Infrastructure as a Service

Infrastructure as a Service, or IaaS is a service offering by most cloud providers that provides *virtual machines* and the accompanying infrastructure as a service. This lecture will discuss the details of how an IaaS service is built.

Virtual machines

Virtualization is a surprisingly old technology. The first virtualized system was the IBM System/370 mainframe with the VM/370 operating system in 1972. The system was different from how we understand virtualization today, but the goal was the same: separate workloads from each other.

When you think about mainframes you have to consider that these machines were *very* expensive and machine time was a scarce resource. Most programs back in those days were *batch jobs*. They processed a large set of data at once and then terminated.

Initially CPUs in personal computers did not have application separation. The x86 line of Intel CPUs only received the *protected mode* feature with the 80286 in 1982. The operating system (MS/DOS) would run in in *real mode* and applications could then switch into the new mode to isolate applications from each other. One such application making use of the new mode was Windows that ran on top of MS/DOS.

Protected mode introduced the concept of *rings* in the CPU. The operating system *kernel* would run in ring 0, device drivers would run in ring 1 and 2 while applications would run in ring 3. The lower ring number meant the higher privilege level.

> **Note**
>
> Device drivers today typically run on ring 0 instead of 1 or 2.

This ring system allowed the operating system to restrict the higher ring numbers from accessing certain functions or memory locations. However, most applications in the day would violate the restrictions of protected mode and could not run in the new mode.

> **Note**
>
> If you try and set up a really old computer game like Commander Keen in DOSBox you will realize that you have to provide the game itself with very hardware-specific settings. You will, for example, have to provide details for your

> sound card. This is because the game itself incorporated sound card drivers for Sound Blaster 16 or Gravis Ultrasound cards. A game that would do this could not run in protected mode.

To work around the problems with protected mode the 80386 successor introduced virtual mode. The new virtual 8086 mode (VM86) introduced a number of compatibility fixes to enable running old real mode programs in a multitasking environment such as Windows without problems.

For instance the CPU would create a simulated *virtual* memory space the program could write to and translate the virtual addresses to physical addresses internally. It would also capture sensitive instructions and turn them over for control to the kernel.

> **Note**
>
> VM86 does not capture *every* instruction the application runs in virtual mode, only the sensitive CPU instructions. This enables legacy applications to run at a reasonable speed.

In the mid 2000's CPUs became so powerful that it made sense to not only virtualize applications but whole operating systems including their kernel. This allowed multiple operating systems to run in parallel. However, without CPU support only *software virtualization* could be achieved. In other words early virtualization software had to *simulate* a CPU in ring 0 to the guest operating system. Some virtualization techniques, such as Xen required the guest operating system to run a modified kernel to facilitate them running in ring 3. Others employed a number of techniques we won't go into here.

While initially expensive hardware support followed suit. In 2005 Intel added the VT-x (Vanderpool) feature to its new Pentium 4 CPUs followed by AMDs SVM/AMD-V technology in 2006 in the Athlon 64, Athlon 64 X2, and Athlon 64 FX processors.

VT-x and AMD-V added new ring -1 to accommodate *hypervisors*. This new ring allowed for separation between several operating systems running at ring 0. Later CPU releases added features such as Direct Input/Output virtualization, network virtualization or even graphics card virtualization. These features allowed for more efficient virtualization and sharing hardware devices between several virtual machines.

> **Note**
>
> Intel also introduced a ring -2 for the Intel Management Engine, a chip that functions as an OOBM in modern Intel chips. The ME runs its own operating system, a MINIX variant and has been the target of severe criticism for its secrecy and power over the machine. Several bugs have been found in the ME that let an attacker hide a malware inside the ME.

Virtualization also gave rise to Infrastructure as a Service. AWS was the first service that offered virtual machines as a service starting in 2006 with a Xen-based offer. They not only offered

virtual machines but they did so that a customer could order or cancel the service using an Application Program Interface.

This allowed customers to create virtual machines as they needed it and they were billed for it on an hourly basis. (Later on AWS and other cloud providers moved to a per-second billing.)

The presence of an API makes the difference between IaaS and plain old virtual machines as a service. IaaS allows a customer to scale their application dynamically according to their current demand.

Typical instance types

When the cloud became popular in the late 2000s several providers attempted to offer a service that was fully dynamic in their sizes. The customer could set how many GB of RAM they needed and how many CPU cores. However, this model has been phased out by most providers since it is difficult to manage such a dynamic environment.

Instead cloud providers nowadays opt to offer fixed machine sizes (think of t-shirt sizes). To accommodate high-CPU and high RAM workloads there are several different instance types, typically:

- **Shared CPU:** These are small instances where a single CPU core is shared between multiple virtual machines, sometimes leading to high steal time. Sometimes this offering includes a *burst* capability (such as the Amazon T instances) where a VM can temporarily use more CPU.
- **Standard, dedicated core CPU:** These instance types receive one or more physical cores leading to a more stable performance without the ability to burst beyond their limits.
- **High CPU:** These instance types are usually hosted on physical servers that have a very high CPU to RAM ratio. Accordingly, the virtual machine offering includes more CPU than the standard offering.
- **High RAM:** This offering is the exact opposite of the high CPU offering. The machines on offer here include more RAM with very little CPU.
- **Storage:** These instance types contain large amounts of local storage (see below in the storage section).
- **Hardware-specific:** These instance types offer access to dedicated hardware such as graphics cards (GPUs) or FPGAs.

Automation

As discussed before, that makes an IaaS cloud provider a cloud provider is the fact that they offer an API to automate the provisioning and deprovisioning of virtual machines as needed. However, that's not all. Simply starting a virtual machine is not enough, the software needs to be installed in it.

Initially this problem would be solved by creating *templates* for the operating system that launches. In larger cloud setups these templates included a pre-installed agent for configuration management that would report to a central service and fetch its manifest of software to install.

Thankfully in the last decade a lot has happened and Cloud Init has established itself as a defacto standard in the IaaS world. Every cloud provider nowadays offers the ability to submit a *user data* field when creating a virtual machine. This user data file is read by Cloud Init (or its Windows alternative Cloudbase Init) and is executed at the first start of the virtual machine.

A DevOps engineer can simply inject a script that runs at the first start that takes care of all the installation steps required.

Tools like Terraform or Ansible assist with managing the whole process of provisioning the virtual machines and supplying it with the correct user data script.

Virtual machine pools

One other use of user data are virtual machine pools. Each cloud provider adopts a different name for them, ranging from instance pools to autoscaling groups. The concept is the same everywhere: you supply the cloud with a configuration how you would like your virtual machines to look like and the cloud will take care that the given number of machines are always running. If a machine crashes or fails a health check the cloud deletes the machine and creates a new one.

The number of machines in a pool can, of course, be changed either manually or in some cases automatically using rules for automatic scaling.

Combined with the aforementioned user data this can be a very powerful tool to create a dynamically sized pool of machines and is the prime choice for creating a scalable architecture.

These pools are often integrated with the various load-balancer offerings cloud providers have in their portfolio to direct traffic to the dynamic number of instances. Some cloud providers integrate them with their Functions as a Service offering as well allowing you to run a custom function whenever a machine starts or stops. This can be used to, for example, update your own service discovery database.

Storage

Hopefully we have managed to convey the *dynamic nature* of the cloud in our description so far. This brings up the question of where data should be stored. This section will explain the different storage options that you can use to store *files* directly from your virtual machine. We will touch on databases and other storage options in a later section.

How do filesystems work?

Before we dive into the different options let's take a look at how filesystems work. In a non-cloud environment you would insert a hard drive into your computer. This hard drive can either be a spinning disk (often referred to as HDD) or a solid-state drive (SSD).

Spinning disks have a read head that goes over the platter that spins under it. This means that information access on HDD requires the head to get into the correct position to read the data which takes time. This positioning is done by the controller chip that's on the disk itself.

SSD's have different problems. Information access is instant since any block can be accessed immediately but the chip on the disk still needs to keep track of which cell is unusable, caching, etc.

In other words the controller chip deals with the *low-level functions* on the disk for both HDD and SSD. The drive itself is accessible via either a SATA (Serial ATA) or a SAS (Serial Attached SCSI) connection. In case of servers these disks are often assembled into a RAID array for redundancy and higher performance. Most RAID controllers will also implement caching which often also requires a built-in battery unit to function properly.

In both cases the data is stored in blocks of fixed sizes (such as 4 kB). The filesystem itself contains the mapping of which file consists of which blocks on the disk.

This is an important distinction: block storage solutions, no matter if they are network-bound or use a more classic connection such as fibre channel only know about the blocks on the disk. They do not know anything about files. Since filesystems are complex beasts block storage devices can only be used by a single machine at any given time.

Network filesystems on the other hand are more resource intensive but can track which file is open from which machine and can be used in a distributed fashion.

Local Storage

The most obvious storage option, of course, is a local storage. By local storage we mean a hard drive (SSD or HDD) that is integrated directly into the machine that's running the hypervisor.

This storage option has the benefit of offering a high performance disk at a relatively affordable price. The drawback is that the disk is local. If the physical machine running your VM has a problem your data may be lost.

For all storage options the implementation of *backups* is critical but with local storage building redundancy on top of the virtual machine may be required to build a reliable system.

Network Block Storage

Network block storage means a block storage that is delivered over the network. When talking about network we are not only talking about your standard IP / Ethernet network but also a direct SCSI over Fibre Channel infrastructure. The latter is often used in expensive enterprise storage systems that provide attachable disks to several physical machines.

TODO: add illustration for fibre channel storage systems

The simplest and easiest to set up network block storage is probably ISCSI, a protocol that runs the SCSI storage protocol over a commodity IP network. In other words one computer can take a block device and let another computer use it. However, as mentioned previously in this case the computer using the ISCSI device is in full control over the files that are stored on the storage device. It is also worth mentioning that ISCSI does not guarantee any redundancy beyond the RAID or storage system the offering computer already has.

More advanced storage systems such as Ceph RBD or replicated enterprise storage systems offer redundancy such that when one storage instance fails others can take its place without outage. Cloud provider offerings such as EBS by Amazon also offer this kind of redundancy with severe limitations as to how many EBS volumes can be attached to a virtual machine.

Network File Systems

In contrast to network block storage network file systems offer access to data not on a block level, but on a file level. Over the various network file system protocols machines using these file systems can open, read and write files, and even place locks on them.

The filesystem has to keep track of which machine has which file open, or has locks on which file. When machine edit the same file in parallel the filesystem has to ensure that these writes are consistent. This means that network file systems are either much slower than block-level access (e.g. NFS) or require a great deal more CPU and RAM to keep track of the changes across the network (e.g. CephFS).

Object storage

Object storage systems are similar to network file systems in that they deal with files rather than blocks. However, they do not have the same synchronization capabilities as network file

systems. Files can generally only be read or written as a whole and they also don't have the ability to lock a file.

While object storages technically *can* be used as a filesystem on an operating system level for example by using s3fs this is almost always a bad idea due to the exceptionally bad performance and stability issues.

Operating system level integration should only be used as a last resort and object storages should be integrated on the application level. We will discuss object storage services in detail in our next lesson.

Network

The next big topic concerning IaaS services is networks. Before we go into the cloud-aspect let's look at how the underlying infrastructure is built. As indicated in the first lecture it is strongly recommended that you familiarize yourself with the basics of computer networking, such as the Ethernet, IP and TCP protocols as you will need them to understand this section.

How cloud networks are built

So, let's get started. Imagine a data center from the first lecture. Your task is to build an IaaS cloud provider. You put your servers that will serve as your hosts for virtual machines in the racks. These servers will be connected to the Top-of-Rack switches (yes, two for redundancy) using 10 GBit/s network cables. The switches are themselves connected among each other and across racks with several 100 GBit/s.

Now comes the tricky part: how do you create private networks for each customer? One option would be to use 802.11Q VLAN tags separating each network from each other. However, that limits you to 2^{12} = 4096 private networks. That may seem like much but for a public cloud provider that is not sufficient. Therefore, public cloud providers use overlay protocols like VXLAN to work around this problem.

We describe this not for you to learn but to highlight that public cloud networks are *complex*. A cloud provider cannot guarantee a fixed bandwidth between two virtual machines (unless they are put on the same physical server using affinity groups). This is one of the reasons that often the ideal setup for cloud architectures is one that uses multiple, medium-sized instances rather than few large sized ones.

Underlying network architectures offered by cloud providers

When we look at the network offerings by cloud providers there are three types:

1. Virtual machines receive private IP addresses and a gateway or load balancer handles the public-to-private IP translations. This is the case with the larger cloud providers such as AWS, Azure and GCP at the time of writing.
2. Virtual machines have one public IP address on the first network interface and additional private networks can be attached as new, separate network interfaces. This is the case with most smaller IaaS providers such as DigitalOcean, Upcloud or Exoscale.
3. Fully dynamic network configuration that allows the customer to define their network setup and IP assignment dynamically. This is typically offered by IaaS providers that target enterprise customers who wish to migrate their classic on-premises infrastructure and require the flexibility they had when using their own hardware. This is the case with 1&1 IONOS.

TODO: Vultr? Linode? Deutsche Telekom Cloud? Alibaba Cloud?

TODO: add illustration

Out of group 2 it is worth mentioning that the services that are available on the public network (firewalls, load balancers) are often not available on private networks.

Firewalling

Network load balancers

VPNs, private interconnects, and routing services

DNS

Monitoring

Automation