Ganeti Enterprise Virtualisation

Iustin Pop

Google Switzerland

March 2, 2012

Outline

Ganeti overview
- Basics
- Architecture

Deployment scenarios
- Generic scenarios
- Google usage

Cluster operations
- Internal operations
- Google-specific workflows

Overview

I will talk about. . .

- what Ganeti is
- deployment scenarios
- usage in Google:
  - in the corporate (internal) infrastructure
  - this is *not* used for user-facing products (search, gmail, . . .)
- software used, tools and infrastructure

Terminology I might use (accidentally):

node physical machine (Xen `dom0`, KVM host)

instance virtual machine (Xen `domU`, KVM guest)

Outline

Ganeti overview

- Basics
- Architecture

Deployment scenarios

- Generic scenarios
- Google usage

Cluster operations

- Internal operations
- Google-specific workflows

Basics

- virtualisation management
- at cluster, not individual machine level
- designed for internal as opposed to external usage

Concepts

Main idea is to abstract low-level details:

- the hypervisor being used
- details of resource allocation
- VM allocation

And instead interact with the cluster as an entity, instead of the individual machines.

Stable features

- virt: supports Xen and KVM
- storage: file-based storage, LVM, DRBD
- manual failover of VMs in case of machine failures
- cluster sizes of 1 to few hundred of physical machines
- bridged and routed network configurations
- guest OSes are installed via “OS definitions”

Experimental features

- LXC as light-weight virtualisation
- shared storage and RBD
- use as machine management layer (e.g. integration with IPMI)

Outline

Ganeti overview

- Basics
- Architecture

Deployment scenarios

- Generic scenarios
- Google usage

Cluster operations

- Internal operations
- Google-specific workflows

Cluster architecture

- centralised model: one node is designated the master node and controls all other nodes
- a number of daemons running on the nodes, depending on node role:
  - master
  - master candidate
  - regular
- two methods of controlling the cluster: CLI and HTTP

Regular nodes

- every physical machine runs the *node daemon*
- this daemon talks to the hypervisor, storage, etc.
- receives commands from the master daemon via cluster RPC
- the *ganeti watcher* should be run from `cron` on each machine for maintenance
- both the watcher and the node daemon need root privileges

Master node/master candidates

- one machine is designated as the *master node*
- it runs the *master daemon*, which is used to control the cluster
- also runs the *RAPI daemon*, which offers a remote, HTTP/REST based API for interacting with the cluster
- the master role can be manually changed to any other machine which has been designated as *master candidate*
  - if so configured, all master candidates run the *confd daemon*, for fast and scalable querying of the cluster configuration
  - this daemon is required for 'routed' network configurations
- none of these daemons need root-level privileges

Interacting with the cluster

- two options: command-line and HTTP-based API
- the command line interface is the “main” method
    - can control all aspects of the cluster
    - offers additional “niceties” for working with the cluster
- *RAPI* is an HTTP/REST based API that offers full control over virtual machine state but only partial over cluster state
    - cannot be used to create/destroy a cluster
    - cannot be used to join a new node to a cluster
    - has a simplistic permission model (not for end-users!)
    - used by web-frontends
- on-going work to bring RAPI to feature-parity with CLI

Security model

- the *node daemon* is a trusted entity
- the *master daemon* controls the node daemons, but cannot be used for “taking over” nodes (i.e. running arbitrary commands)
  - in case of losing the cluster-internal SSL certificate or subversion of the master daemon, the nodes themselves *should* still be safe
  - but VMs could be removed, reinstalled, etc.
- the *rapi daemon* itself controls the master daemon, but cannot control all aspects of cluster state
- ideal goal would be to have Ganeti control VMs, but not have access to their data
  - limit permissions on the node daemon
  - limit instance export destinations

Outline

Ganeti overview

- Basics
- Architecture

Deployment scenarios

- Generic scenarios
- Google usage

Cluster operations

- Internal operations
- Google-specific workflows

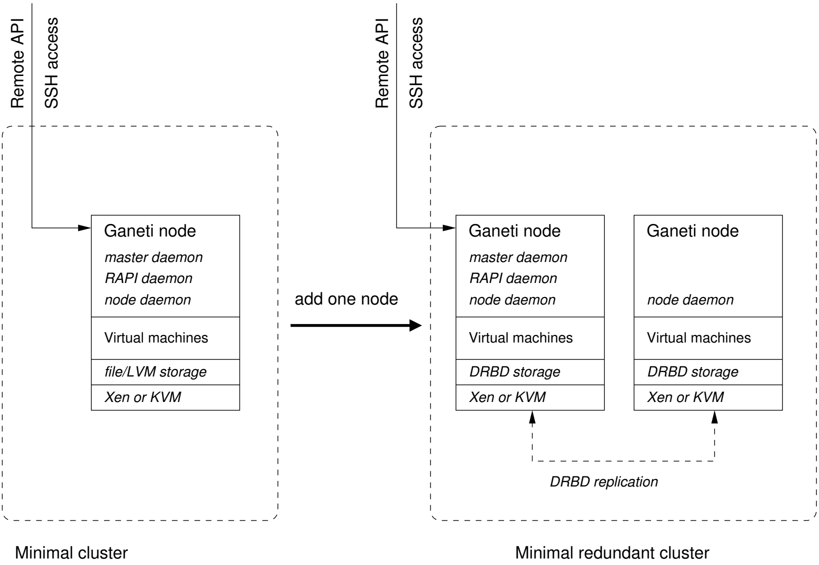
Minimal cluster

- can run on a single machine
- and be extended later to many more
- even on a single machine, we still have all daemons
    - except for *ganeti-confd*, used in big deployments
    - the RAPI daemon is not optional, even though it could be
- file or LVM-based storage
- can fully utilise hardware resources

Minimal redundant cluster

- needs two machines for DRBD or shared storage
- can then live-migrate instances between the nodes
- or manual fail-over after one node has failed
- due to redundancy requirements, not all memory can be used

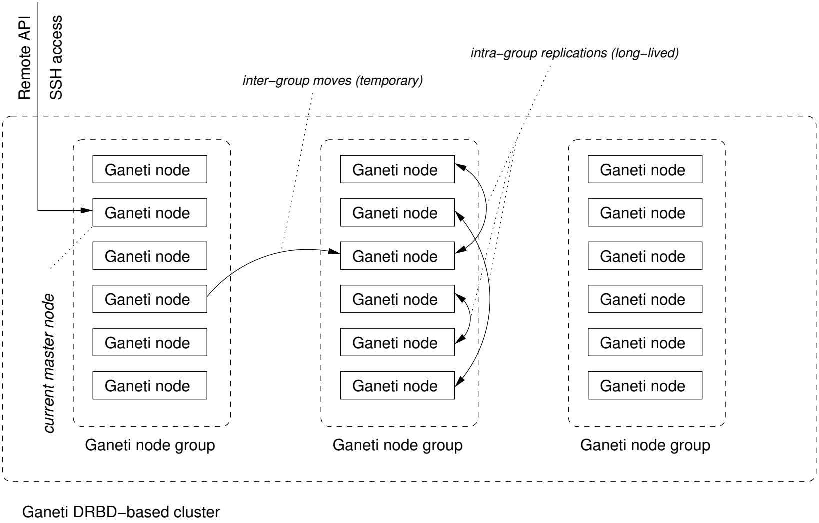
Small cluster diagram



Large clusters

- current version works up to a few hundred nodes
- there are scalability issues after 100 nodes
    - these do not “break” the cluster, but reduce parallelism or can impede operations in certain failure cases
    - on-going work to improve this
- nodes should be grouped in *node groups*
    - concept used to limit the scope of cluster-internal locks and increase parallelism
    - for example, make each rack of machines a node group
    - the node group is also the default mobility domain for instances
- efficiency varies, usually around 75-80% of memory
    - the rest of the memory is reserved for redundancy

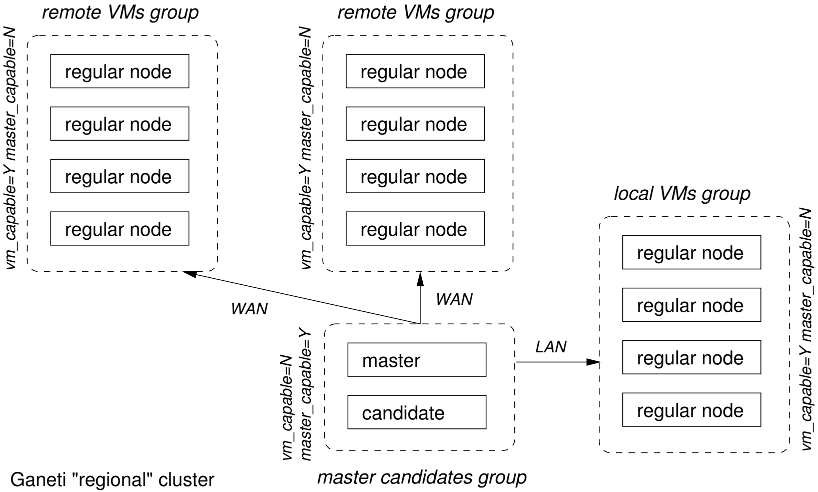
Large cluster diagram



Special configurations

- beside the discussed node roles, nodes have two extra flags:
  - *vm-capable*
  - *master-capable*
- these allow “dedicating” nodes to either running VMs or just for controlling the cluster
- since the replication/mobility domain is inside the node group, the master node can be run remotely or as a VM itself

Regional cluster diagram



Outline

Ganeti overview
- Basics
- Architecture

Deployment scenarios
- Generic scenarios
- Google usage

Cluster operations
- Internal operations
- Google-specific workflows

Google usage

- comprises servers located in offices
  - support local office infrastructure
  - latency-sensitive services (e.g. DNS, caches)
  - (very) small numbers of machines
  - spread across many offices
  - ‘small’ cluster model
- and servers located in datacenters
  - various purposes
  - just a few datacenters
  - but many machines per datacenter
  - ‘large’ cluster model

Technology

- we virtualise mostly Linux servers
- we deploy Xen. . .
    - on standard (off-the-shelf) x86 hardware (`amd64`)
    - on top of a standard Linux distribution (Debian)
    - in paravirtualised mode
- no SAN/NAS: just DRBD & LVM storage
- using both open-source components (Ganeti itself) and internal tools

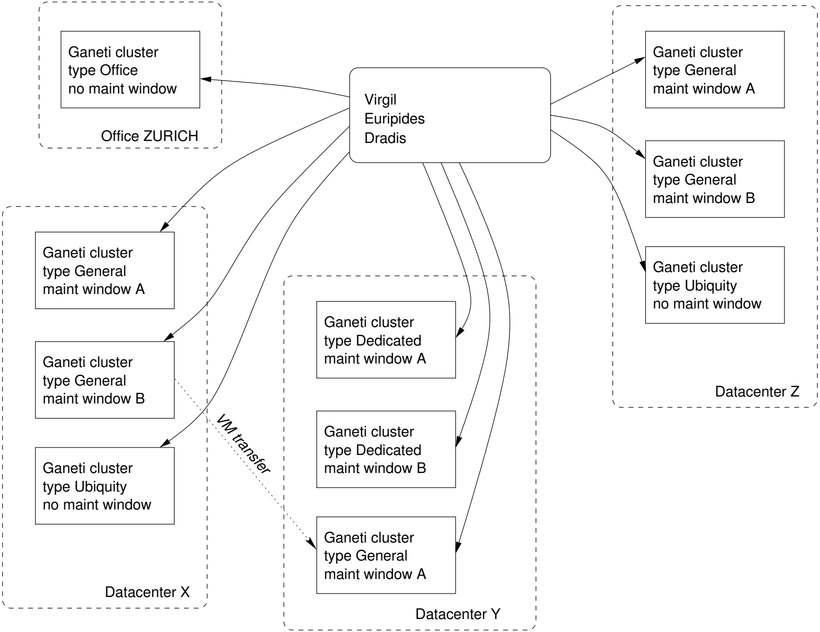
Internal components

- Ganeti manages machines in a cluster
- what manages a fleet of clusters?
- this is done via internal software (not open source)
    - both generic:
        - monitoring
        - machine database
    - and specific to Ganeti-in-Google:
        - web interface to the clusters (code name *Virgil*)
        - cluster-level configuration management *(Dradis)*
        - machine (repair) workflow manager *(Euripides)*
    - these are related to hardware work-flows, not virtualisation
- the generic components have open-source alternatives
    - "Ganeti Web Manager" is an open source web console
- no known equivalents for Dradis and Euripides
- large deployments will likely need to reimplement them

Fleet organisation

- clusters are categorised according to customer type
- clusters of the same type and in the same region are split in two “maintenance windows”
    - allow for maintenance work on only half of the clusters in a region
    - compensates for the fact that the cluster is a single point of failure for a given VM
- Virgil talks to all the clusters and provides fleet overview
- such meta-concepts are implemented in Ganeti via *cluster tags*
- tags are used for other tasks that cannot be expressed natively in Ganeti

Fleet diagram



Other internal tools

- machine history console
  - displays physical machine history
  - ties into monitoring, hardware repairs process, life-cycle, etc.
- `rolling-reboot` tool
  - allows rebooting an entire cluster without VM impact
  - uses live migration and sequential reboots
- `ganeti-capacity`: a capacity planning tool
  - computes simulated cluster capacity
  - VM specs versus physical resources, space, power
  - will be open sourced, not related to internal systems
- and many other small tools
  - notification of owners per cluster/physical machine
  - monitoring and resource dashboards
  - etc.

Outline

Ganeti overview

Basics

Architecture

Deployment scenarios

Generic scenarios

Google usage

Cluster operations

Internal operations

Google-specific workflows

Internal cluster operations

- *htools* component shipped with Ganeti can
  - balance the cluster
  - compute cluster capacity
  - compute node evacuation strategy
  - do automatic selection of nodes for VM placement
- plugin versus API
  - node evacuation and instance placement use internal "IAllocator" plugin framework
  - the others are command line tools that talk to Ganeti via its external APIs
- the documentation explains how to use all of these
- downside: not working yet with the shared storage backends

Instance placement

- given a new instance with given memory, disk, CPU size/count, where to place it in the cluster?
- there are both “hard” constraints that limit the placement:
  - instance might not share same node with others
  - some nodes might not be ‘open’ for allocation
  - some nodes might have exhausted ‘soft’ resources
- and “cluster metrics” that guide the placement:
  - equalised usage of memory/disk/CPUs
- all of these are condensed into a cluster score
- placement tries to minimise the score

Node evacuation

- where to relocate instances from a failed node?
- this is similar to “re-allocating” these instances from scratch, as if they weren’t already on the cluster
- however, they already exist and were running before the failure
- thus some soft constraints can be violated for this case:
  - we can ‘dip’ into the memory pool reserved for redundancy
  - we can temporarily oversubscribe other nodes

Cluster balancing

- cluster operations can result in nodes having different usage
  - e.g. instance resize/removal, node addition, etc.
- the cluster balancer (`hbal`) can optimise the cluster layout
  - same base algorithm as describe previously
  - hence "better" means an overall lower cluster score
- it generates (and executes) a set of Ganeti commands
- operation can be tweaked in multiple ways, e.g.:
  - no live migration, or only live migration
  - no changes to some instances
  - stop at a certain score, or after a given number of steps

Outline

Ganeti overview

- Basics
- Architecture

Deployment scenarios

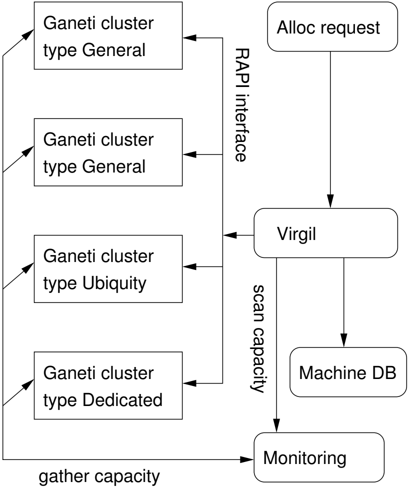
- Generic scenarios
- Google usage

Cluster operations

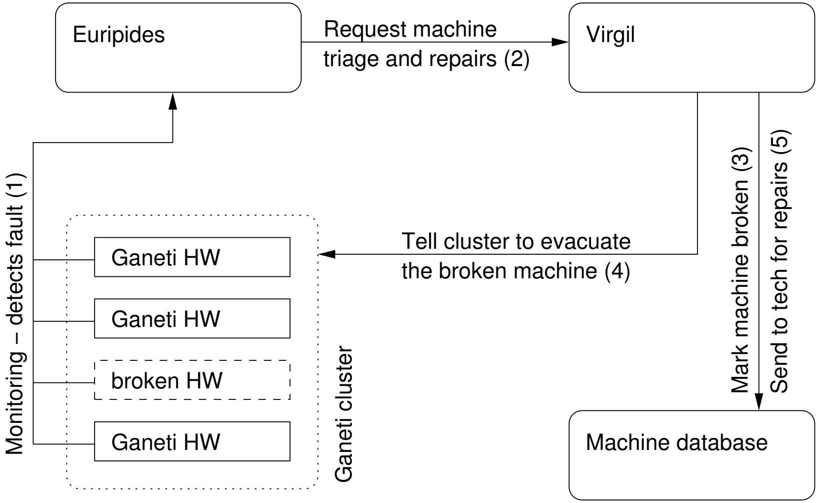
- Internal operations
- Google-specific workflows

VM allocation

1. Virgil gets an allocation request (region, cluster type)
2. creates machine record (DNS, other systems)
3. selects "best" cluster based on VM spec, capacity data
4. and tells it to create the VM
5. cluster selects best physical machine(s) to host the VM
6. VM is created, and OS installation scripts are run
   - install software
   - configure authentication


Ganeti cluster type General
Ganeti cluster type General
Ganeti cluster type Ubiquity
Ganeti cluster type Dedicated
RAPI interface
Alloc request
Virgil
scan capacity
Machine DB
Monitoring
gather capacity

Handling machine failures


Euripides

Request machine triage and repairs (2)

Virgil

Monitoring – detects fault (1)

Ganeti HW

Ganeti HW

broken HW

Ganeti HW

Ganeti cluster

Tell cluster to evacuate the broken machine (4)

Mark machine broken (3)

Send to tech for repairs (5)

Machine database

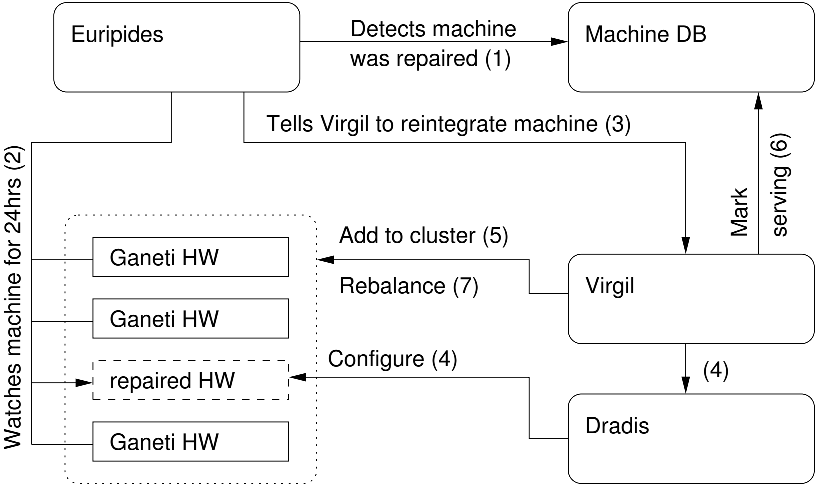
Handling machine failures

1. monitoring detects a HW problem (e.g. disk error, memory problem, etc.)
2. Euripides (for non-critical problems) tells Virgil a machine needs to be sent to repairs
    - for critical events (machine dead), on-call person is paged, instructs Euripides how to proceed
3. Virgil first marks the machine as “not in production”
4. then tells the cluster to evacuate the VMs from it
5. finally requests repairs by local tech

Note

- for “known” errors, the process if fully automated
- otherwise, an “exception” case is created for investigation

Handling repaired/new machines



Handling repaired/new machines

1. Euripides detects new or repaired machine in Machine DB
2. at first, it's being kept "under watch" for a period of time
3. it tells Virgil to integrate new machine
4. Virgil calls Dradis to configure the machine appropriately
5. Virgil tells the cluster to add the new machine
6. finally the new machine is marked as serving
7. the cluster will be rebalanced in order to utilise the machine

Note

- assuming no errors in the OS installation, configuration, etc., the process is fully automated

Questions?

Thanks!

Links

Ganeti homepage `http://code.google.com/p/ganeti`

Code repositories `http://git.ganeti.org/`

Documentation `http://docs.ganeti.org/ganeti/current/html/`

Ganeti Web Manager `http://code.osuosl.org/projects/ganeti-webmgr`

Image-based OS template `http://code.osuosl.org/projects/ganeti-image`

Presentation on virtual workstations `http://neatx.googlecode.com/files/herding-virtual-workstations-fisl-2009.pdf`