OS Installation Redesign: One Year Later

Michele Tartara - Ganeti Core Team - Google
GanetiCon 2014 - September 2, 2014

One year ago, Athens: GanetiCon 2013

Design sessions

- Many productive design sessions
- An agreement reached on how to proceed on all topics

- except: How to redesign the instance OS installation

Why a redesign?

Previous state

- OS definition: set of scripts
  - One for each operation
  - `create`, `import`, `export`, ...
- Executed as **root** on the host node
  - When one of those operations is invoked

Issues of the old design

- Fine in a 100% trusted environment, but:
- scripts MUST be trusted
- everything they access MUST be trusted
- No (automatic) support for personalization
- No (automatic) support for disk images
- No user-defined VM images
  - A crafted disk image might compromise the host kernel

Last year's design session

Why was there no agreement?

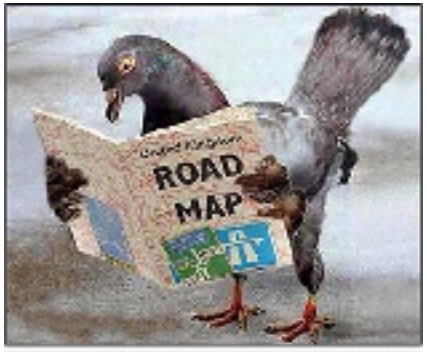
- Scripts with root powers are too powerful and flexible
- They had been around for a long time
- Everybody had developed its own workflow
  - And hoped to change it as little as possible
  - But they were really different from each other
- A one hour design session was not enough to unravel the issue

A longer design journey

The following months

- Lots of communications with the interested internal and open source community parties
- Over different channels
  - ganeti-devel mailing list
  - Private emails
  - Videochats
  - Meetings at conferences somewhere in the world
  - Homing pidgeons

trying to define the new design and the roadmap for its implementation



User requests (I)

- Secret parameters
  - Pass parameters to the OS installation scripts that are not stored in the configuration file or in the logs.
  - Useful for:
    - Private keys
    - Username/passwords

User requests (II)

- Image-based install
  - No need for OS scripts
  - Take an image and dump it AS IS on the 1st disk
  - User provided images are a security issue
    - even a non-root mount might lead to a compromised system

User requests (III)

- Personalization of image-based install
- Image-based install is quick and convenient
- Frequently instances should differ a bit
  - Users
  - IPs
  - Configuration
- Scripts have OS parameters
- What about images?
- Adding/modifying files should be possible

User requests (IV)

- Root scripts in a safe environment
  - Root scripts are insecure... but (sometimes) powerful and useful!
  - Untrusted scripts are currently not possible
  - Plenty of power and flexibility
  - What about keeping them, but in a more secure way?
  - Run inside a VM
    - If compromised, the node is still safe
    - Which VM?
      - Resources might be scarce...

User requests (V)

- Communication with the instance
  - An OS definition might need:
    - A trusted part (accessing sensitive information)
    - An untrusted part (dealing with user data)
  - One can run in the host, one in the VM performing the install
  - They need to communicate
  - In general: need for a bidirectional communication channel
  - Other uses are possible too

User requests (VI)

- Temporary installation state
  - During the installation, different parameters needed
  - Installation media, different boot order, extra NICs
  - Have a special set of parameters
    - Stored with the config
    - Used only for (re)install

User requests (VII)

- Installation metadata
  - An image-based install:
    - receive metadata in a standard location during its first start-up
    - self-configure the instance based on it

Design decisions and implementation status

OS Parameters

- Divided in 3 categories
  - `Public`: traditional behaviour. Logged and stored freely.
  - `Private`: parameter saved in the config, not shown in logs
  - `Secret`: parameter not saved. Reinstall impossible unless it is manually specified again. Not shown in logs, not serialized to disk.
- Implementation status
  - Implemented
  - Parameters are still serialized to disk
    - Coming soon: blocked on MasterD/LuxiD refactoring
  - Beware: if log level is set to DEBUG, some parameters might occasionally still appear. They are completely hidden only at lower levels.

Image-based install

- `gnt-instance add --os-parameters os-image=install_disk.img inst1`
- The image is dumped on the first disk of the instance
- Not only local files: URLs too!
- OS scripts can be executed too, afterwards, for further personalization

Personalization package

- Image-based install don't need to be identical
- Configs, users, private keys can be injected
- Personalization package
  - Compressed archive
  - Contains same structure of the filesystem
  - Extracted on `/` of the first disk
    - From a safe environment (no risks while mounting the image)

Communication mechanism (I)

- Based on a link-local network interface (last interface, added after the user-defined ones)
- Host-instance only communication. No cross-instance communication.
- Mainly meant for install. Can be used also during normal runtime if needed.
- TAP interfaces created by ganeti: `gnt.com.%d` (`%d` is a host-unique number)
- The host listens on `169.254.169.254/32`
- Static point-to-point, interface-based route set on the host
- No iptables magic
  - Common, so unconfigured instances always know where to look for it.

Communication mechanism (II)

- Interface name (NOT IP!!) distinguishes the connection.
- Guest side
  - Host-unique MAC address
  - Host-unique IP address, provided by DHCPon the host
    - DHCP dynamically configured by Ganeti
    - dnsmasq is preconfigured. Scripts can be adapted for any other server
    - Only listens to network interfaces/MAC addresses configured by Ganeti
    - Virtually no conflict with other DHCP servers

Communication mechanism (III)

Why using network interfaces?

- Network support
  - Ubiquitous
  - OS independent
  - Hypervisor independent
  - Unconfigured network interfaces are common, no side effects
- VirtIO, XenBus
  - hypervisor dependent
  - might require (para)virtualization drivers
- Virtual floppy, virtual USB device
  - Detected and configured automatically by the guest OS
  - Frequently in prominent positions of the UI
  - Might lead to unexpected behaviour for the user if communication is active outside install-time

Communications mechanism (IV)

- Implementation status
  - Done
  - Allows host-instance network connections on any port
  - Firewalling is up to the user if required
    - Activated by
      - Enabling the communication network: `gnt-cluster modify --instance-communication-network=net1`
      - Enabling communication for a specific instance: `gnt-instance add/modify --communication=yes|no`

Metadata service (I)

- A new daemon provides metadata about the instance
  - Static IP address
  - A ganeti specific pipe-like communication channel that the scripts can use to communicate
  - OS scripts parameters
- The instance can use it to self-configure itself
- Implementation status
  - Partial
  - Daemon listening on `169.254.169.254:8080`: Done

Metadata service (II)

- Implementation status
  - Daemon implemented
  - Implemented endpoints
    - `/os/os-install-package`: an archive containing OS scripts and related files
    - `/os/parameters.json`: the OS parameters to be accessed by the scripts
    - `/os/package`: personalization package
  - More to come
    - `/read`, `/write`: endpoints of the communication pipe
    - `/os/meta_data.json`: arbitrary metadata to be received and used by the instance

Helper VM

- Running root scripts on the host is dangerous
- Root powers are usually required to mount images/prepare VMs
- Solution: run them in a VM
- Helper VM
  - User-provided
  - Ganeti will have a default script for creating one
- Ganeti communicates with the helper VM
- Instructs it on what to do, how to install/modify instances

Helper VM

Where does it run?

- With its own resources?
  - Take them into account in all tools
  - Have them!
  - Only one running at a time: possible long waits
  - ...or multiple copies, huge waste of resources!
  - :-(
- In the domain/with the resources of the instance it is working on
  - Resources already accounted for
  - Surely more than enough
  - When the helper VM runs, the instance is down anyway
  - :-)

Helper VM

What does it do?

- Runs the untrusted OS scripts (currently, only create)
- Will extract the personalization package on top of the installation image
  - If both are specified, and there are no OS scripts
- Other future uses
  - Zeroing out unused disk space before an instance move to make it quicker
  - Other unsafe operations?

Trusted VS untrusted scripts

- New category of OS install scripts introduced
- **Untrusted** scripts
  - Functionally equivalent to the trusted ones
  - Run inside the helper VM
  - Can use the communication mechanism to interact with the trusted ones
    - Over sockets
    - Over the metadata `/read` and `/write` (TODO)

Acknowledgements

- Thanks to everybody that helped with suggestions, requisites, use cases
- Thanks to Jose for finishing and detailing the design and for the implementation

Thank You!

Questions?