . .

Notice that I added the term Service Provider, to get Internet to your home you select an Internet Service Provider between all of the available ones, you usually make that choice with care depending on the offer, price and your needs. To choose a Cloud solution you follow the same steps, except that businesses might choose to use a few Cloud Service Providers to support their needs.

Infrastructure-as-a-Service (IaaS) is the most requested Cloud service. Basically IaaS aims to provides you with the same components as you would traditionally have buy and put On-Premises : Network, Server, Storage. With the Cloud you don't have to worry about the upfront investment for the hardware.

Almost every Cloud Service Provider as an offer for IaaS which usually include : virtual servers, isolated virtual network, data store. You pay-as-you-go : for the number of hours you use the virtual server for example, it usually is an advantage for businesses but not always . . .

2. Migration of Servers

To manage On-Premise servers most businesses use hypervisor such as VMWare ESXi to manage virtual servers that they own and host on their hardware. Virtualization solution of this kind usually leverage a cluster of physical servers, this allows to create, delete or even move virtual machines around the cluster.

To manage Cloud server businesses don't have to know about the hypervisor, that

part is manage by the service provider, the customer (businesses) choose what they want through the web portal or APIs of the provider. The end result is the same as On-Premise.

On-Premises you have your own Data-Center(s) at different location, in the Cloud your services are at several locations usually called region which are close to you (Paris, London ...) each of the region can be seen as a big virtual Data-Center.

At some point, you will eventually be confronted to issues while trying to migrate (move) your servers around Data-Center Locations : Imagine your company needs to relocates in the Cloud due to legislation, or open a new main Data-Center and needs to move everything there . . . Like everything there is limitations to what you can do, for both On-Premise and Cloud, the most common issue is the compatibility in the underlying hardware when you try to migrate servers to other physical hardware.

However when On-Premise that's something you have control of. If your are opening a new Data-Center you can think of that and chose compatible hardware, or make sure that you can install the correct drivers before migrating. In the cloud not so much, you are dependent on the hardware option and availability of you Cloud Service Provider, you do not necessarily have the same level access as On-Premises because everything is behind the your provider APIs.

3. Object of the document

During my work at a company, we had the task of moving our Cloud servers from one region to another one. For most of the servers this was pretty straight forward, like we would have done with an On-Premise hypervisor, we selected and asked to move the virtual server to another location, which has the same underlying characteristic.

However for very few of them we encountered a complication, it was impossible to move them. . . The underlying characteristic on which they were currently running on was not available in the other location where we wanted to move them. That's quite the issue which requires a solution because moving them is not an option, it must be done.

This document aims to solve this problematic. It begins with a bit of history and theory which are required to understand the issue. Then it develops the issue and what's at stake. It continues by presenting hypothesis of solution and testing them in the same condition. Finally it conclude on the best way to overcome the issue, given the needs.

VI. State of the Art - Virtualization

1. Origin

There is no exact trace of when the idea of virtualization began, however there is general consensus that it started in the late 1960s - early 1970s : The goal was to develop efficient and solid solution to share a computer resources between a large subset of users, this is commonly called a time-sharing solution and aims to reduce cost and improve efficiency.

This computing model was revolutionary : back in the early computing days, to make an operation you had to program it using FORTRAN (a scientific purposed language) and Key Punch Card. The card will tell the computer what to do, you had to give them to the computer operator when it was your turn. If you didn't "reserved" enough resources or made an error the job would fail, thus you would basically "reserved" more which meant long pauses before the next user turn, this was called batch processing.

Time-sharing solved this by allowing multiple user to give jobs to the machine, called mainframe, when one user job was in a pause state rather than waiting the processing unit would be filled with another user job, and so on. Thus we utilize all the available resources of the mainframe, or at least we make more efficient usage of it. IBM was the first to commercialize time sharing solution, but the idea is claimed to come from the MIT.

The time-sharing principle worked because the processing unit was more power full than any Input/Output devices, thus time was spent waiting. The way this was done was by partitioning the large mainframe into several logical instances. Each of the instances where "dedicated" to a user, to access it the user would use it's own console. Today virtualization is still based on those principle of optimizing and logically splitting resources usage.

2. Modern Concept

When you run a software it will ask to reserve some resources on the machine for itself, for example Random Access Memory (RAM), it will sometimes ask to monopolize a device or bus making it impossible to use by other software. To fulfill what the software

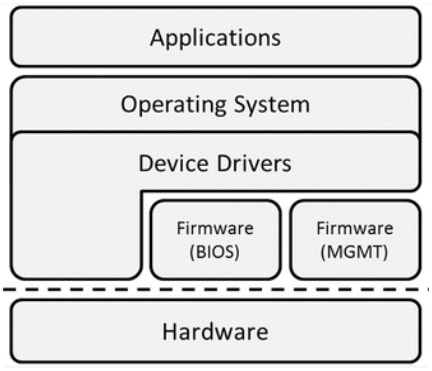
ask, there is between the software and the hardware the operating system (OS).

With the computer performance we now have resources left-over (non utilized) is a big combat. You could run several software on an operating system to try to utilize the underlying hardware at its best, but it's not always something you want, can do or that make sense. Someone once said divide to conquer, putting you software on several different operating is a good thing for maintenance, failure boundaries… Also some software will work on an operating system and some other will need another one.

The solution, for many years was to have many physical server and to try to fit the good hardware in, which was an investment and not scalable. But nowadays the solution is virtualization, it has become a standard practice for any IT team: improvement of agility, flexibility, scalability while providing cost savings. Business use virtualization to run multiple operating systems on the same hardware.

Throughout the years there has been several virtualization solution on the market, each with it's own perks and downsides, but they all have the same common idea of abstraction : hide a part of a process between two elements. To virtualize it is necessary to detach the software from the physical hardware, to do so it relies on a piece of software called hypervisor that will manage and provide virtual versions of the computer components to the virtual machines.

But why do we need abstraction to run multiple operating system on a same host ?



The OS it self is a giant piece of software, its purpose is it to manage the allocation and interactions with memory, processor, buses, devices… The OS is the master of the computer, to work it uses interconnects with the mother board circuitry which only supports a single arbiter. Thus we need to make an abstraction in order to run multiple OS.

Figure 1: Legacy way to host an operating system

3. Hypervisors

Today, server virtualization can be break down in two parts: virtual servers and containers. Containers does not offer a way to run multiple OS on the hardware, because they go further than virtual server by making abstraction of the OS, they virtualize the OS.

Thus I will only focus on virtual servers which are meant to virtualize hardware.

To create virtual servers more commonly called virtual machine (VM) you need a Virtual Machine Monitor or in it's shorter form an hypervisor : it offer an engine to create VMs, to manage them you need a management console. Classes of hypervisors exists :

- Native or Bare-Metal (Type 1) : *VMware ESXi - KVM ? - Hyper-V ? - Xen ?*

This class of hypervisor software replace the traditional OS you put on the hardware, they are on OS themselves, they are directly linked to the hardware. There is nothing between the hypervisor and the hardware. VMs runs on a level above the hypervisor. These kind of hypervisor is very simple by itself, to manage the machine you need a separate management console which can be a software package or web-based.

- Hybrid Hypervisor (Type 3) : *KVM ? - Hyper-V ? - Xen ?*

This type is not recognized by everyone nor well known. Solution from this type are usually classified as Type 1, because they are technically close to it. This type of hypervisor is in direct communication with the hardware which makes it qualify for Type 1. At the same time it appears to run on top of an OS, but it only appears to. There is actually a strong bind between the main OS kernel and the hypervisor, they are at the same layer so it's definitely not a Type 2. It's almost like if the main OS is running in a transparent VM. Hence why solution based on this type are usually referred to as Type 1.

- Hosted (Type 2) : *VMWare Workstation - Virtualbox*

This class of hypervisor software is installed on a traditional OS, they are in this case a third party software, they rely on the existing OS. Thus it adds a layer on top of the existing OS, VM are started on top of that third party layer. Type 2 hypervisor are all-in-one software, they are the engine and the management console.

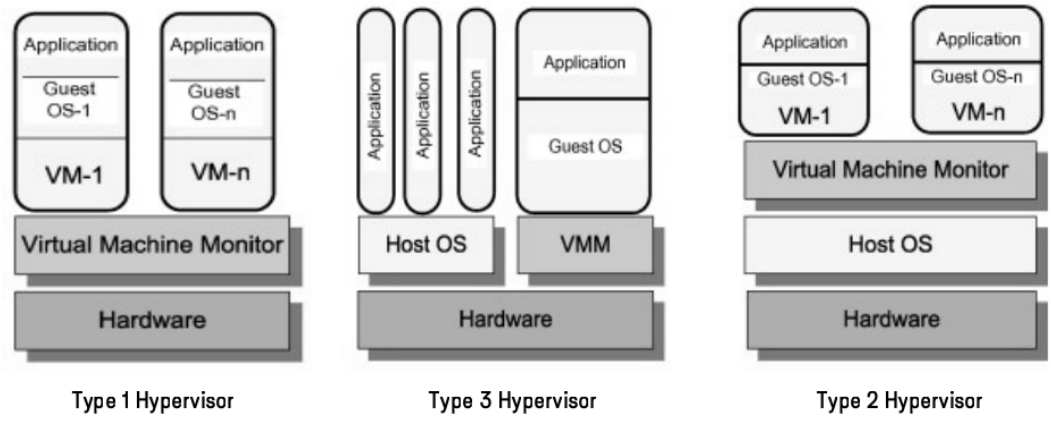


Figure 2: Three ways to virtually host an operating system

Compared to Type 1 hypervisor, Type 2 are less efficient, so businesses usually don't use them for production, however they are suitable to make test or development environment. Because of the way Type 1 (and Type 3) hypervisor are designed they are more efficient : VMs are close to the hardware which is better for performances. Thus this type is suitable for production environment.

Type 1 hypervisor also allows over-allocation of resources to support computing spikes. Type 3 can support it too, but not out-of-the-box. Over allocation means that you virtually allocate more than what you physically have: if your hardware has 64GB of RAM, and you create 10 8GB VM. As long as all the VMs do not use their 8GB at the same time you will not have issue.

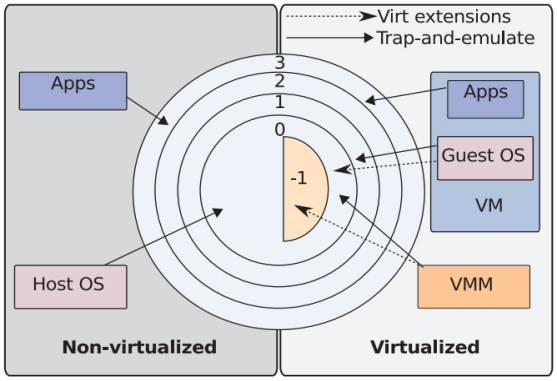
4. Protection Ring

Introduced by Multics time-sharing operating systems around 1973, the concept of protection ring was revolutionary. Also called hierarchical protection domains, they are a mechanism implemented and available on the CPU which aims to provide data protection, fault tolerance and security layers. The lower number is the most privileged (most trusted), the idea is that only the most privileged ring can access directly the hardware.

When a higher number rings wants to access hardware they need to communicate with the lower rings using call gates. Call gates are predefined instruction designed to improve security by preventing program from misusing or maliciously accessing hardware. On the well known Intel x86 CPU architecture, there are 4 available rings each ring as a "recommended" use : Ring 0 - Kernel, Ring 1 & 2 - Driver, Ring 3 - Applications.

It's worth mentioning that Having to through layers using Call Gates to access Kernel code causes delay. The level of rings provided by the CPU are a design and it's not mandatory to use all of them. For example Windows and Linux only use ring 0 for the kernel mode (OS and drivers), and ring 3 for user mode (Applications). This doesn't necessarily mean the OS is less secured as other alternative are used. Microsoft replaced ring 1 & 2 (for drivers) with it's own mechanism : signed drivers.

When considering Type 1 & 3 hypervisors, they used to reside on the ring 0, which meant that the Guest OS was in the ring 1. In order to work properly hypervisors needed to intercept the OS Calls and translate them, which add an impact on performances.



Intel and AMD, in order to improve performances, added negative rings to their architecture. The hypervisor can thus live in negative ring (-1) as close as possible to the hardware while not disturbing guest native operation, because the OS still lives at ring 0.

Figure 3: CPU protection rings designs

5. Virtualization Mechanism

Nowadays several virtualization methods exists, I will concentrate on server virtualization but resources (network, storage) and application virtualization also exist. Server virtualization can be Virtual Machines or Containers. As I said earlier, the focus here is on hardware virtualization in order to run multiple OS on the same real hardware. Thus I will only treat VMs, three virtualization mechanisms are available for them :

- Full Virtualization (FV) [Software-Based Virtualization] : KVM, VMware

Full virtualization completely abstract the physical hardware : the guest OS running in the VM is not aware that is it being virtualized, its kernel is not modified and thus it operates as it would on real hardware. The hypervisor does all the job to make this virtualization possible. With this method the guest OS is in ring 1 and the hypervisor in ring 0.

The hypervisor will catch on the fly the guest OS instruction and translate them in binary. User Apps however can run without any modification or translation from the hypervisor, this is called direct execution. This means that on the guest OS the user level is running at native speed and the kernel level has a bit of delay due to the translation, to help with that delay the hypervisor uses cache to be faster on common instruction from the guest.

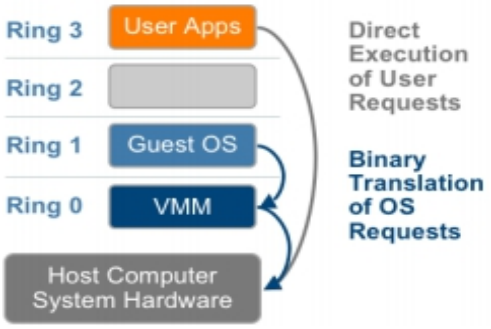


Figure 4: Full-virtualization approach & ring levels

- Para Virtualization (PV) [Software-Based Virtualization] : Xen

In para-virtualization the guest OS is aware that it is virtualized, its kernel is modified to send Hypercalls to the hypervisor for OS instruction that can't be virtualized. An interface is also provided by the hypervisor for other critical OS instruction such as

interrupts, memory access. The guest OS in ring 0, and the hypervisor is close but below.

Compared to full virtualization, para virtualization is more accessible, in the sense that modifying a guest OS is more doable and easy than having to build all the binary translation required by the full virtualization. However compatibility and portability of para-virtualization is often mediocre as not a lot of OS can support kernel modification. Para-virtualization can offer both incredible and poor performance depending on the case.

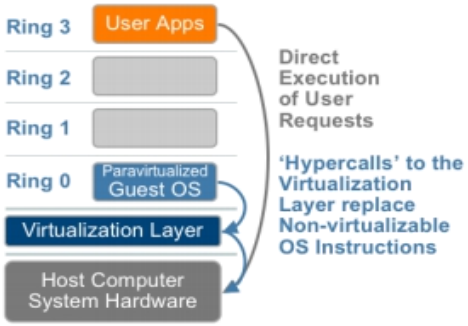


Figure 5: Para-virtualization approach & ring level

Some solution offer para-virtualized device on top of a full virtualization. This means minimal changes to the kernel OS are done, it just add driver for para-virtualized networking, storage. . . Portability in this case is high and performance for the the devices are as good as native because there is no emulation of the device.

- Hardware Assisted Virtualization (HVM) [Hardware-Based Virtualization] : Xen

Also known as native virtualization or accelerated virtualization, hardware-assisted virtualization use a different approach than Binary Translation nor Hypercalls. OS calls from privileged (positive) rings are automatically trapped to the hypervisor who then handle the instruction as usual.

This was made possible by the addition of more privileged (root or negative) rings in the newer CPU (Intel VT-x & AMD-V). This added a new set of CPU instruction dedicated to hypervisors. The main advantage is the reduction of overhead and thus lagging time. But this wasn't always the case the first-generation of hardware-assisted virtualization was not performing well enough to be considered viable.

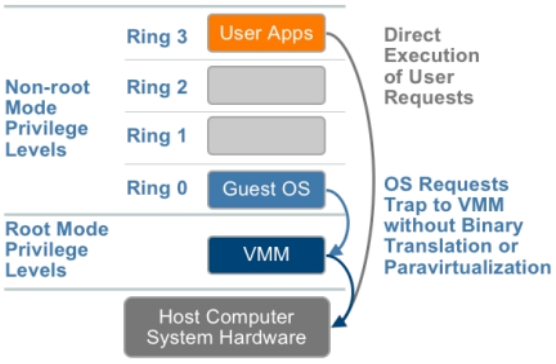


Figure 6: Hardware-assisted virtualization approach & ring level

Now with the newer generation of hardware-assisted virtualization performance are as good as para-virtualized with the simplicity of not tweaking the OS. Like with full-virtualization, it's possible to have para-virtualized device in hardware-assisted virtualization (PVHVM), thus enhancing even more the performance.

6. Memory Management

The OS whether it's running on bare-metal or virtualized see a contiguous (linear) memory space which is virtual, this virtual memory in not necessarily referring to the physical RAM. If your virtual memory exceed your RAM amount then part of your drives storage is used (swapping). Typically part of the CPU, memory management unit (MMU) uses page tables to map virtual and physical memory, segmentation is also used to speed up retrievals, the OS kernel is aware of those translations and tells the MMU what to do.

With VMs you allocate virtual memory to the guest machine too, this other layer of virtual memory is seen by the guest OS like physical RAM. The hypervisor virtualize the MMU for each guest. The hypervisor will use the real MMU to map the guest allocated memory to the real memory. To be more efficient and prevent two translation the hypervisor usually use shadow pages which map the guest OS virtual memory directly to the real memory (see red arrow).

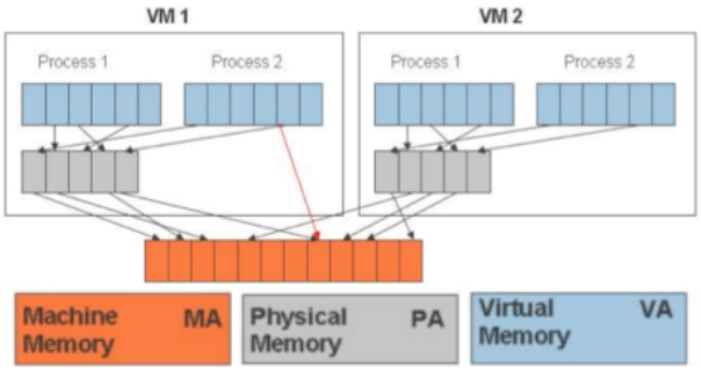


Figure 7: Memory virtualization

7. Device Management

The last point to evoke on virtual server is the device management, an OS doesn't work alone there is a hand full amount of devices that it needs: network interface card, hard drives, less common but it can also be printers, graphics card. . . When interacting with devices the OS receive and send information this is called Input/Output (IO). There is three way to manage device in a virtualized environment.

- **Emulation :**

The hypervisor will create and expose a virtual common virtual that doesn't need specific driver to be recognized. Performance are severally impacted because CPU cycles are lost just to emulate the behavior of a real component (e.g. E1000 NIC).

- **Para-Virtualized :**

Already discussed as an option in the virtualization mechanism section, if you seek better performance device para-virtualization is the way you should go. The hypervisor will

create and expose a device that doesn't really exist, an interface. To support it you need to install specific driver (e.g. VirtIO). This enables fast I/O between the guest and the hardware managed by the hypervisor. Because they are interfaces and the hypervisor still manage the real device, you can have several guest accessing the same real device.

- **Pass-through :**

If you seek native (bare-metal) performance PCI pass-through is the best bet. The device appears to the VM as if it was physically attached. This usually as one major drawback : The guest to which you pass-through is the only one that can now use the device because the hypervisor is not in the path anymore. But if you have a motherboard, hypervisor and device that support Self-Virtualized Hardware (SR-IOV) then it's possible to do pass-through to multiple VMs, because the device create virtual instance of itself.

8. Conclusion

Today it's estimated that at least 80% of small businesses are using virtualization. You can count on your hands all the solutions that are well known and commonly use. The timeline below show their first release date. All of them are still used today, but compared to when they first released the feature offering has evolved tremendously.

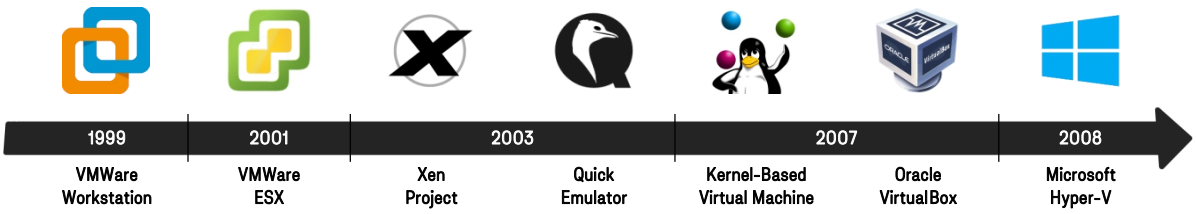


Figure 8: Common virtualization solutions first-release timeline

For years Full Virtualization was the easiest way to virtualize with correct performances even though some latency was there due to binary translation and device emulation. A complete Para-virtualization (CPU + Devices) was the way to try to achieve better performance but this added overhead and maintenance cost due to the kernel modification that where required.

Many businesses thus founded a sweet spot by levering Para-Virtualized device on a full virtualization. However the future is Hardware-Assisted Virtualization, easy to setup, without any overhead, they are performing very well on their own because no binary translation nor kernel modification are needed. When hardware assisted is combined with Para-Virtualized device (PVHVM) you get the pinnacle of what can be done today.

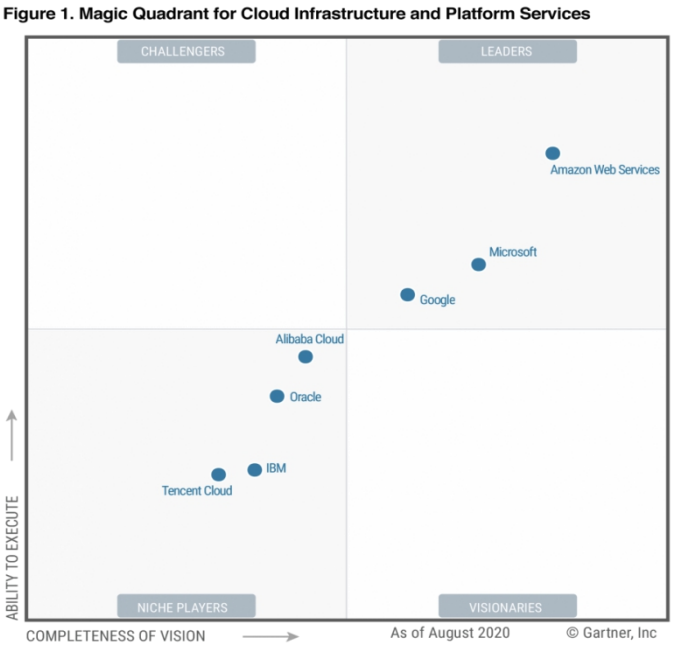
VII. Cloud Providers in Brief - AWS EC2

1. Overview on Providers

Cloud provider are companies offering other companies several IT services (virtual server, storage, database, etc). Cloud computing is based on the idea of pay-as-you-go, only for what you need and use. Plus Cloud provider offering are usually thought to be fault-tolerant, highly-available and easily-scalable to accommodate growth.

On paper this makes it is very attractive for businesses, and it is when we look at Cloud Provider turnover we usually see millions if not billion of dollars. However not everything has benefit to be the cloud, there is still a place and time for On-Premise use.

The most known Cloud Service Providers today where already known IT company such as Google, Microsoft, IBM, Oracle. But there is also exceptions with the E-commerce site Amazon & Alibaba, which also are Cloud Provider.



Amazon Web Services (AWS), for ten years has been the Leader in Gartner's Cloud Infrastructure and Platform Services Magic Quadrant. Thus I will only concentrate on AWS because they are the leader and therefore there offer should be representative of the other Providers offers. Plus it will help to understand the problematic of this whole document because it is related to an AWS offer.

Figure 9: Cloud Leaders - 2020

2. Zoom on Amazon Web Services

Amazon Web Services (AWS) started in 2004, with Simple Queue Service (SQS),

a scalable messaging model which didn't made the unanimity when released, but nowadays it is greatly appreciated. Amazon Web Services then launched Simple Storage Service (S3) in 2006, the service now host more than 82 Billion objects.

Also launched in 2006, was AWS virtual server offer is called EC2 at the beginning this was only available in the US Northern Virginia Region. EC2 quickly was adopted, in order to offer better performance (latency) to European consumers AWS opened it's first European Region in 2007 in Ireland.

It took a few years before new AWS European Region came to life. In 2014, Germany (Frankfurt) was opened, two years later in 2016 the London Region opened, followed by Paris in 2017... AWS is always trying to open new locations close to businesses, in the aim to provide lower latency but this can also help to meet country legal IT requirements.

Today, AWS has 25 regions (blue circles) worldwide and some are dedicated to government, which shows how trusted they are. They already project to open more (red).

Figure 10: AWS Cloud Data-Center Locations

With millions of customers and just shy of a 100 security standards and compliance certifications. AWS offers over 200 services built on top of the most secure, stable and scalable cloud platform available. The core services offered are compute, storage, content delivery network, databases and networking.

VIII. Problematic

1. Context

A United Kingdom based company who has been established for several years, has been putting its servers in the Cloud for a bit more than ten years. When they started they journey with Cloud, they choose Amazon Web Service (AWS) as their provider. At that time the closest location available to them with that provider was the Ireland Region (Created in 2007). Thus they used that region because no other better option was available.

In 2016, the AWS London Region was opened but the company didn't chose to move there for good reasons. Ireland it just next to the United Kingdom, such migration would imply quite the overhead just to gain a slightly lower latency, plus the Ireland region hourly pricing for most services is cheaper than the London one. Thus the logical decision was definitely to not move anything and concentrate on the main activity : producing software.

2. Objective

If you follow a bit the news, you probably have heard about Brexit, acted at the beginning of this year (2021), the United Kingdom is no longer a member of the European Union. With that comes a number of uncertainty on the legal aspect of IT systems.

When a company store and process data it is regulated by several laws. Those laws are applicable and coming from multiple levels. There are the "local" legislation such as country laws, county laws . . . That you must follow but on top of that the data you treat is not always local, it can come from the other end of the world and if that "remote" country have specific laws then you should follow them too.

This tend to make the picture quite complicated for multinational company with customer worldwide, if everything is done On-Premise the company itself is responsible for every legal aspect. When in the Cloud it's all about the shared responsibility model. The cloud provider will for example meet the "local" storage requirement, the business will meet the "remote" data laws by storing in the cloud only what is authorized.

3. Issue

Since the company is in the United Kingdom (UK), with local and worldwide customers, they are touched by all the uncertainties of leaving the European Union (EU). For example the UK was following General Data Protection Regulation (GPDR) before Brexit, now that they left the EU they kept almost word for word the current GPDR and called it the UK-GPDR.

For now those are the same rules which doesn't complicate things. However nothing is Frozen and the GPDR, or the UK-GPDR could change in the future. The company will thus need to make sure they still follow both data protection act if they treat "local" (UK) and "remote" (EU) customer data. This was just an example, there is many other legislation in place to follow depending on customers data provenance.

The company at the beginning wasn't keen on moving to London because no real benefits could be seen for something that was a bit more expensive. Now with Brexit things have changed, it's not about performance nor expenses but legislation. Since the company is based in the UK, it makes sense to host their cloud systems in the UK to make sure the provider accommodate, at least, the UK rules on the storage side. The company will just to have to ensure what they collect and treat is legal depending on whether it's "local" or "remote" data.

4. Stakes

Moving from the AWS Ireland Region to the AWS London Region, is pretty straight forward for most of the server (~95%) that are "recent" because their instance type are available in London. However for the remaining ones that are "historic" and Linux systems the migration is impossible in their current state because the instances type does not exist in the London Region. Opened in 2016 this Region don't have the previous generation instances type.

We thus need to find a way to migrate those old server to newer instance type, before we can move them to London. We could start newer instances from scratch and deploy the proper services again, but that would take sometime. Ideally we need an alternative solution, which should be decently simple but most importantly fast and reliable: it should not compromise the server stability.

IX. Solution Hypotheses Listing

1. Overview

The older instance type, also called previous-generation instances are referring to instances type like T1. The newer instance type also called next-generation instances are referring to instance type like T2 or higher.

The company "historic" servers are T1 instances hosted in Ireland, in order to move to London it's necessary to first move them to a newer instance type such as T2, because London only support next-generation instances.

In total I had 4 main ideas to try upgrading the instances to newer type before being able to migrate them, the last idea can be subdivide in many approach as its a manipulation of the file-system :

1. From Scratch - Build a new server directly using the newer types
2. Console Upgrade - Seek for an option to change the type prior to migration
3. Swapping Disk - Simply move the drive from the older type to the newer type
4. Conversion Methods - Based on disk swapping we modify the file-system

2. Details

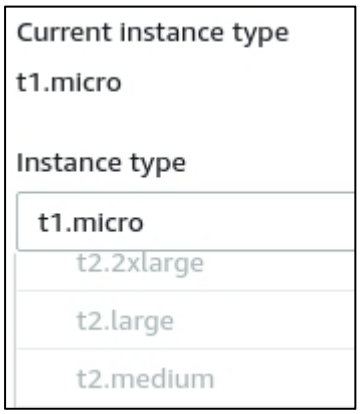
Without digging anywhere I already had some part of these ideas. Of course I looked at AWS documentation to see if anything was documented, unfortunately didn't found anything useful apart the fact that it could be possible. I finished by seeking on forums for people who also faced that problematic, and there definitely was.

- 1. From Scratch :

The first idea is an obvious solution that will work every time : it's not really a migration since you redeploy all the services, code and dependencies you need on a clean newer instance type. But this solution is the one we would like to avoid, it takes time because redeploying everything is doable but require quite the work just for one server. This is kept as the backup solution in case nothing better can be found.

- 2. Console Upgrade:

The second idea comes from reading the AWS documentation called : EC2 upgrade paths. They offer a table with AWS recommendation for upgrading older instances type to newer instances type. It's written that T1 & M1 older generation instances are suitable for a T2 upgrade, but they don't provide any recommendation on going higher such as T3.



They precise that to change the type you can do it right from the console : this is the basic procedure when you want to switch between generation/type that are directly compatible. I know that for our case it's impossible because otherwise we would have already managed to migrate the Ireland T1 to London as T2. A quick tour in the console can prove it, when changing the T1 instance type the newer instance type are not available (grayed-out).

Figure 11: AWS Console - Changing Instance Type (View)

- 3. Swapping Disk:

This one is very close to the hypothesis "2. Console Upgrade" where the operation was to try and ask the hypervisor to migrate the machine. When such migration is done by the hypervisor, the block devices (hard drives) are moved to another virtual server and a few things are kept such as the virtual server details (Instance ID…). This third hypothesis does almost the same but manually.

On a freshly launched T2 instance, we attach the T1 drive where the one was and try to boot. This idea came up a lot as a question on forums, it's a good starting point but it won't be enough, the instance won't start. In AWS documentation, it's written that the instances virtualization type are different, one of them use a modified boot-loader, so it's booting process is quite different hence why it won't start : it won't find the drive.

- 4. Conversion Methods

It's not for nothing that T1 are called previous-generation, and T2 next-generation. T1 instances are para-virtualized (PV) and T2 are hardware-accelerated virtualized (HVM). There is quite the differences between PV and HVM. PV needs some modification to be able to boot, while the other one (HVM) behaves as a bare-metal installation.

This explain why hypothesis 2 and 3 won't work and can be rule out. In this hypothesis we do more than just swapping the drive, we will copy some part of the file system from the new T2 instance drive to the old T1 drive. The copied parts can be the boot, kernel, app. . .

X. Chosen Hypothesis Thought Development

1. Multiple approaches

From the previous listing it's clear that the only plausible approach other than starting from scratch (hypothesis 1.), is to try and perform an instance conversion (hypothesis 4.). In the details of this last hypothesis I explained that different parts of the file system could be moved in order to try to boot the instance.

On forum and post about this several opinion appeared… With the two extreme where some says copying the boot files of the newer type to the older type is enough, and some other see the problem from the other where they copy only the software and data from the older to the newer instance.

So what is the best approach ? Everyone said their approach is working for them, so we could assume they should work for our use case. They could work but it's not sure, I could just try them all and take the first one working but that can be time consuming if unfortunately only the last works…

To make a decision on the best way to proceed, more research are needed, first of all it's necessary to understand how the two instance generation provide the boot environment, how does Linux boot and how is the Linux file-system is organized.

2. Instance Boot Overview

AWS uses Xen as it's hypervisor, thus the AWS HVM EC2 Instances and AWS PV EC2 Instances are actually Xen HVM and PV virtual machine. Knowing that we can look at Xen Wiki to understand a bit more the opaque differences that reside between an HVM and PV instance boot.

Xen works with domains: Dom0 and multiple DomU. Dom0 is a unique privileged domain which is responsible of the back-end firmwares (drivers) for the real hardware. DomU are unprivileged isolated domains, they are the guests launched and managed by Dom0. Inside each DomU there are front-end firmwares, which communicate with the back-end ones. Simply put, Dom0 is the hypervisor and DomU are the virtual machines.

Grub is the most used boot loader by Linux. The PV guest use a special version of this boot-loader that is modified to support the Para-Virtualized environment. This boot-loader needs to be specified to the Dom0, because it's the hypervisor that launch it. Every device is para-virtualized. Once running the boot loader will load the kernel from the guest (DomU) if it's found.

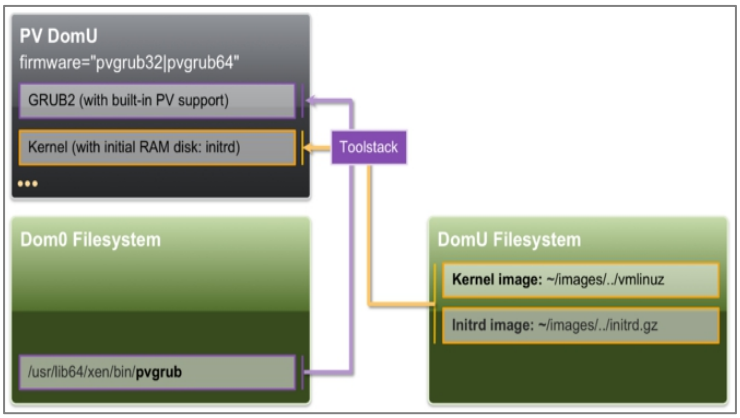


Figure 12: PV Boot Process

HVM guest, are closer to bare-metal installation, Dom0 has firmwares for the real hardware which will either be para-virtualized or emulated. Dom0 here doesn't manage the boot loader hence why it's closer to as bare-metal installation. Once the HVM DomU is up its virtual BIOS will look for a boot-loader in the guest. If found it will try to launch the kernel.

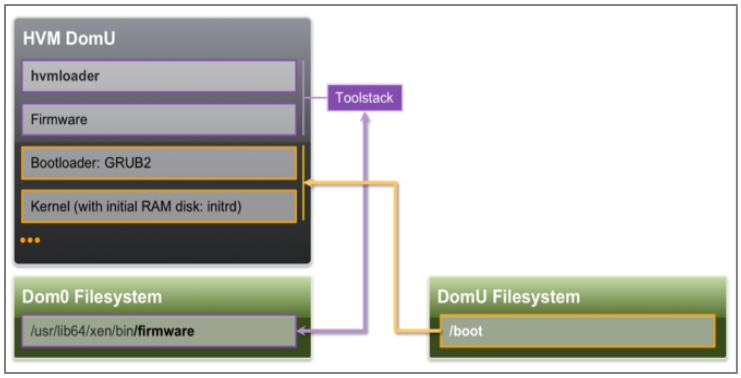
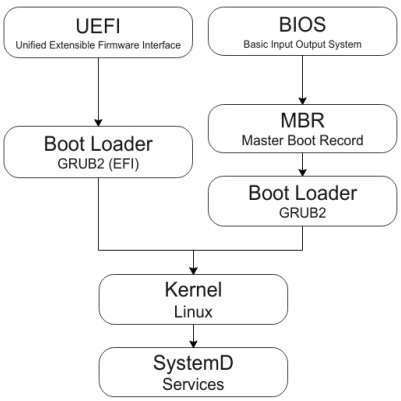


Figure 13: HVM boot process

3. Linux Boot Sequence

Every computer system will start it's boot process by some steps until it arrives to the common point of the boot-loader, a small software is used to target and load a kernel. On Linux kernels the drivers will be loaded before it goes on to the last phase.



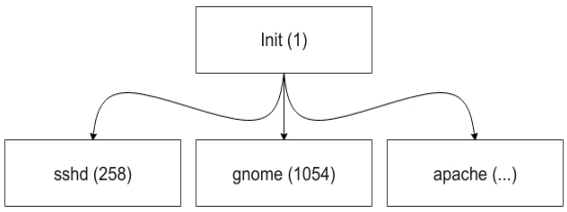
Once the main process are up the auxiliary process (services) are started. This diagram is very simplistic, at each step many small action are taken.

The kernel will be loaded from the vmlinuz. Initrd an initial RAM disk is also loaded. This initial RAM disk will install some part of the firmware and modules.

Figure 14: Linux Universal Boot Sequence

4. Linux OS (Kernel & Distribution)

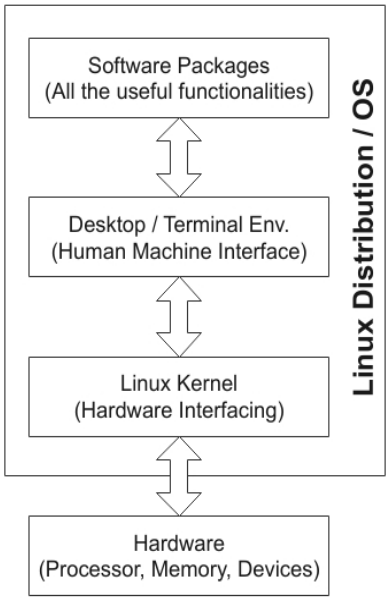
The kernel is the base of any Linux Operating System. When the Kernel is targeted by the boot loader, it has to self extract before it can start to mount the system. To do so has explained in the last section the Kernel will mount the initial ram disk ('initrd') which provides an initial root file system.



Once done the kernel then tries to launch the 'init' process, this process always has the first Process Identifier (PID = 1), it is the mother of any other process.

Figure 15: Linux Mother Process & Chain

If this process can't start or crashes a 'kernel panic' error is raised. The 'init' process worked but had some issues, notably delay because every service was launched serially. Thus a few replacements where proposed, the most adopted today is 'systemd'. Which for example support parallel service start launch.



The Operating System, or distribution, is just a set of elements composed of a Linux Kernel and services that makes up the user environment and its applications. The user environment is an interface available to make the system usable by us humans.

In the same manner the applications needs a way to interact with the hardware without complications. That's the Kernel job it does the heavy lifting of recognizing the hardware and providing interfaces for them using the drivers, it also does process and memory management.

Figure 16: Composition of a Linux Distribution

Release of a new kernel are very frequent, but usually each distribution doesn't use the latest but the most stable one for their needs. The newer kernel versions are developed in a way to maintain backwards compatibility using flag parameters on the existing system calls, if not possible new system calls are usually added (unless there is a good reason to change the existing one). This way older applications should still work with a newer kernel and can be update to take advantage of the new features.

5. Mount Points

On Linux having secondary drives or partitions is transparent. They are mounted at the place of a folder, so unless you are aware that the specific folder you open is mounted you won't see the difference. On Windows it's different, the main drive or partition is defined as "C:". The secondary one has another letter like "D:", both have child folders.

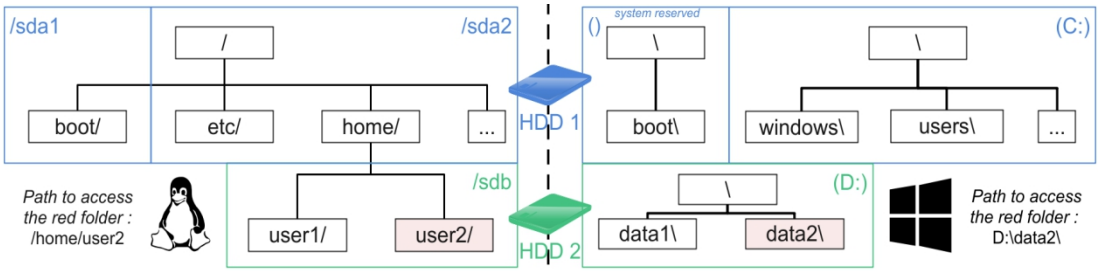


Figure 17: Mount point differences between Linux and Windows

Historically on the boot folder on Linux was separated and placed on another partition, because old BIOS didn't supported large boot partition, it also helped to boot faster. Doing so reduced the file system load, very useful on large disk that at the time could take some time to retrieve files such as the kernel. Modern system have lifted that issue but in many cases we continue to separate the boot folder in a partition.

Today uses of this separation can be for safety, multiple boot, or even for encryption, but the boot partition is kept unencrypted. For example on my laptop I have a dual boot with Windows, on the Linux side I encrypt the swap, home and backup partition. Notice here was my system knows the hard drive is an NVME SSD.

```
NAME          MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
nvme0n1       259:0    0 931.5G  0 disk
├─nvme0n1p1   259:8    0    16M  0 part
├─nvme0n1p2   259:9    0     3G  0 part  /boot/efi
├─nvme0n1p3   259:10   0  33.5G  0 part
│ └─cryptswap 253:0    0  33.5G  0 crypt [SWAP]
├─nvme0n1p4   259:11   0 279.4G  0 part  /
├─nvme0n1p5   259:12   0 457.3G  0 part  /mnt/lab
└─nvme0n1p6   259:13   0 158.3G  0 part
  └─chome     253:2    0 158.3G  0 crypt /home
```

Figure 18: Block Device List (lsblk) - My PC

AWS EC2 instances run on Xen Hypervisor, the Linux Instance only have one single root partition which includes the boot data. The reason is that the instances are on SSD, which can process big chunk of data quickly, in addition this simplifies the file system. Notice here that the system doesn't know it's running on SSD, it sees XVD* which correspond to Xen Disk Storage.

```
NAME  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0   7:0    0 33.3M  1 loop /snap/amazon-ssm-agent
loop1   7:1    0 28.1M  1 loop /snap/amazon-ssm-agent
loop2   7:2    0 99.2M  1 loop /snap/core/10958
loop3   7:3    0   97M  1 loop /snap/core/9665
loop4   7:4    0 55.5M  1 loop /snap/core18/1997
xvda1 202:1    0   15G  0 disk /
```

Figure 19: Block Device List (lsblk) - AWS EC2

So in our case not so important to know how Linux mount the drives and partition because everything important for the boot and system to work is inside the root partition.

How do I know that ? simply because the only partition I can see when listing block devices is "/" which means the root.

We can ignore the "/snap/" ones because they are added after the system as boot up they are used for third-party software from the snap store. Now that we know we won't have to deal with several partition which will definitely ease the work, We can look at the Linux file-system structure.

6. Linux File-System

Thankfully Linux Operating System only use files, which kind of make things simpler, but also more risky as system critical elements such as settings are easily available. Conversely, on Windows Operating System those settings are part of a database called registry, of course this is stored in files somewhere but that complicates the picture.

Composed of several folders, it as seen some evolution throughout the years. Nowadays the file-system will usually be composed of the folders from the table below.

Folder	Type	Class	Description
/bin	Hard Drive	Software	User Binaries
/boot	Hard Drive	System	Boot Loader Files & Kernel
/dev	*Ram-Based*	*System*	*Device Files*
/etc	Hard Drive	System	System Configuration Files
/home	Hard Drive	Data	Home Directory
/lib	Hard Drive	Sys/Temp	System Libraries
/media	Hard Drive	Data/Temp	Removable Device
/mnt	Hard Drive	Data/Temp	Mount Directory
/opt	Hard Drive	Software	Optional Apps
/proc	*Ram-Based*	*System*	*Process Information*
/run	*Ram-Based*	*System*	*Run-time Data*
/sbin	Hard Drive	Software	System Binaries
/srv	Hard Drive	Data	Service/Site Data
/sys	*Ram-Based*	*System*	*System Information*
/tmp	*Ram-Based*	*Temporary*	*Temporary Data*
/usr	Hard Drive	Software	"Read-Only" Application
/var	Hard Drive	Soft/Temp	Longer Temporary Data

Figure 20: Usual Linux File-System Structure Description

An entire document could be dedicated to only the Linux file-system and understanding in details what every folder roles is, however that's not the goal here. We just aim to convert an old instance to a newer instance while keeping the data. To do so we can proceed by elimination and if needed learn more about some folder purposes.

7. Chosen Approach

The temporary folder class can already be put aside, their content is populated during the boot process, or data is kept over reboot but only for a couple of days. Then a choice as to be done on the approach chosen either moving the system or data/software :

A - From the 17 folders of the file system, only 7 are related to the system itself (boot, kernel, firmware, distribution package). In these folders only 3 are stored on the hard drive (etc, boot, lib). The etc folder contains the configuration of our system and we want to keep that as-is. This makes only 2 folders (boot, lib) to deal with if we aboard the problem by moving the system folders from the new instance drive to the old one.

B - From the 17 folders of the file system, 9 folders total are related to software and data, these are all on the hard drive, thus they persist over reboots, which is logic you don't want to loose this data. It seems that approaching the problem from this side moving the data and software to the newer instance is going to be way more complicated notably because it's more heavy on the data aspect.

The migration solution needs to be as fast as possible, thus the approach chosen is :
A - moving the system to the newer instance. The manipulations should be pretty fast. We will override the boot entry, kernel from the old instance drive by the new instance one (moving kernel is possible due to their backwards compatibility design), this should allow the system to boot with another kernel which isn't modified for Para-Virtualization.

We will also override a part of the lib folder for the modules/firmware, because at my first try I encountered an issue with missing firmwares. I understood that some sub-folders from this library folder were stored on the hard drive. While some other sub-folders where ram-based, populated by Initrd during the boot process.

During my research I couldn't find which sub-folders where disk or ram-based. The reason I saw for that was that assuming you use vanilla kernels it will varies depending on the version and distribution used. The only reasonable solution was to try.

XI. Chosen Hypothesis Tests & Result

1. Recap

The hypothesis chosen is to try and convert the older Ireland Instance Type (T1) which are Para-Virtualized (PV), has a newer Instance Type in London which are Hardware-Assisted (HVM). AWS EC2 Instances are classified by type and size. The size is the allocated hardware capacity such as micro with 512MB of RAM, small with 1GB… While migrating we will keep the same size of instance because this need didn't change (migrating shouldn't consume more RAM).

The HVM instance type start at T2 and go higher. As previously mentioned AWS in its upgrade path documentation only recommends going with T2 when coming from T1. this is probably because the hardware is almost the same or at least very close between these type, so we don't have unrecognized hardware due to missing firmware. In the same manner when launching the T2 instance we will choose the same Linux Distribution and the closest release available to the one from the T1 Instance.

To do that conversion two main approach were possible, either moving the system files from the newer instance to the old instance drive and keep using the old drive in the new instance. Or the opposite keep the new instance drive but move only the software and data from the old instance. Looking at the file-system structure it makes more sense to go with the first approach were we move the system files (boot files, kernel…) as there is less unknown and less data to move, so it's faster.

2. Testing in the same context

At the end of the last part I said that I couldn't find enough information on the "lib" folder. It as several sub-folders for system libraries, firmwares … the problem is that some are disk based and other are RAM-based, which makes it hard to know which sub-folder are filled during the boot process and which stays and should be copied.

The only solution is thus to prepare a test procedure, and to make sure that the test can be run in the same context, with the same variable. In our case it's pretty simple, because insuring that the context is the same can be done through a backup: a snapshot

or AMI (Amazon Machine Image) since we are using virtual machine in the cloud. Doing so will allow me to deploy a new hard disk (snapshot) or complete instance (AMI) from an older point in time which will always be the same.

Anyway this pretest backup of the system would have been mandatory as a good practice. Indeed if I wasn't dedicated a server for these test, but I was using a real production or development server you want to secure the operational state of the machine before you apply any changes. This way if things go south then you still can roll back to the operational state without major downtime.

3. The procedure

The procedure will be composed of 8 major parts including the backup part which is done only one time for all iteration of the procedure. Below is the short version :

0. **Backup :** Take a Backup of T1 Instance
1. **Restore :** Deploy a New T1 from Backup
2. **Analyze :** Get the T1 information (hardware, version, journal…)
3. **Preparation :** Get the closet T2 to T1, control T1 is up-to-date
4. **Conversion :** Mount T1 drive to T2 and move the system folder
5. **Finalization :** Unmount T2 drive and place T1 drive as the root of T2
6. **Checks :** Control the state of the journal and your services
7. **Clean-Up :** Get rid of no longer used element (T2 drive, T1 instance)

The procedure aims to limit the number of shutdown and start-up of an instance because shutting-down and booting-up again is billed as a full hour every time, however rebooting doesn't count as starting a new hour. Basically systems will only be shutdown when their root drive needs to detach and moved such as in step 4. and 5.

4. Procedure Iterations (Trials)

In this part I details in each part of the procedure, in total I needed two try before I manage to perform a "decent" conversion. By that, I mean it's possible that I missed something. . . However with all the indicators I researched and used to me it seems that the conversion is a success : the system boots and no errors coming from the conversion are seen in the journal. Most of the system checks screenshot are available in Appendix A. The complete set of commands/actions used in the procedure is available in Appendix B.

0. Backup :

For my test, I didn't used any production or development server but a dedicated test instance that I built from scratch. I choose to use Ubuntu and emulate a web server with Apache2 because this was fast to setup and it provides a default test website.

This will be enough to see if the system boot up and the services are started properly and healthy. It's important to keep the network settings has DHCP, because each instance get it's IP from it, in the cloud networking is a key component including for management.

Once the system was operational and I could access the website from the internet. I stopped it and took a snapshot.

Name	Snapshot ID	Size	Status
T1 APACHE	snap-0fdcf277b8adc5e49	15 GiB	completed

Figure 21: Original System Backup

1. Restore :

For the first try I didn't need to proceed with this step I just started again the T1 instance. However for the second try, the T1 drive wasn't "pure" anymore due to the conversion trial, thus I delete it. I then created a new T1 drive from the snapshot, attached it back to correct place on the T1 instance and checked that the system was working as expected.

2. Analyze :

Using command inside the T1 instance I checked the journal for already existing errors. I also retrieved the Linux Distribution release, Kernel version, Instance Type.Size (I can get this from AWS too) and finally the hardware component (CPU, RAM, Network, Storage).

CPU and RAM aren't that much important because it's manage by the hypervisor, I just took them as references. However Network and Storage are good indicators for choosing the newer instance in the next part, that we know we already have driver for.

Linux Distribution Release	Ubuntu 16.04.06 LTS (Xenial)
Linux Kernel Version	4.4.0-1128-AWS
Status Of Apache2 Service	Active (Website Accessible)
Hardware Information Network	Xen Virtual Ethernet Card - Driver xen_netfront
Hardware Information Storage	Xen Virtual Storage - Driver xen_blkfront
Hardware Information CPU/RAM	Intel Xeon E5-2670 V2 @ 2.50GHZ | 512 MB
Instance Type & Size	T1.Micro
Grub Package Installed	PC, PC-bin, Common, 2-Common, GFX, Legacy-EC2
Journal Errors	3 (2 from Kernel Firmware/Device 1 from ACPID) *

** Those errors where already there when I launched the instance from an AMI provided by AWS, thus I'm not worried about them **

Figure 22: Old System Characteristic

3. Prepare :

Updating the software package inside T1 was successful nothing broke, the website was still available. It doesn't matter much, the distribution was in version 16.04.06 before the update, after the update it's minor went up, it is now in 16.04.07.

Searching for the closest Instance and Distribution to the T1 System wasn't complicated. I applied what I said in the recap and spun up an AWS AMI for Ubuntu Bionic 18.04 LTS on a T2.Micro Instance. I connected inside the new T2 instance to retrieve the same kind of information as in step two, to confirm system were close.

Linux Distribution Release	Ubuntu 18.04.05 LTS (Bionic)
Linux Kernel Version	5.4.0-1038-AWS
Status Of Apache2 Service	Not Found (Not Installed)
Hardware Information Network	Xen Virtual Ethernet Card - Driver xen_netfront
Hardware Information Storage	Xen Virtual Storage - Driver xen_blkfront
Hardware Information CPU/RAM	Intel Xeon E5-2676 V3 @ 2.40GHZ | 960 MB
Instance Type & Size	T2.Micro
Grub Package Installed	PC, PC-bin, Common, 2-Common, GFX, ~~Legacy-EC2~~
Journal Errors	2 (2 from Kernel cannot get HVM parameter) *

** Again I'm not worried about those error, the instance comes like that from AWS and thousand of people use it daily **

Figure 23: New System Characteristic

The only difference on the hardware are the CPU and RAM but this is managed by the hypervisor, we already have firmwares for Storage and Network because they are identical so that's great. Grub is installed too with the same package except the legacy EC2 which is normal, this grub package was used to manage the files that PV-GRUB uses.

Out of curiosity I also did the same checks inside a T3.Micro instance just to get an idea of why AWS was recommended only T2 for T1 upgrade, first things I could see was the network hardware wasn't the Xen Virtual Ethernet anymore but Elastic Network Adapter (ENA), which we don't have currently have the firmware for. The purpose here is just to move to London and not get the latest generation possible, thus I won't dig further on T3.

4. Conversion :

For this step the baseline is to stop and detach the drive from the T1 Instance. Then mount the T1 drive to the running T2 Instance. Once mounted we override the boot folder from T2 drive to the T1 drive, the result of this operation is the copy of the BootLoader, Kernel, Initrd files.

Then we trick the system into thinking that the T1 drive is it's actual root drive by re-mapping the device, system and processes folder, this allow us to re-install the Boot-Loader and automatically set it's entries accordingly to the new Kernel available on the T1 drive. Everything written above is correct and make the system boot on the first try. However I could see a missing firmware error in the journal.

As far as I know a few methods can be used to solve that error. The easiest one if we know we don't need this firmware because no hardware need it is to deactivate it. But this won't always be a valid solution sometimes the firmware can is needed, thus you can try to re-install the firmware library. Make sure it work with you specific kernel headers (version). If the firmware is stored in a 'lib' sub-folder that is RAM-based, thus populated at boot it's even more complicated as you need to update the initial RAM disk.

The last method I found, which I choose to test in the second try, is to copy from the "lib" folder the sub-folders which are associated to your missing firmware. In my case it was the "kernel-version/kernel/drivers" this sub-folder didn't seem to be RAM-based hence why I had issue with missing files. Since the system is already setup to use the file of that firmware it's easier to copy the sub-folder when we do the conversion rather than looking to install it again like I was explaining in the last paragraph. This indeed solve the issue.

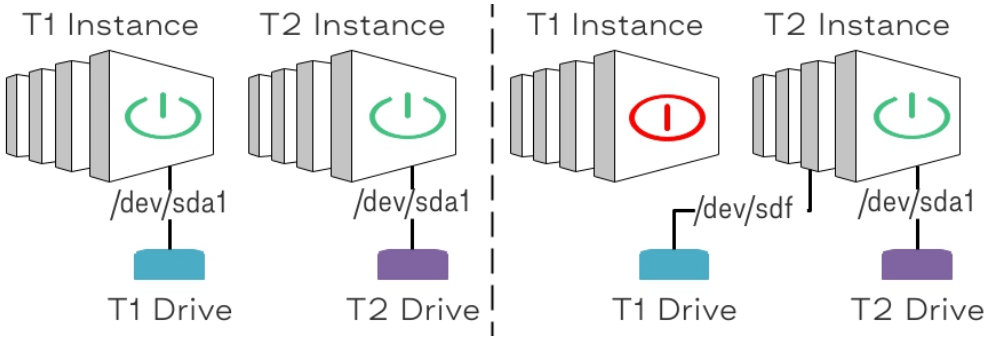


Figure 24: Instances Hard Drive Evolution Through Time (Part 1)

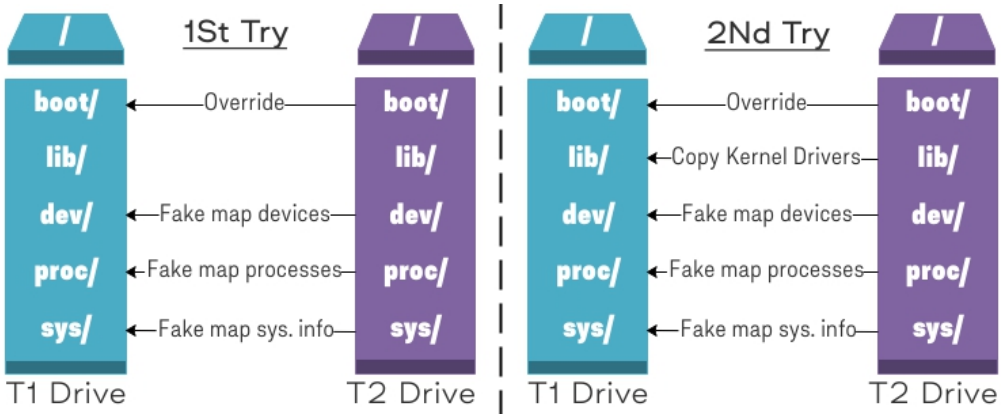
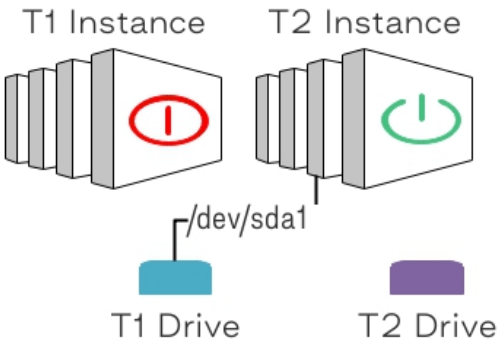


Figure 25: Manipulation Done On The File-System In Each Trial

5. Finalization :



In this step nothing complicated, our T1 drive should now be ready do work on the T2 instance. We shutdown the T2 instance detach both drives, and reattach only the T1 drive at the root location (/dev/sda1). Then re-start the T2 instance and it should boot up.

Figure 26: Instances Hard Drive Evolution Through Time (Part 2)

6. Checks :

During both tries the system felt healthy, no crashes, no issue to remote manage the instance, the AWS automatic check (hypervisor checks) all passed. In addition the Apache2 service was running perfectly without any intervention and the website accessible.

However like I explained in the previous step, it's important to check the journal because even though the whole system seemed happy, under the hood the system small non disruptive problem were present. Below are the errors output of the journal for both trials and the original outputs of both instance T1 and T2 before the conversion.

Original

Instance - T1	kernel: dmi: Firmware registration failed. kernel: mce: Unable to init device /dev/mcelog (rc: -5) iscsid[1092]: iSCSI daemon with pid=1093 started! acpid[1127]: cannot open input layer
Instance - T2	kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22! kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22!

Trials/Test

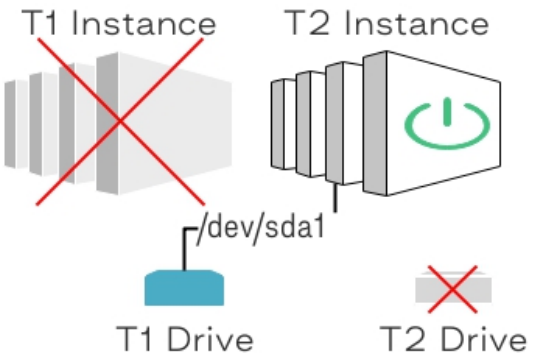
Test 1 - New Error	kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22! kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22! systemd-modules-load[408]: Failed to find module 'ib_iser' systemd[1]: Failed to start Load Kernel Modules.
Test 2 - No Error	kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22! kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22! iscsid[1120]: iSCSI daemon with pid=1121 started!

Figure 27: Instance Journal Error Output Comparison Between Tests & Original

Now if we concentrate on the output of the trials we see that during Test 1 a new error appeared about a missing firmware, this is an indicator that the conversion wasn't complete. In the second try we see that this issue is resolved. We get the same error for some 'HVM' parameter as we used to with T2, that's normal since we use the same kernel that we copied. The last line was also there in the T1 output and is wrongly classified, it's not an error but a information that the iscsi service is started.

When I originally started instances the instances I checked their journal and errors where already there, since they are AWS provided AMI. An AMI is a system to launch an instance in the cloud, the OS is already pre-installed with the correct drivers… Thus I'm not worried about these errors, thousand of people use them daily without reporting any issue related to these errors, if AWS made them available like that they shouldn't be an issue.

7. Clean-Up :



In the clean up part nothing complicated either, we don't need the T2 drive anymore, and it doesn't contain anything important so we can delete it. The T1 instance won't be used anymore as its drive is now mounted as the root drive of T2, an instance without a drive is useless we can also delete it.

Figure 28: Post-Procedure Clean-Up

To finish there is still the backup that was taken at the beginning, this can either be a snapshot or AMI, the price tag associated to this is not huge compared to a running instance. In a real production environment I would kept this backup for a few weeks just to be safe, the security of being able to roll back overweight the small price tag of the backup.

5. Results

The conversion by moving the system files from the newer instance hard drive to the older instance hard drive seemed successful on the first try, where I didn't touched the library folder at all, however some missing firmware error occurred, in my case I could have just deactivated the loading of those firmware as I didn't them.

However it's better to have a more universal solution. Thus I ran a second iteration of the procedure and this time I touched the library folder a bit in order to copy the missing firmware. With that the system is operational, stable and when we compare the journal before and after the conversion no new/unknown error are present.

Reminder: most of the system checks screenshot are available in Appendix A.
The complete set of commands/action used in the procedure is available in Appendix B

XII. Conclusion

Migration of perfectly healthy server as always been a serious task, even with virtual servers. You need to ensure that the virtualization type matches, that the hardware is compatible or do prepare the system to work with newer drivers… When you start to do such modification you can jeopardize the healthiness of the machine and that's the major fear businesses have while migrating live server. Cloud vendor understood that and they try to only broaden their offer and not replace everything by the latest tech.

The providers data-center locations are called Regions, provider for try to create new regions all the time to get closer to their customer for performances and legal reasons. Unfortunately when they open new Regions it's not possible do buy, install and provide customer with the exact same offering than another region, sometimes the hardware or technology are so old or not powerful enough nowadays that it would not make sense to add them because very few customer would use them.

The statement above actually caused quite the headache when a company tried to migrate between two region. A small amount of the server from the original Region had an older virtualization mechanism (para-virtualization) which wasn't available in the newer Region thus a migration plan was needed. After listing all the hypothesis and narrowing down do the most plausible one, research were done to understand if this could work or not and how to proceed, from this a migration plan was written and tested.

The outcome of the tests that were done proves that even if a Cloud Provider doesn't provide a way to migrate their instances from a para-virtualization mechanism to a hardware-assisted mechanism you can still do it. Most likely the reason the provider doesn't a way to migrate is because the manipulation we did are intrusive to the system and if you refer to the could shared model for IaaS system manipulation is the customer responsibility, while the provider is responsible for the infrastructure available.

Indeed the process pretty much comes to swapping the system files (kernel, bootloader, drivers) from the newer generation onto the older one to make the system work again while still being healthy. At least this is true while migrating from T1 to T2 instance in AWS, with a Ubuntu distribution. Finally since the Linux system all have the same architecture we can suppose this could for any other distribution with some twist. However this wasn't tested with Windows and most likely won't work with the plan we did due to the windows Registry, if it's possible at all.

XIII. References

Virtualization History - Oracle Documentation
https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html

Time-Sharing - IBM Documentation
https://www.ibm.com/docs/en/zos/2.4.0?topic=operations-controlling-time-sharing

Time-Sharing Principle - Community Wiki
https://en.wikipedia.org/wiki/Time-sharing

Compatible Time-Sharing System - Article by IEEE Computer Society
https://history.computer.org/pubs/2011-06-ctss.pdf

Energy Efficient Server (Chapter 5 - BIOS) - Book by Corey Gough, Ian Steiner, [...]
https://link.springer.com/chapter/10.1007/978-1-4302-6638-9_5

Benefits of virtualization - IBM Blog by Jordan Shamir
https://www.ibm.com/cloud/blog/5-benefits-of-virtualization

Hypervisor Differences - Blog by Nilabh Verma
https://www.devopsage.com/difference-between-type-1-and-type-2-hypervisor/

Hybrid Hypervisor - Article by IEEE
https://ieeexplore.ieee.org/document/5703225

Hyper-V Type 1 Hypervisor ? - Forum Thread
https://superuser.com/questions/836116/hyper-v-appears-to-runs-on-top-of-the-host-os-so-why-is-it-considered-a-native

Virtualization Security - State Of The Art by Fatma Bazargan, Chan Yeob Yeun [...]
https://www.researchgate.net/publication/268291860_State-of-the-Art_of_Virtualization_its_Security_Threats_and_Deployment_Models

Virtualization Mechanisms - White Paper by VMWare
https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

OS and Virtualization - Computer Science Course by Washington University
https://courses.cs.washington.edu/courses/csep551/14au/video/archive/html5/video.html?id=csep551_14au_5

Moving to the Cloud (Chapter 9 - Rings) - Book by Dinkar Sitaram, Geetha Manjunath
https://books.google.com.np/books?id=Nq4J0_tK0lsC&printsec=frontcover#v=snippet&q=all%20three%20types&f=false

Hardware and Software Based Virtualization Differences - Doc by Oracle
https://docs.oracle.com/en/virtualization/virtualbox/6.0/admin/hwvirt.html

Call Gates' Ring Transitioning in IA-32 Mode - Blog by Mohammad Sina Karvandi
https://rayanfam.com/topics/call-gates-ring-transitioning-in-ia-32-mode/

Memory Protection Architecture - Forum Thread
https://security.stackexchange.com/questions/127124/ring-1-and-ring-2-memory-protection-architecture

Protection Ring - Community Wiki
https://en.wikipedia.org/wiki/Protection_ring

Negative Rings Architecture - Blog by "RealWorldCyberSecurity" User
https://medium.com/swlh/negative-rings-in-intel-architecture-the-security-threats-youve-probably-never-heard-of-d725a4b6f831

Hypervisor IO & Devices management - Blog by Abel Gordon
https://www.stratoscale.com/blog/compute/hypervisor-comparison-of-io-virtualization-models/

SR-IOV Device - Blog by Scott Lowe
https://blog.scottlowe.org/2009/12/02/what-is-sr-iov/#:~:text=So%20what%20is%20SR%2DIOV,specification%20will%20help%20promote%20interoperability.

Cloud Computing History - IBM Blog by Maximilliano Destefani Neto
https://www.ibm.com/blogs/cloud-computing/2014/03/18/a-brief-history-of-cloud-computing-3/

Cloud Computing History - Blog by Jacob Ben-David
https://blog.turbonomic.com/cloud-computing-two-decades-in-review

Grub Boot Loader - Community Wiki
https://wiki.archlinux.org/index.php/GRUB

Boot Partition - Ubuntu Documnetation
https://help.ubuntu.com/community/BootPartition

Necessity Of The Boot Partition (1) - Forum Thread
https://superuser.com/questions/522971/is-a-boot-partition-always-necessary

Necessity Of The Boot Partition (2) - Forum Thread
https://unix.stackexchange.com/questions/256/is-it-good-to-make-a-separate-partition-for-boot

Details of Grub-Legacy-Ec2 Package - Ubuntu Database
https://launchpad.net/ubuntu/bionic/+package/grub-legacy-ec2

Linux Common Boot Process (1) - Blog by Joice Joseph
https://geekstuffweb.wordpress.com/2014/08/06/linux-boot-process/

Linux Common Boot process (2) - Blog by James Kiarie
https://www.tecmint.com/linux-boot-process/

CentOS / RHEL Boot process (1) - Blog by "admin" User
https://www.thegeekdiary.com/centos-rhel-7-booting-process/

CentOS / RHEL Boot process (2) - Blog by "Deepika" User
https://www.thegeeksearch.com/understanding-centos-rhel-8-boot-process/

SystemD replace Init - Blog by "Editor" User
https://www.tecmint.com/systemd-replaces-init-in-linux/

Linux Initial Ram Disk (1) - Community Wiki
https://en.wikipedia.org/wiki/Initial_ramdisk

Linux Initial Ram Disk (2) - Website by Jeremy Huntwork, Matthew Burgess
https://www.linuxfromscratch.org/blfs/view/svn/postlfs/initramfs.html

Linux Interchangeable Kernel - Forum Thread
https://unix.stackexchange.com/questions/392512/updating-the-linux-kernel-while-leaving-rest-of-system-as-is/392523

Linux Kernel - Doc by Redhat
https://unix.stackexchange.com/questions/235335/why-is-there-a-linux-kernel-policy-to-never-break-user-space

AWS Early Beginning - Blog by Jeff Barr
https://aws.amazon.com/blogs/aws/aws-blog-the-first-five-years/

AWS Cloud Leader (1) - AWS Website
https://aws.amazon.com/about-aws/global-infrastructure/#:~:text=AWS%20has%2080%20Availability%20Zones,Indonesia%2C%20Spain%2C%20and%20Switzerland.

AWS Cloud Leader (2) - Blog by Danilo Poccia
https://aws.amazon.com/blogs/aws/aws-named-as-a-cloud-leader-for-the-10th-consecutive-year-in-gartners-infrastructure-platform-services-magic-quadrant/

AWS Instances Timeline - Based on Jeff Bar Information
https://instancetyp.es/

AWS Instance Recommended Upgrade Path - AWS Website
https://aws.amazon.com/ec2/previous-generation/#Upgrade_Paths

AWS Supported Virtualization Mechanism - AWS Documentation
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/virtualization_types.html

AWS PV-Grub - AWS Documentation
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/UserProvidedKernels.html

Xen Domains Principle - Community Wiki
https://wiki.xenproject.org/wiki/Dom0

Xen Guest Boot Principle - Community Wiki
https://wiki.xenproject.org/wiki/Booting_Overview

Linux File-System (1) - Blog by David Both
https://opensource.com/life/16/10/introduction-linux-filesystems

Linux File-System (2) - Blog by Paul Brown
https://www.linux.com/training-tutorials/linux-filesystem-explained/

Debian Firmware Modules - Community Wiki
https://wiki.debian.org/ModulesAll#Module_Documentation

Discussion on the conversion from T1 to T2 (1) - Forum Thread
https://forums.aws.amazon.com/thread.jspa?threadID=155526

Discussion on the conversion from T1 to T2 (2) - Forum Thread
https://stackoverflow.com/questions/26676933/migrate-from-t1-micro-to-t2-micro-amazon-aws

PV to HVM Conversion Steps (1) - Blog by “nixhat” User
http://www.nixhat.com/2014/01/migrate-aws-pv-to-hvm-instance/

PV to HVM Conversion Steps (2) - Blog by Casey Labs
http://www.caseylabs.com/convert-a-centos-pv-paravirtual-instance-to-hvm/

PV to HVM Conversion Steps (3) - Blog by Mark Allman
https://allmanpc.medium.com/converting-an-aws-linux-pv-server-to-a-hvm-server-60def4888ef2

T1 to T2 Conversion Steps - Blog by Javier Sianes
http://jsianes.blogspot.com/2014/07/aws-convert-t1-instances-to-t2.html

Discussion on the conversion from PV AMI to HVM AMI - Forum Thread
https://serverfault.com/questions/439976/create-an-aws-hvm-linux-ami-from-an-existing-paravirtual-linux-ami

XIV. Figure Sources

This section show the sources of the figures I used and did not made myself.

Figure 1: Figure 5.1 - Hierarchy of software components used in power management
Chapter 5 - BIOS and Management Firmwares
Energy Efficient Servers by Corey Gough, Ian Steiner, Winston Saunders, 2015

Figure 2: Figure 5.3 - Isolated Systems
Chapter 5.4 - Virtual Machines
Cloud Computing by Dan C. Marinescu, 2013

Figure 3: Figure 1 - CPU protections rings levels
Chapter 3 - Execution of unmodified guest OSs
A summary of virtualization techniques by Fernando Rodríguez-Haro, Felix Freitag, [...] , 2012

Figure 4: Figure 5 - The binary translation approach to x86 virtualization
Technique 1 - Full Virtualization using Binary Translation
Understanding Full Virtualization, Paravirtualization, and Hardware Assist by VMWare White Paper

Figure 5: Figure 6 - The Paravirtualization approach to x86 Virtualization
Technique 2 - OS Assisted Virtualization or Para-virtualization
Understanding Full Virtualization, Paravirtualization, and Hardware Assist by VMWare White Paper

Figure 6: Figure 7 – The hardware assist approach to x86 virtualization
Technique3 - Hardware Assisted Virtualization
Understanding Full Virtualization, Paravirtualization, and Hardware Assist by VMWare White Paper

Figure 7: Figure 8 – Memory Virtualization
Memory Virtualization
Understanding Full Virtualization, Paravirtualization, and Hardware Assist by VMWare White Paper

Figure 12: (No figure name)
3 - PV Guest Boot Process
Booting Overview by Xen Wiki

Figure 13: (No figure name)
2 - HVM Guest Boot Process
Booting Overview by Xen Wiki

Migration of historic systems hosted in the cloud

by

Adrien FIAND

APPENDIX A

-

System Checks Screenshots

September 11, 2021

T1 Instance Original Checks

Distribution Release & Kernel Version

```
ubuntu@ip-172-31-20-52:~$ cat /etc/os-release | grep VERSION=
VERSION="16.04.6 LTS (Xenial Xerus)"
ubuntu@ip-172-31-20-52:~$ uname -r
4.4.0-1128-aws
ubuntu@ip-172-31-20-52:~$ ls /boot | grep vm
vmlinuz-4.4.0-1110-aws
vmlinuz-4.4.0-1128-aws
```

Dummy Service (Web Server / Apache2)

```
ubuntu@ip-172-31-20-52:~$ systemctl status apache2.service
● apache2.service - LSB: Apache2 web server
   Loaded: loaded (/etc/init.d/apache2; bad; vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Thu 2021-04-29 10:55:39 UTC; 4h 24min ago
```

Hardware Information (Storage, CPU, RAM, Network)

```
ubuntu@ip-172-31-20-52:~$ hwinfo --storage
08: None 00.0: 0180 Storage controller
  [Created at pci.1402]
  Unique ID: Fnxm.j9MZgu1IFZ6
  SysFS ID: /devices/vbd-2049
  SysFS BusID: vbd-2049
  Hardware Class: storage
  Model: "Xen Virtual Storage 0"
  Vendor: int 0x6011 "Xen"
  Device: int 0x1003 "Virtual Storage 0"
  Driver: "vbd"
  Driver Modules: "xen_blkfront"
```

```
ubuntu@ip-172-31-20-52:~$ hwinfo --cpu
01: None 00.0: 10103 CPU
  [Created at cpu.460]
  Unique ID: rdCR.j8NaKXDZtZ6
  Hardware Class: cpu
  Arch: X86-64
  Vendor: "GenuineIntel"
  Model: 6.62.4 "Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz"
```

```
ubuntu@ip-172-31-20-52:~$ hwinfo --memory
01: None 00.0: 10102 Main Memory
  [Created at memory.74]
  Unique ID: rdCR.CxwsZFjVASF
  Hardware Class: memory
  Model: "Main Memory"
  Memory Range: 0x00000000-0x2126afff (rw)
  Memory Size: 512 MB
```

```
ubuntu@ip-172-31-20-52:~$ hwinfo --netcard
06: None 00.0: 0200 Ethernet controller
  [Created at pci.1402]
  Unique ID: Zz0U.USuSioBKp55
  SysFS ID: /devices/vif-0
  SysFS BusID: vif-0
  Hardware Class: network
  Model: "Xen Virtual Ethernet Card 0"
  Vendor: int 0x6011 "Xen"
  Device: int 0x0003 "Virtual Ethernet Card 0"
  Driver: "vif"
  Driver Modules: "xen_netfront"
```

Instance Type.Size Through Meta-Data

```
ubuntu@ip-172-31-20-52:~$ curl 169.254.169.254/latest/meta-data/instance-type && echo " "
t1.micro
```

Grub Version & Installed Package

```
ubuntu@ip-172-31-20-52:~$ grub-install --version
grub-install (GRUB) 2.02~beta2-36ubuntu3.29
```

```
ubuntu@ip-172-31-20-52:~$ dpkg -l grub-pc grub-pc-bin grub-legacy-ec2 grub-gfxpayload-lists grub2-common grub-common
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                          Version              Architecture         Description
+++-=============================-====================-====================-=============================================================
ii  grub-common                   2.02~beta2-36ubuntu3.2 amd64              GRand Unified Bootloader (common files)
ii  grub-gfxpayload-lists         0.7                  amd64                GRUB gfxpayload blacklist
ii  grub-legacy-ec2               21.1-19-gbad84ad4-0ubu all                Handles update-grub for ec2 instances
ii  grub-pc                       2.02~beta2-36ubuntu3.2 amd64              GRand Unified Bootloader, version 2 (PC/BIOS version)
ii  grub-pc-bin                   2.02~beta2-36ubuntu3.2 amd64              GRand Unified Bootloader, version 2 (PC/BIOS binaries)
ii  grub2-common                  2.02~beta2-36ubuntu3.2 amd64              GRand Unified Bootloader (common files for version 2)
```

Journal Errors Output

```
ubuntu@ip-172-31-20-52:~$ journalctl -p 3 -xb
-- Logs begin at Thu 2021-04-29 16:36:13 UTC, end at Thu 2021-04-29 16:51:11 UTC. --
Apr 29 16:36:13 ip-172-31-20-52 kernel: dmi: Firmware registration failed.
Apr 29 16:36:13 ip-172-31-20-52 kernel: mce: Unable to init device /dev/mcelog (rc: -5)
Apr 29 16:36:23 ip-172-31-20-52 iscsid[1092]: iSCSI daemon with pid=1093 started!
Apr 29 16:36:23 ip-172-31-20-52 acpid[1127]: cannot open input layer
```

T2 Instance Original Checks

Distribution Release & Kernel Version

```
ubuntu@ip-172-31-23-166:~$ cat /etc/os-release | grep VERSION=
VERSION="18.04.5 LTS (Bionic Beaver)"
ubuntu@ip-172-31-23-166:~$ uname -r
5.4.0-1038-aws
ubuntu@ip-172-31-23-166:~$ ls /boot/ | grep vm
vmlinuz-5.4.0-1038-aws
```

Dummy Service (Web Server / Apache2)

```
ubuntu@ip-172-31-23-166:~$ systemctl status apache2
Unit apache2.service could not be found.
```

Hardware Information (Storage, CPU, RAM, Network)

```
ubuntu@ip-172-31-23-166:~$ hwinfo --storage | grep -A50 -E '13'
13: None 00.0: 0180 Storage controller
  [Created at pci.1416]
  Unique ID: JaHa.j9MZgu1IFZ6
  SysFS ID: /devices/vbd-768
  SysFS BusID: vbd-768
  Hardware Class: storage
  Model: "Xen Virtual Storage 0"
  Vendor: int 0x6011 "Xen"
  Device: int 0x1003 "Virtual Storage 0"
  Driver: "vbd"
  Driver Modules: "xen_blkfront"
```

```
ubuntu@ip-172-31-23-166:~$ hwinfo --cpu
01: None 00.0: 10103 CPU
  [Created at cpu.460]
  Unique ID: rdCR.j8NaKXDZtZ6
  Hardware Class: cpu
  Arch: X86-64
  Vendor: "GenuineIntel"
  Model: 6.63.2 "Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz"
```

```
ubuntu@ip-172-31-23-166:~$ hwinfo --memory
01: None 00.0: 10102 Main Memory
  [Created at memory.74]
  Unique ID: rdCR.CxwsZFjVASF
  Hardware Class: memory
  Model: "Main Memory"
  Memory Range: 0x00000000-0x3d2a1fff (rw)
  Memory Size: 960 MB
```

```
ubuntu@ip-172-31-23-166:~$ hwinfo --netcard
13: None 00.0: 0200 Ethernet controller
  [Created at pci.1416]
  Unique ID: Zz0U.USuSioBKp55
  SysFS ID: /devices/vif-0
  SysFS BusID: vif-0
  Hardware Class: network
  Model: "Xen Virtual Ethernet Card 0"
  Vendor: int 0x6011 "Xen"
  Device: int 0x0003 "Virtual Ethernet Card 0"
  Driver: "vif"
  Driver Modules: "xen_netfront"
```

Instance Type.Size Through Meta-Data

```
ubuntu@ip-172-31-23-166:~$ curl 169.254.169.254/latest/meta-data/instance-type && echo " "
t2.micro
```

Grub Version & Installed Package

```
ubuntu@ip-172-31-23-166:~$ grub-install --version
grub-install (GRUB) 2.02-2ubuntu8.21
```

```
ubuntu@ip-172-31-23-166:~$ dpkg -l grub-pc grub-pc-bin grub-legacy-ec2 grub-gfxpayload-lists grub2-common grub-common
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                          Version             Architecture          Description
+++-=============================-===================-=====================-=================================================================
ii  grub-common                   2.02-2ubuntu8.21    amd64                 GRand Unified Bootloader (common files)
ii  grub-gfxpayload-lists         0.7                 amd64                 GRUB gfxpayload blacklist
un  grub-legacy-ec2               <none>              <none>                (no description available)
ii  grub-pc                       2.02-2ubuntu8.21    amd64                 GRand Unified Bootloader, version 2 (PC/BIOS version)
ii  grub-pc-bin                   2.02-2ubuntu8.21    amd64                 GRand Unified Bootloader, version 2 (PC/BIOS binaries)
ii  grub2-common                  2.02-2ubuntu8.21    amd64                 GRand Unified Bootloader (common files for version 2)
```

Journal Errors Output

```
ubuntu@ip-172-31-23-166:~$ journalctl -p 3 -xb
-- Logs begin at Thu 2021-04-29 16:13:25 UTC, end at Thu 2021-04-29 17:01:09 UTC. --
Apr 29 17:00:59 ip-172-31-23-166 kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22!
Apr 29 17:00:59 ip-172-31-23-166 kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22!
```

Conversion Checks

If nothing is precised then screenshot are valid for both trials

Distribution Release & Kernel Version

```
ubuntu@ip-172-31-23-166:~$ cat /etc/os-release | grep VERSION=
VERSION="16.04.7 LTS (Xenial Xerus)"
ubuntu@ip-172-31-23-166:~$ uname
5.4.0-1038-aws
```

Dummy Service (Web Server / Apache2)

```
ubuntu@ip-172-31-23-166:~$ systemctl status apache2
● apache2.service - LSB: Apache2 web server
   Loaded: loaded (/etc/init.d/apache2; bad; vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Thu 2021-04-29 15:59:55 UTC; 5min ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1236 ExecStart=/etc/init.d/apache2 start (code=exited, status=0/SUCCESS)
    Tasks: 55
   Memory: 7.4M
      CPU: 249ms
   CGroup: /system.slice/apache2.service
```

Instance Type.Size Through Meta-Data

```
ubuntu@ip-172-31-23-166:~$ curl 169.254.169.254/latest/meta-data/instance-type && echo " "
t2.micro
```

Journal Errors Output (1St Trial)

```
ubuntu@ip-172-31-23-166:~$ journalctl -p 3 -xb
-- Logs begin at Thu 2021-04-29 17:15:12 UTC, end at Thu 2021-04-29 17:20:41 UTC. --
Apr 29 17:15:12 ip-172-31-23-166 kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22!
Apr 29 17:15:12 ip-172-31-23-166 kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22!
Apr 29 17:15:12 ip-172-31-23-166 systemd-modules-load[408]: Failed to find module 'ib_iser'
Apr 29 17:15:12 ip-172-31-23-166 systemd[1]: Failed to start Load Kernel Modules.
```

Journal Errors Output (2nd Trial)

```
ubuntu@ip-172-31-23-166:~$ journalctl -p 3 -xb
-- Logs begin at Thu 2021-04-29 19:23:26 UTC, end at Thu 2021-04-29 19:23:57 UTC. --
Apr 29 19:23:26 ip-172-31-23-166 kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22!
Apr 29 19:23:26 ip-172-31-23-166 kernel: Cannot get hvm parameter CONSOLE_EVTCHN (18): -22!
Apr 29 19:23:32 ip-172-31-23-166 iscsid[1120]: iSCSI daemon with pid=1121 started!
```

Migration of historic systems hosted in the cloud

by

Adrien FIAND

APPENDIX B

-

Complete Technical Procedure

September 11, 2021

In bold are the summary of the action to accomplish
In italic are the commands used in the Linux instances
In normal are the tasks done from in the AWS Console
In gray and brackets are some useful comments

Step 0 (Backup)

- Take AMI or Snapshot the old generation server that is using PV_EC2. (ex: T1.Micro)
[I will call it PV_EC2 in the doc.]

Step 1 (Restore PV Instance)

- (AMI) Restore that AMI in a new PV_EC2.
Using the same Instance Type and Size.

- (SNAPSHOT) Restore the snapshot on a Drive.
Mount it as the root of the PV_EC2.

Step 2 (Analyze)

- SSH into PV_EC2 instance > Get the OS version, Kernel, Hardware Info, Grub
cat /etc/os-release
uname -r
hwinfo --netcard && hwinfo --storage && hwinfo --memory && hwinfo --cpu
dpkg -l grub-pc grub-pc-bin grub-legacy-ec2 grub-gfxpayload-lists grub-common
grub-install --version

Step 3 (Preparation)

- Perfom PV_EC2 packages update, if any is available, then shutdown
sudo apt update && apt upgrade
sudo shutdown now

- Launch a new HVM_EC2

Take the closet Version to the PV_EC2 OS (ex: Linux Ubuntu 18.04)

Take the closet newer generation that supports HVM (ex: T2)

Use the same size as the PV_EC2 (ex: Micro)

Make sure to put it in the same VPC & AZ as the PV_EC2 (for EBS volumes swap)

[I will call it HVM_EC2 in the doc.]

Step 4 (Conversion)

- From AWS Console

Get the IDs of the volumes attached to PV_EC2 & HVM_EC2 (remember them)

Detach the root volume (shoud be /dev/sda1) from PV_EC2 instance

Attach this volume to HVM_EC2 (/dev/sdf for example - remember it)

- SSH into HVM_EC2 instance > Check mount point of the volume you just attached

sudo fdisk -l

[Note depending on OS it might be /dev/sdf like in AWS Console or /dev/xvdf ...]

- Mount the PV_EC2 root volume you just attached to HVM_EC2

sudo mkdir -p /mnt/xvdf && sudo mount /dev/xvdf /mnt/xvdf

- You can control it's mounted using ls (if there is folder like etc, boot it's good)

ls -l /mnt/xvdf

- Once mounted, we copy the HVM_EC2 boot folder over the PV_EC2 one (overwrite)

sudo rsync -avzXA /boot/ /mnt/xvdf/boot/

[This copies the bootloader, kernel, initrd]

- If needed copy some part or the complete lib folder from HVM_EC2 to PV_EC2 drive

sudo cp -prf /lib/modules/$(uname -r)/kernel/driver/ /mnt/xvdf/lib/modules/$(uname -r)/kernel/driver/

[This copies the kernel specific (uname -r) firmwares]

- We need to 'trick' the system into thinking the mounted PV_EC2 drive is the root drive

sudo mount -o bind /dev /mnt/xvdf/dev

sudo mount -o bind /dev/pts /mnt/xvdf/dev/pts

sudo mount -o bind /proc /mnt/xvdf/proc

sudo mount -o bind /sys /mnt/xvdf/sys

[This is needed in order to be able to update the bootloader (grub) of that drive].

- Fake/Jail HVM_EC2 root directory to be the PV_EC2 drive

chroot /mnt/xvdf

- Make the PV_EC2 drive bootable “again”

grub-install --no-floppy --recheck --force /dev/xvdf && update-grub2

[On ext2 fs. a warning might be thrown about block size, it can be ignored]

- Leave the fake root directory and shutdown HVM_EC2

exit chroot: CTRL+D

sudo shutdown now

Step 5 (Finalization)

- From AWS Console

Detach the root volume (shoud be /dev/sda1) from HVM_EC2 instance

Detach the PV_EC2 volume (shoud be like /dev/sdf) from HVM_EC2 instance

Attach back PV_EC2 volume to HVM_EC2 instance (as the root /dev/sda1)

Step 6 (Checks)

- SSH into HVM_EC2 instance > Check the services are up and if the journal has errors

systemctl status apache2

journalctl -p 3 -xb

Step 7 (Clean-Up)

- From AWS Console

Terminate the PV_EC2 Instances (it should not have any EBS anymore)

Delete the original HVM_EC2 drive (it should be detached since step 5)

Keep the snapshot or AMI a little longer just in case…