Introducing SlapOS Architecture

by SlapOS Team.

This tutorial explains the simple concepts which underlie SlapOS architecture. SlapOS is a distributed, open source, Cloud system. With SlapOS, anyone can become a Cloud provider, selling Software as a Service (SaaS), Platform as a Service (PaaS) or Infrastructure as a Service (PaaS). With SlapOS it does not matter if one uses their own private hardware infrastructure or public, shared infrastructure. SlapOS can accommodate the diversity of Cloud resources and gather the Cloud resourses as if they were yours. SlapOS also helps optimizing resource usage between different Cloud providers.

Agenda

- Masters and Slaves
- Computer Partitions
- Networking

This tutorial has 3 parts. In the first part we explain the concept of master and slave nodes in SlapOS. In the second part we explain the concept of computer partition in SlapOS. In the third part we explain how SlapOS approaches networking.

Master and Slaves

SlapOS is based on a Master and Slave design.Here, we are going to provide an overview of SlapOS architecture. We will be explaining the role of the Master and Slave nodes in particular, as well as the software components on which they rely on to operate a Distributed Cloud.

Overview

Slave nodes request to the Master node which software they should install, which software they should run and report to the Master node how much resources each running software has been using for a certain period of time. The Master node keeps track of the available slave node capacity and available software. The Master node also acts as a Web portal and Web service so that end users and software bots can request software instances which are instantiated and run on Slave nodes.

Master nodes are stateful. Slave nodes are stateless. More precisely, all information required to rebuild a Slave node is stored in the Master node. This may includes the URL of a backup service which keeps an online copy of data so that in case of failure of a Slave node, a replacement Slave node can be rebuilt with the same data.

It is thus very important to make sure that the state data present in the Master node is well protected. This could be implemented by hosting the Master node on a trusted IaaS infrastructure with redundant resource. Or - better - by hosting multiple Master nodes on many Slave nodes located in different regions of the world, thanks to appropriate data redundancy heuristic.

We are approaching here the first reflexive nature of SlapOS. A SlapOS Master is normally a running instance of SlapOS Master software instantiated on a collection of Slave nodes, which, together, form a trusted hosting infrastructure. In other terms, SlapOS is self-hosted.

Master Node

Let us now review in more detail the role of the SlapOS Master node. SlapOS keeps track of the identity of all parties which are involved in the process of requesting Cloud resources, accounting Cloud resources and billing Cloud resources. This includes end users (Person) and their company (Organisation). It includes suppliers of Cloud resources as well as consumers of Cloud resources. It also includes so-called computer partitions which may run a software robot to request Cloud resources without human intervention. It also includes Slave nodes which need to request to SlapOS Master which resources should be allocated. SlapOS generates X509 certificates for each type of identity: X509 certificates for people like you and me who login, an X509 certificate for each server which contributes to the resources of SlapOS and an X509 certificate for each running software instance which may need to request or notify SlapOS Master. A SlapOS Master node with a single Slave node, a single user and 10 computer partitions will thus generate up to 12 X509 certificates: one for the slave, one for the user and 10 for computer partitions.

Any user, software or Slave node with an X509 certificate may request resources to SlapOS Master node. SlapOS Master node plays here the same role as the back office of a marketplace. Each allocation request is recorded in SlapOS Master node as if it were a resource trading contract in which a resource consumer requests a given resource under certain conditions. The resource can be a NoSQL storage, a virtual machine, an ERP, etc. The conditions can include price, region (ex. China) or specific hardware (ex. 64 bit CPU). Conditions are somehow called Service Level Agreements (SLA) in other architectures but they are considered here rather as trading specifications than guarantees. It is even possible to specify a given computer rather than to rely on the automatic marketplace logic of SlapOS Master.

By default, SlapOS Master acts as an automatic marketplace. Requests are processed by trying to find a Slave node which meets all conditions which were specified. SlapOS thus needs to know which resources are available at a given time, at which price and with which characteristics.

Lastly, SlapOS Master also needs to know which software can be installed on which Slave node and under which conditions.

Slave Nodes

SlapOS Slave nodes are pretty simple compared to the Master node.

Every Slave node needs to run software requested by the Master node. It is thus on the Slave nodes that software is installed. To save disk space, Slave nodes only install the software which they really need.

Each Slave node is divided into a certain number of so-called computer partitions. One may view a computer partition as a lightweight secure container, based on Unix users and directories rather than on virtualization. A typical barebone PC can easily provide 100 computer partitions and can thus run 100 wordpress blogs or 100 e-commerce sites, each of them with its own independent database. A larger

server can contain 200 to 500 computer partitions.

SlapOS approach of computer partitions was designed to reduce costs drastically compared to approaches based on a disk image and virtualization. But it does not prevent from running virtualization software inside a computer partition, which makes SlapOS at the same time cost efficient and compatible with legacy software.

Master Software

The reference implementation of SlapOS Master node is based on ERP5. SlapOS Master node is actually derived from ERP5 implementation for a Central Bank. The underlying idea is that currency clearing and Cloud resource clearing are very similar. They should thus be implemented with the same software. Since ERP5 was already implemented to run a Central Bank in 8 countries, it was a natural choice. Moreover, ERP5 has demonstrated its scalability for large CRM applications (ex. Beteireflow) and its trustability for accounting. Thanks to NEOPPOD, its distributed NoSQL database, ERP5 can provide the kind of transactional nature and scalability which is required for a stateful marketplace.

Implementing SlapOS Master on top of ERP5 was a direct application of ERP5 Universal Business Model (UBM) technology, a model which unifies all sciences of management and which has been acknowledged by numerous IEEE publications as a major shift in enterprise application design. Each Computer is represented by an Item in UBM. Allocation requests, resource deliveries and resource accounting are represented by a Movement in UBM. The movement resource can be: software hosting, CPU usage, disk usage, network usage, RAM usage, login usage, etc. Software hosting movements start whenever the running software starts in the computer partition and stop whenever the running software stops. Resource usage movements start and stop for accounting during each period of time, independently of the software running state. The software release which is run on the computer partition is also an Item in UBM, just like the subscription contract identifier. The parties (client, supplier) are represented as Node in UBM. More surprisingly, each Network is considered also as a Node in UBM, just as a storage cell is represented as a Node in logistics.

Slave Software

SlapOS Slave software consists of a POSIX operating system, SlapGRID, Supervisord and Buildout.

SlapOS is designed to run on any operating system which supports GNU's Glibc and Supervisord. Such operating systems include for example GNU/Linux, FreeBSD, MacOS/X, Solaris, AIX, etc. We hope in the future that Microsoft Windows will also be supported as a host (Microsoft Windows is already supported as a guest) through Glibc implementation on Windows and a port of Supervisord to Windows.

SlapOS relies on mature software: Buildout and Supervisord. Both software are controlled by SlapGRID, the only original software of SlapOS. SlapGRID acts as a glue between SlapOS Master node (ERP5) and both Buildout and Supervisord. SlapGRID requests to SlapOS Master node which software should be installed and executed. SlapGRID uses Buildout to install software and Supervisord to start and stop software processes. SlapGRID also collects accounting data produced by each running software and sends it back to SlapOS Master. Let us now study with more detail the roles of Supervisord and Buildout.

Supervisord is a process control daemon. It can be used to programmatically start and stop processes with different users, handle their output, their log files, their errors, etc. It is a kind of much improved init.d which can be remotely controlled. Supervisord is lightweight and old enough to be really mature (i.e. no memory leaks).

Quoting the Buildout website, "*Buildout is a Python-based build system for creating, assembling and deploying applications from multiple parts, some of which may be non-Python-based. It lets you create a Buildout configuration and reproduce the same software later.*" Buildout originated from the Zope/Plone community to automate deployment of customized instances of their software. Lead by Jim Fulton, CTO of Zope Corporation, Buildout became a stable and mature product over the years.

Buildout is used in SlapOS to define which software must be executed on a Slave node. It plays a key role in SlapOS industrial successes. Without it, SlapOS could not exist. However, Buildout is also often misunderstood - sometimes purposely - by observers who criticize its use in SlapOS. Many people still do not realize that there is no possible software standard on the Cloud and that Buildout is the solution to this impossibility. Experts know for example that any large scale production system which is operated on the Cloud (ex. a social network system) or privately (ex. a banking software) uses patched software. Relational databases are patched to meet performance requirements of given applications as soon as data grows. If a Cloud operating system does not provide the possibility to patch about any of its software components, it is simply unusable for large scale production applications. SlapOS is usable because its definition of what is a software is based on the possibility of patching any dependent software component.

Where is my patch?

Still people who name a software such as "KVM" or "MySQL" believe that this is enough (and for them, SlapOS provides aliases for the words "KVM" and "MySQL" which link to an explicit Buildout definition). However, the reality is not that straightforward. For example, some releases of KVM support NBD protocol over IPv6 but some not. Some releases of KVM support Sheepdog distributed block storage but some not. Some releases of KVM support CEPH distributed block storage but some not. Most users who run KVM to try a software do not care about IPv6, Sheepdog or CEPH. But those users who run KVM on SlapOS need IPv6 support to access NBD, and this is for now only available as a patch. Those who want resilient storage may want Sheepdog support which is only available from version 0.13. And those who want CEPH support also need a patch. However, those users who want the IPv6 patch may prefer not to use the CEPH patch which is not yet stable officially. And those who want CEPH patch may distrust the IPv6 patch. All in all, there is no way to agree on a single version of KVM. All the different releases of KVM may have to be installed on SlapOS Slave nodes in order to meet market requirements. Since the patch possibilities are so wide, the easiest way to know afterall which KVM is being installed on a SlapOS node is simply to list where its original source code was obtained from and which patches were applied. This is exactly what Buildout does, in just a few lines of configuration. Buildout also eliminates any complex or time consuming process to distribute binary packages on a wide range of hardware architecture, thanks to a trusted, distributed, caching mechanism which does not even centralize signature.

The problem we are discussing here about KVM is even more complex than MySQL. There are now multiple sources of MySQL: the official one (MySQL), the one by MySQL original author Michael Widenius (MariaDB), the one by Percona InnoDB experts and the one by Cubrid which is not MySQL but claims to be 90% compatible with it. Among each source of MySQL sources, there are different versions. Default compilation options may also differ. Authors of large scalable applications know very well that the performance of their applications can be dramatically impacted by subtle changes to the SQL optimizer. Changing the version of source of MySQL may simply lead to a performance collapse. We always remember an example of application for which we had to change the default parameters in MySQL header file in order to scan 32 rows instead of 8 for query optimization. Therefore, if we did not have the possibility to choose which source of MySQL to use and which patch to apply to it, we just could not have run entreprise applications with SlapOS and shown industrial success stories.

Arguments and counter-arguments against Buildout

The use of Buildout by SlapOS is disruptive compared to traditional approaches of software distribution. It has enabled industrial success faster. But it also has lead to slower adoption of SlapOS by certain communities, often for incorrect rationale. We are going to discuss further.

What about disk images?

Some people consider that Buildout is irrelevant since Cloud should be based on disk images and virtual machines. What those people do not realize is that not only SlapOS can run about any disk image format but that Buildout can be used to automate the production of disk images, much better probably than many other tools. And it is open source.

What about distributions' packaging systems?

Some people consider that Buildout is irrelevant since it is possible to achieve the same with packaging systems of GNU/Linux distributions. What they do not realize is that not only Buildout can rely on existing GNU/Linux distribution packages (at the expense of portability) but that Buildout can also be used to automate the production of packages for multiple GNU/Linux distributions in little effort. Also, Buildout format is much more concise when it comes to patching or adding dependencies to existing software thanks to the "extends" mechanism. Lastly, Buildout provides a kind of packaging format which can reuse language based packaging formats (eggs, gems, CPAN, etc.) in a way which is neither specific to a given GNU/Linux distribution nor to GNU/Linux itself. In a sense, Buildout integrates much better with native language distribution systems than GNU/Linux packaging systems do. And native language distribution systems are currently becoming the de facto standard for developers.

What about separation between software and instance?

Some people consider that Buildout prevents sharing the same executable among multiple instances of the same application. This is a common misconception, which is also wrong. SlapOS is a typical example of how to deploy once a single software made of shared libraries and executable binaries and create hundred instances of it without any binary code duplication, without wasting resident RAM.

I need something that is language agnostic

Some people consider that Buildout is designed for python only. What they do not realize is that Buildout is already used to build software based on C, C++, Java, Perl, Ruby, etc. And it would not be an issue to extend SlapOS and support any Buildout equivalent. But we are not aware of any system builder such as Buildout which can support as many different architectures and languages in such a flexible way.

Come on, I'm on Windows

Some people consider that Buildout is not for Windows or that it does not support proprietary software in binary form, without source code. Again, this is a misconception. Buildout is just an automation tool. Whenever source code is not available, Buildout can take a binary file as input. This is what is often done for example to build Java applications based on .war distribution archives, or to deploy OpenOffice binaries which would else take 24 hours to compile. Buildout is also compatible with Windows. Automating the installation or the replication of Windows based software with Buildout is possible. Buildout would even be an excellent candidate to automate the conversion of Windows disk images from one host environment to another. Generally speaking, running SlapOS natively on Windows could be very useful both for SlapOS and... for Windows.

It destroys the work made by GNU/Linux distributions

Overall, what makes Buildout so debated by some observers is that it shows a different path for software distribution, especially for open source software distribution. Instead of focusing - as GNU/Linux

distributions do - on providing a consistent set of about any possible open source application with perfectly resolved dependencies and maximized sharing of libraries, it focuses on building a single application only and its dependencies in a way which maximizes the portability between different GNU/Linux distributions and POSIX compliant operating systems. Application developers only need to care about their own application and stabilize its distribution. Unlike what happens with most GNU/Linux distributions, they do not need to care about possible consequences of changing one shared library on other applications hosted on the same operating system. Buildout is after all an approach to software distribution in which the most complex software has about 100 dependencies to resolve, compared to 10,000+ interdependent packages in a traditional GNU/Linux distribution. Buildout puts the burden of maintenance on each application packager and removes the burden of managing global dependencies, thus allowing parallel and faster release cycles for every application. All this is with a very concise approach.

Not convinced yet?

If this discussion does not make you convinced yet that Buildout is an efficient solution to specify a software executable and deploy it on the Cloud, please consider the following problem to solve: automate the packaging of ERP5 open source ERP and all its dependencies (OpenOffice, patched Zope, patched MariaDB, etc.) on all major GNU/Linux distributions in such a way that it is possible to provide the same behavior on every GNU/Linux distribution and to run 100 instances of ERP5 on the same server, each of which can have its own MariadDB daemon and Zope daemon. Obviously, if you find a better solution, please let us know.

SLAP Protocol

SlapOS is based on the SLAP protocol. Both SlapOS Master reference implementation based on ERP5 and SlapGRID reference implementation in python could be replaced. An implementation of the SLAP protocol was for example already made in Java on the client side in a few days. Implementing SLAP for about every language should be just as easy.

The SLAP protocol is a polling protocol. Every SlapOS Slave Node contacts through HTTP SlapOS Master Node for 4 different purposes: to define capacity, to collect the list of software to install, to collect the list of computer partitions to configure and to post accounting information.

At boot time, each Slave Node contacts SlapOS Master node to notify it that the boot process was completed and provides a list of available computer partitions, in particular their identifier and IPv6 address. This is the **set-capacity** request. This request is then launched again every 24 hours in order to take into account possible changes of network configuration, which normally should not happen but which sometimes do.

Every 5 minutes, SlapOS Slave node requests the list of software which should be installed. As for most parts of SLAP protocol, the values which are exchanged are promises to reach, not actions to take. SlapOS Master thus returns the complete list of software which are expected to be installed by the Slave node, not taking into account whether such software was installed or not. Reversely, if a software which was installed is no longer in the list, it implies that it should be removed. Just remember, SlapOS Slave Nodes are supposed to be stateless, just as the SLAP protocol.

Every 5 minutes, SlapOS Slave node requests the list of computer partitions to configure. This is handled by a different process. The underlying idea is that installing a software could take between a couple of minutes (if it was already compiled and cached for the same architecture) to a couple of hours (if it needs

to be compiled for the architecture). Configuring an instance should take on the other hand less than a couple of seconds and ideally less than a second. Each time SlapOS Slave node requests the list of computer partitions, this will eventually lead to the reconfiguration of **all** partitions. A large server could contain 300 partitions. If the configuration of a single partition takes one second, it takes 5 minutes to reconfigure all partitions. Obviously, SlapGRID tries to optimize partition configuration and will only reconfigure those partitions which configuration has changed since the last run. But, in case an incident happens, such as an earthquake or electricity shortage in a region, it is possible that all computer partitions of a given server need to be reconfigured at the same time, even though this is not desirable. In order to make sure that such massive reconfiguration does not lead to system collapse, we have taken the design decision to run configuration with a single process and a single thread, so that most cores of the host server are still available for running what they are actually supposed to run, instead of running configuration software.

Every day, accounting information is collected from every computer partition. It is the role of the software instance running in the computer partition to produce a file which contains usage and incident reports in TioXML format. All files are aggregated and posted to SlapOS Master which then uses them for further accounting and billing. One should take note that the accounting information which is exchanged is very abstract and can cover both physical usage (ex. CPU, RAM, disk), virtual usage (ex. number of users, number of transactions) and incidents (ex. failure to access data for 5 minutes). TioXML format is easy to extend in order to cover about any possible billing requirement.

We are currently considering extending the get-cp-list request with HTTP long polling or Web Sockets in order to make the system more reactive and at the same poll SlapOS Master less often. For now, it is not a priority. The goal of SLAP protocol will probably never consist of instantly providing a Cloud resource. For instant provisioning, we rather recommend a predictive pre-allocation approach. Rather than allocating on demand, one should pre-allocate based on previsions or for safety and simple pass to the requester the pre-allocated resource. We even think that slowing down the provisioning of resources is a good approach to reduce the risk of speculation on the availability of Cloud resources and thus an efficient way to increase Cloud Resilience. Further research combining Computer Science and Economy could eventually prove or infirm our assertion. Anyway, we think that more scalability could be reached through an HTTP-based push protocol. It remains to be seen how well such a protocol can resist to frequent network interruptions over intercontinental Internet transit routes.

Computer Partitions

The concept of Computer Partition is fundamental to understand the structure of a SlapOS Slave Node. A Computer Partition can be seen as a lightweight container or jail. It provides a reasonable level of isolation, based on the host operating system user and group management. It does not provide however the same level of isolation as the one which exists between virtual machines, unless of course computer partitions are used to run virtualization software, something SlapOS can do. We came with the idea of computer partition after trying other approaches. Around 2004, we started using chrooted filesystems and linux-vserver jails. We also tried to run virtual machines on the same server hardware. We found that both linux-vserver jails and virtual machines required maintaining one complete filesystem per instance of application. This generated much additional effort compared to having to maintain only one filesystem. Also it was impossible to run hundreds of filesystems or virtual machines on the same host because of the huge overhead of each filesystem and virtual machine. This meant that reaching low cost hosting for standard open source applications was close to impossible with this approach. We then discovered Buildout and found that it was possible to split Buildout into two independent profiles: one profile to build the software in a self contained way and one profile to configuration files in a directory with links to a shared software directory. The concept of Computer Partition was created. Thanks to this concept, it is now possible to reach a hosting cost of less than 1 EUR / month per hosted application. Competition with Cloud monopolies becomes possible for all independent software vendors.

Let us now review the details of a Computer Partition.

Computer Partition N

- dedicated global IPv6
- dedicated local IPv4
- dedicated slaptapN
- dedicated slapuserN
- /srv/slapgrid/slappartN
- optional /dev/sdaX and IPv4

Every computer partition consists of a dedicated IPv6 address, a dedicated local IPv4 address, a dedicated tap interface (slaptapN), a dedicated user (slapuserN) and a dedicated directory (/srv/slapgrid/slappartN). Optionally, a dedicated block device and routable IPv4 address can be defined.

SlapOS is usually configured to use IPv6 addresses. Although use of IPv6 is not a requirement (an IPv4 only SlapOS deployment is possible) it is a strong recommendation. IPv6 simplifies greatly the deployment of SlapOS either for public Cloud applications or for private Cloud applications. In the case of public Clouds, use of IPv6 helps interconnecting SlapOS Slave Nodes hosted at home without having to setup tunnels or complex port redirections. In the case of private Cloud, IPv6 replaces existing corporate tunnels with a more resilient protocol which provides also a wider and flat corporate addressing space. IPv6 addressing helps allocating hundreds of IPv6 addresses on a single server. Each running process can thus be attached to a different IPv6 address, without having to change its default port settings. Accounting network traffic per computer partition is simplified. All this would of course be possible with IPv4 or through VPNs but it would be much more difficult or less resilient. The exhaustion of IPv4 addresses prevents the allocation of so many public IPv4 addresses to a single computer. After one year of experimentation with IPv6 in France, using Free IPv6 native Internet access (more than 50% of worldwide IPv6 traffic), we found that IPv6 is simple to use and creates the condition for many innovations which would else be impossible.

Even though IPv6 is used to interconnect processes globally on a SlapOS public or private Cloud, we found that most existing software is incompatible with IPv6. Reasons vary. Sometimes, IP addresses are stored in a structure of 3 integers, which is incompatible with IPv6. Sometimes, IPv6 URLs are not recognized since only dot is recognized as a separator in IP addresses. For this reason, we provide to each computer partition a dedicated, local, non routable IPv4 address. Legacy software listens on this IPv4 address. A kind of proxy mechanism is then used to create a bridge between IPv6 and IPv4. In the case of HTTP applications, Apache usually plays this role, in addition to the role of applicative firewall (mod_security) and strong security (TLS). In the case of other protocols, we usually use Stunnel for the same purpose. We will discuss this approach in the next chapter and study in particular how Stunnel can turn a legacy application into an IPv6 compatible application without changing any line of the original code.

For some applications, IP is not the appropriate ISO level. We provide to such applications a tap interface which emulates a physical Ethernet interface. This interface is usually bridged with one of the servers' physical Ethernet interfaces. Tap is often used by virtualization software such as KVM to provide access to the outer world network. This is for example how the default KVM implementation of SlapOS is configured. But it could also be used for other applications such as virtual private networks or virtual switches which require a direct access to Ethernet. In a Computer with 100 computer partitions, tap interfaces are usually named slaptap0, slaptap1, etc. until slaptap99.

Every computer partition is linked to a user and a directory. In a Computer with 100 computer partitions, users are usually named slapuser0, slapuser1, etc. until slapuser99. Directories are usually set to /srv/slapgrid/slappart0, /srv/slapgrid/slappart1, etc. until /srv/slapgrid/slappart99. Directory /srv/slapgrid/slappart0 is owned by user slapuser0 and by group slapuser0. Directory /srv/slapgrid/slappart1 is owned by user slapuser1 and by group slapuser1. Slapuser0 is able to access files in /srv/slapgrid/slappart0. Slapuser1 is not able to access files in /srv/slapgrid/slappart0. Moreover tap interface slaptap0 is owned by slapuser0, tap interface slaptap1 is owned by slapuser1, etc. Q: what

about IPv6 individual addresses, who own them?

For some applications, it could be necessary to attach to some partitions a raw block device. This could be useful to maximize disk I/O performance under certain configurations of KVM, and to access directly a physical partition of an SSH disk. This possibility has been included in the design of SlapOS, although it is not yet fully implemented.

For some applications, such as providing a shared front-end and accelerated cache, a dedicated IPv4 address is required. This possibility has been included in the design of SlapOS, although it is not yet fully tested (but it should be before Q3 2011).

To summarize security, a Computer Partition is configured to have no access to any information of another Computer Partition. Access rights in SlapOS have thus 3 different levels: global access, computer partition only access and superuser only access. SlapOS slave nodes are normally configured in such a way that global hardware status has global access right. Installing a monitoring software is thus possible without further customization. Every software running in a computer partition has access to all files of the computer partition, owned by the same user. Software running in a computer partition has no possibility to access or modify files owned by the superuser. As a general design rule, we refuse to grant any superuser privilege to applications or computer partitions. Only SlapGRID and Supervisord are executed with superuser privilege.

Computer Partition N

- Process(N, 0)
- Process(N, 1)
- ...
- Process(N, q)

A single computer partition is intended to host a single elementary application such as a database, an application server or a test runner. Yet, multiple UNIX processes maybe required for this purpose. If we consider the case of a Zope Web application server, two processes at least are allocated. One process for Apache acts as secure applicative firewall (mod_security + mod_ssl). Another process is the Zope application server itself. In the case of a database, one process is the database itself and another process is Stunnel application which maps IPv6 ports to local IPv4 ports.

The number of processes is even higher for applications. Running ERP5 requires no less than 12 processes: backend_apache, certificate_authority, conversion_server, crond, erp5_update, kumo_gateway, kumo_manager, kumo_server, memcached, mysql_update, mysqld, zope_1. In this case, the computer partition acts as a one place fits all containers for ERP5 and all its dependencies. A similar approach would be followed for any shrinked wrapped applications, including Apache/PHP/MySQL applications. This is acceptable since the concept of "elementary" still relates to the idea that only one instance of the application is launched and that, most of the time, is not used. Multiple computer partitions can thus be allocated on a single computer. However, this approach does not consider the possibility to scale up.

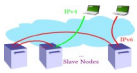
Some users even use a single computer partition to run multiple instances of the same application server. Computer partition is no longer elementary in this case. It acts as a mini-cluster and ends up consuming all resources of a computer. We are no longer in the original intention of elementary usage. This kills both the scalability of the application and the possibility to optimize resources in SlapOS through fine grained resource allocation.

SlapOS Networking

It is a design choice of SlapOS to consider that the only commonality between nodes of a distributed Cloud is IP and that there is no possibility to rely on network management services such as BGP to implement value added networking. SlapOS networking is thus based on flat IP addressing model. There is no notion of virtual local area network (VLAN) at the core of SlapOS. There is no notion of quality of service at the core of SlapOS. There is no encryption and no security at the core of SlapOS. It is the role of applications to implement such concepts by allocating appropriate resources and encapsulating them into insecure and unpredictable IP transit.

It would be an interesting research topic to discuss how to provide quality of service or virtual local area network management service on top of insecure and somehow unpredictable IP transit. We hope that someone will contribute to this research by implementing for example a complete Infrastructure as a Service (IaaS) stack on top of SlapOS with the idea to deploy over a collection of computers spread all over the world. This topic is however out of scope of SlapOS core design.

IPv6

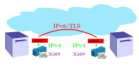


The use of IPv6 is recommended in order to create a global, distributed, peer-to-peer, unencrypted network of intercommunicating processes with a single, flat, addressing space. In an ideal SlapOS implementation, all software instances allocated on computer partitions of Slave Nodes can communicate with each other through IPv6 connections. Some users, represented on the drawing with a laptop, access SlapOS processes using IPv6 directly. This is the case of developers who need to access processes directly without a front end. Most legacy users however access SlapOS applications processes through IPv4 and application front-ends. Application front-ends are thus allocated both on IPv4 and IPv6 on special computer partitions with a dual IPv4 and IPv6 addressing.

The use of IPv6 is sometimes questioned by observers. For end users, IPv4 front ends provide access to the IPv6 backend. The use of IPv6 is thus transparent. On the other hand, any reasonable developer is able nowadays to setup an IPv6 tunnel using Miredo for example or to setup an IPv6 tunnel through tunnel brokers such as Hurricane Electric. Until now, we have been able to implement IPv6 access in about any condition: on mobile 3G connections, on home ADSL, in a university in China, etc. In the worst case, we simply connect through IPv4 and HTTP to a remote virtual machine hosted on SlapOS and accessible through a front-end. We then use that virtual machine instead of our local machine.

Yet, some large organisations refuse to implement IPv6. In this case, IPv6 can be replaced by IPv4 in SlapOS as long as a VPN is deployed to provide a global, flat addressing space with enough available addresses. It should be possible to allocate 100 IPv4 addresses on each SlapOS Slave Node. Distributed VPN technologies such as Tinc could eventually be integrated at the core of SlapOS to implement a margeIPv4 flat addressing space without sacrificing the key concept of distribution of resources which is at the core of SlapOS.

Stunnel: Security and Legacy



The main problem with IPv6 is that it is poorly supported by most applications. Another problem with

IPv6 is IPSEC. Although IPSEC is a beautiful technology, it is not easy to deploy it in a way which provides encryption and authentification on a per UNIX user base. It is also difficult to deploy in a completely decentralized way.

Stunnel provides a solution to both problems. Whenever a secure communication is needed between two applications, a Stunnel process is created at both ends. Stunnel maps local IPv4 addresses to global IPv6 addresses and encrypts all communication. Stunnel is also used to restrict access to a few X509 certificates.

Stunnel is used for example in SlapOS to connect to MySQL database servers hosted on public IPv6 servers. MySQL client itself only supports partly IPv6 and does not encrypt connections. With Stunnel, it is possible to access MySQL over IPv6 with encryption and possibly strong authentication. The same approach is used to access Memcached servers. Memcached was originally designed for trustable Local Area Network (LAN). By encapsulating Memcached protocol into Stunnel, we can get all IPv6 support, encryption and authentication.

Generally speaking, we found while implementing SlapOS that most software components which are used by large Web infrastructure such as social networks, SaaS or search engines are designed for trustable environments and private clusters. Porting those applications to distributed Clouds and untrustable networks requires an additional effort to make the connection secure. Rather than using a centralized VPN approach, we found that Stunnel could be used as a very efficient peer-to-peer VPN, and at the same time solve the IPv6 migration problem.

Stunnel itself provides enough performance compared to available IP network transit bandwidth. According to Stunnel authors, Stunnel performance on a Core 2 Duo architecture can reach up to 600 Mbit/s (http://www.Stunnel.org/?page=perf).

<table>
<tr><td rowspan="4">Data throughput</td><td>RC4-MD5</td><td>75MB/s</td><td>600Mbit/s</td></tr>
<tr><td>AES128-SHA</td><td>55MB/s</td><td>440Mbit/s</td></tr>
<tr><td>AES256-SHA</td><td>47MB/s</td><td>296Mbit/s</td></tr>
<tr><td>DES-CBC3-SHA</td><td>15.5MB/s</td><td>124Mbit/s</td></tr>
<tr><td rowspan="3">New connections</td><td rowspan="2">without session cache
(1024-bit RSA key)</td><td>Software</td><td>290 conn/s</td></tr>
<tr><td>Hardware</td><td>estimated* approx.
1 000 conn/s</td></tr>
<tr><td>with session cache</td><td colspan="2">2 150 conn/s</td></tr>
<tr><td rowspan="2">Max. concurrent sessions</td><td>Unix poll() / Win32</td><td colspan="2">over 10 000</td></tr>
<tr><td>Unix select()</td><td colspan="2">500</td></tr>
<tr><td>Virtual memory usage</td><td colspan="3">155KB/concurrent connection</td></tr>
</table>

Next

It is now time to discuss what to do next with SlapOS now that we have an idea of its architecture. As a general guideline, we recommend practical implementation and practical use. Experimentation is a great source of ideas, of innovation and helps further understanding the strength and weaknesses of any system, including SlapOS.

Where to go next?

- Install SlapOS Client
- Use KVM with SlapOS
- Use mysql with SlapOS
- Use kumofs with SlapOS
- Use ERP5 with SlapOS
- Install Slave Node

The first thing to do is to install SlapOS Client on a UNIX-like PC. This will help you understand better how Buildout technology works. It will also help you install the latest version of SlapOS whenever needed.

Next, you should try to allocate various software through SlapOS, either by using the Web user interface of VIFIB (www.slapos.org) or by using the SlapConsole command line interface. Try to allocate KVM virtual machines, MySQL relational database and KumoFS NoSQL database. Use Stunnel to connect to MySQL and KumoFS. Try ERP5 open source ERP, which even includes an open source Web office. And at the end try allocate from VIFIB a SlapOS Master node.

After doing all this, you will understand that SlapOS is very simple system based on a very simple architecture which disrupts the way you have envisaged IT until now, which can make you more productive and which can help you move to SaaS business in very short time.