Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters

Emmanuel Jeanvoine, Luc Sarzyniec, Lucas Nussbaum

Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters

Emmanuel Jeanvoine, Luc Sarzyniec, Lucas Nussbaum

AlGorille team (Inria Nancy Grand Est – LORIA)

Installing an operating system can be very tedious when it must be reproduced on several computers, for instance, on large scale clusters. Since it is not realistic to install the nodes independently, disk cloning or imaging with tools such as Clonezilla[1], Rocks [5], SystemImager [6] or xCAT [8] is a common approach. In that case the administrator must keep updated only one node (sometimes called *golden node*), that will be replicated to other nodes. This article presents Kadeploy3, a tool designed to perform operating system provisioning using disk imaging and cloning. Thanks to its efficiency, scalability, and reliability, it is particularly suited for large scale clusters.

1 A Reliable Deployment Process with Kadeploy3

Kadeploy3 belongs to the family of the *disk imaging and cloning* tools. It takes as input an archive containing the operating system to deploy, called an *environment*, and copies it on the target nodes. As a consequence, Kadeploy3 does not install an operating system following a classical installation procedure and the user has to provide an archive of the environment (as a *tarball*, for Linux environments).

Kadeploy3 does not directly take control of the nodes since it would require some specific and uncommon hardware support. Instead, it uses common network boot capabilities based on the PXE protocol [4] and it manages the associated PXE profiles.

Using such a mechanism, combined to the capability to dynamically update the PXE profiles of the nodes and with the capability to reboot the nodes in a reliable way thanks to out-of-band control interfaces (Baseboard Management Controller, Remote Supervisor Adapter) or the power distribution unit's capabilities, it is possible to take the control of the nodes and to specify what they are booting.

As shown in figure 1, a typical deployment with Kadeploy3 is composed of three major steps, called *macro steps*.

1. *Minimal environment setup*: the nodes reboot into a *trusted* minimal environment that contains all the tools required for the deployment (partitioning tools, archive management, ...) and the required partitioning is performed.
2. *Environment installation*: the environment is broadcast to all nodes and extracted on the disks. Some post-installations operations can also be performed.
3. *Reboot on the deployed environment.*

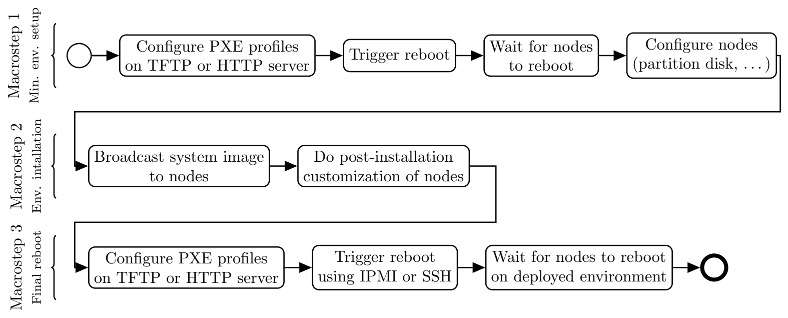
Macrostep 1
Min. env. setup
Configure PXE profiles on TFTP or HTTP server
Trigger reboot
Wait for nodes to reboot
Configure nodes (partition disk, ...)
Macrostep 2
Env. intallation
Broadcast system image to nodes
Do post-installation customization of nodes
Macrostep 3
Final reboot
Configure PXE profiles on TFTP or HTTP server
Trigger reboot using IPMI or SSH
Wait for nodes to reboot on deployed environment

Fig. 1. Kadeploy deployment process, composed of three macro-steps

Each *macro step* can be executed via several different mechanisms to optimize the deployment process depending upon required parameters and the specific infrastructure. For instance, the *Reboot on the deployed environment* step can perform a traditional reboot or it might instead rely on a call to `kexec(8)` for a shorter reboot.

Reconfiguring a set of nodes involves several low-level operations that can lead to failures for various reasons, e.g., temporary loss of network connectivity, reboot taking longer than planned, etc. Kadeploy3 reliability is achieved thanks to two ways. First the deployment process has a powerful error management and second, critical reboot operations required for the node control are based on reboot commands escalation in order to be able to take control of the nodes in any situation.

1.1 Reliability of the Deployment

Kadeploy3 is designed to detect failures as quickly as possible and improve deployment reliability by providing a *macro step* replay mechanism on the nodes of interest. To illustrate that, let's consider the last deployment *macro step* that aims at rebooting on the deployed environment. Kadeploy3 implements, among others, the following strategies:

1. directly load the kernel inside the deployed environment thanks to `kexec`;
2. perform a *hard* reboot using an out-of-band management hardware without checking the state of node.

Thus it is possible to describe strategies such as: try to launch the first strategy; then if some nodes fail, try to launch the second strategy several time if required.

Since all the steps involved in the deployment process rely on system calls (hard disk operations, network communications, specific hardware management), a special attention has been paid to the error handling. Thus, Kadeploy3 collects the result of every operation (exit status, `stdout`, `stderr`), even when it is performed on remote nodes. As a consequence, some steps can be replayed on nodes where a problem occurs..

Furthermore, some operations are likely to last too long (e.g., network boot, file-system creation, etc). Kadeploy3 also provides administrators with the capability to define specific timeouts for some operations in order to adapt the deployment process to the infrastructure. That allows to identify some problems quickly and to replay some operations on the related nodes.

1.2 Reliability of Reboot Operations

Since reboot operations are essential to control the cluster nodes, and ultimately the entire deployment process itself, they must behave correctly and reliably. Several methods can be used to reboot a nodes, for instance:

1. direct execution of the `/sbin/reboot` command;
2. use the capability of out-of-band management hardware with protocols such as IPMI. Various kinds of reboot can be executed: reset, power cycle, etc;
3. use the power management capability of the power distribution unit (PDU).

Performing a `/sbin/reboot` is the best solution with regards to speed and cleanliness. But it may not be an option if the target node is unreachable via in-band methods such as SSH (e.g., the node is already down, the OS has crashed, an unfriendly operating system is installed, etc). In this scenario, we would use IPMI-like features if available. Also, it might be better for speed to perform a reset rather than a power cycle since it bypasses the power on self test, but sometimes this is not sufficient. Finally, if onboard management hardware is unreachable, we may be required to use the capabilities of a remotely manageable PDU.

Kadeploy3 provides administrators with a way to specify several levels of commands in order to perform escalation if required. This allows them to perform highly reliable deployments if the clusters have the appropriate hardware. Unfortunately, depending on the methods chosen, reboot escalation comes at a cost and a balance must be struck between desired reliability and the time to deployment.

2 Scalability

In addition to have a reliable node control mechanism, deploying large scale clusters in a reasonable time require to be able to execute efficiently several commands and to send large files on a large number of nodes.

2.1 Parallel commands

The deployment workflow contains several operations that reduce to executing a command on a large set of nodes.

Thanks to SSH, one can execute commands remotely and retrieve their outputs. Launching SSH commands on a large number of nodes in sequence does not scale at all. Furthermore, launching all commands simultaneously can impose an extreme load on the server and can saturate all of its file descriptors.

Several tools have been built to overcome these limitations. For instance, Pdsh [3] and ClusterShell [2] are designed to execute SSH commands on many nodes in parallel. Both tools use windowed execution to limit the number of concurrent SSH commands and both also allow retrieval of command outputs on each node.

We choose to leverage TakTuk [9] as our mechanism for parallel command execution and reporting. TakTuk is based upon a model of hierarchical connection. This allows it to distribute the execution load on all the nodes in a tree and to perform commands with low latency. Using such a hierarchical mechanism would normally require the tool to be installed on all nodes. Fortunately, TakTuk includes a convenient auto-propagation feature that assures the tool's existence on all necessary nodes. The tool also uses an adaptive work-stealing algorithm to improve performance, even on heterogeneous infrastructures.

2.2 File broadcast

The broadcast of the system image to all nodes is a critical part of the deployment. In cluster environments where the most important network for applications is using Infiniband or Myrinet, the Ethernet network is often composed of a hierarchy of switches (e.g., one switch per rack) that is hard to leverage for a high-performance broadcast. File distribution to a large number of nodes via any sequential push or pull method is not scalable. Kadeploy3 provides system administrators with three scalable file distribution approaches during the *Environment installation macro step* to minimize deployment time.

With tree-based broadcast, a file is sent from the server to a subset of nodes, which in turn send the file to other subsets until all the nodes have received. The size of the subsets, called tree arity, can be specified in the configuration. A large arity can reduce the latency to reach all nodes but transfer times might increase because global bandwidth is equal to the bandwidth of a network link divided by the tree arity. The opposite effect occurs when the arity is small. In general, this broadcast method does not maximize bandwidth and should be used primarily for the distribution of small files. This method is also inefficient when used in hierarchical networks. We implement tree-based broadcast using TakTuk.

Chain-based broadcast facilitates the transfer of files with high bandwidth. A classical chain-based broadcast suffers from the establishment time of the chain in large scale clusters. Indeed, since each node must connect to the next node in the chain (usually via SSH), a sequential initialization would drastically increase the entire broadcast period. Thus we perform the initialization of the chain with a tree-based parallel command. This kind of broadcast is near-optimal in a hierarchical network if the chain is well ordered since, as shown in Figure 2, all the full-duplex network links can be saturated in both directions, and

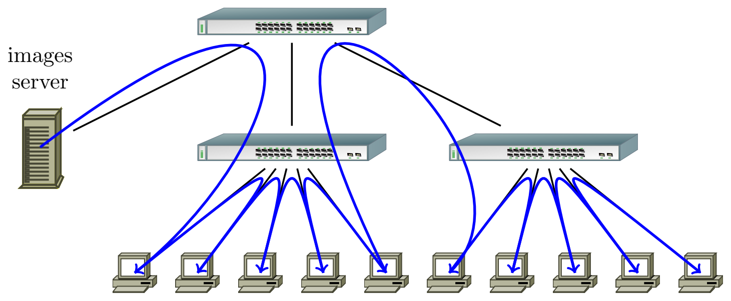

Fig. 2. Topology-aware chained broadcast. Data is pipelined between all nodes. When correctly ordered, this ensures that inter-switches links are only used once in both directions.

the performance bottleneck becomes the backplane bandwidth of the network switches. For this method, we implement chain initialization using TakTuk and perform transfers using other custom mechanisms.

BitTorrent-based broadcast is able to send files at large scale without making any assumptions about the quality of the network. Furthermore, BitTorrent is able to efficiently handle churn, an important property in large scale systems like petascale and future exascale clusters. Currently, our experiments show that there are two scenarios in which the performance of this broadcast method is inferior to the other methods. The first pathological case is one in which we are broadcasting on a small-scale cluster with a high-speed network, and the second is one in which we are broadcasting small files. In both cases, BitTorrent exhibits high latency and the overhead of the protocol dominates the time to broadcast. The large number of established connections between nodes induced by the protocol can lead to bottlenecks depending on the network topology.

In a default configuration, Kadeploy3 uses tree-based broadcast for the files used in the deployment process (e.g., disk partition map) and chain-based method for the environment broadcast which is usually a large file. However, this behavior can be modified in the configuration.

3 Other advanced features

In addition to being reliable and scalable, Kadeploy has many useful features.

Multi-cluster support

Kadeploy3 can be configured to manage several clusters at the same time, through a hierarchical set of YAML configuration files. In a Grid-like environment, it is also possible to initiate and control deployments on several Kadeploy servers from a unique Kadeploy client.

Hardware and software compatibility

Kadeploy3 does not generally rely on vendor-specific mechanisms. Vendor-specific remote control systems used to trigger nodes reboots can be easily used, even if they do not support the IPMI protocol.

Environments can be stored either as tarballs (for Linux environments) or as raw partitions, which enables the deployment of Windows or BSD-based systems.

Rights management and environments library

Kadeploy3 can be used to provide users with a Cloud-like experience with bare-metal system reservation. It can integrate with a cluster batch scheduler used to manage reservations, to delegate system deployment rights to specific users for the duration of a job. Kadeploy3 can also manage a set of environments and their visibility (public, private) in order to provide default environments, on which users can base their work to create and register custom environments.

Statistics collection

It is often hard to identify defunct nodes in a cluster, especially when failures are transient. Kadeploy3 integrates a statistics collection mechanism that enables the detection of nodes that often fail during the deployment process.

4 Performance Evaluation

4.1 Grid'5000 experimental testbed

Kadeploy3 has been used intensively on the Grid'5000 testbed (`http://www.grid5000.fr`) since the end of 2009 (and previous versions of Kadeploy were used since 2004). In that time, approximately 620 different users have performed 117,000 deployments. On average each deployment has involved 10.3 nodes. The largest deployment involved 496 nodes. To our knowledge, the deployed operating systems are mostly based on Linux (all flavors) with a sprinkling of FreeBSD. Although the Grid'5000 use case does not exercise all the goals targeted by Kadeploy3 (e.g., scalability), it shows the tool's adequacy with regard to most characteristics, such as reliability.

4.2 Curie Petascale Supercomputer

We had the opportunity to evaluate Kadeploy3 on the Curie[7] supercomputer owned by GENCI (`http://www.genci.fr/`) and operated by CEA (`http://www.cea.fr`). 2088 nodes were available to perform the test and the goal was to deploy the production environment. After a single administrative cycle, 2015 nodes where successfully deployed. This prove the efficiency and the reliability of Kadeploy3 in a large scale production infrastructure.

4.3 Virtual testbed

Validating scalability on large physical infrastructures can become complex (we only had access several hours to the Curie supercomputer since it is used for production purpose) because it requires privileged rights on many components (e.g., access to management cards, modification of PXE profiles, etc). Thus we chose to build our own large-scale virtual testbed on Grid'5000, leveraging important features such as link-layer isolation, and Kadeploy3 of course.

We performed an experiment where we used 635 physical nodes of the Grid'5000 testbed. Depending on the nodes capabilities, we launched a variable number of KVM virtual machines. In total, 3,999 virtual machines were launched and participated in a single virtual network (despite the physical nodes where located on four different geographical sites). Then we installed all the required servers: DHCP, TFTP, MySQL, HTTP server, Kadeploy3. Once the testbed was launched, we were able to perform deployments within a single cluster of 3,999 nodes. During the largest run, a 430 MB environment was installed on 3,838 virtual machines in less than an hour. 161 virtual nodes were lost due to network or KVM issues. A significant amount of time was also wasted because of the high latency between distant geographical sites (10-20 ms), which affected some infrastructure services like DHCP and the PXE protocol.

5 Wrapping-up

We think that Kadeploy3 can help system administrators of large scale clusters to save a precious time by reducing the OS provisioning time. The best way to be convinced is to have a try. Kadeploy3 is free software (CeCill 2 license) written in Ruby and available from `http://kadeploy3.gforge.inria.fr/`. Source code can be downloaded as well as Debian and RPM packages. Kadeploy3 is configured thanks to few YAML files and to help administrators, a complete guide describes the entire installation and configuration steps.

References

1. Clonezilla. `http://clonezilla.org`.
2. ClusterShell. `http://cea-hpc.github.com/clustershell`.
3. Parallel Distributed Shell. `http://sourceforge.net/projects/pdsh`.
4. Preboot Execution Environment (PXE) Specification. `http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf`.
5. Rocks: Open-Source toolkit for real and virtual clusters. `http://www.rocksclusters.org`.
6. SystemImager. `http://systemimager.org`.
7. The Curie supercomputer. `http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm`.
8. xCAT: Extreme Cloud Administration Toolkit. `http://xcat.sourceforge.net`.
9. Benoit Claudel, Guillaume Huard, and Olivier Richard. TakTuk, adaptive deployment of remote executions. In *International Symposium on High Performance Distributed Computing (HPDC)*, pages 91–100, 2009.