Saturnring Guide

v0.51

Sachin Agarwal

6/23/2015

Saturnring network iSCSI block storage management system architecture and user guide. Licensed under the Apache 2.0 open source license.

CONTENTS

LIST OF FIGURES

THE MILE HIGH VIEW

INTRODUCTION

Saturnring is a scalable network block storage system. It builds a management layer on top of the iSCSI network block storage protocol in order to orchestrate multiple iSCSI servers that export block devices over a high-speed network to multiple client computers and servers. Think of it as a cost-effective software-defined SAN built with commodity servers, networking and storage-media and open-source software.

A use-case may be a few physical servers with multiple high-capacity SSD drives that export different-sized slices of these drives to 100s or even 1000s of VMs in an infrastructure-as-a-service (IaaS) cloud using the iSCSI block storage protocol. Saturnring lets administrators and users manage their iSCSI block devices via a RESTful API and a web portal.

The use case described above is becoming more common as VM users demand SSD-class IO performance. One way to meet this requirement is to install solid state disks in hypervisors (servers on which VMs are running) and make the disk available to a VM. There are two operational problems with this approach. First, a VM is tied to that physical hypervisor; if there is a hardware malfunction then the data on the solid state disk will not move to another healthy hypervisor even if the VM was re-instantiated there. Second, there is no option to expand a VM's storage capacity if needed inserting another physical SSD disk into the hypervisor and provisioning it in the VM. Third, the hypervisor's SSD disk is either entirely assigned to 1 VM, or the hypervisor administrator has to divide up the solid state disk into multiple block devices for each VM needing solid state disk in that hypervisor. Hundreds or thousands of hypervisors will make this task hard, non-scalable, and error-prone.

Saturnring makes solid state disk is available via low latency iSCSI connections to VMs that need it. It provides the ability to automate the iSCSI lifecycle – provisioning, management, and deletion of iSCSI target LUNs – across multiple block devices on multiple storage servers, and so the ability to drive up solid state utilization, the freedom move VMs across different hypervisors, the flexibility to horizontally scale "solid state" storage using inexpensive hardware and a pre-existing network, and the choice of selecting any suitable solid state disk (depending on e.g. the SSD write wear of the applications).

This is not to say that Saturnring is not suited to manage spinning-disk or other storage media. In its current form, Saturnring can manage iSCSI targets irrespective of the underlying storage media. In fact it also has the ability to manage heterogeneous media types that can be chosen at the time of provisioning. In another manifestation, Saturnring can be used to orchestrate Amazon AWS or Google cloud SSD-VMs to create a network block storage cluster for other compute VMs.

DESIGN PRINCIPLE

The key assumption in Saturnring's design is that clients of the iSCSI storage – services consuming the storage – implement data replication via application logic. The backend iSCSI servers do not replicate data in the cluster. They are built to be independent of each other. This assumption means that individual iSCSI servers can fail, their storage may become offline or be irretrievably lost. Relaxing this condition allows commodity hardware to be used as storage backend. For example, an Amazon AWS VM with local SSD disk can become an iSCSI server. Saturnring and applications using the non-replicated backend iSCSI storage assume that multiple iSCSI servers do not error at the same time. Saturnring provides algorithmic mechanisms to make iSCSI targets members of a particular anti-affinity group – the Saturnring storage provisioner will strive to spread out the anti-affinity VMs across as many iSCSI servers as available.

The risk may seem overwhelming but it is effectively mitigated by the sophisticated application-level replication in many cloud-based technologies which are primary use cases for Saturnring storage – for example - Nosql databases such as Elasticsearch, Cassandra or MongoDB. Amazon AWS or similar cloud offerings recommend using multiple availability zones to mitigate risk. So, it may make more financial sense to keep 1-copy of the data in each availability zone rather than 2 or more copies in the same availability zone. Solutions like Ceph recommend having at least 2 replicas connected via low-latency links to ensure safety of the entire storage cluster's data (since data is striped across all storage servers). Saturnring's storage servers are independent of each other and data is not striped across the servers so the failure of a storage server will only make the data in that server unavailable. In addition, Saturnring does not need a high-speed replication network to preserve the low-latency characteristics of SSD storage. A write is acknowledged as soon as it is persisted on the iSCSI storage server and no backend copying is needed.

Traditionally iSCSI has been associated with SAN networks with expensive network interconnects (e.g. Infiniband) for low latency operation. However with the commoditization of 10Gbit Ethernet and the availability of mature open-source iSCSI implementations, it is now possible to build low latency iSCSI-over-Ethernet storage solutions.

Saturnring forces the application developers to think in the "cloud way". The business case for cross-containment HA architecture becomes apparent. This architecture is absolutely necessary in any private or public cloud IaaS application deployment given the multiple points of failure – network switches, hypervisors, storage, etc. Saturnring leverages the cross-containment application HA design principle to reduce the number of data replicas to as little as possible, significantly driving down costs.

KEY CONSIDERATIONS

Saturnring brings iSCSI -- a well-understood, proven and debugged veteran network storage technology -- to the modern cloud-based world. Here are some technical and non-technical features:

1. Users provision and manage their own storage (no storage admin is required): iSCSI sprawl is automatically managed and user quotas enable multi-tenant low overhead processes. Features like LDAP/AD integration make Saturnring easy to roll out.

2. All provisioning is through a RESTful API; this makes Saturnring easy to integrate with cloud management software. E.g. Openstack or Open Nebula.

3. Saturnring relies on SCST – a proven open-source iSCSI server solution – for the iSCSI data plane.

4. Saturnring makes no assumption on the underlying block device it will manage – in fact it is agnostic to which block device is being used underneath the LVM setup it controls on the iSCSI server. This makes it possible to deploy block storage that exactly meets the needs rather than be forced to buy a certain underlying SSD or disk device. Saturnring does have the ability to shepherd provisioning requests according to the block storage device specified in the request.

5. Multiple network interface iSCSI portals are supported – this is useful when say, clients require VLAN support for their storage service.

6. All software components are open-source. All hardware components can be chosen independently (subject to technical requirements). This flexibility drives costs down but also comes with great responsibility because hardware choices will affect the reliability down the road. Saturnring requires a certain degree of proficiency in Linux and storage concepts as will be clear later in this document, so while software and hardware are cheap, the administrator costs should be adequately understood.

TERMINOLOGY

- ➢ **CRON** is a time-based job-scheduler in Linux (and other Unix-like operating systems). It is used in Saturnring to trigger periodic polling of iSCSI servers' status.

- ➢ **Dmcrypt** is a disk encryption subsystem and part of the device mapper infrastructure. It is used to encrypt logical volumes that are the backing stores of iSCSI targets. Data stored in the logical volume is secured in this way.

- ➢ **Git** is a revision control system. It is used in Saturnring to keep a version history of LVM, SCST and dmcrypt configuration files, as well as to keep SSH keys under version control. This proves useful while looking at the history of changes and even reverting configurations.
  Internet Small Computer System Interface. It can be used to implement network block storage through its ability to send SCSI storage commands and data over a network.

- ➢ **iSCSI**
  - o iSCSI client – Computer that consumes iSCSI storage via an iSCSI block device, created when the iSCSI client connects to the iSCSI server and requests the iSCSI target.
  - o iSCSI initiator - Unique identifier of an iSCSI client. In Saturn's current iSCSI implementation targets and initiators have a 1-to-1 relationship for access control and to prevent multiple VMs from accessing the same iSCSI-served block device.
  - o iSCSI server - Physical server containing storage media where the iSCSI LUN actually lives. Users "log in" to the iSCSI "portal" hosted on the iSCSI server via the iSCSI protocol and access their iSCSI target(s).
  - o iSCSI target - Unique identifier to identify and "connect" to the iSCSI LUN on an iSCSI server.

- ➢ **LVM** is a logical volume manager for the Linux kernel that manages disk drives and similar mass-storage devices. Saturnring uses the more recent LVM2 software to manage the storage media and slice it based on provisioning requirements. For more information about LVM, volume groups (VGs), logical volumes (LVs), physical volumes (PVs) and thin provisioning using LVM, visit the excellent Linux documentation project entry for LVM (http://tldp.org/HOWTO/LVM-HOWTO/).

- ➢ **Redis** is a key-value cache and store. In the context of Saturnring it is used in for Redis queues.

- ➢ **SCST** is the generic SCSI target subsystem for Linux that creates iSCSI targets from underlying block devices (e.g. LVM logical volumes) that can be exported over a network. Saturnring uses SCST as its iSCSI server.

- ➢ **Supervisord** Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems. Supervisord is used in Saturnring to manage redis queue worker processes.

- ➢ **Saturnring Cluster** The set of all the iSCSI servers controlled by Saturnring

SATURNRING ARCHITECTURE

Keywords to read-up on the Internet: SCST, iSCSI, LVM, Redis, Supervisord, Django web framework

Saturnring orchestrates iSCSI block devices across multiple iSCSI servers. It manages their complete lifecycle from creation to deletion and makes intelligent choices over their placement among the iSCSI servers. It is important to note that the iSCSI data plane – is implemented via the ready-made industrial strength SCST subsystem. Moreover the control plane is designed to be independent of the data-plane, meaning that *pre-existing iSCSI targets will continue to exist and function normally even if the entire Saturnring provisioning infrastructure is disabled.*

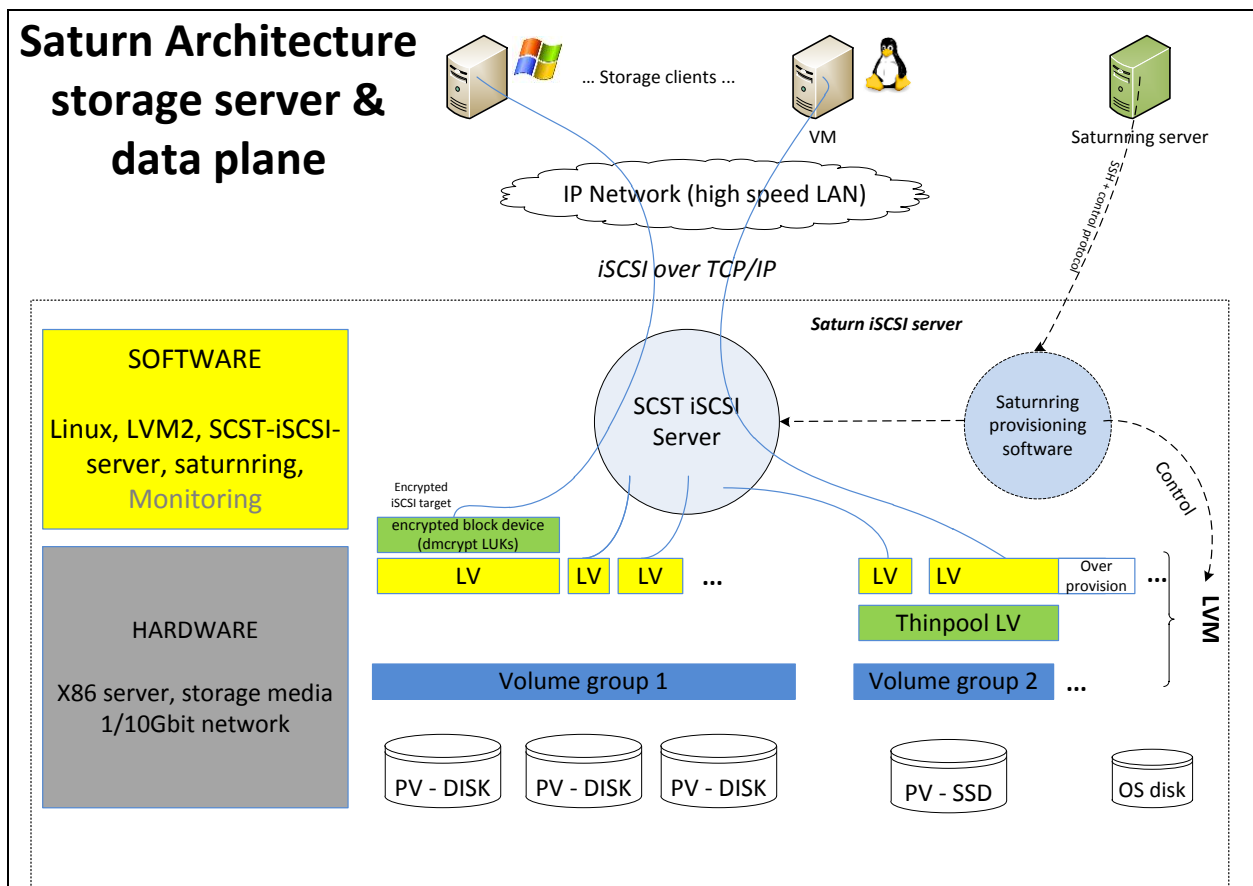**FIGURE 1: ISCSI STORAGE CONFIGURATION AND DATA/CONTROL PLANE**

Figure 1 shows the iSCSI server configuration and the distinction between the data and control plane in Saturnring. The iSCSI server houses one or more "data block devices" – e.g. disk drives – that store the data. LVM is setup to use any combination of these devices as physical volumes (PVs). One or more volume groups (VGs) may be created

over these PVs. Logical volumes (LVs) can then be carved out of the VGs. iSCSI targets connect the LVs to storage clients and store data on the LVs. An iSCSI server can have different types of storage media – e.g. SSD and spinning disk. There are facilities for defining different media types while provisioning storage in a Saturnring cluster.

Optionally LVs may be encrypted via dmcrypt to ensure at-rest data protection; so for example, if the PV disk is removed then the contents of the encrypted LVs are not compromised. The choice of doing encryption on the LV level stems from the observation that encryption introduces a small but noticeable penalty on performance and so should not be enabled for all iSCSI targets by default (for example, by encrypting the entire PV). As an aside, note that the iSCSI server's OS disk is separate from the data disks – this allows moving the data-disks physically to another backup server should the server die. LVM has facilities to do such manual inter-server migration of disks.

The LVM and iSCSI SCST server is controlled by the provisioning Saturnring software. There is a clear distinction between the data plane carrying iSCSI traffic and the Saturnring control plane for managing the iSCSI server(s). Any existing iSCSI sessions are not impacted when the Saturnring server becomes unavailable. The users will lose the ability to manage storage (provision/delete storage for example) but the pre-existing iSCSI sessions will not fail. This is analogous to managed virtualization solutions like Openstack or Opennebula – the hypervisor software is independent of the management software and the unavailability of the latter does not impact pre-existing VMs.

The SCST iSCSI server exports LVs as iSCSI targets to the client VMs or servers. The iSCSI network block protocol is used as the underlying network storage mechanism. For reliable and fast storage a reliable, low latency and high throughput network is necessary. Building and operating such a network can be challenging; the tight coupling (TCP sessions) between network and iSCSI storage requires stringent design and operational excellence for iSCSI to work. Even then, applications built on top of any network storage protocol (not just iSCSI) need to be resilient to occasional TCP session losses that result in unavailable storage. Some risk mitigation strategies were discussed in the section Key Considerations. When possible, alternatives to block storage should also be considered, for example, data object-stores like Openstack swift or Amazon S3 serve data via a stateless (usually HTTP-based) protocol. These types of storage services are inherently more resilient.

The Saturnring server in Figure 1 controls iSCSI servers via invoking bash commands and scripts over SSH tunnels. A small corpus of re-usable bash scripts (marked Saturnring provisioning software) are stored on each storage server to speed up the process. Advanced users may tweak these bash scripts to control some of the provisioning aspects and fine-tune the SCST, dmcrypt and LVM parameters.

Saturnring does not come with its own monitoring software. The administrator can setup any monitoring software that reports server health of the Saturnring provisioning server and the iSCSI servers. Apart from the usual monitoring metrics like CPU, memory, network, and disk IO, it is prudent to monitor a few critical services – for example – the SCST process on each iSCSI server, any dmcrypt or iSCSI server errors in the kernel logs and the Apache process on the Saturnring server. Any specific monitoring tools for the underlying storage devices – for

example wear level indicators for SSD drives – should also be included in the monitoring dashboards. Open source tools like Cacti, Nagios or Zabbix are good free choices for monitoring software.
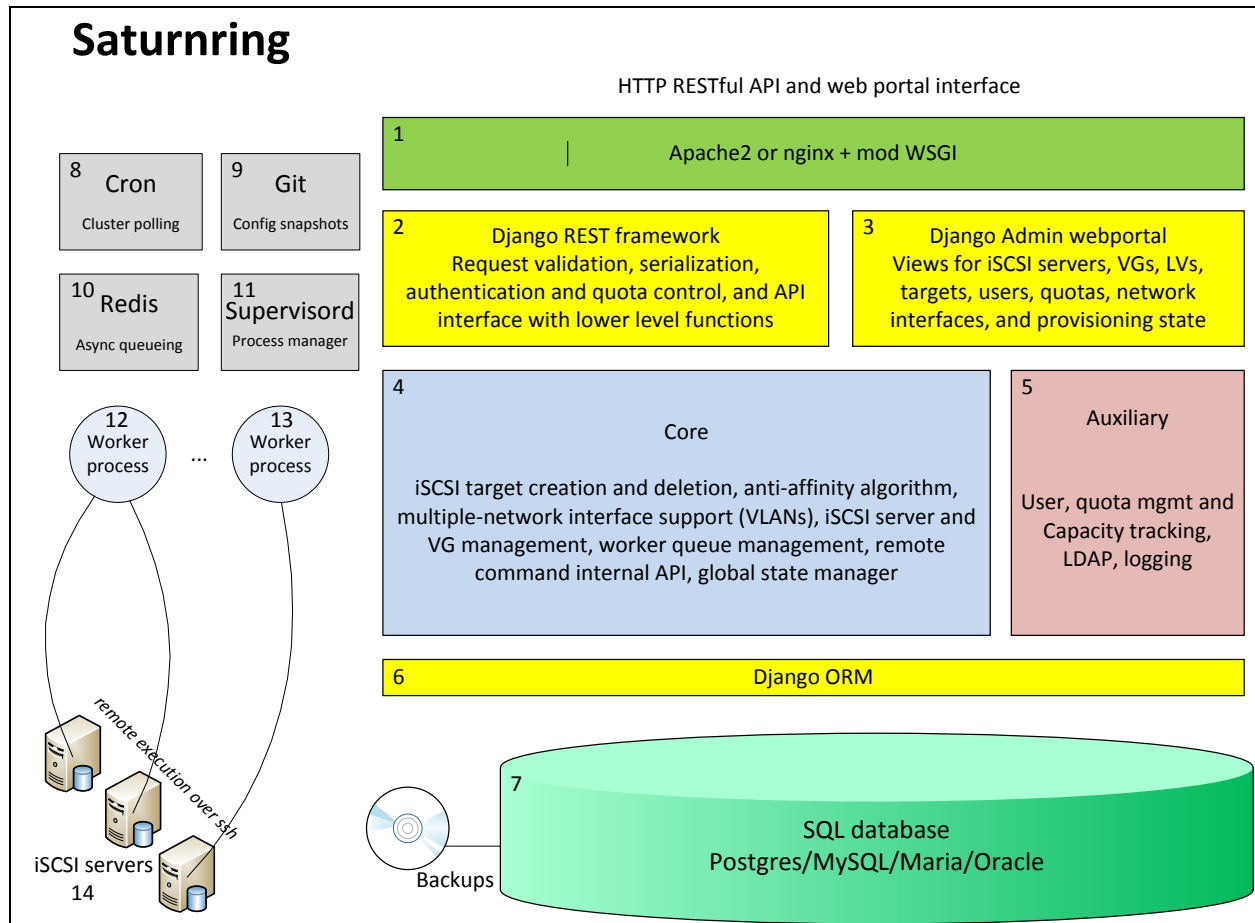

FIGURE 2: HIGH LEVEL SOFTWARE ARCHITECTURE OF SATURNRING

Figure 2 illustrates the Saturnring provisioning system. Storage is controlled either via HTTP(s) RESTful API calls (usually embedded within iSCSI clients to manage storage via scripts) or via a web-browser in the Saturnring portal. The corresponding web services are implemented in the Saturnring server based on top of the Django web and RESTful framework.

Behind the scenes Saturnring is primarily a database with models representing users, quotas storage hosts, volume groups, logical volumes, targets network interfaces etc. - the shared state in the Saturnring system is implemented via this SQL database. Saturnring makes decisions on where and how to manage storage based on the information provided by the database. Any SQL database supported by the Django ORM can be used, although Sqllite should be avoided due to known locking issues. It is important to backup this database periodically. This is usually a low-IO database being written to when the state of the Saturnring cluster changes and while polling the cluster for iSCSI metrics (defaults to once per minute). Multiple Saturnring servers can be run off the SQL database to ensure high

availability (assuming that the SQL database itself is HA). All locks for critical sections of code are implemented via the database to make such HA setups possible. For a very large Saturnring cluster the Redis server with Supervisord can be architected to control multiple "worker processes" on multiple hosts. In most small clusters this is not required.

Saturnring is implemented using the Django web framework. It presents a heavily modified Django-admin interface is presented as its web portal. The Django REST framework is used to implement the API.

INSTALLATION GUIDE

Keywords to read-up on the Internet: Vagrant, Virtualbox, Ubuntu, Django Web Framework, apache2

VAGRANT INSTALLATION – FOR DEVELOPMENT AND TESTING

Vagrant is a system for creating and configuring virtual development environments. It is used to create an virtual machine instances of the Saturnring server and the iSCSI servers. These VMs are virtualbox VMs being managed by Vagrant. They accurately emulate a real world Saturnring configuration. It is the fastest method to test-drive Saturnring in a virtualized environment.

A Saturnring is built out of multiple components - the iSCSI server(s), the Django-driven Saturnring portal and API and Apache webserver with mod-wsgi extensions, the backend database (sqlite or other Django-compatible relational DB) and a redis-server and job workers for running periodic tasks. A Vagrant file and shell provisioner scripts are included to automatically setup these components for illustration.

Instead of supplying pre-baked and customized VM images for quick setup the idea is to provide scripts that can be adapted to instantiate Saturnring on any private or public cloud or on bare-metal. The Vagrant file setups up Virtualbox VMs that take on the roles of the Saturnring server, 2 iSCSI servers, and an iSCSI client. Vagrant brings up vanilla Ubuntu 14.04 images, and the shell provisioner scripts do the work of adapting the vanilla VMs into these different roles. These bash scripts are an easy segway to setting up Saturnring in any other virtual or bare-metal environment, or for creating custom images to be used in the cloud.

An unhindered Internet connection and a computer capable of running at least 2 VMs (256M RAM per VM, 1 vCPU per VM, 20GiB disk) is assumed here. 'Host' refers to the PC running the VMs, the SSH login/password for all VMs is vagrant/vagrant, and the Vagrant file defines an internal network 192.168.56/24 and a bridged adaptor to let VMs access the Internet.

Here are the step-by-step installation instructions

STAGE 0: SOFTWARE INSTALLATION AND CODE DOWNLOAD

1. Install Virtualbox: http://www.virtualbox.org
2. Install vagrant: http://docs.vagrantup.com/v2/installation/
3. On the Virtualbox host machine (your PC) Clone into https://github.com/sachinkagarwal/saturnring/ in local directory

    ```
    mkdir -p ~/DIRROOT

    cd ~/DIRROOT

    git clone https://github.com/blackberry/saturnring/
    ```

4. Navigate to /saturnring/deployments/vagrant

```
cd ~/DIRROOT/saturnring/deployments/vagrant
```

STAGE 1: BRINGING UP SATURNRING PORTAL/API SERVER (192.168.56.20)

5. Use Vagrant to bring up the Saturnring VM, you should see a lot of bootup activity happening on the VM (takes a while). You may have to download the Ubuntu 14.04 Vagrant box from

   https://cloud-images.ubuntu.com/vagrant/trusty/current/trusty-server-cloudimg-amd64-vagrant-disk1.box.

   ```
   vagrant up saturnring
   ```

6. If all went well, you should be able to navigate to

   ```
   http://192.168.56.20/admin
   ```

   from a web brower on the host machine. Check by logging in with credentials "admin/changeme".

STAGE 2: BRINGING UP THE ISCSI SERVER(S)

7. Bring up an iSCSI VM defined in Vagrantfile vagrant up iscsiserver1 (192.168.56.21)

   ```
   vagrant up iscsiserver1
   ```

8. Log into the saturnring VM and copy SSH keys for Saturning to access the iSCSI server

   ```
   vagrant ssh saturnring

   cd ~/nfsmount/saturnring/saturnringconfig

   ssh-copy-id -i saturnkey vagrant@192.168.56.21
   ```

   (password is vagrant)

9. Log into the saturnring portal as admin superuser and add the new iscsi server. For this simple example, Dnsname=Ipaddress=Storageip1=Storageip2=192.168.56.21. Failure to save indicates a problem in the configuration steps. Saturnring will not allow a Storagehost being saved before the configuration is right.

10. From the VM host issue a "initial vgscan" request to the Saturnring server so that it ingests the storage made available by iscsiserver1 at IP address 192.168.56.21 (Networking is defined in the Vagrantfile)

    ```
    curl -X GET http://192.168.61.20/api/vgscan/ -d
    "saturnserver=192.168.61.21"
    ```

    Confirm in the web browser portal (under VGs) that there is a new volume group

11. Repeat Stage 2 for iscsiserver2 (192.168.56.22) if you want (it is useful to have 2 iscsiservers if you want to try the anti-affinity provisioning)

SIDE NOTE: SPEEDING UP VAGRANT VM START TIMES

The installation scripts are setup to adapt a plain vanilla Ubuntu image into a saturnring and iscsiserver(s) VM each time. This may be too slow for rapid development because several 100MB of packages are downloaded from Ubuntu package repositories every time the VMs are recreated. The process can be speeded up by taking a Vagrant snapshot of the Saturnring and an iscsiserver VM immediately after running the installation steps described above. If the snapshot is then used as the starting point for instantiating VMs the boot time will be drastically reduced.

To implement this we perform these steps

1. Instantiate a saturnring VM and an iscsiserver VM as described above
2. Find out the Virtualbox-assigned names of each VM

   ```
   vboxmanage list runningvms
   "saturnring_saturnring_1430832300410_25018" {29562b70-b035-42bb-8409-d5a8a2e51ffe}
   "saturnring_iscsiserver1_1430833211668_58259" {e3314d87-6e90-4d56-8826-b6f409990f6c}
   ```
3. On the vagrant host create a "box" corresponding to each of the VMs

   ```
   vagrant package --base "saturnring_saturnring_1430832300410_25018"
   ```
   (this will shut down the VM gracefully)
   ```
   vagrant box add package.box --name saturnringbaked
   vagrant package --base "saturnring_iscsiserver1_1430833211668_58259"
   ```
   (this will shut down the VM gracefully)
   ```
   vagrant box add package.box --name iscsibaked
   ```

There should now be 2 boxes containing these snapshots

```
vagrant box list
iscsibaked        (virtualbox, 0)
precise32         (virtualbox, 0)
saturnringbaked   (virtualbox, 0)
ubuntu/trusty64   (virtualbox, 14.04)
```

Copy over the Vagrantfile-prebaked into Vagrant file and start using the prebaked images. If you want to revert to using the installation from plain vanilla Ubuntu, copy the Vagrantfile-vanilla into Vagrant and proceed.

When VMs are created from the pre-baked images the same Vagrant provisioner is run (the .sh files in the install directory), but since all software packages are already installed the VM starts quickly. Any changes (like adding an additional package) will be picked up by the Vagrant provisioner.

START DEVELOPING!

The saturnring VM (access via `vagrant ssh saturning`) mounts the hosts' code directory (same directory where the Vagrantfile is stored) under the `/vagrant` location. The Django project code is accessible at `/vagrant/ssddj`; or equivalently it can be edited on the host and then run on the saturnring VM. Remember to restart the Apache2 process to pick up new code. If you are using the Django web server for development then code changes are usually picked automatically. There are exceptions to this, for example the queue processes that run in Saturnring may not reload the code when it is changed. See the Section "Worker Queue Processes" on how to restart these processes.

GUIDE FOR ADMINISTRATORS

Keywords to read-up on the Internet: Django web framework, LDAP

INTRODUCTION

Saturnring provides a web portal to manage the Saturnring cluster. The webportal allows administrators and users to operate on this database. The database in turn is used to drive the

LOGIN SCREEN AND OVERVIEW

The administrator primarily interacts with Saturnring through the web portal. The webportal can be reached via the browser by pointing it at

http://192.168.56.20/admin

This will return a page like shown in Figure 3. Apart from the login form, there is the name of the Saturnring cluster (as defined in the saturn.ini file), the number of iSCSI Saturn servers in the cluster, and a 4-tuple stating the total storage in the iSCSI cluster, the quota assigned to users (quota-promised), how much of the storage is actually in use (size of the LVs created across all iSCSI servers) and the maximum LUN size that could fit in any VG across the cluster. The last quantity becomes important as the cluster fills up and it becomes impossible to create a large iSCSI target on any one VG, even though there may be a substantial amount of residual capacity across the VGs in the cluster.
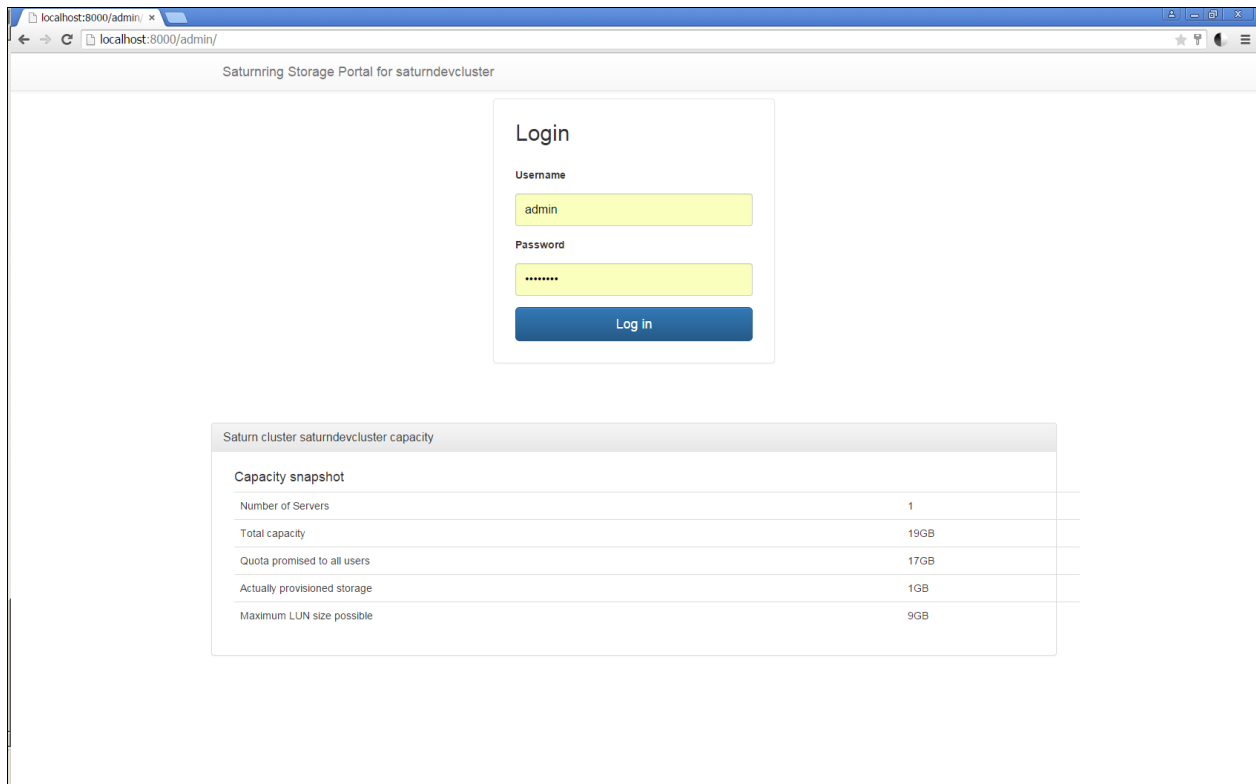
FIGURE 3: LOGIN SCREEN

The admin interface is adapted from the Django admin interface. The interface is a powerful scope into the data-driven Django framework, providing convenient and relatively readymade bugfree mechanisms to interact with the Saturnring database. A drop-down bar below the login credentials form provides a snapshot of the cluster's capacity situation even without logging in. When using the Vagrant install scripts described in Section "Vagrant Installation – for development and testing" the default login and password for admin is admin/changeme. More users can be created by the admin user as described later in this chapter.
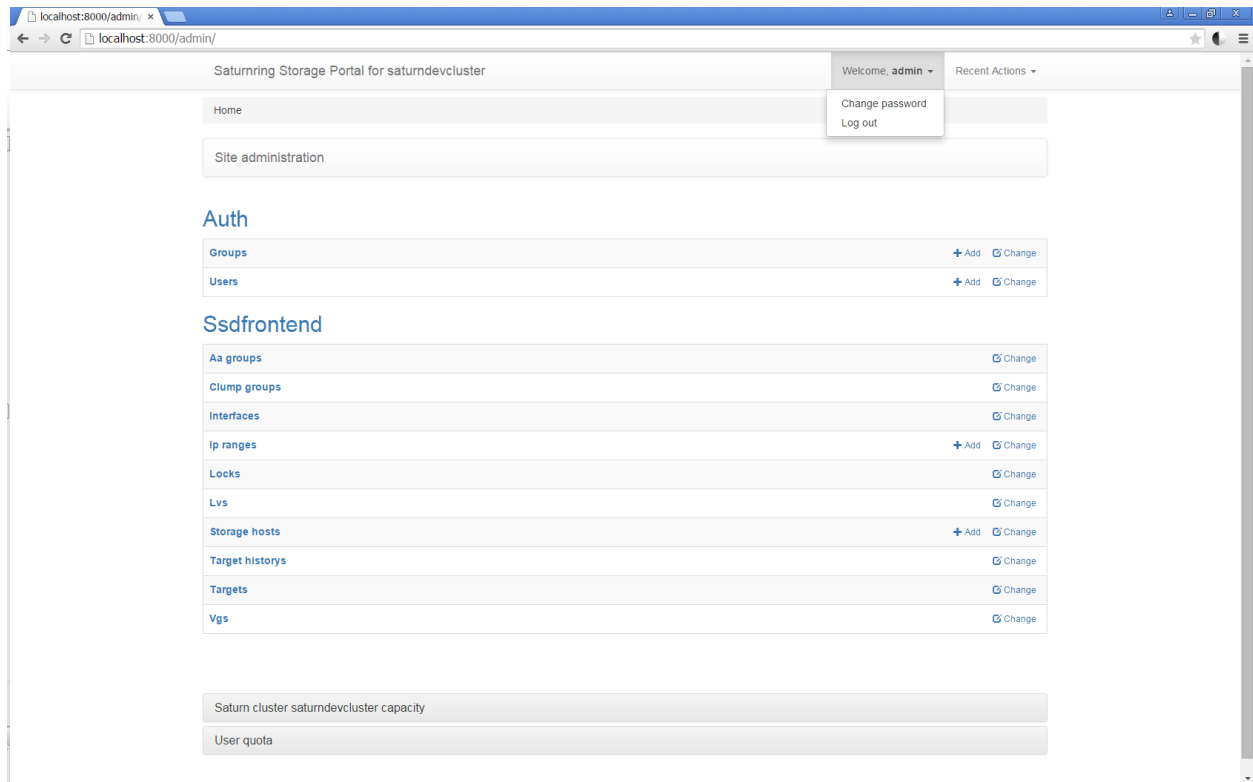
FIGURE 4: POST-LOGIN SCREEN FOR ADMIN USER

Figure 4 shows the post-login screen for the Admin user. There are two broad categories of controls. The "Auth" controls with the ability to define users and assign them to groups with permissions to specific tables in the back-end Saturnring database and the "SSDfrontend" controls that defines various aspects of the Saturnring system. These aspects will be described later in this chapter. The admin password is changed by clicking the welcome banner at the top right of the screen. There is also a dropdown list of "recent actions" for the user at the top right of the screen. Changes made via the interface are displayed in reverse chronological order.

The bottom of the screen displays two bars that can be clicked and expanded to show cluster and user statistics (the admin user is also permitted to provision iscsi storage; so the user quota and currently used quota is shown, along with the maximum allowed iSCSI target size.

USER AND QUOTA MANAGEMENT
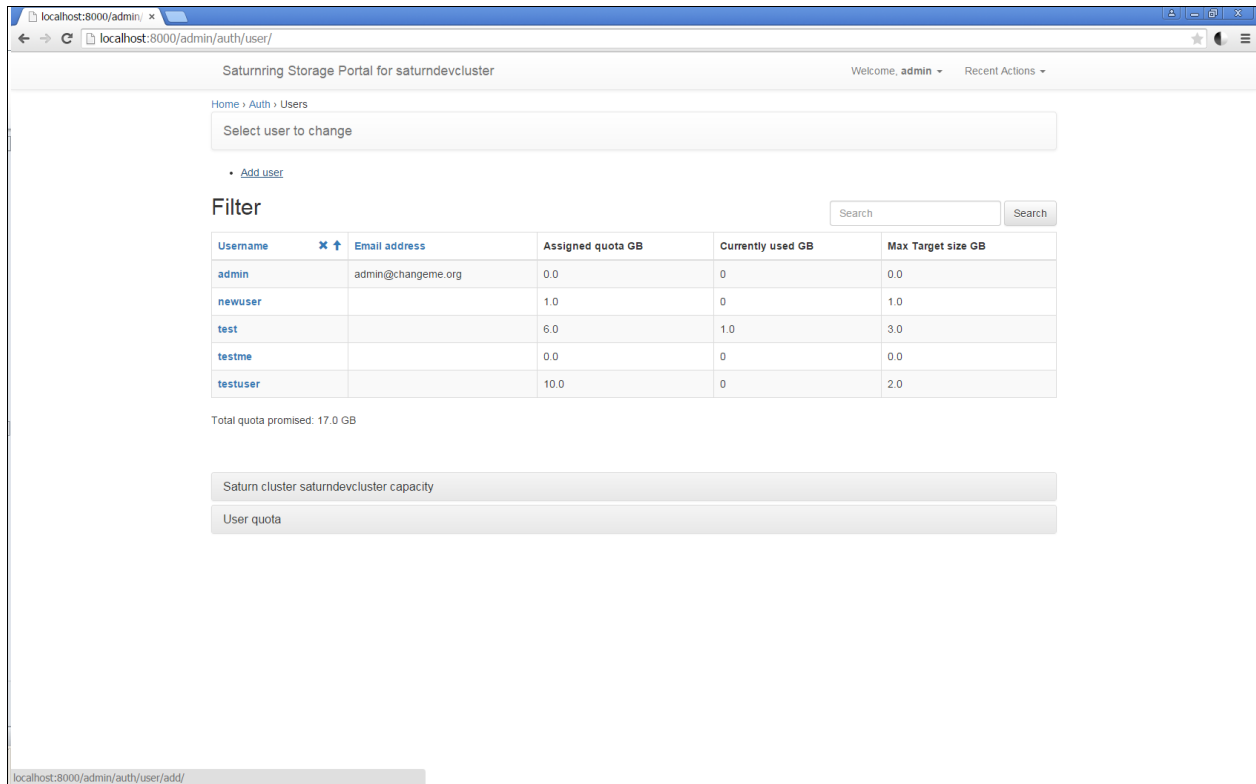
FIGURE 5: USER LIST AND QUOTA OVERVIEW

User management is accessible via Home > Auth > Users (notice the navigation pointers at the top left).

The list of users and their quota information is presented as a table. New users can be added and existing users can be searched for here. A new user can be added by clicking on the "Add user" link. The add-user link leads to the form shown in Figure 6. This form takes in the basic information about the user – username, password, and quota information and creates the user. Quota information comprises of two entities.

1. Max target size (GB): The maximum allowed size of an iSCSI target created by the user specified in GB.

2. Max allocated size (GB): The maximum allowed sum of all iSCSI target sizes created by the user, specified in GB.

The form also verifies quota availability. If the total quota promised to all users plus the quota of the new user is greater than the cluster capacity then an error is thrown and the administrator is prompted to correct the quota.

FIGURE 6: ADDING A NEW USER

To allow portal access a new user needs to be classified as staff by checking the "staff status" checkbox as shown in Figure 7. Next, the user's admin portal views have to be given permissions to view targets and target histories belonging to the user. This is implemented either via specifying a "group" containing the requisite permissions or via adding individual permissions. Add these permissions to a new user:

```
SSDFRONTEND | TARGET | CAN CHANGE TARGET
SSDFRONTEND | TARGET | CAN DELETE TARGET
SSDFRONTEND | TARGET HISTORY| CAN ADD TARGET HISTORY
SSDFRONTEND | TARGET HISTORY| CAN CHANGE TARGET HISTORY
SSDFRONTEND | TARGET HISTORY| CAN DELETE TARGET HISTORY
```

Finally, Figure 8 shows the form where quotas can be changed (for new or existing users). This form can be accessed for existing users by clicking on the existing user and scrolling down.

FIGURE 7: SETTING PERMISSIONS

FIGURE 8: CHANGING QUOTAS

LDAP AND ACTIVE DIRECTORY

Saturnring can authenticate against an open-LDAP or active-directory server. The implementation is based on python-ldap (which in turn is based on Open-ldap-2.x) and django-auth-ldap. It should work with any LDAP-compatible authentication service. The code has been tested with a Windows Server 2008r1 Active directory installation. Here is the corresponding confiuration in the Saturn.ini configuration file:

```
[activedirectory]
enabled=1
ldap_uri=ldap://192.168.61.30
user_dn="CN=Users,DC=saturn,DC=ad"
staff_group="CN=staff,DC=saturn,DC=ad"
bind_user_dn="CN=adbinduser,CN=Users,DC=saturn,DC=ad"
bind_user_pw="adbinduserpassword"
```

Saturnring configures authentication so that both local and LDAP authentication is possible. For example, the superuser (admin) account may be local. Local accounts run off the Django database. With an external authentication source, all users in the staff_group defined in the saturn.ini file can control their Saturn storage via the API. The superuser will need to change their default quotas and enable their accounts to change targets and target-histories if these external accounts are to access the portal.

iSCSI SERVER AND STORAGE MANAGEMENT

Navigate to: `Home > Ssdfrontend > Storage hosts > Add storage host`

New iSCSI servers need to be added to the cluster via the "Add storage host" link shown in Figure 9. Adding a new iSCSI server requires filling out the form shown in Figure 10. Before saving, the ssh keys need to be in place for password-less login (See the Section Adding SSH Keys). For the simple Vagrant example, Dnsname=Ipaddress=Storageip1=Storageip2=192.168.56.21. The "save" operation will fail if Saturnring cannot scp-copy over scripts to the new iSCSI server. This usually arises from network connectivity or SSH key issues. After saving the new iSCSI server information make a "initial scan" request to the Saturnring server so that it ingests the storage VGs made available by iscsiserver1 at IP address 192.168.56.21. This is done via the following API call:

```
curl -X GET http://192.168.61.20/api/vgscan -d "saturnserver=192.168.61.21"
```

Confirm in `Home>Ssdfrontend>Vgs` that the new volume group(s) belonging to the new server are now available to Saturnring.

Any iSCSI server can be "disabled" by checking off the "Enabled" shown in Figure 10. This means that users can no longer provision or delete storage targets on the disabled iSCSI server. Pre-existing iSCSI sessions are not affected.
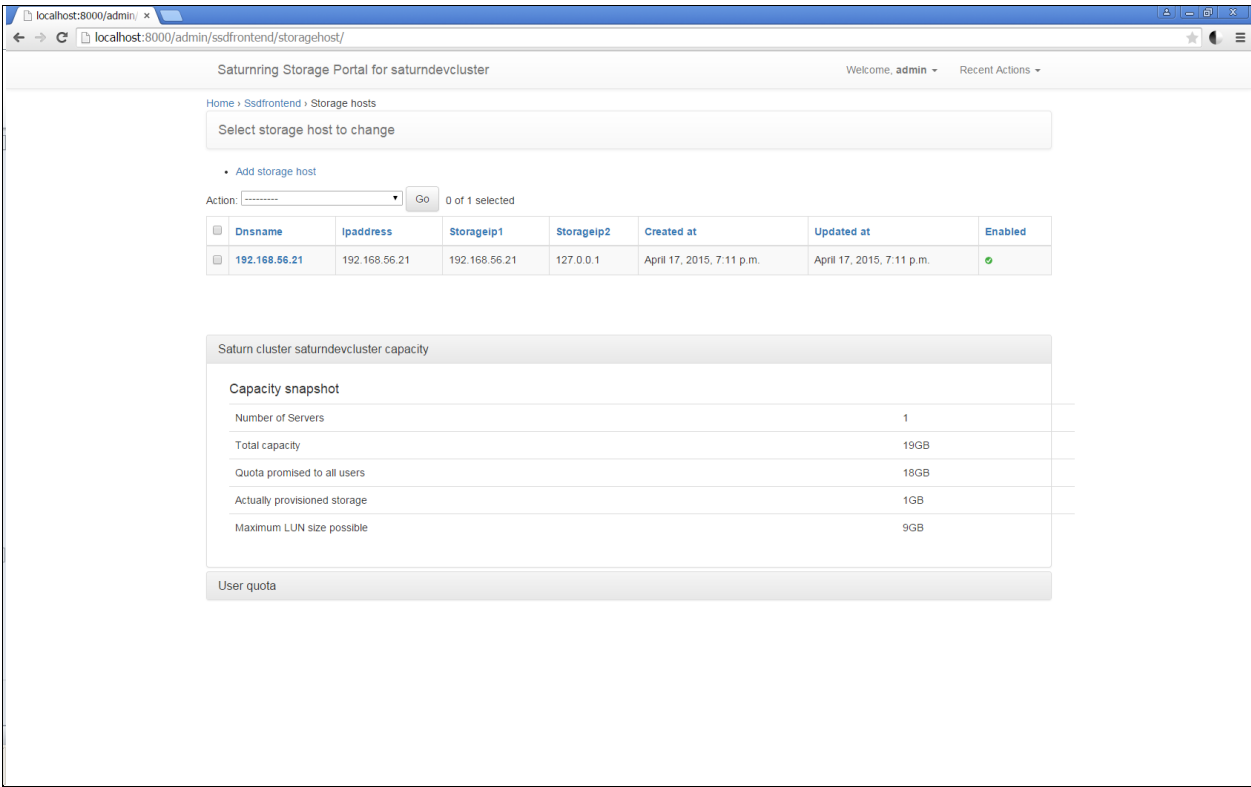
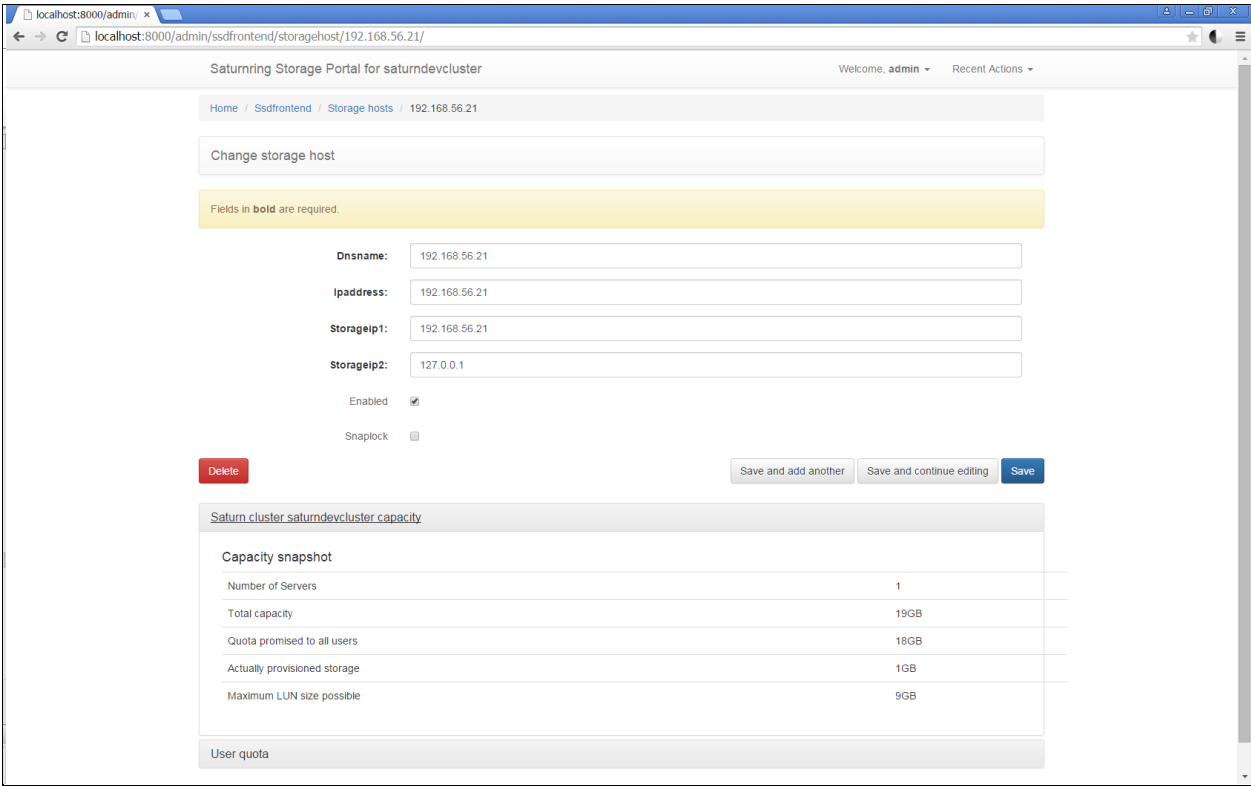**FIGURE 9: ISCSI SERVER LIST**

FIGURE 10: ADDING A NEW ISCSI SERVER

Saturnring Guide

Volume groups are automatically discovered when the "vgscan" API call is executed against a storage host as discussed above. Navigating to `Home > VGs` will display a list of all volume groups in the cluster. The storage host containing the VG is displayed as the "VG host" column.
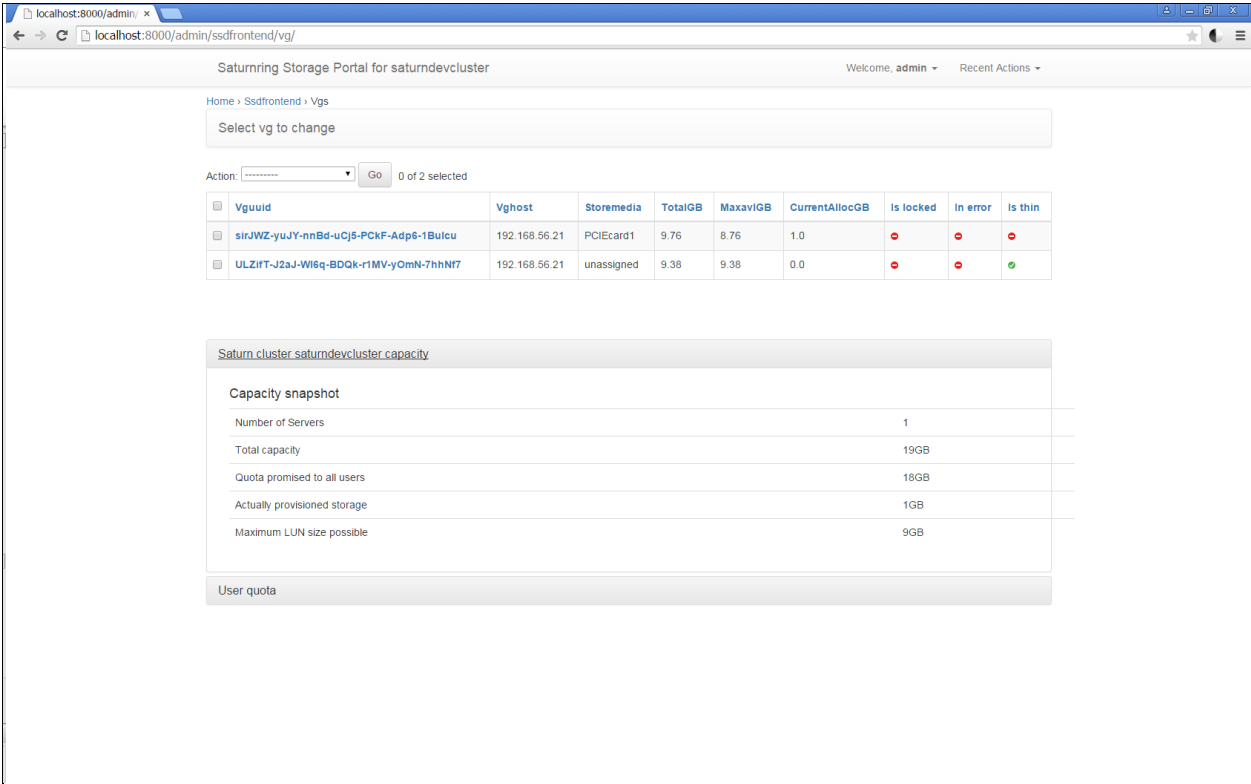

FIGURE 11: VOLUME GROUPS

FIGURE 12: EDITING A VOLUME GROUP

TARGETS AND LOGICAL VOLUMES

Navigate to Home > Targets

iSCSI targets information is stored in the Saturnring database in the "Targets" table. The administrator can view all the targets in the cluster and search through them in the Targets list view. This Targets list view shows columns with the target and initiator IQNs, creation time, size in GB, whether the underlying LV is LUKs-encrypted or not, anti-affinity and clumpgroup information, read/write throughput in kB per minute, whether the iSCSI session is up, physical location (storage host, volume group, logical volume) where the iSCSI target lives, and the owner of the target. Many of these fields are searchable and sortable. Clicking on any iSCSI target yields more information about the target.

iSCSI targets can be deleted by selecting them and using the drop-down action menu, as illustrated in Figure 21. An administrator can delete any user's storage; however sessions which are active (sessionup column has a green dot) cannot be deleted by anyone. When a target is deleted its LV is also recycled (i.e. the LV is deleted and the storage returns to the VG). An entry is created in the Target History table to record the creation and deletion time of the target as well as the total kilobytes written and read off the target during its lifetime.
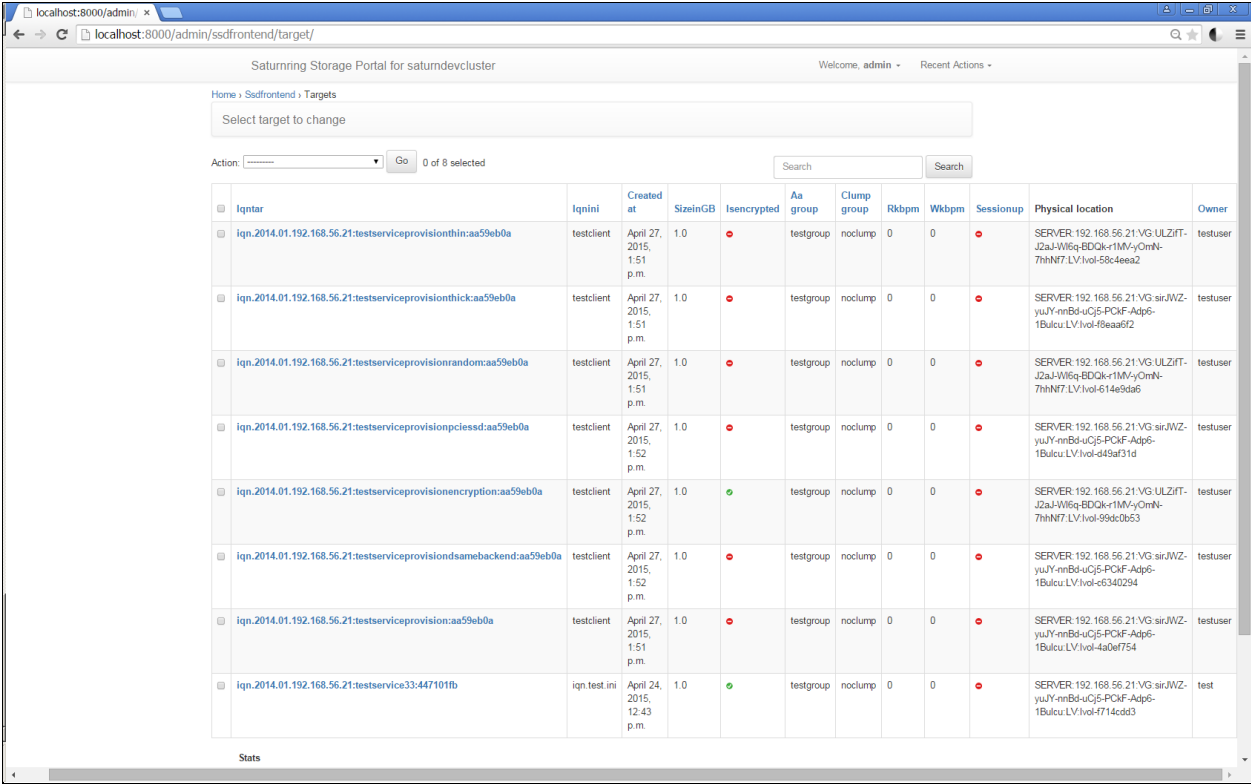
FIGURE 13: ALL TARGETS IN THE CLUSTER
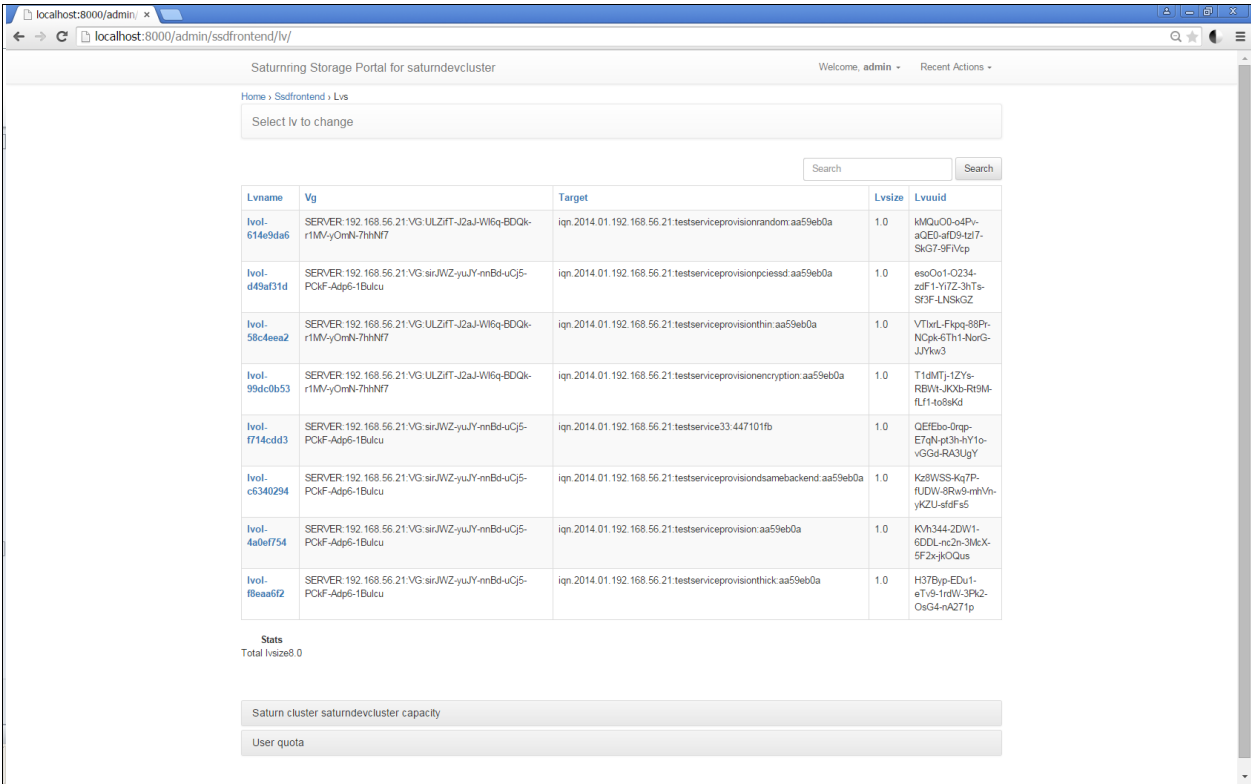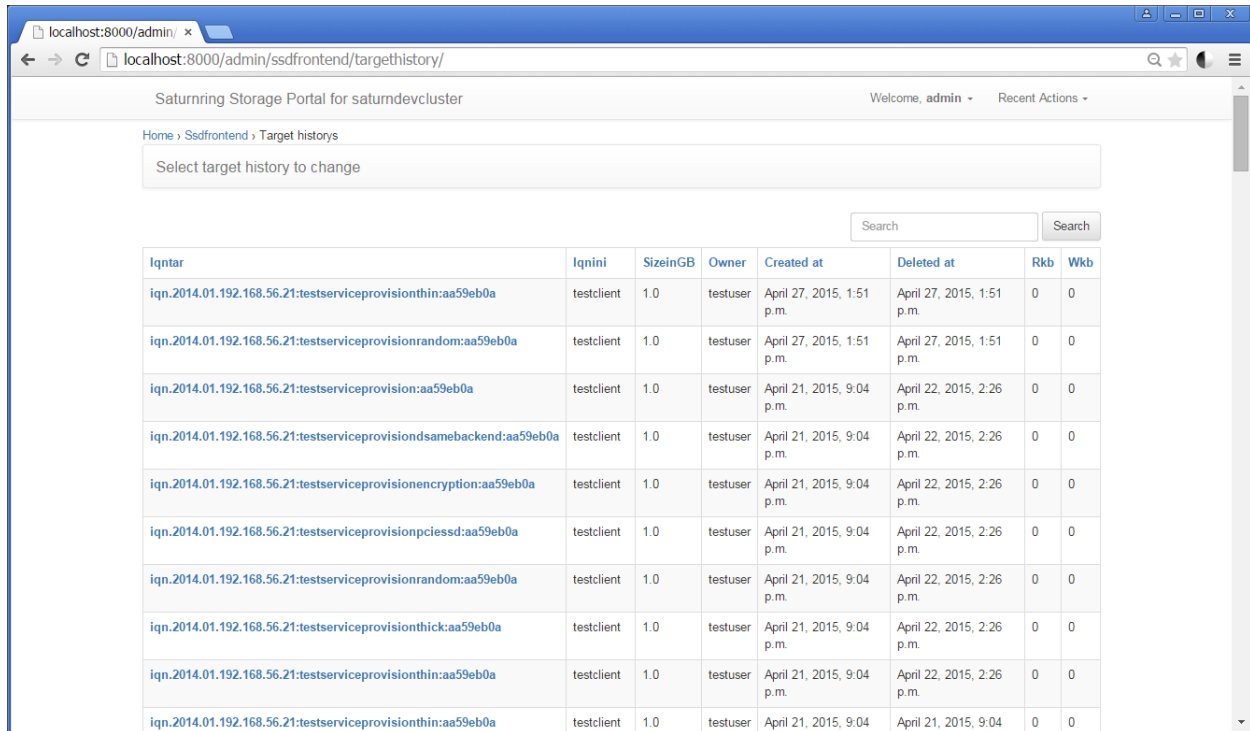
FIGURE 14: ALL LOGICAL VOLUMES IN THE CLUSTER

Navigate to Home > LVs to see a list of all logical volumes as shown in Figure 14. Logical volumes are associated with a target, and are created and deleted in lock-step with target creation and deletion.

TARGET HISTORY

FIGURE 15: TARGET LIFECYCLE HISTORY

Navigate to Home > Target Historys to see historical information about when targets were created and deleted. Note that only deleted targets are added to the target history list. Figure 15 illustrates this table. Information about the size and the total data read and written to the target (in kB) is also recorded. All this information can be used to create a billing extension to Saturnring, for example.

NETWORKING

This feature is optional. If there is no VLAN then Saturnring will provision the iSCSI targets on the IP address specified as the "storage IP" during provisioning (see the section titled Example 1: Setup a simple iSCSI target*)*

The current version of Saturnring includes basic (beta) ability to create an iSCSI target on a specific subnet. This may be useful when users' storage needs to be presented over a specific VLAN. The idea is that network interfaces for VLANs are created on all storage hosts via any out-of-band mechanism (possibly using a configuration management tool like puppet, chef or ansible) for a user's VLAN. These interfaces are automatically detected by Saturnring's scanning mechanism on all storage hosts. Next, the administrator defines IP ranges and makes the corresponding user the owner of this IP range. While provisioning iSCSI targets owners can specify the iprange in the form of a subnet string to create a iSCSI target on one of the storage hosts with an interface in that ip range.

Saturnring allows the administrator to define IP ranges in terms of standard network subnet definitions. For example an IP range may be 192.168.61.0/24. The IP range is owned by a user and mapped to all storage hosts that have network interfaces in this IP range. Saturnring scans each storage host periodically to learn all its network interfaces; these are recorded and displayed under Home > Ssdfrontend > Interfaces.

FIGURE 16:NETWORK INTERFACES

FIGURE 17: ASSIGNING INTERFACES TO IPRANGE
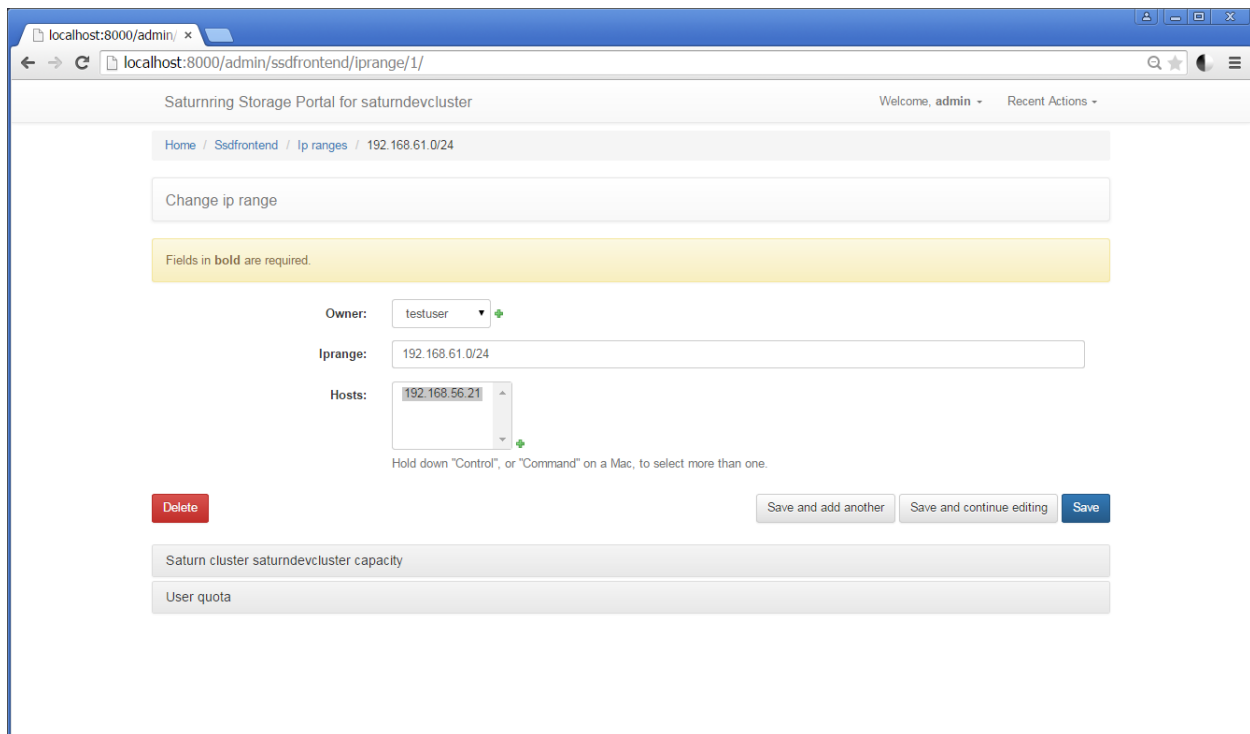
FIGURE 18: IP RANGES

LOW LEVEL CONFIGURATION

SATURN.INI CONFIGURATION FILE

The saturn.ini file is located in the `<install directory>/saturnring/ssddj` directory. The file (self documented via comments) contains several parameter values for the saturnring setup, each iSCSI server, database, LDAP configuration, keys, default directories etc. An example from the Vagrant installation is included in the appendix of this document to illustrate what features are configurable. Administrators may also want to look at the bash script used to create the Saturn.ini file during the vagrant install process: `<install directory>/saturnring/saturnring_postbootup_as_vagrant_user.sh`.

SETTINGS.PY CONFIGURATION FILE

The settings.py Django configuration file is available in the `<install directory>/saturnring/ssddj/ssddj` directory. It can be used to fine-tune configure the Saturnring service with more granularity, although many of its parameters are configured via the saturn.ini file. Some knowledge of Django and Python is needed to work with this file.

ADDING SSH KEYS

Saturnring controls iSCSI servers by logging into them via SSH and executing bash commands. A sudo-enabled account (with sudo's password requirements disabled) is used on the iSCSI servers. The SSH key can be copied over

31

from the saturnconfig directory (specified in saturnring.ini under the `saturnring > iscsiconfigdir` key) using the `ssh-copy-id` utility in Linux. For example,

```
cd ~/nfsmount/saturnring/saturnringconfig

ssh-copy-id -i saturnkey vagrant@192.168.56.21
```

CHANGING POLLING INTERVALS

Saturnring polls each iSCSI server periodically via a cronjob to ascertain the state of its LVM entities and iSCSI sessions. This mechanism is also behind the detection of the session state – whether the iSCSI target session is active or not. The polling interval can be configured in the crontab. The default is set to 1 minute. Setting this lower will increase the polling frequency, but remember that saturnring is logging to the iSCSI servers and running various bash commands to gather the polled information, tying up the worker process for the iSCSI server in that interval. It is not recommended reducing this polling interval below 30 seconds.

Saturnring Guide

GUIDE FOR USERS
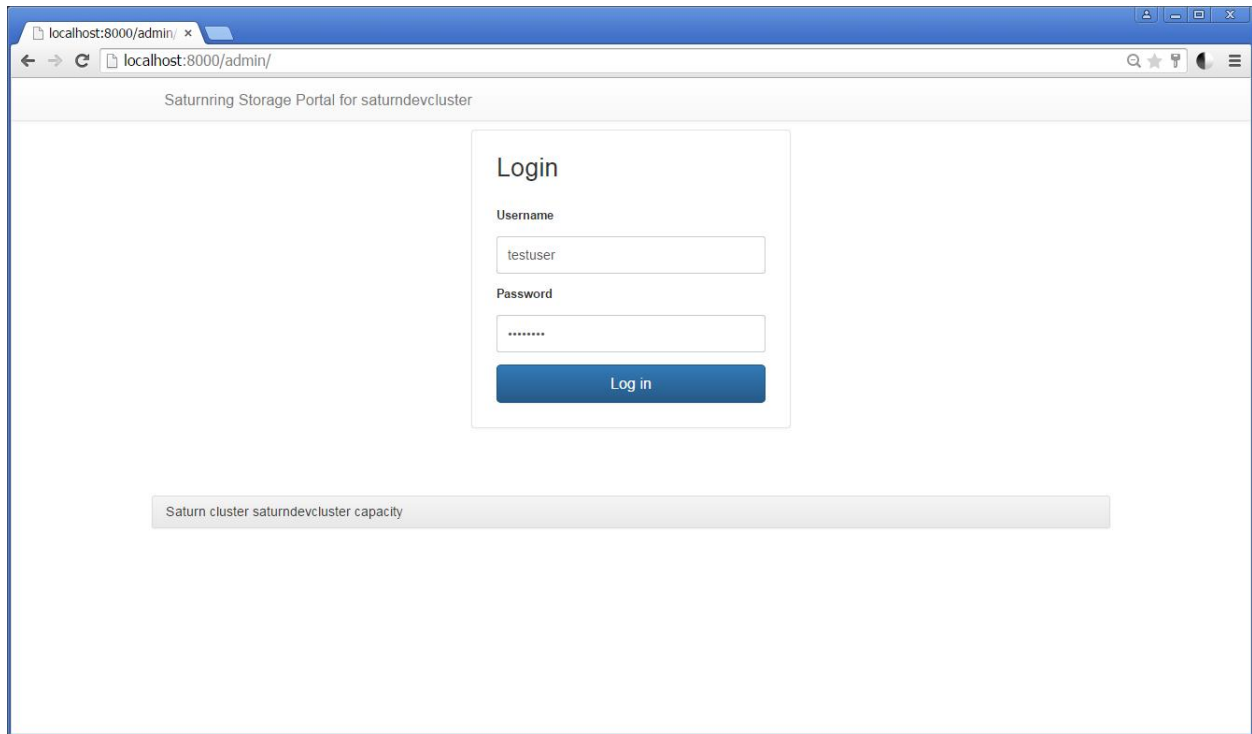
FIGURE 19: USER LOGIN

A user can log into the Saturnring portal to view her storage quotas, get a list of iSCSI targets that were created by her account, delete targets, change her password, and view a target history log of all her deleted targets. Figure 20 shows the user's dashboard. The username and password is the same as that used in the API. The user also has access to the "Saturn cluster capacity" dropdown bar that gives her a birds-eye view of the resource capacity available in the cluster.

To view and/or delete iSCSI targets the user navigates to Home > Ssdfrontend > Targets and selects any targets she wants to delete and then chooses the delete selected iSCSI targets action from the dropdown as illustrated in Figure 21. All deleted targets are seen by navigating to Home > Ssdfrontend > Target Historys.
Note that as of now the GUI interface cannot create iSCSI targets; users can only view or delete their iSCSI targets from the GUI. Use the API (Chapter Saturnring API) to create iSCSI targets.

Saturnring Guide
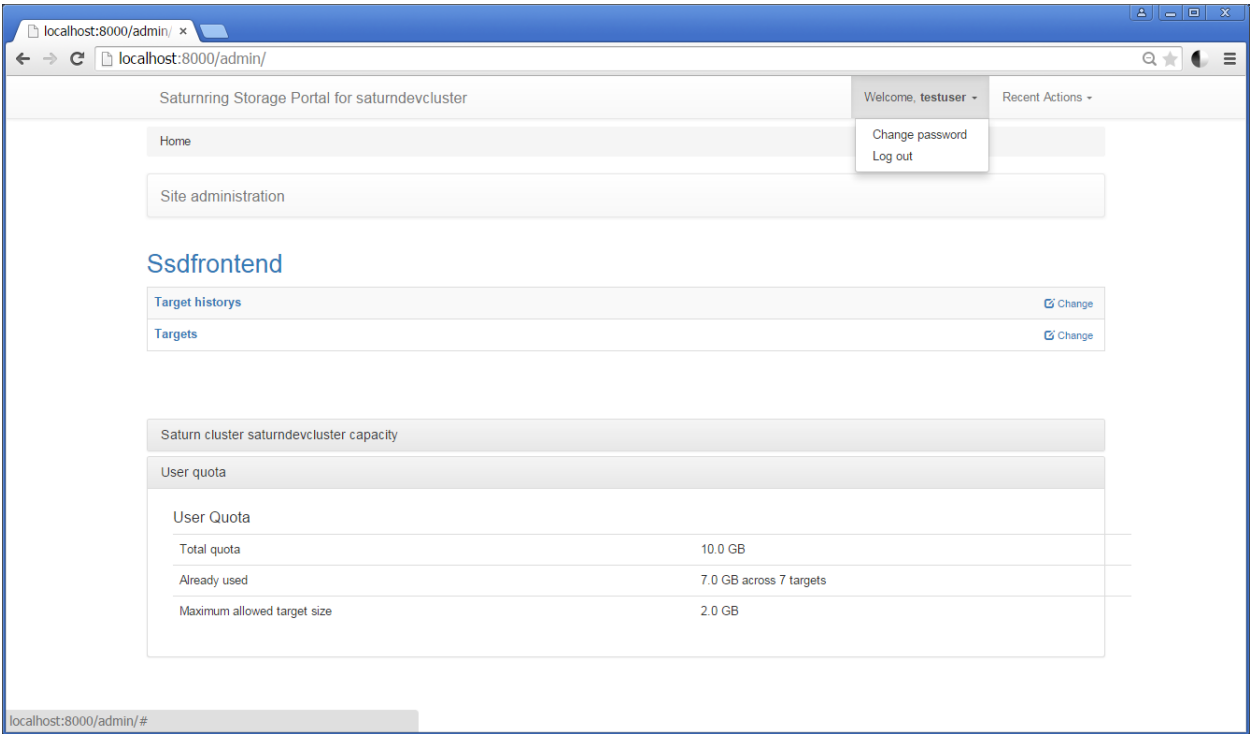
FIGURE 20: USER DASHBOARD
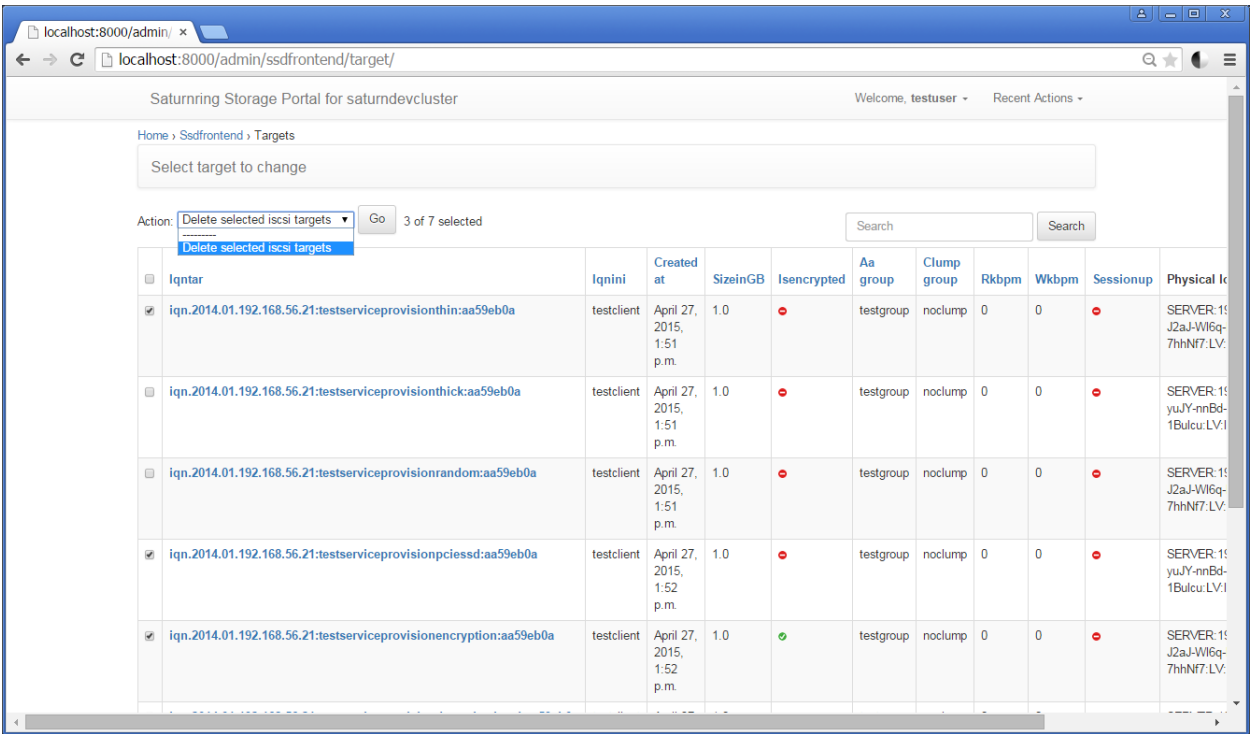
FIGURE 21: VIEWING AND DELETING TARGETS

34

SATURNRING API

Keywords to read-up on the Internet: CURL, RESTful API, iSCSI initiators, targets, portals

The primary purpose of the API is to control (provision/delete) storage via a HTTP request. The API is the only mechanism for provisioning new storage. It can also be used to query for existing targets and delete targets. A few other auxiliary functions are also accessible via the API. The API is useful for automating storage lifecycle and circumventing manual intervention for tasks like creating and deleting iSCSI storage. The API can be invoked via any HTTP client, illustrated here via curl as shown below. Most of the API calls require a username and password to authenticate before resources can be assigned or changed.

PROVISIONING NEW STORAGE AND QUERYING PRE-EXISTING iSCSI TARGET METADATA

New iSCSI storage – iSCSI targets – can only be created via a HTTP GET request to the Saturnring server. Although using a "GET" request is semantically unusual for creating state in the backend (a "PUT" would be more appropriate), the same Provisioning GET request is also capable of returning information about a pre-existing iSCSI target if for example, the iSCSI target is being re-attached to a VM that has been restarted. This simplifies cloud VM start-up scripts for automation and resubmission.

EXAMPLE 1: SETUP A SIMPLE iSCSI TARGET

Suppose a user needs to setup a 1GB iSCSI target in one of the backend iSCSI servers. Here is the workflow of how this will be implemented in Saturnring.

1. The user will obtain a user name, password and quota allocation from the Saturnring admin. (See the Section titled "User and Quota Management" in Chapter "

Guide for Administrators" on how this is implemented.)

2. The user would then instantiate the VM and run the following command (possibly run as part of a script during VM creation)

```
curl -X GET http://192.168.56.20:8000/api/provisioner/ \
    -d "sizeinGB=1.0" \
    -d "serviceName=mongodbbackendhost1disk1" \
    -d "clientiqn=2014.dbhost1.ini" \
    -d "aagroup=mongodbaagroup" \
    -u 'testuser:password' \
| python -mjson.tool
```

The (serviceName, clientiqn) tuple uniquely specify an iSCSI target. The serviceName may be any string excluding special characters (underscore _ and minus – are allowed). The clientiqn is unique to a iSCSI client; if more than one iSCSI target is needed on a client then the service names of each target need to be different. For example, "disk1" and "disk2" etc. The same serviceName can be used on different iSCSI clients.

The above curl statement returns a json string such as the one shown below.

```
{
    "aagroup__name": "mongodbaagroup",
    "already_existed": 0,
    "clumpgroup__name": "noclumpgroup",
    "error": 0,
    "iqnini": "2014.dbhost1.ini",
    "iqntar": "iqn.2014.01.192.168.56.21:mongodbbackendhost1disk1:7fbb4dc6",
    "isencrypted": true,
    "sessionup": false,
    "sizeinGB": 1.0,
    "targethost": "192.168.56.21",
    "targethost__storageip1": "192.168.56.21",
    "targethost__storageip2": "127.0.0.1"
}
```

The user can then use the iSCSI client on the MongoDB VM to log into the iSCSI target and use the storage. On Linux the open-iscsi package may be used. After setting making the initiator name (/etc/iscsi/initiatorname)

"2014.dbhost1.ini" and re-starting open-iscsi the appropriate iSCSI discovery and login commands can be issued (See the man page for iscsiadm).

Each of the keys in the JSON string shown above are explained now:

1. `aagroup_name`: This is the anti-affinity group name. Use the same string while creating another iSCSI target if it is required that these targets be created on different iSCSI servers. Note that this is best-effort operation: if there is no other iSCSI server with the required capacity then the anti-affinity request is ignored and the storage is still provisioned. It is prudent for a user to confirm anti-affinity worked by checking the target host IP for targets in the same anti-affinity group.

2. `already_existed`: This flag is set to 0 if a new target is being created. It is set to 1 if the target already exists. So for example running the same cURL statement again will return the same JSON string but with the already_existed key set to 1. A target is unique up to the (iqnini, servicename) tuple specified in the provisioning call. Therefore if the iSCSI initiator name (iqnini) and servicename were previously used to create an iSCSI target then the same target will be reported in the JSON string as a response to the provisioner call. This is the underlying mechanism for a workflow where a virtual machine instance can be deleted and recreated (with the same provisioner request to Saturnring) and it will re-acquire the previously created iSCSI target. The already_existed flag can be used to decide if a file system needs to be created/formatted on the device or not. So for example if already_existed=0, then this is a new target and so a file system needs to be created. However if already_existed=1 then a filesystem and possibly data already exists on the target from a previous VM and a new file system should (probably) not be created.

3. `error`: When there is a provisioning error (for example, there is no Storage host capable of accommodating the storage requested) this field will be non-zero; in addition most of the other fields in the JSON string returned by Saturnring will be missing and there will be a field titled description that describes the error. It is good practice to check for the error value before proceeding with any other post-provisioning steps.

4. `iqnini`: This is the initiator IQN. It is a unique string per client host specified while making the provisioning call (see the client example at saturnring/deployments/vagrant/clientscripts/storage-provisioner.sh). If the client has a DNS name then this hostname can be a part of the iqnini string for tracking client-target relationships in the Saturnring portal down the road.

5. `iqntar`: This is the unique IQN of the target storage provisioned on one of the iSCSI servers. At present it is of the form
   `2014.01.<DNS of iSCSI server>:<Servicename>:<Trancated MD5 of clientiqn>`.

6. `sessionup`: The sessionup property is relevant when an already existing target is being "provisioned again (see discussion about already_existed above). A sessionup=True would indicate that another client (with the same iqnini and servicename and hence access to the target) has already got an active iSCSI

session. In this case it is best not to try to login to this iSCSI target (bad things can happen if r/w target access is given to multiple clients). Targets with sessionup indicating true cannot be deleted.

7. `sizeinGB`: This is the requested storage size in GB – this is the size of the underlying LV backing the target.

8. `targethost`: Targethost is the DNS name or IP address of the iSCSI server. If multiple IP addresses are assigned to an iSCSI server then this IP address should be the management IP address.

9. `targethost_storageip1`: The targethost storageip is the IP address to be used for the iSCSI connection.

10. `targethost_storageip2`: The targethost storageip is the IP address to be used for the iSCSI connection. This may or may not be identical to targethost_storageip1. If there are 2 different network paths to the iSCSI server then two IP addresses can be used for iSCSI multipath setups.

11. `isencrypted:` Value of 1 indicates that the underlying LV is LUKs/dmcrypt encrypted. If the is_encrypted flag is not specified during provisioning then the default is True.

CLUMP GROUPS

There may be specific instances when a user wants to force all targets from an initiator to be created on the same backend iSCSI server (somewhat opposite to an anti-affinity-group). The need for clumping all targets of an initiator arose because the Linux iSCSI client imposes a bottleneck on very high throughput in a single session and so it is advantageous to create multiple iSCSI targets and stripe (via md or lvm for example) on the client. Note that the bottleneck here is on the iSCSI client and not the server. Another reason may be to create a client-side LVM setup to non-disruptively extend block device size in the future, for example using the resize2fs command for ext2/3/4 file systems.

Using multiple underlying iSCSI block device targets introduces the problem of the client's storage becoming vulnerable to failure of more than one iSCSI server at any given time. For example, if M Cassandra nodes create such striped disks using all the N iSCSI servers then the failure of any one of these N iSCSI servers will bringing down all the M Cassandra nodes, and hence the database. On the other hand if clumpgroups are used to force all targets of an initiator to be created on the least possible number of iSCSI servers (usually 1, unless that iSCSI server runs out of resources while provisioning subsequent targets of a clumpgroup), then the failure of n out of N iSCSI servers will only knock out (n/N*M) Cassandra nodes. Clumpgroups take precedence over anti-affinity groups, meaning that if two provisioning requests specify the same clumpgroup and the same anti-affinity group, then the two requests will be provisioned on the same backend iSCSI server even though the anti-affinity group is the same.

Clumpgroups can be specified as shown in the example below; here two consecutive provisioning calls result in the creation of 2 targets on the same backend iSCSI server:

```
curl -X GET http://192.168.56.20:8000/api/provisioner/ \
>      -d "sizeinGB=1.0" \
```

```
>       -d "serviceName=mongodbbackendhost1disk1" \
>       -d "clientiqn=2014.dbhost1.ini" \
>       -d "aagroup=mongodbaagroup" \
>       -d "clumpgroup=host1" \
>       -d "isencrypted=1" \
>       -u 'testuser:password' \
> | python -mjson.tool
```

```
{

    "aagroup__name": "mongodbaagroup",
    "already_existed": 0,
    "clumpgroup__name": "host1",
    "error": 0,
    "iqnini": "2014.dbhost1.ini",
    "iqntar": "iqn.2014.01.192.168.56.21:mongodbbackendhost1disk1:7fbb4dc6",
    "isencrypted": true,
    "sessionup": false,
    "sizeinGB": 1.0,
    "targethost": "192.168.56.21",
    "targethost__storageip1": "192.168.56.21",
    "targethost__storageip2": "127.0.0.1"
}
```

Next, another provisioning call with the same clumpgroup creates another target on the same iSCSI server. Note that the serviceName is different.

```
curl -X GET http://192.168.56.20:8000/api/provisioner/ \
>       -d "sizeinGB=1.0" \
>       -d "serviceName=mongodbbackendhost1disk2" \
>       -d "clientiqn=2014.dbhost1.ini" \
>       -d "aagroup=mongodbaagroup" \
>       -d "clumpgroup=host1" \
>       -d "isencrypted=1" \
>       -u 'testuser:password' \
> | python -mjson.tool
```

```
{
```

```
    "aagroup__name": "mongodbaagroup",
    "already_existed": 0,
    "clumpgroup__name": "host1",
    "error": 0,
    "iqnini": "2014.dbhost1.ini",
    "iqntar": "iqn.2014.01.192.168.56.21:mongodbbackendhost1disk2:7fbb4dc6",
    "isencrypted": true,
    "sessionup": false,
    "sizeinGB": 1.0,
    "targethost": "192.168.56.21",
    "targethost__storageip1": "192.168.56.21",
    "targethost__storageip2": "127.0.0.1"
}
```

ENCRYPTION

The current Saturnring implementation provides "data-at-rest" encryption – the LV containing data for an iSCSI target is encrypted using dmcrypt/LUKs, unless overridden via the "`isencrypted`" opt-out switch – for example by setting "`isencrypted=0`" in the provisioning call above. Saturn encryption employs dmcrypt - a transparent disk encryption subsystem in Linux kernel versions 2.6 and later and in DragonFly BSD. It is part of the device mapper infrastructure, and uses cryptographic routines from the kernel's Crypto API. LUKs - the Linux Unified Key Setup enhancements to dmcrypt are used to manage the encrypted logical volume (LV) devices that are subsequently exported via iSCSI to client VMs. Since encryption is on a per-LV basis, the same volume group (VG) can support both encrypted and un-encrypted Saturn iSCSI targets. Therefore unencrypted and encrypted LVs can be carved off the same VG pool and there is no need to plan capacity separately.

There is only 1 master key used across all encryption targets per Saturnring cluster; this key is only visible to the storage administrators. There are dormant hooks in the code to let users provide their own keys in the future. The master key can be swapped for a user-key at that time: dmcrypt LUKs allows switching keys online and without rebuilding the encrypted block device at a later time.

ISCSI TARGET DELETION

ISCSI targets can be deleted using an API call "delete". Deletion is also possible via the portal, as illustrated in Figure XXX.

1. Delete a specified target belonging to the user

```
curl -X GET http://192.168.56.20:8000/api/delete/ \
>    -d "iqntar=iqn.2014.01.192.168.56.21:mongodbbackendhost1disk1:7fbb4dc6" \
```

```
>    -u 'testuser:password' \
> | python -mjson.tool


{
    "detail": "{}",
    "error": 0
}
```

2. Delete all targets assigned to a specified initiator belonging to the user

API call takes initiator iqn and user authentication credentials as input. The initiator iqn is the one specified while provisioning the storage. Use this with care – it deletes all iSCSI targets belonging to the user with the specified initiator iqn.

```
curl -X GET http://192.168.56.20:8000/api/delete/ \
>     -d "iqnini=testclient" \
>     -u 'testuser:password' \
> | python -mjson.tool
 {
    "detail": "{}",
    "error": 0
}
```

3. Delete all targets on a specified iSCSI server belonging to the user

API call takes iscsi host name and user authentication credentials as input. Use this with care – it deletes all iSCSI targets belonging to the user on the specified iSCSI target host.

```
curl -X GET http://192.168.56.20:8000/api/delete/ \
>     -d "targethost=192.168.56.21" \
>     -u 'testuser:password' \
> | python -mjson.tool
{
    "detail": "{}",
    "error": 0
}
```

4. Deleting all targets for a specified initator created on a specified iSCSI server is also possible. This example also illustrates another pattern of working with the API that may be more applicable to script automation.

```
#User defines these variables
####################################################
SATURNRINGUSERNAME="fastiouser"

SATURNRINGPASSWORD="fastiopassword"

TARGETHOST="192.168.61.21"

####################################################
IQNINI=`cat /etc/iscsi/initiatorname.iscsi | grep ^InitiatorName=  | cut -d=
-f2`

RTNSTR=$( curl -s -X GET "${SATURNRINGURL}" --user
"${SATURNRINGUSERNAME}":"${SATURNRINGPASSWORD}" --data
clientiqn="${IQNINI}"'&'targethost="${TARGETHOST}" )

echo $RTNSTR | python -mjson.tool
```

CLUSTER STATISTICS

An Excel file containing cluster statistics can be downloaded by pointing the browser address bar the following API URL:

```
curl -X GET http://192.168.56.20:8000/api/stats/
```

This multi-tab Excel workbook contains cluster, iSCSI server and user-based storage and quota statistics.

TROUBLE-SHOOTING

Keywords to read-up on the Internet: Redis server, Python-rq, Django-rq, supervisord, mod-WSGI+Django, Django development web server, open-iscsi, iscsiadm, scstadmin

MANAGEMENT PLANE: SATURNRING

FIGURE 22: LOGGING SERVER AT HTTP://<SATURNRING>:9021

LOG FILES

Log files are important information-gathering tools to debug saturning issues. The Vagrant installation puts the logs in the location `/nfsmount/saturnring/saturnringlog`. The saturn.log contains the log of the master process running the saturnring server. The master process delegates work to multiple worker processes Multiple files named `rqworker-#.log` contain each worker's logs. The Apache2 access.log and error.log are also available in this directory. Tailing the logs while debugging Saturnring is very useful in pinpointing and correcting errors. The last 2000 log entries are also viewable in a web-browser via the logserver web-GUI interface (http://saturnringserver.example.com:9021), a screenshot of which is shown in Figure 22. The logserver is an independent python program to which Saturnring's socket log handlers write to (code @ ssddj/logserver/logserver.py). It is also controlled by the supervisor daemon (see discussion on worker queue processes below).

WORKER QUEUE PROCESSES

Saturnring delegates long-lasting tasks to a group of worker processes. These tasks usually take several seconds to complete. Delegation frees up Saturnring to attend to other tasks. iSCSI target creation and deletion are examples of such tasks. Given the independent architecture of Saturnring's iSCSI servers it is possible to execute parallel tasks on different iSCSI servers. For example a target can be deleted on one iSCSI server and another target can be created on another iSCSI server concurrently. Different worker processes can control this parallel execution.

The internal design of Saturning sets up one redis queue per worker process and all delegated operations on an iSCSI server are assigned to the same queue. Note that more than one iSCSI server may share the same queue so if there are many more iSCSI servers than worker queues then it may make sense to increase the worker queue count in the saturn.ini file (`numqueues` parameter).

Sometimes, worker processes and the queues they service can get stuck and die, resulting in Saturnring being unable to provision/delete storage or even update statistics about the cluster. It then becomes necessary to restart the work processes and the redis-queues if necessary.

The saturnring GUI can show the status of redis-queues. After logging in as admin in the saturnring portal, enter the URL the browser address bar: <saturnringURL>/django-rq. This will yield a screen like the one shown in Figure 23. The table has 3 columns corresponding to the queue name, the number of jobs currently in the queues, and the number of worker processes servicing each queue. Only queue# should have exactly 1 worker assigned to them, and the number of jobs should be 0 or reduce in number as time progresses (hit the browser refresh button). If the number of "failed" queue entries is greater than 0 then clicking on it will yield a list of failed jobs. Clicking on any failed job will show the relevant exception that caused the job to fail. Note that each job is just a python code segment being executed by the worker process.
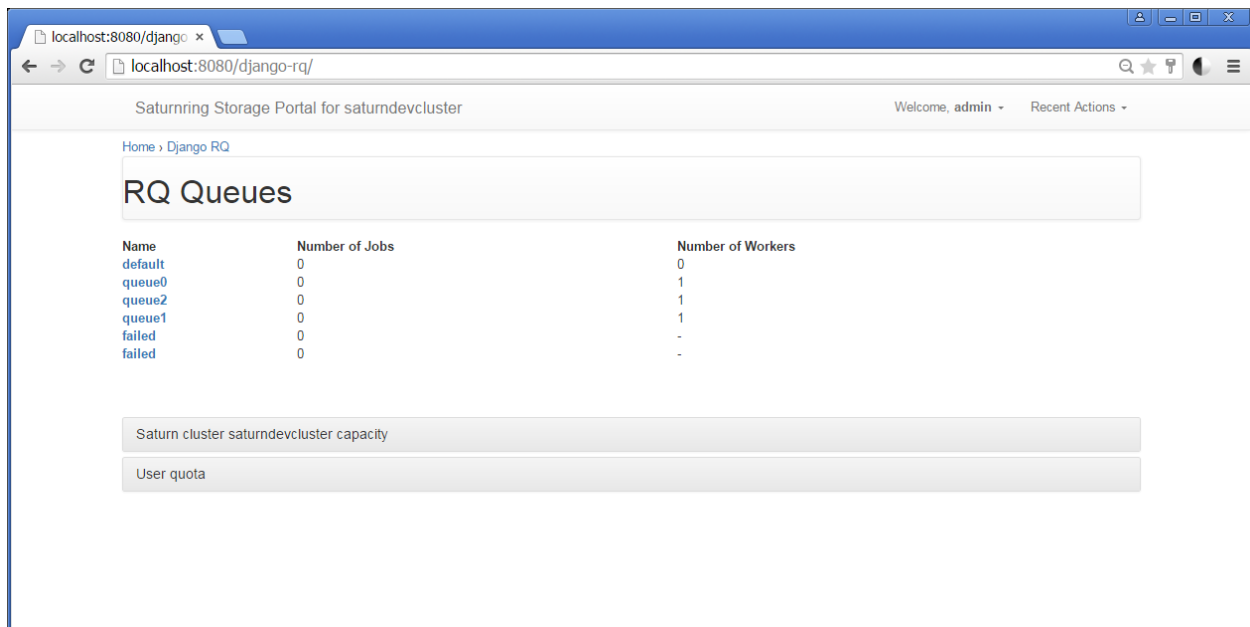
FIGURE 23: REDIS QUEUE STATUS

If something is amiss, log into the saturnring server as a sudo user (e.g. vagrant in the example) and check the state of the worker processes.

The worker processes are controlled via supervisord.

```
vagrant@saturnring:~$ sudo supervisorctl
django-rqworker-0              RUNNING    pid 10656, uptime 20:19:50
django-rqworker-1              RUNNING    pid 10655, uptime 20:19:50
django-rqworker-2              RUNNING    pid 10657, uptime 20:19:50
supervisor>
```

In case any of the processors has exited, try to reload the processes from the supervisor prompt

```
supervisor> reload all
```

45

Sometimes the processes have to be manually killed and then reloaded in supervisorctl. The rqworker process is started via the script named "rqworker.sh" (in the saturnring VM as `/vagrant/redisqconf/rqworker.sh` in the Vagrant example).

```
vagrant@saturnring:~$ more /vagrant/redisqconf/rqworker.sh
#!/bin/bash
source /vagrant/saturnenv/bin/activate
python /vagrant/ssddj/manage.py rqworker $1
```

Supervisor starts (numworkers=3) workers when Saturnring starts

vagrant@saturnring:~$ sudo ps -aux | grep rqworker

vagrant@saturnring:~$ sudo ps -aux | grep rqworker

vagrant   11048   0.1   2.8 127740 28664 ?   S   19:22   0:02 /vagrant/saturnenv/bin/python /vagrant/ssddj/manage.py rqworker queue1

vagrant   11049   0.1   2.8 127740 28580 ?   S   19:22   0:02 /vagrant/saturnenv/bin/python /vagrant/ssddj/manage.py rqworker queue0

vagrant   11051   0.1   2.8 127740 28580 ?   S   19:22   0:02 /vagrant/saturnenv/bin/python /vagrant/ssddj/manage.py rqworker queue2To restart the workers follow the commands marked in bold.

```
vagrant@saturnring:~$ sudo ps -aux | grep rqworker
vagrant   11048   0.1   2.8 127740 28664 ?              S       19:22      0:02
/vagrant/saturnenv/bin/python /vagrant/ssddj/manage.py rqworker queue1
vagrant   11049   0.1   2.8 127740 28580 ?              S       19:22      0:02
/vagrant/saturnenv/bin/python /vagrant/ssddj/manage.py rqworker queue0
vagrant   11051   0.1   2.8 127740 28580 ?              S       19:22      0:02
/vagrant/saturnenv/bin/python /vagrant/ssddj/manage.py rqworker queue2
vagrant   11619   0.0   0.0  10460    936 pts/0     S+      19:59      0:00 grep --
color=auto rqworker
vagrant@saturnring:~$ sudo supervisorctl
django-rqworker-0                 RUNNING    pid 11049, uptime 0:37:30
django-rqworker-1                 RUNNING    pid 11048, uptime 0:37:30
django-rqworker-2                 RUNNING    pid 11051, uptime 0:37:30
logserver                         RUNNING    pid 11050, uptime 0:37:30
supervisor> stop all
django-rqworker-1: stopped
django-rqworker-0: stopped
logserver: stopped
django-rqworker-2: stopped
```

```
supervisor> status
django-rqworker-0                STOPPED    Jun 23 08:00 PM
django-rqworker-1                STOPPED    Jun 23 08:00 PM
django-rqworker-2                STOPPED    Jun 23 08:00 PM
logserver                        STOPPED    Jun 23 08:00 PM
supervisor> exit
vagrant@saturnring:~$ vagrant@saturnring:~$ sudo ps -aux | grep rqworker
<empty>
vagrant@saturnring:~$ sudo supervisorctl
django-rqworker-0                STOPPED    Jun 23 08:00 PM
django-rqworker-1                STOPPED    Jun 23 08:00 PM
django-rqworker-2                STOPPED    Jun 23 08:00 PM
logserver                        STOPPED    Jun 23 08:00 PM
supervisor> start all
django-rqworker-1: started
django-rqworker-0: started
logserver: started
django-rqworker-2: started
supervisor>
```

Sometimes the "failed" queues become very large because the per-minute polling of each iSCSI server is failing (for example, because the SSH key no longer works). Removing all these entries in the failed queue (and in fact removing all entries from all queues) can be implemented by purging the redis-server's database. (Note: All jobs sent to the queue have a timeout of 45 seconds, so unless the jobs are queuing up at a faster rate and failing, these queues should clear out on their own).

In the Vagrant example, log into saturnring as a sudo user (vagrant) and then issue these commands. Note that the supervisor-controlled worker processes are also restarted.

```
vagrant@saturnring:~$ sudo redis-cli
127.0.0.1:6379> flushall
OK
127.0.0.1:6379> exit
vagrant@saturnring:~$ sudo service redis-server restart
Stopping redis-server: redis-server.
Starting redis-server: redis-server.
vagrant@saturnring:~$ sudo supervisorctl
django-rqworker-0                EXITED     Apr 30 03:46 PM
django-rqworker-1                EXITED     Apr 30 03:46 PM
```

```
django-rqworker-2                    EXITED      Apr 30 03:46 PM
supervisor> start all
django-rqworker-1: started
django-rqworker-0: started
django-rqworker-2: started
supervisor> status
django-rqworker-0                    RUNNING     pid 20596, uptime 0:00:07
django-rqworker-1                    RUNNING     pid 20595, uptime 0:00:07
django-rqworker-2                    RUNNING     pid 20607, uptime 0:00:06
supervisor>
```

Important note: The above reset of the queue infrastructure is necessary when say, a number of update jobs fail or get queued up when an iSCSI server dies (so the periodic updates don't succeed). Since queues are shared, not clearing the queue may result in other servers' jobs being held in pending state.

LOCKS

Some sections of Saturnring code are "critical sections". For example the anti-affinity and VG choice algorithm needs to process requests serially so that the information about capacity in each VG is accurate when determining where to place an iSCSI target. The model corresponding to a VG is placed in a "locked state" as shown in Figure 24. If a provisioning request fails, these locks may not be released. An administrator can navigate to Home > Ssdfrontend > Vgs and check for any "Is locked" column entries checked green. Clicking on the corresponding VGuuid and unchecking the "Is locked" flag will allow saturnring to provision on this VG again. It is prudent to look at the saturnring logs to find out what caused the provisioning code to error that lead to the VG being left in a locked state.
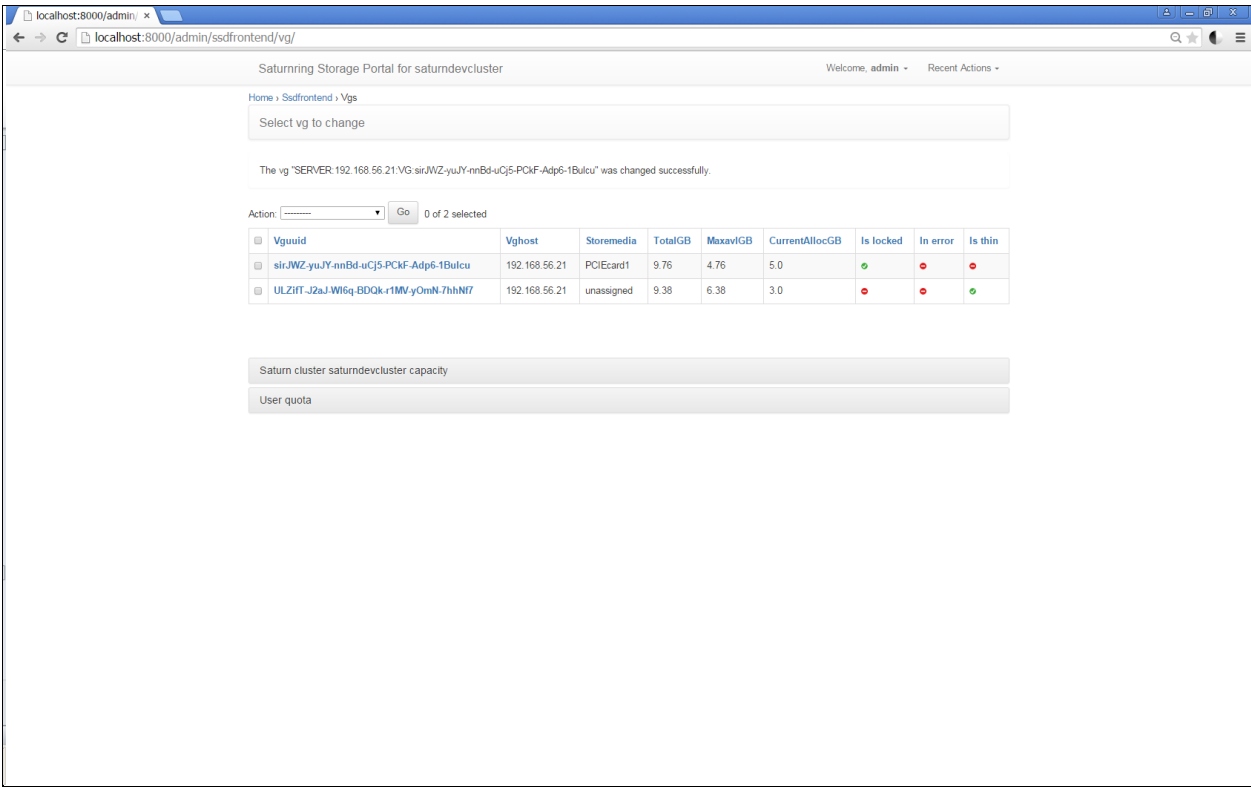
Saturnring Guide

FIGURE 24: VOLUME GROUPS AND LOCKS

The astute administrator may have noticed the "Locks" link in the main dashboard when the admin user logs in. There is one "allvglock" visible by navigating to Home > Ssdfrontend > Locks. This lock is used in a small section of the provisioning code. If the provisioning code breaks during this time the allvg lock will remain in a "Locked" state and subsequent provisioning calls will block. This can be fixed by clicking on "allvglock" and releasing the lock and saving it, as shown in Figure 26. In the passing, this "allvglock" checkbox can be used to disable provisioning all across the cluster. Users' provisioning requests will queue up while this lock is in place.
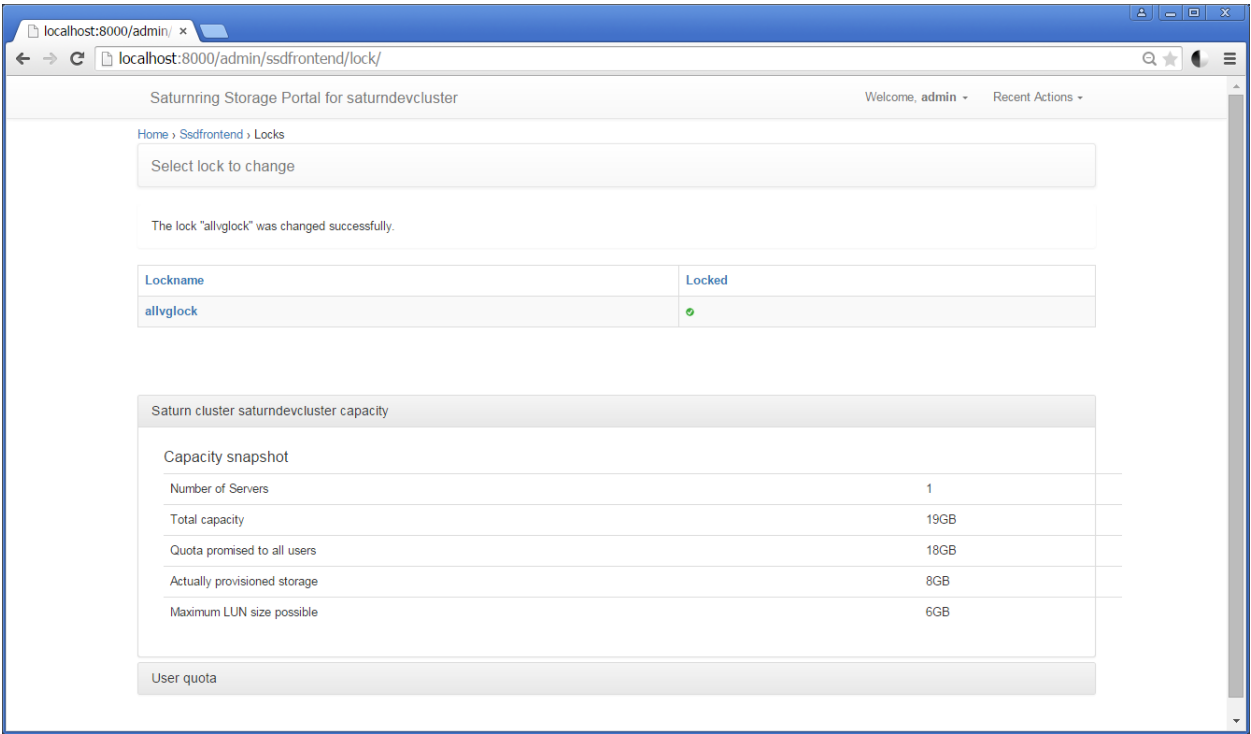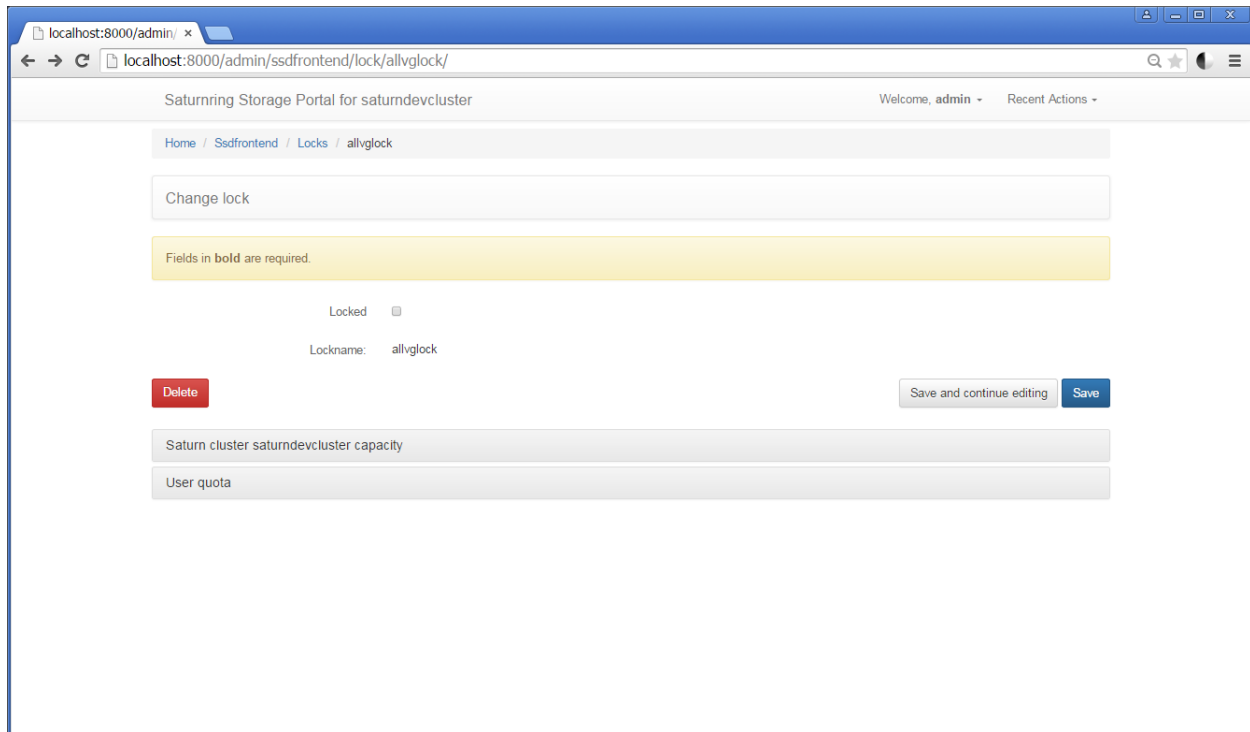

FIGURE 25: GLOBAL LOCK

FIGURE 26: TOGGLING THE GLOBAL LOCK

WEB SERVER & DJANGO ADMIN

The Django application and API are implemented with an Apache2 + mod-WSGI extension for Django/Python in the Vagrant example. For debugging Django also provides a standalone webserver. This can be run to determine issues that may not be obvious when running the Apache2 server to serve the application. To run the Django webserver (this can be done while the apache2 server is also running)

1. Log into the saturnring VM and change to the code directory.
    ```
    cd /vagrant/ssddj
    ```
2. Setup the python virtual environment
    ```
    source ../saturnenv/bin/activate
    ```
    When this command executes successfully it will prepend (saturnenv) to the bash prompt. This indicates that all the modules needed by the application are available to Python now.
3. Now start up the Django webserver and look at the scrolling output for any errors. This command will run a webserver on port 8000. The Django admin portal should be available and so should the API (all on port 8000 though)
    ```
    python manage.py runserver 0.0.0.0:8000
    ```

Note: the django-admin command can be used to perform several advanced operations like database migrations, resetting admin superuser passwords etc. More information is available by typing

51

```
python manage.py
```

in the python virtual environment.

Note that the Apache2 process needs to be restarted when there is a code change. The Django webserver automatically picks up code changes. But it is only meant for non-production debugging.

DATA PLANE: ISCSI

iSCSI trouble shooting is a complex topic, here are some pointers to get started.

1. Read and understand the man pages for the SCST iSCSI server interface utility scstadmin and the client iSCSI interface utility (iscsiadm in case of open-iscsi).
2. Look at the kernel logs (e.g. dmesg) and look for iSCSI errors in the log on both the iSCSI client and the iSCSI server. Cutting and pasting error messages in a search engine is often the quickest way to find out what may be amiss.
3. iSCSI is a network block storage protocol – look for any networking issues
4. Always back up data and/or replicate; if data is important then it should be replicated on physically separate iSCSI servers using the anti-affinity saturnring mechanisms.

APPENDIX

EXAMPLE OF A SATURN.INI FILE

```
[saturnring]

#Cluster name used in the GUI and XLS stats file output

clustername=saturndevcluster


#Location where the Saturnring picks up scripts to install

#on the iscsi server(saturnnode) for things like creating/deleting iSCSI
targets

bashscripts=globalstatemanager/bashscripts/


#The user needs to copy the corresponding public key to the saturn node's
user

#(specified in the [saturnnode] section using e.g. ssh-copy-id

privatekeyfile=/nfsmount/saturnring/saturnringconfig/saturnkey


#This is where saturnring keeps the latest iscsi config file

iscsiconfigdir=/nfsmount/saturnring/saturnringconfig/


#Django secret key (CHANGE in production)

django_secret_key=pleasechangemeinproduction


#Logging path

logpath=/nfsmount/saturnring/saturnringlog

logfile=saturn.log

#Server that accepts pickled sockethandler log messages and puts them in the
log gui+saves them

logserverhost=localhost
```

```
logwebmonitorport=9021

logfilerotatesize=50000000

logfilerotatecount=10




#Number of queues. A higher number will create more worker processes;

#Useful for clusters with many iSCSI servers. Note that each iSCSI server

#is always serviced by the same worker, so increasing this number will not

#speed up a single-iSCSI server cluster. The parameter is useful when many

#iSCSI servers are serviced by the same number of limited workers/

numqueues=3



#Proxyfolder

#This is the proxy subfolder if the application is being run behind a proxy.

#Used to manage appropriate BASE URLs

proxyfolder=



#Database settings

[database]



#sqlite or postgres

dbtype=sqlite

dbname=saturntestdb.sqlite

dbdir=/vagrant/sqlitedbdir

#These parameters are only applicable to postgres

dbhost=dbhost

dbport=5432

dbuser=postgres
```

```
dbpassword=postgres


#iSCSI server settings (also referred to as saturnnode or storage host)

#All iSCSI servers are assumed to have identical configurations

[saturnnode]

user=vagrant

#Location on the saturnnode where the scripts are installed.

install_location=/home/vagrant/saturn/

bashpath=/bin/bash

pythonpath=/usr/bin/python


#LDAP/AD settings

[activedirectory]

enabled=0

ldap_uri=ldap://ldapserver.url

user_dn==OU=Users,OU=Ring,OU=ouname,DC=ad0,DC=dcname

staff_group=CN=Cloud Customers,OU-Security Groups, DC=ad0,DC=dcname

bind_user_dn=ldapreadaccount,CN=Users,DC=ad0,DC=dcname

bind_user_pw=sup3rs3cur3


#Configuration to run unit tests.

[tests]

saturnringip=192.168.56.20

saturnringport=80

saturniscsiserver=192.168.56.21
```