Saturnring User Guide

Beta document – work in progress.

Table of Contents

Why Saturnring?

Computer CPUs and memory IO speeds far outstrip disk IO speeds, especially when small blocks of data are read from random locations on the disk. This lag shows up as “iowait” time - where the CPU waits on disk IO to complete. Solid state storage addresses this problem by making random disk access fast and independent of the access sequence because no moving parts are involved. Applications can perform 10s of thousands of IO operations with low latency on a solid state disk instead of having to rely on expensive and unwieldly disk arrays for even moderate IOPs requirements.

The next question is how to make SSD-class storage available to virtual machines being spun up in a IaaS cloud. One way 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, the 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, unscalable, and error-prone. There are few storage-admins willing to sign up for that scale of fragmented-storage management.

If solid state disk is available on “storage” servers (physical or virtual machines) with low latency network connections to VMs that need it, then iSCSI block storage protocol can be used to export storage to any VM in the network. Add to this the ability to automate the iSCSI storage lifecycle – provisioning, management, and deletion – across multiple block devices on multiple storage servers, and we achieve the ability to drive up solid state utilization, the freedom move VMs across different hypervisors, the flexibility to horizontally scale “solid state” storage using cheap hardware and the pre-existing network, and the choice of selecting any suitable solid state disk (depending on e.g. the write wear of the applications).

This, in a nutshell, is what Saturnring provides.

Saturnring Architecture

Saturn relies on recent versions of Linux LVM (Logical volume manager) to divide up block device(s) into logical volumes (LVs), which are then exported via an iSCSI server (currently, SCST) as iSCSI targets. Clients of this storage - iSCSI initiators - use the corresponding LVs as block storage devices. Saturn provides mechanisms for orchestrating multiple multiple block devices on multiple target hosts. Saturn targets can be assigned anti-affinity groups (AAGs) at creation time, meaning that targets belonging to the same AAG will be placed on different target servers if possible. This is useful in controlling failure domains for applications like Cassandra, setting up RAID-1 volumes in the client VMs whose backing-targets are on different hosts, or active-active replicated storage back-ends for relational databases.

Unlike clustered data storage systems (e.g. GPFS, Gluster, CEPH etc.) Saturn makes no effort of replicating data on the back-end (apart from the hardware RAID controller on each saturn node doing a RAID 1-0 across all its drives to protect against single drive failure). There are no multiple copies being created on write. Instead Saturn defers high-availability and data protection to the application (e.g. NoSQL database replication or software RAID 1 across 2 target LVs on the initiator). Saturn's focus is on preserving low latency, high IOPs and high throughput properties of SSDs or other “fast” storage over the cloud network. Saturn's value add is manageability and RESTFul block storage sharing and provisioning amenable to cloud applications.

Terminology

iSCSI server - Physical server containing solid state media where the iSCSI LUN actually lives. Users “log in” to the iSCSI “portal” hosted on the iSCSI server via the iSCSI prototcol and access their iSCSI target(s).

iSCSI target - Unique identifier to identify and “connect” to the iSCSI LUN on an iSCSI server.

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.

iSCSI initiator - Unique identifier of an iSCSI client (a VM - consumer of the Saturn service). 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.

Components

Saturnring uses the SCST iSCSI target software on saturnring servers exporting block storage to clients. Saturnring servers use LVM to divide their underlying storage – e.g. A jbod of SSD drives – into logical volumes, each of which is exported as an iSCSI target. Logical volumes are created on a volume group (called “storevg” by default). A thin pool logical volume (called “thinpool”) is created on storevg. Logical volumes are carved off this thin pool. Thin pools are useful for thin-provisioning storage – they allow users to create LVs that are larger than the size of all available extents in the volume group, with the underlying assumption that most LVs are over-provisioned and will not actually use all the space allocated to them. Read more about LVM thin pools here: http://red.ht/T1WHsQ.

Saturnring manages multiple iSCSI servers with a single user-console and an API. By managing the creation, management and deletion of iSCSI targets across multiple servers it completely standardizes target naming, user management, and rules for choosing which iSCSI servers host which iSCSI target. Because all iSCSI resources are tracked via the saturnring portal/database, it addresses the problem of iSCSI sprawl and unreclaimed forgotten storage - the Saturnring admin has a birds eye view of every iSCSI target on every saturnring iSCSI server.

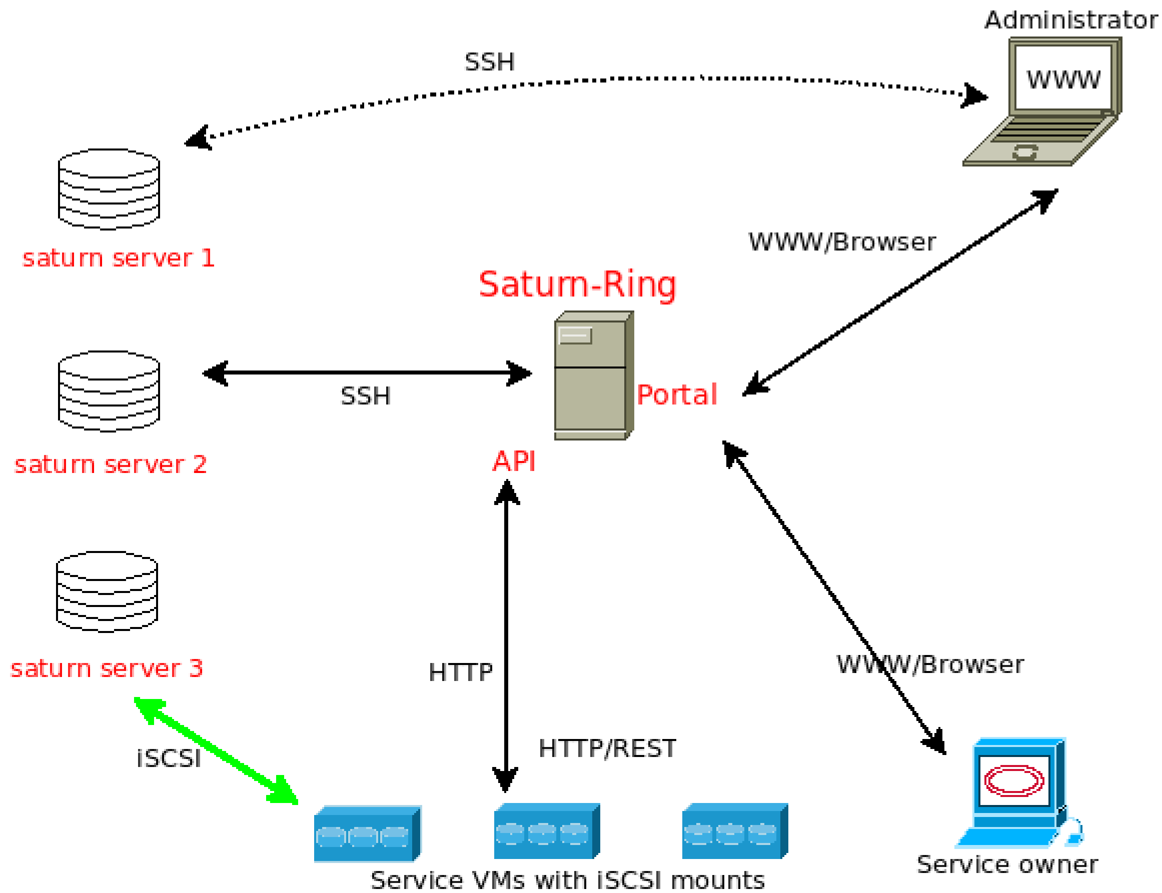

Illustration 1: Saturnring Architecture

Saturnring is implemented using the Django Python web framework. Saturnring provides the portal to manage users and storage (for users and admins). It also exposes a HTTP RESTful API to enable automatic provisioning of storage via scripts, for example, when a virtual machine is started and needs iSCSI block storage.

Saturnring controls (creation, deletion, and monitoring of iSCSI targets) saturnring iSCSI servers over SSH and using bash scripts. These code for the scripts is available at sddj/globalstatemanager/bashscripts.

Saturnring scans and updates the status of iSCSI servers via periodic (default : 1 minute) asynchronous "scans" of each server. These scans are performed via a separate redis-queue worker process (4 such processes are installed by default but this number can be adjusted for the number of iSCSI servers and the desired update frequency). A redis server is used to dispatch scanning jobs to the worker processes. A supervisord process is used to manage the workers. These status updates check which iSCSI target sessions are up and active, collect some basic data about the amount of data being written and read off each iSCSI target etc.

ISCSI targets are created with the following parameters (these can be changed by modifying ssddj/globalstatemanager/bashscripts/createtarget.sh):

```
thin_provisioned=1, rotational=0, write_through=1, blocksize=4096
```

Therefore 4K block sizes should be specified while creating filesystems on top of the target in the iSCSI initiator. The SCST blockio driver is used, along with write_through=1, and so there is no caching being used on the iSCSI server. Writes are acknowledged after the the acknowledgement from the underlying storage is received.

Installation Guide

Saturnring is built out of multiple components - the iSCSI server(s), the Django-driven Saturnring portal and API and Apache webserver with modwsgi 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 in the git distributable 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 with a little effort.

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 3 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 192.168.61.0/24 and a bridged adaptor to let VMs access the Internet.

STAGE 0: Software installation

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

   ```
   mkdir -p ~/DIRROOT
   cd ~/DIRROOT
   git clone https://github.com/sachinkagarwal/saturnring/
   ```

   Then, navigate to /saturnring/deployments/vagrant

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

STAGE 1: Bringing up Saturnring portal/API server (at 192.168.61.20)

1. 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 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
    ```

2. If all went well, you should be able to navigate to http://192.168.61.20/admin from a web brower on the VM host machine

3. Log into the Saturnring VM

    ```
    vagrant ssh saturnring
    ```

4. Activate the Saturnring Python environment

    ```
    cd saturnring
    source saturnenv/bin/activate
    ```

5. Create a Storage admin superuser

    ```
    cd /home/vagrant/saturnring/ssddj
    python manage.py createsuperuser
    ```

    (follow the prompts to setup a superuser)

6. Exit to the host and confirm that you can log into the Saturnring portal using the superuser credentials in the web browser.

STAGE 2: Bringing up the iSCSI server(s)

1. On the VM host, navigate to cd ~/DIRROOT/saturnring/deployments/vagrant

2. Bring up an iSCSI VM defined in Vagrantfile vagrant up iscsiserver1 (192.168.61.21)

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

```
vagrant ssh saturnring
cd ~/saturnring/ssddj/config
ssh-copy-id -i saturnkey vagrant@192.168.61.21
```

(password is vagrant)

4. Log into the saturnring portal as admin superuser and add the new iscsi server. For this simple example,

   ```
   dnsname=Ipaddress=Storageip1=Storageip2=192.168.61.21
   ```

   Failure to save when the “save” button is clicked indicates a problem in the configuration above steps. Saturnring will not allow a Storagehost being saved before all the config is right.

5. From the VM host or one of the above 2 fired up VMs, Make a "initial vgscan" request to the Saturnring server so that it ingests the storage made available by iscsiserver1 at IP address 192.168.61.21 (Networking is defined in the Vagrantfile)

   ```
    curl -X GET http://192.168.61.20/api/vgscan/ -d "saturnserver=192.168.61.21"
   | python -mjson.tool
   ```

   should return something like

   ```
   [
       {
           "CurrentAllocGB": null,
           "created_at": "2014-07-30T15:32:03.780Z",
           "enabled": true,
           "in_error": false,
           "is_locked": false,
           "maxthinavlGB": 8.79,
           "opf": 0.99,
           "thintotalGB": 8.79,
           "thinusedmaxpercent": 99.0,
           "thinusedpercent": 0.0,
           "updated_at": "2014-07-30T15:32:03.781Z",
           "vgfreepe": 11.0,
           "vghost_id": "192.168.61.21",
           "vgpesize": 0.0,
           "vgsize": 9.76,
           "vgtotalpe": 2499.0,
           "vguuid": "NeDQRY-J2hk-yBwC-LPbV-mRn6-fd9D-d6pete"
       }
   ]
   ```

6. Confirm in the portal (under VGs) that there is a new volume group

Repeat STAGE 2 for iscsiserver2 (192.168.61.22) if you want (it is useful to have 2 iscsiservers if you want to try the anti-affinity provisioning)

STAGE 3: Testing via an iscsi client VM (192.168.21.23)

1. Log into the Saturnring web portal as superuser and under users create an account for a test user (fastiouser/fastiopassword). Do not change the storage quota while creating the user. Make the user a staff user (checkbox) and give it permission to add, remove and modify targets.
2. On the VM host navigate to /saturnring/deployments/vagrant

   ```
   cd ~/DIRROOT/saturnring/deployments/vagrant
   vagrant up iscsiclient
   ```
3. Log into the iscsi client

   ```
   vagrant ssh iscsiclient
   ```

   There are example scripts for simple provisioning (storage-provisioner.sh), high-availability RAID-1 provisioning (high_availability_storage.sh) and Networking VLAN iSCSI setups (storage-provisioner-network.sh) in the /vagrant/clientscripts directory.
4. Edit the script storage-provisioner.sh (this script is also available in the home directory of the vagrant user) and set appropriate values for these variables. For example

   ```
   ##################################################
   SIZEINGB=1.0
   SERVICENAME="fastiorequired"
   SATURNRINGUSERNAME="fastiouser"
   SATURNRINGPASSWORD="fastiopassword"
   ANTI_AFFINITY_GROUP=${SATURNRINGUSERNAME}"unique-string"
   SATURNRINGURL="http://192.168.61.20/api/provisioner/"
   ##################################################
   ```
5. Then run the storage-provisioner.sh script

   ```
   sudo ./storage-provisioner.sh
   ```

   The output should look like

   ```
   vagrant@iscsiclient:~$ sudo ./storage-provisioner.sh
   ```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
open-iscsi is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
{
    "aagroup__name": "fastiouserdatabaselun",
    "already_existed": 0,
    "clumpgroup__name": "noclump",
    "error": 0,
    "iqnini": "iqn.1993-08.org.debian:01:6e48d074fff1",
    "iqntar": "iqn.2014.01.192.168.61.21:fastiorequired:57e626a7",
    "sessionup": false,
    "sizeinGB": 1.0,
    "targethost": "192.168.61.21",
    "targethost__storageip1": "192.168.61.21",
    "targethost__storageip2": "192.168.61.21"
}
iqn.2014.01.192.168.61.21:fastiorequired:57e626a7
192.168.61.21
192.168.61.21:3260,1 iqn.2014.01.192.168.61.21:fastiorequired:57e626a7
Logging in to [iface: default, target:
iqn.2014.01.192.168.61.21:fastiorequired:57e626a7, portal:
192.168.61.21,3260] (multiple)
Login to [iface: default, target:
iqn.2014.01.192.168.61.21:fastiorequired:57e626a7, portal:
192.168.61.21,3260] successful.
```

The script will provision an iSCSI target on one of the iSCSI servers setup in STAGE 2. An iSCSI session from the iscsiclient to an iscsi server will be created. A block device will be inserted in the client VM's /dev directory. dmesg should show the initialization details, including the name of the new block device. More information about the iscsi Session is available on the client via the command

```
iscsiadm -m session -P3
```

6. A filesystem can now be created on the device. Note that SCST is configured to export 4K block size targets. The target block device is thin provisioned, but sometimes thin provisioning's unmap doesnt play well if large files are deleted. For now, its best to create the filesystem with nodiscard options set and use fstrim for asynchronous unmapped block recovery. For example

```
sudo mkfs.ext4 /dev/sda -b 4096 -E nodiscard
sudo mount /dev/sda /mnt -o nodiscard,noatime
```

Deployment Considerations

Here are some ideas for Saturnring in production:

1. Monitoring and Alerting - consider something like Zabbix or Nagios to keep tabs on Saturnring

components

2. Configuration management (Puppet/Chef etc.) or pre-built images will reduce the pain and errors that come with managing multiple servers manually
3. SSDs wear out - they have limited PE cycles; best to keep a close eye on them
4. Saturnring uses a recent LVM2 implementation, look at its documentation for its many features
5. The Vagrant example does not patch the Linux kernel for optimal SCST iSCSI target software performance. Read more here: http://scst.sourceforge.net/iscsi-scst-howto.txt
6. A few very useful knowledge pools
    1. SCST (iSCSI target software) http://scst.sourceforge.net/
    2. Linux open iscsi client http://www.open-iscsi.org/
    3. Linux LVM http://tldp.org/HOWTO/LVM-HOWTO/
    4. <Your favorite search engine>

Portal Operations: Admin

Initial Login

Navigating to the portal address (usually http://saturnringipaddress_or_dnsname/admin) will display the screen shown in Illustration 2.

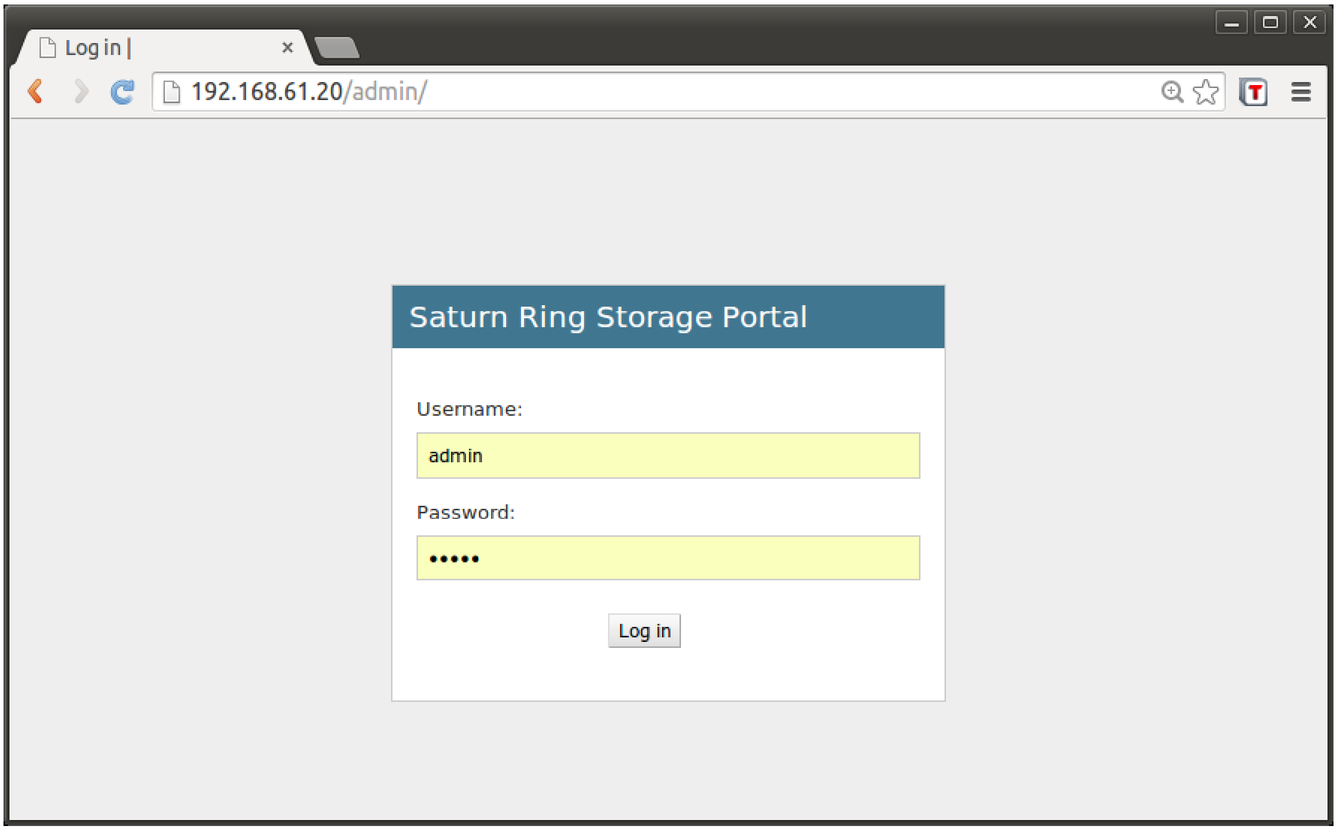

Illustration 2: Saturnring portal login page

After supplying admin/superuser credentails the screen shown in Illustration 3 will be displayed.

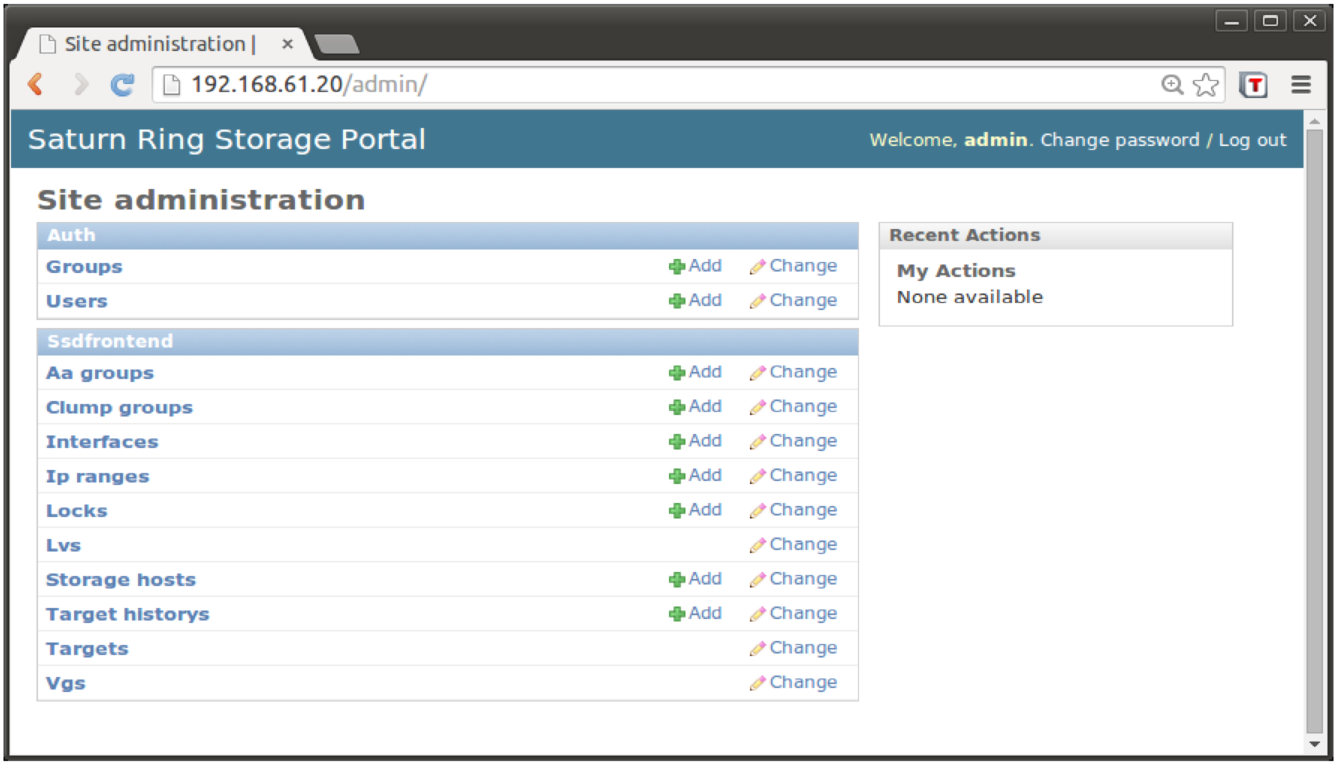

Illustration 3: Post-login screen of the superuser (admin)

There are 2 sub-categories of links. The Auth category for managing users and the SSDfrontend category for managing storage and networking. The recent actions pane shows a list of actions recently completed.

Adding an iSCSI Server

Once Saturnring is up, a new storage host (iSCSI server or Saturn server) that needs to be managed by Saturnring can be added. For this,

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

The screen shown in Illustration 4 depicts the fields needed to add a storage host. The DNS name is the network DNS name of the server; if DNS is not setup then it can also be set to the IP address of the storage host. The IP address, storageip1 and storageip2 can be set to the same interface for simple setups.

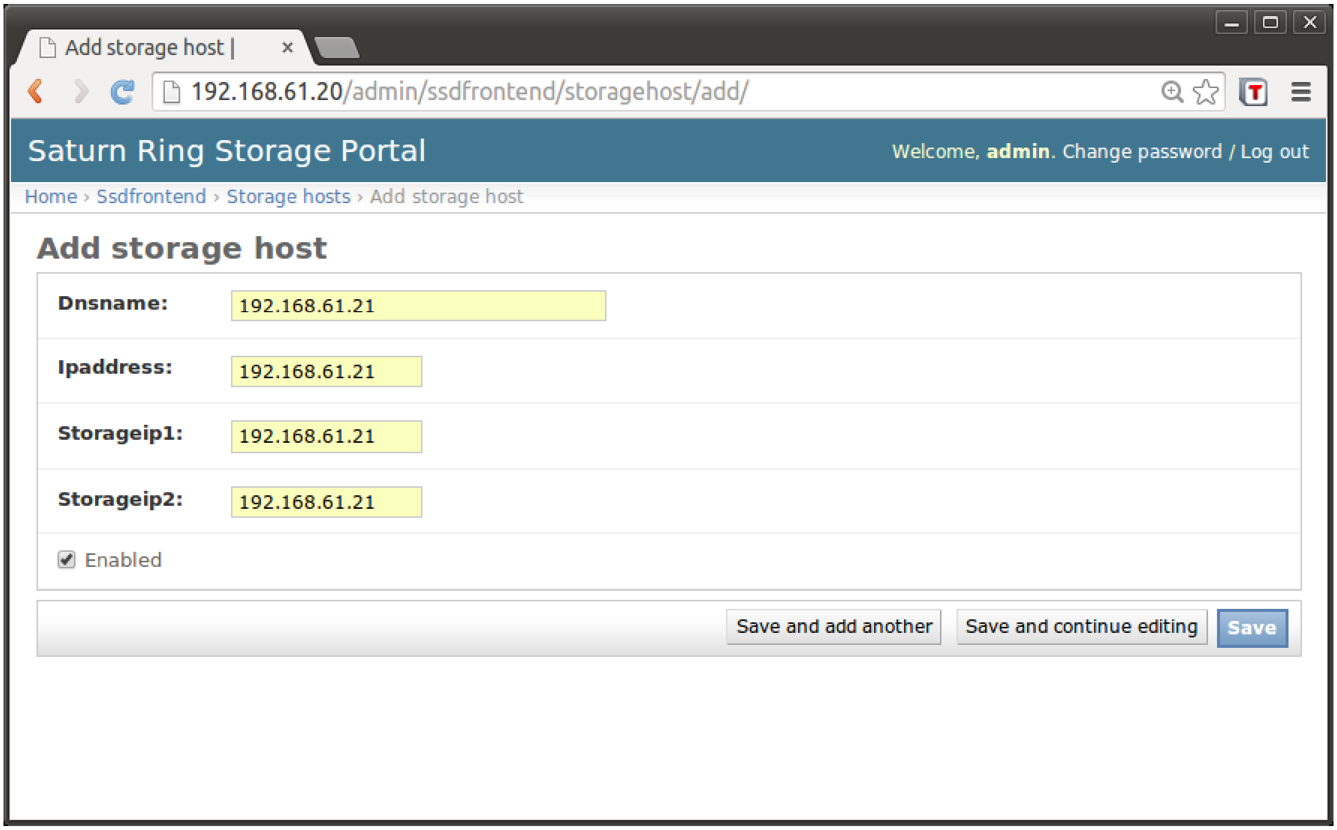

Illustration 4: Adding a new iSCSI saturnring server

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

   For example:

   ```
   vagrant ssh saturnring
   cd ~/saturnring/ssddj/config
   ssh-copy-id -i saturnkey vagrant@192.168.61.21
   ```
2. Log into the saturnring portal as admin superuser and add the new iscsi server.

   For this simple example, Dnsname=Ipaddress=Storageip1=Storageip2=192.168.61.21. Failure to save indicates a problem in the configuration steps (11-13). Saturnring will not allow a Storagehost being saved before all the config is right. This is probably a good thing.

   The storage Ips 1 & 2 can be used to specify iSCSI portals over different networks, perhaps in order to do iSCSI multipath setups etc.
3. Make a "initial scan" request to the Saturnring server so that it ingests the storage made available by iscsiserver1 at IP address 192.168.61.21 (Networking is defined in the Vagrantfile):

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

   Confirm in `Home >Ssdfrontend>Vgs` that the new volume group is now available to Saturnring; click on the new VG there.

Home > Ssdfrontend > Storage hosts should look like Illustration 5 after adding 2 iSCSI saturn servers.

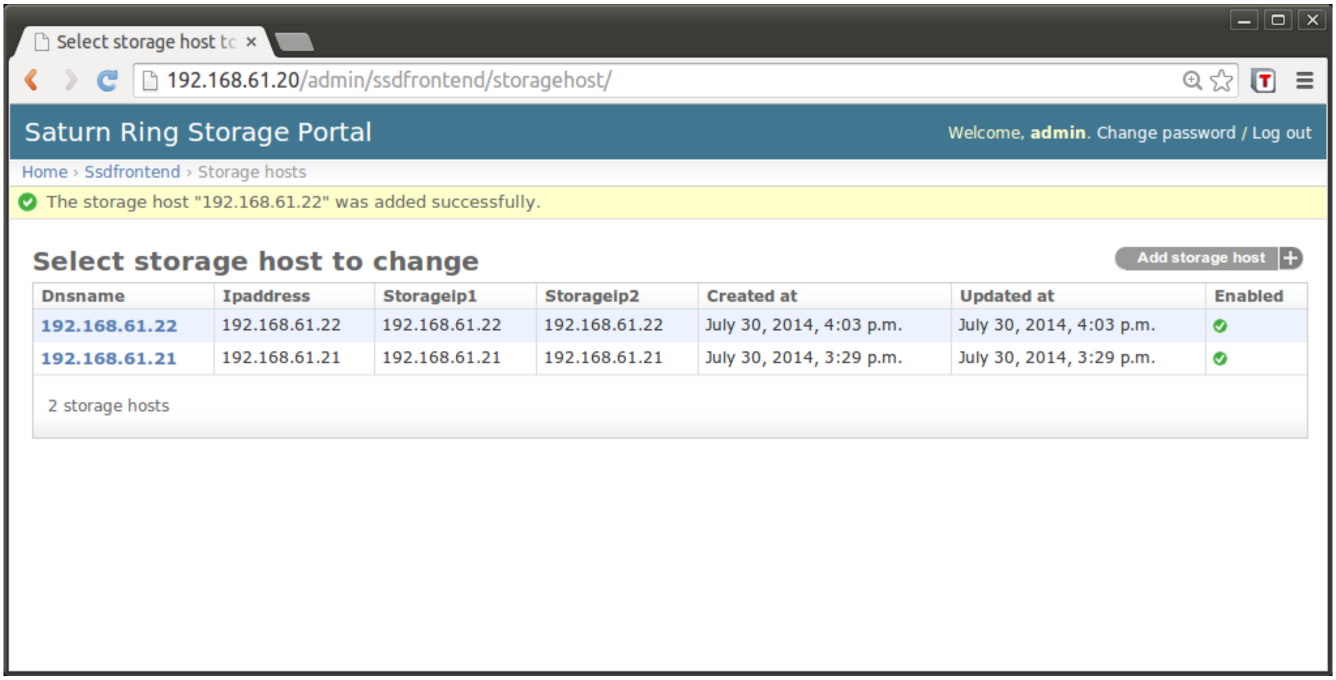

Illustration 5: List of storage hosts

Any Saturn server can be "disabled" by checking off the "Enabled" in Illustration 4. This means that users can no longer provision or delete storage targets on the disabled saturn server. The dates when the server was added to the saturnring cluster and the last date when its settings were updated (e.g. IP address) is also shown.

Volume Groups Administration

Each storage host has a volume group which owns all the physical storage inside the server. In this version of Saturnring all block devices inside a server are assumed to be equal and are added as linear physical volumes (PVs) to construct the volume group. So its not a good idea to mix device types (SSD and spinning disk for example), but its acceptable to mix different device capacities, or, even PCIe and SAS SSD disks in most cases. Home >Ssdfrontend > Vgs shows a list of Vgs (1 VG per storage host); and clicking on a VG displays a screen like shown in Illustration 6.

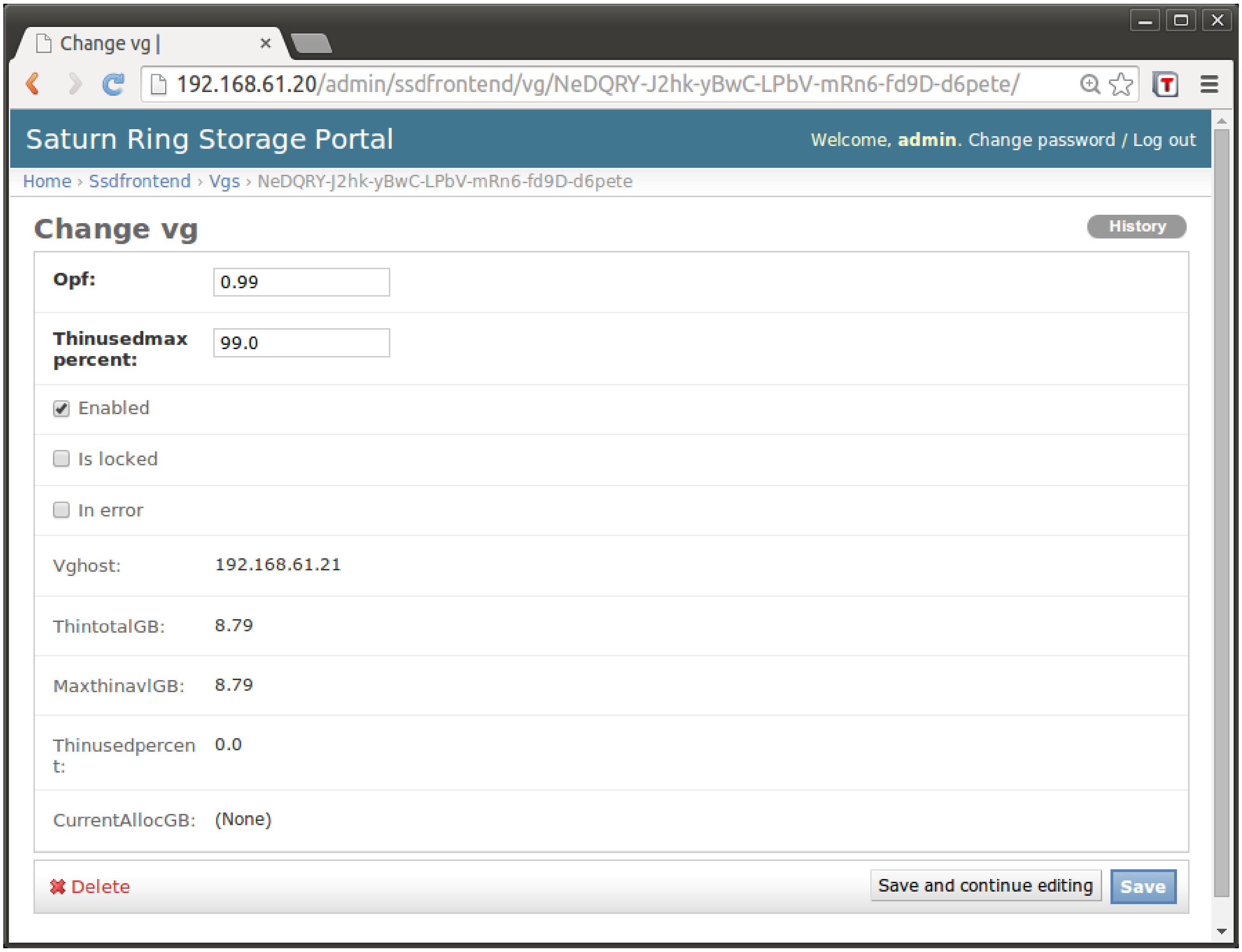

Illustration 6: Changing Volume group properties

There are 2 important properties here

1. **Opf**: This is the over provisioning factor. While using thin provisioning this floating number (0.0-) indicates how much overprovisioning will be allowed before Saturnring stops provisioning more targets on the VG. For example, if there is a 100GB volume group and opf is set to 5.0 then Saturn will allow allocation to targets totalling 5.0x100GB = 500GB. Off course the underlying assumption is that the actual storage used is less than 100GB.

2. **Thinusedmaxpercent**: This property is the percentage of actual storage blocks used. As soon as more than this percent of blocks are used, Saturnring will stop provisioning more targets on the VG. In the above example, with thinusedmaxpercent set to 70%, Saturnring will not provision more targets on the VG if more than 70GB is actually used (summed over all targets previously

provisioned on the VG).

Thin provisioning should be carefully considered (what if the overprovisioned targets need more than the available actual storage?) .

Under normal (error-free) operation only the “Enabled” checkbox should be checked. If there are errors then Saturnring may set the “locked” and “in error” checkboxes. The admin can uncheck these after carefully reviewing the logs to determine root cause of the errors/locking conditions.

Here is a screenshot of Home > Ssdfrontend> Vgs after adding a couple of storage hosts and running initial VG scans.

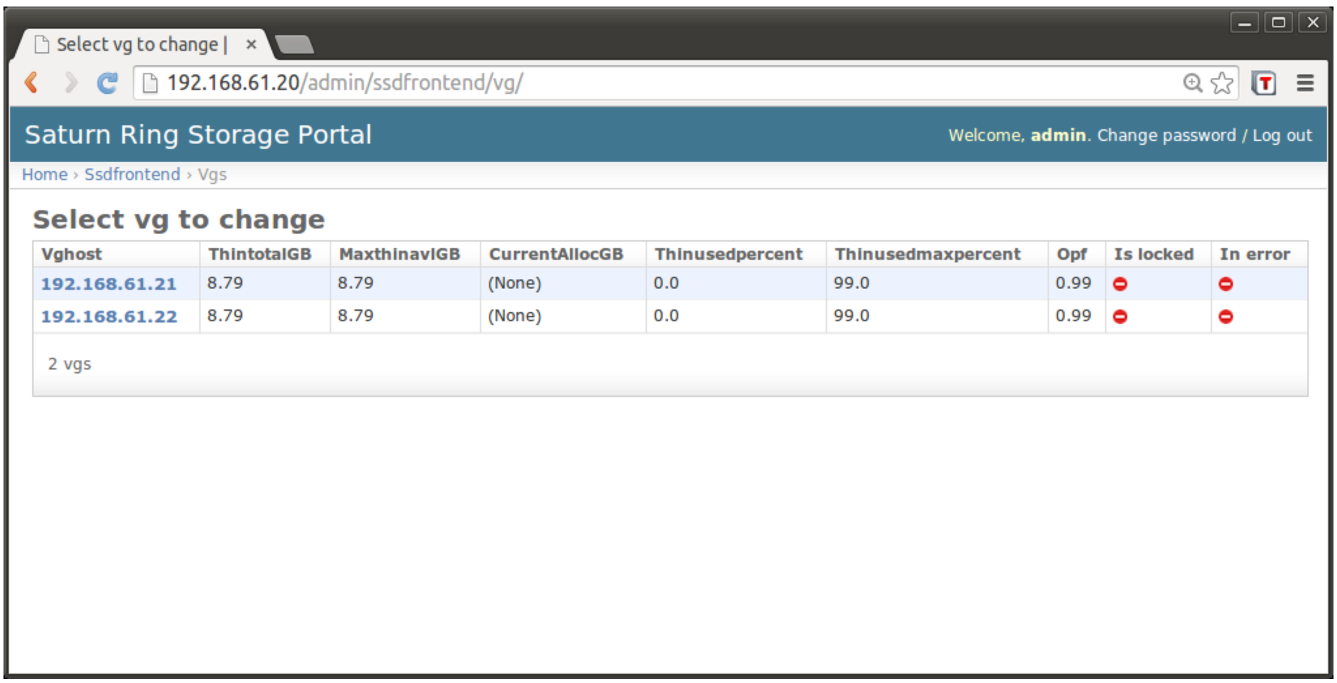

Illustration 7: Volume group list

User Administration

Saturnring empowers every user to create and delete block storage on-demand. This is the key value-add as compared to traditional enterprise arrays because the role of the storage administrator is limited to keeping the system running. The task of managing iSCSI target block devices is delegated to users.

If an external authentication source (such as active directory or LDAP) is used for authentication (discussed in detail in Authentication Against Active Directory/LDAP) then Saturnring creates a corresponding user object in its database the first time a user tries to log into the Saturnring portal. The administrator can subsequently view the user's "local copy" of the account and make quota and permission changes as described below. The discussion about creating new users only applies when creating new users via the in-built authentication system.

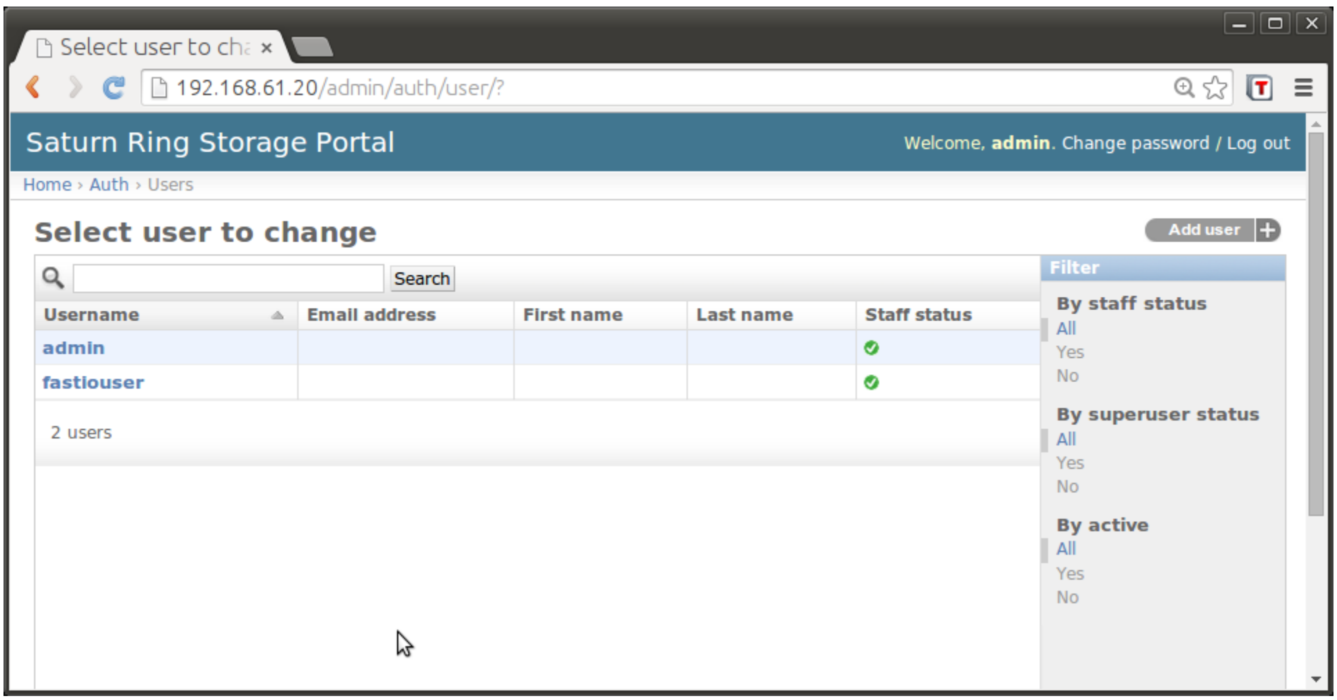

Illustration 8: User list

Users are created with attributes – username, password, email address, etc. A quota property is attached to each user. The storage administrator can set the maximum allowed provisioning (in GB) for each user account.

The admin can perform user management by navigating to Home>Auth>Users. Users may be added/deleted, passwords can be reset, quotas can be changed etc.

A new user is added using the “Add user” button at the top right of the page. Fig. 7 shows a new user being added. Please do not change the quotas in this screen (leave the defaults), quotas are correctly changed in the next screen (Fig. 8).

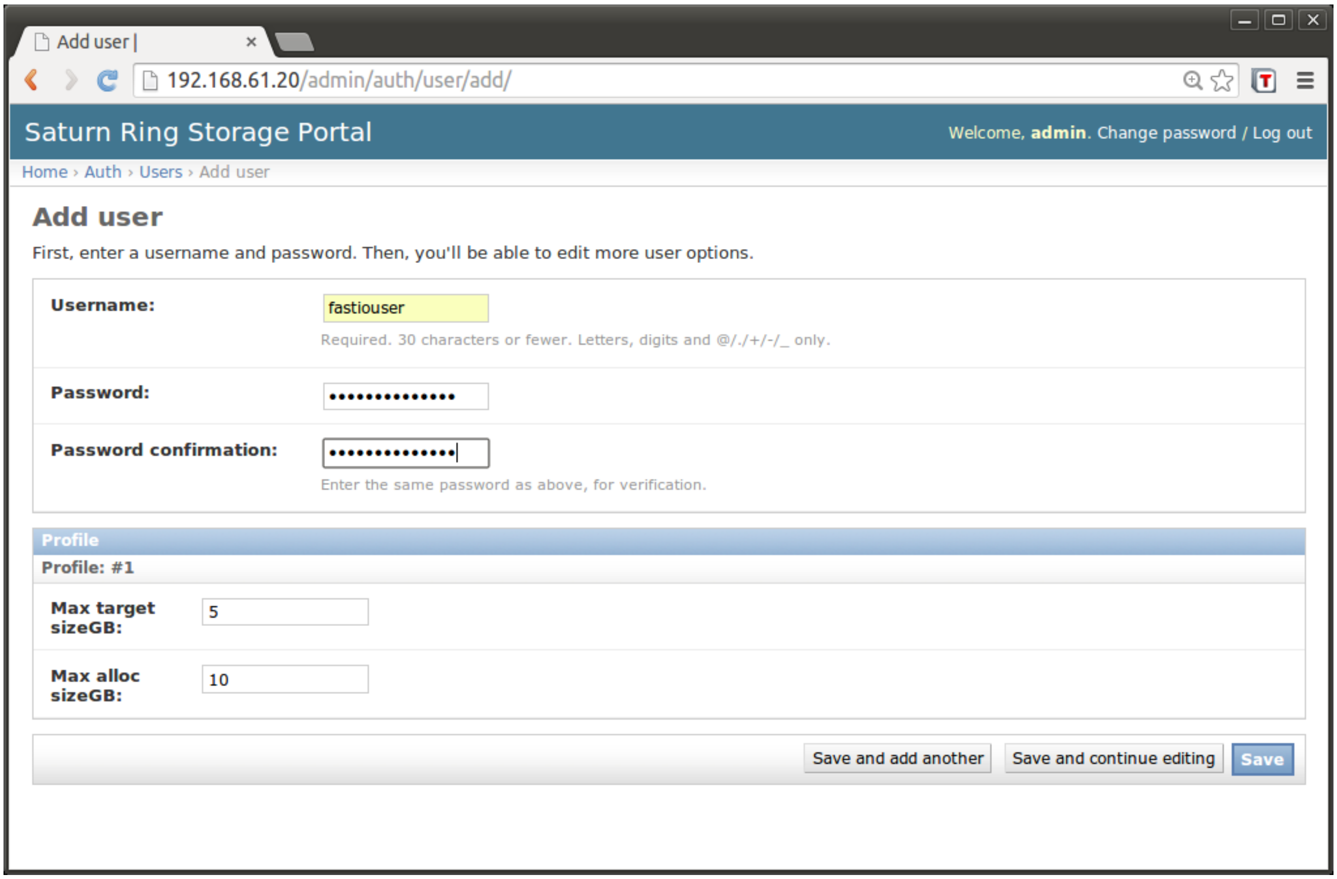

Illustration 9: Adding new user, note - quotas should not be changed in this screen!!

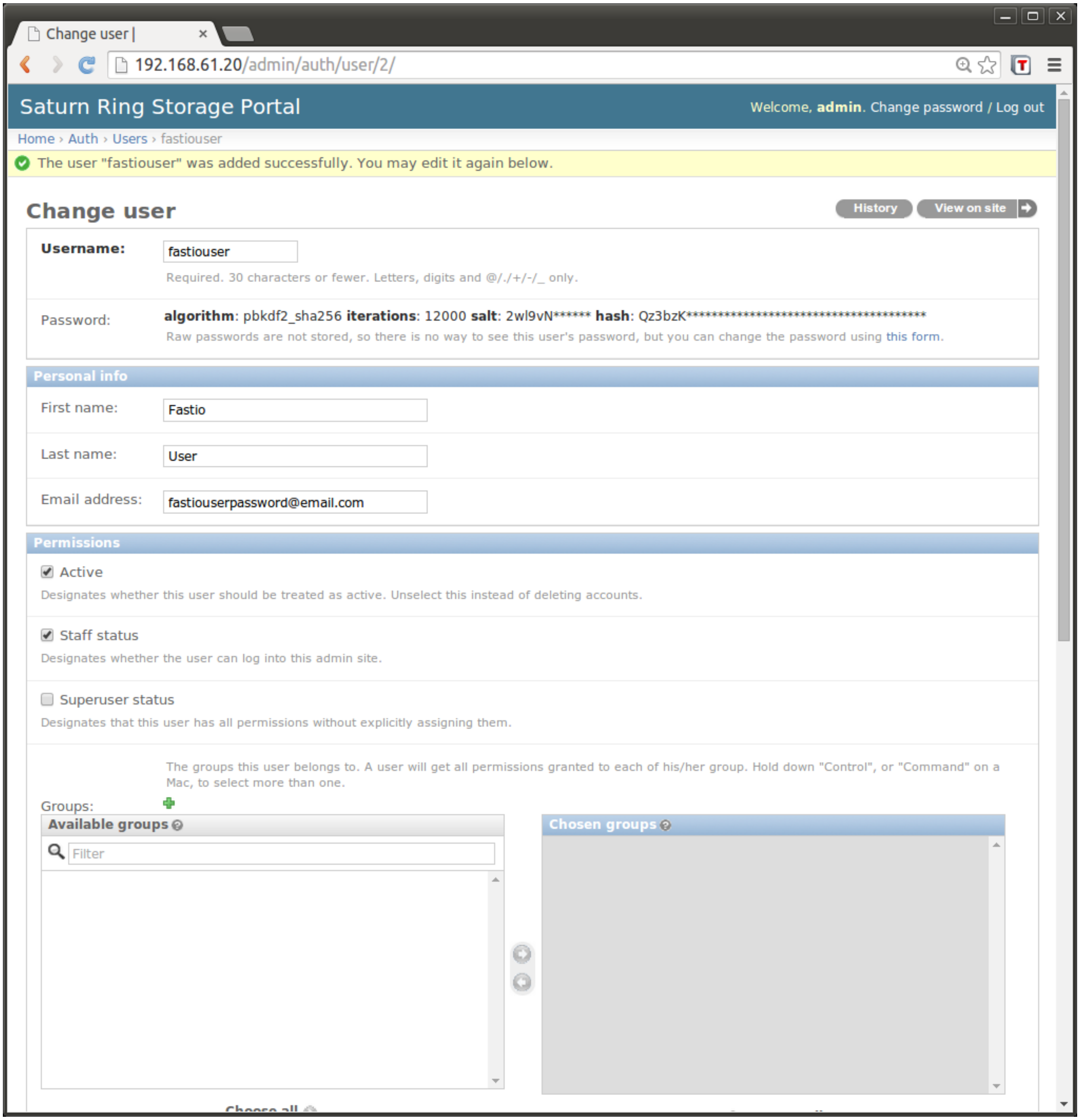

Illustration 10: User properties #1

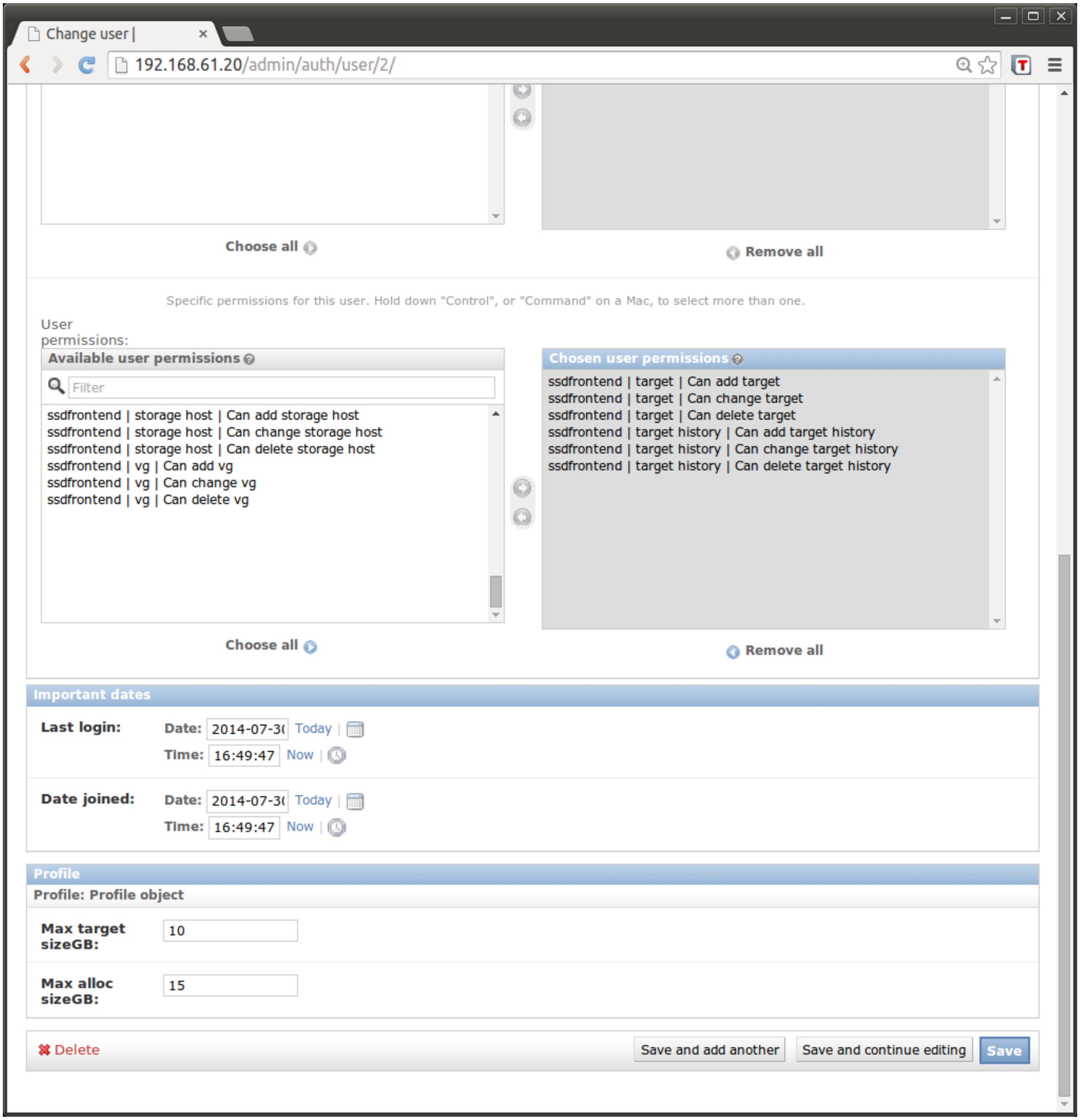

Illustration 11: User properties #2

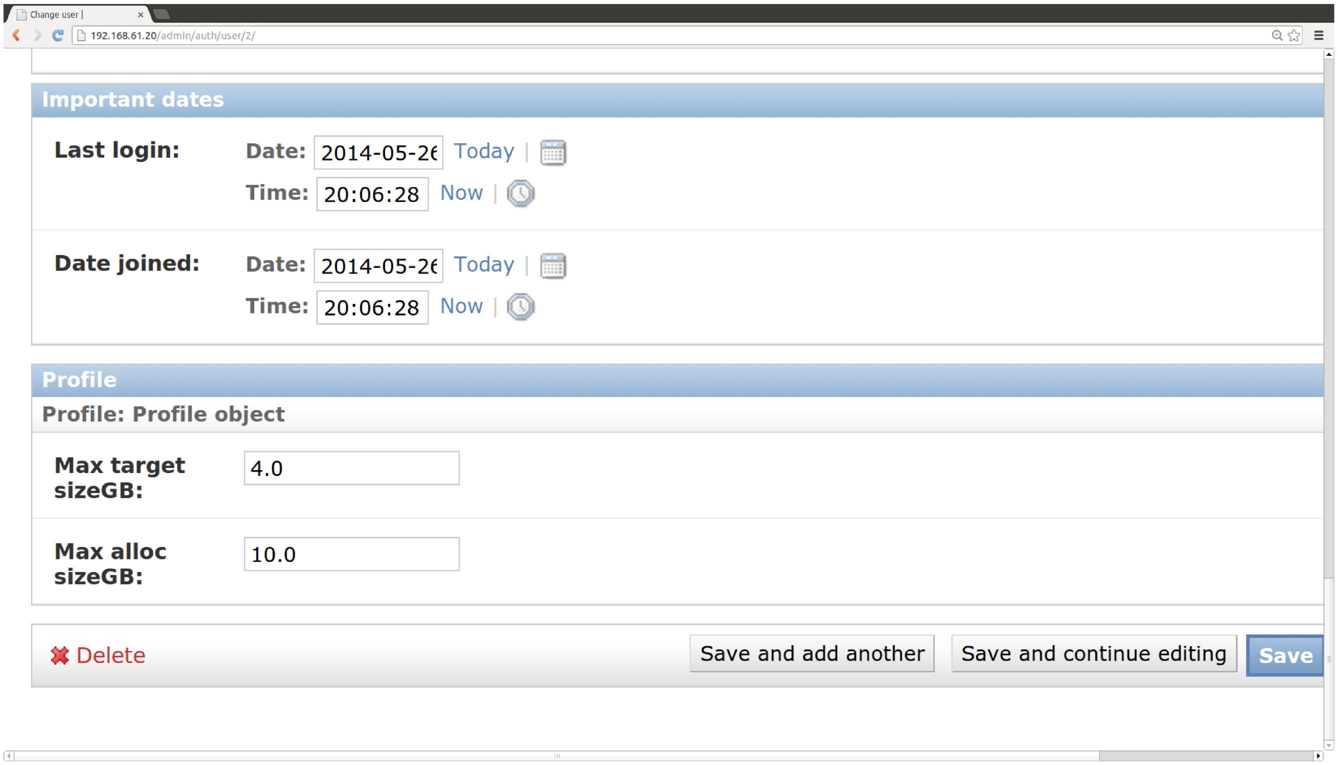

Illustration 12: Setting quotas

Illustration 12 shows a zoomed-in version of Illustration 11 showing how quotas can be managed. There are two parameters here. Max target sizeGB caps the maximum size of any iSCSI target the user can request from Saturnring whereas the Max alloc sizeGB is the total storage a user can allocate, across the Saturnring cluster (sum of all users' targets).

Cluster Statistics

Accessing this portal URL returns an XLS (Microsoft Excel) file with statistics about storage, users, quotas and targets.

```
http://<portal-ip>/api/stats, for example

http://192.168.61.20/api/stats
```

In the current version there is no access control; anyone with that URL can get the statistics excel worksheets.

Portal Operations: User

A user may log into the portal using her credentials. The user gets to see the full list of owned targets, and can select multiple targets to delete via the portal. The user also sees details about deleted targets since the account was created.

After entering user credentials on the portal login page (Illustration 2), the initial screen has a link to Targets provisioned and owned by the user and another link to target historys – targets created and then deleted by the user, as shown in Illustration 13 .

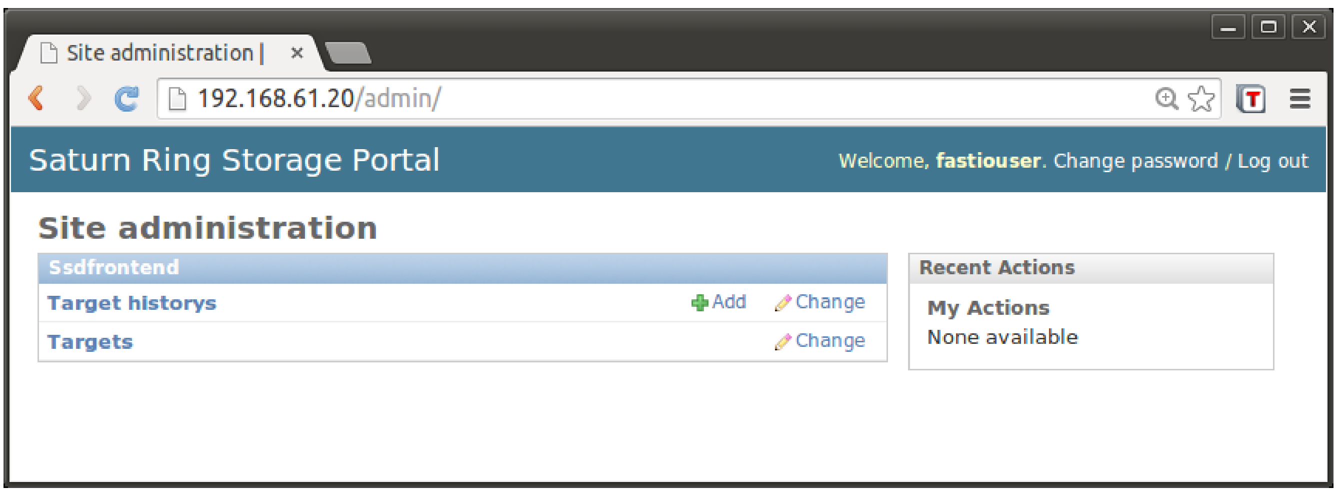

Illustration 13: User logged-in view

Target Administration

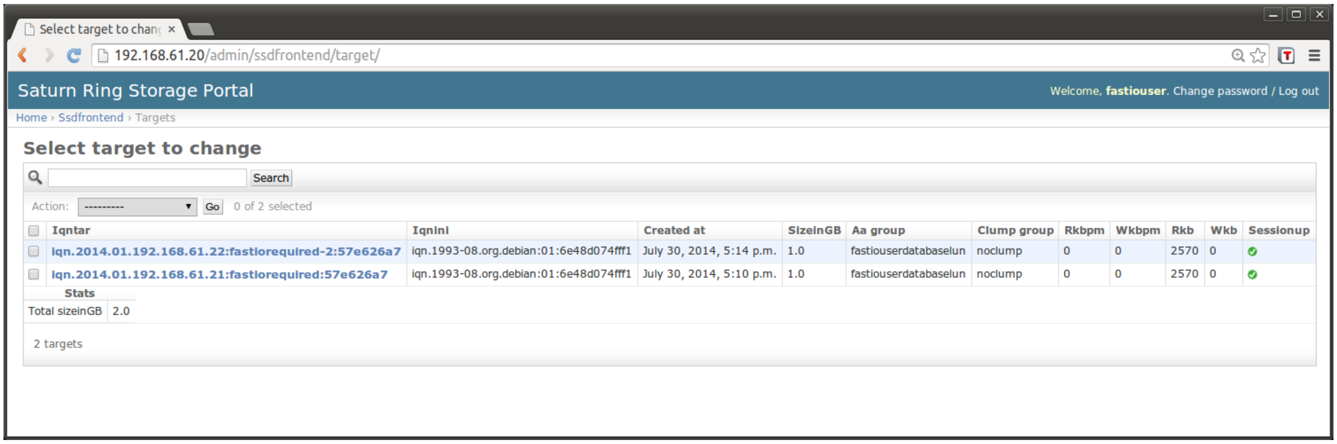

Fig. 5: Targets (admin view)

By navigating to Home>Ssdfrontend>Targets the user can get a bird's eye view of all the users targets across all iSCSI servers provisioned in the iSCSI system.

Clicking on any of the target IQNs will show the properties of that target as shown in Illustration 14.

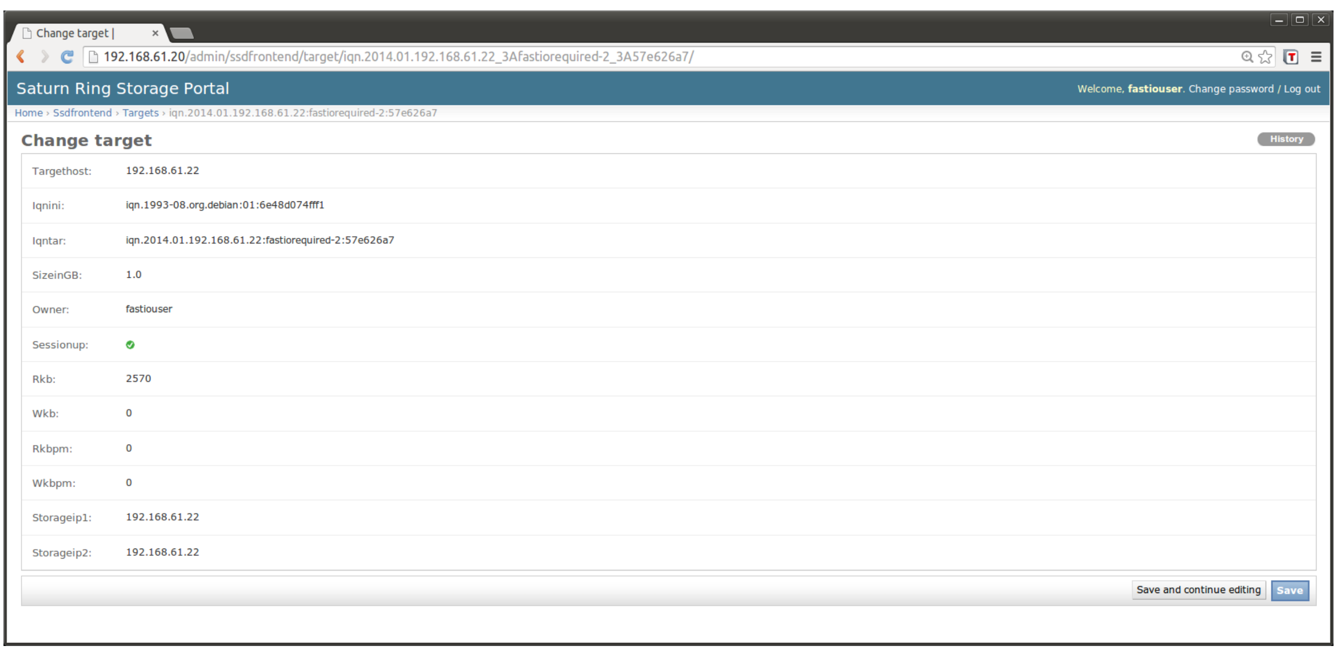

Illustration 14: iSCSI Target properties

There is no current functionality attached to “saving” a target; all fields displayed are read-only.

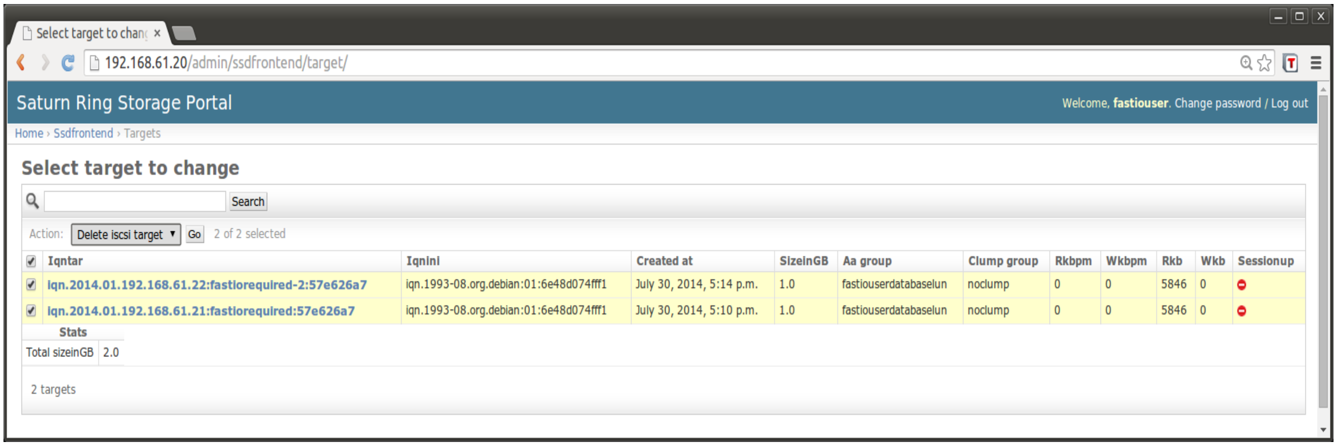

Illustration 15: Deleting target(s)

The user can delete any of her targets by checking the box against the target and then clicking on Action->Delete iscsi target. Deletion only works if the “sessionup” property for the targetis false (red stop sign). The user should first logout of the iSCSI session on the client

```
vagrant@iscsiclient:~$ sudo iscsiadm -m session -u
Logging out of session [sid: 1, target:
iqn.2014.01.192.168.61.21:fastiorequired:57e626a7, portal: 192.168.61.21,3260]
Logging out of session [sid: 2, target:
iqn.2014.01.192.168.61.22:fastiorequired-2:57e626a7, portal: 192.168.61.22,3260]
Logout of [sid: 1, target: iqn.2014.01.192.168.61.21:fastiorequired:57e626a7,
portal: 192.168.61.21,3260] successful.
Logout of [sid: 2, target: iqn.2014.01.192.168.61.22:fastiorequired-2:57e626a7,
portal: 192.168.61.22,3260] successful.
```

The “sessionup” property may take a minute or two to refresh after logging out of the iSCSI session. None of the portal's webpages auto-update currently (need to hit the refresh button).

(!) Deletion results in the irreversible removal of the LVM logical volume backing the storage, so please be sure before issuing the command.

Saturnring API

The purpose of the API is to provision storage via a HTTP GET request. This is useful for automating storage creation when cloud vm instances start up. The API can be invoked via any HTTP client, illustrated here via curl as shown below:

Provisioning Call

```
#User defines these variables
##################################################
SIZEINGB=1.0
SERVICENAME="fastiorequired"
SATURNRINGUSERNAME="fastiouser"
SATURNRINGPASSWORD="fastiopassword"
ANTI_AFFINITY_GROUP=${SATURNRINGUSERNAME}"unique-string"
SATURNRINGURL="http://192.168.61.20/api/provisioner/"
##################################################
IQNINI=`cat /etc/iscsi/initiatorname.iscsi | grep ^InitiatorName=  | cut -d= -f2`
RTNSTR=$( unset http_proxy && curl -s -X GET "${SATURNRINGURL}" --user "$
{SATURNRINGUSERNAME}":"${SATURNRINGPASSWORD}" --data clientiqn="$
{IQNINI}"'&'sizeinGB="${SIZEINGB}"'&'serviceName="${SERVICENAME}"'&'aagroup="$
{ANTI_AFFINITY_GROUP}" )
echo $RTNSTR | python -mjson.tool
```

The Saturnring API server provisions the requested storage and returns a JSON response shown below:


```
{
    "aagroup__name": "fastiouserunique-string",
    "already_existed": 0,
    "error": 0,
    "iqnini": "iqn.1993-08.org.debian:01:ba70a129ba3",
    "iqntar": "iqn.2014.01.192.168.61.21:fastiorequired:aead642d",
    "sessionup": false,
    "sizeinGB": 1.0,
    "targethost": "192.168.61.21",
    "targethost__storageip1": "192.168.61.21",
    "targethost__storageip2": "192.168.61.21"
}
```

This JSON sting contains the following fields:

1. **aagroup_name:** This is the anti-affinity group name. Use the same string while creating another iSCSI target if it is requred 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. It is the user's responsibility to confirm anti-affinity worked by checking the targethost for the targets in the same anti-affinity group. See the example at saturnring/deployments/vagrant/clientscripts/high_availability_storage.sh on how this can be checked.

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. 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 previosuly 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 filesystem needs to be created/formated on the device or not. So for example if already_existed=0, then this is a new target and so a filesystem 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 filesystem should probably not be created.

3. **error:** When there is a provisioning error (for example, there is no Storage host capable of accomadting 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. So its best 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 provioning 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 name of iSCSI server hosting the
storage.Servicename.trancated MD5 hash of iqnini
```

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 its best not to try to login to this iSCSI target (bad things can happen if r/w target access is given to multiple clients). Look at the example saturnring/deployments/vagrant/clientscripts/storage-provisioner.sh for how to use this property.

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. See the example saturnring/deployments/vagrant/clientscripts/storage-provisioner.sh for how to use this in the iSCSI session login command.

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 addreses can be used for iSCSI multipath setups.

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.

This 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 subseqent targets of a clumpgroup), then the failure of n out of N iSCSI servers will only knock out (n/N*M) Cassandra nodes.

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:

```
#User defines these variables
##################################################
SIZEINGB=1.0
SERVICENAME="fastiorequired1"
SATURNRINGUSERNAME="fastiouser"
SATURNRINGPASSWORD="fastiopassword"
ANTI_AFFINITY_GROUP=${SATURNRINGUSERNAME}"unique-string"
SATURNRINGURL="http://192.168.61.20/api/provisioner/"
CLUMPGROUP="anotherclump"
##################################################

IQNINI=`cat /etc/iscsi/initiatorname.iscsi | grep ^InitiatorName=  | cut -d= -f2`
RTNSTR=$( unset http_proxy && curl -s -X GET "${SATURNRINGURL}" --user "$
{SATURNRINGUSERNAME}":"${SATURNRINGPASSWORD}" --data clientiqn="$
{IQNINI}"'&'sizeinGB="${SIZEINGB}"'&'serviceName="${SERVICENAME}"'&
'aagroup="${ANTI_AFFINITY_GROUP}"'&'clumpgroup="${CLUMPGROUP}" )
echo $RTNSTR | python -mjson.tool
SERVICENAME="fastiorequired2"
RTNSTR=$( unset http_proxy && curl -s -X GET "${SATURNRINGURL}" --user "$
{SATURNRINGUSERNAME}":"${SATURNRINGPASSWORD}" --data clientiqn="$
{IQNINI}"'&'sizeinGB="${SIZEINGB}"'&'serviceName="${SERVICENAME}"'&
'aagroup="${ANTI_AFFINITY_GROUP}"'&'clumpgroup="${CLUMPGROUP}" )
echo $RTNSTR | python -mjson.tool
```

Deletion Call

ISCSI targets can be deleted using an API call “delete”. As noted earlier deletion is also possible via the portal.

1. Delete a specified target belonging to the user

API call takes target iqn and user authentication credentials as input.

```
#User defines these variables
##################################################
SATURNRINGUSERNAME="fastiouser"
SATURNRINGPASSWORD="fastiopassword"
SATURNRINGURL="http://192.168.61.20/api/delete/"
DELETETARGET="iqn.2014.01.192.168.61.21:fastiorequired:aead642d"
##################################################
RTNSTR=$( unset http_proxy && curl -s -X GET "${SATURNRINGURL}" --user "$
{SATURNRINGUSERNAME}":"${SATURNRINGPASSWORD}" --data iqntar="${DELETETARGET}" )
echo $RTNSTR | python -mjson.tool
```

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

API call takes initiator iqn and user authentication credentials as input

```
#User defines these variables
##################################################
SATURNRINGUSERNAME="fastiouser"
SATURNRINGPASSWORD="fastiopassword"
SATURNRINGURL="http://192.168.61.20/api/delete/"
##################################################
DELETEALLINITATORTARGETS=`cat /etc/iscsi/initiatorname.iscsi | grep
^InitiatorName=  | cut -d= -f2`
RTNSTR=$( unset http_proxy && curl -s -X GET "${SATURNRINGURL}" --user "$
{SATURNRINGUSERNAME}":"${SATURNRINGPASSWORD}" --data clientiqn="$
{DELETEALLINITIATORTARGETS}" )
echo $RTNSTR | python -mjson.tool
```

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

```
#User defines these variables
```

```
################################################
SATURNRINGUSERNAME="fastiouser"
SATURNRINGPASSWORD="fastiopassword"
SATURNRINGURL="http://192.168.61.20/api/delete/"
TARGETHOST="192.168.61.21"
################################################
RTNSTR=$( unset http_proxy && curl -s -X GET "${SATURNRINGURL}" --user "$
{SATURNRINGUSERNAME}":"${SATURNRINGPASSWORD}" --data targethost="${TARGETHOST}" )
echo $RTNSTR | python -mjson.tool
```

4. Deleting all targets for a specified initator created on a specified iSCSI server is also possible.

```
#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=$( unset http_proxy && curl -s -X GET "${SATURNRINGURL}" --user "$
{SATURNRINGUSERNAME}":"${SATURNRINGPASSWORD}" --data clientiqn="$
{IQNINI}"'&'targethost="${TARGETHOST}" )
echo $RTNSTR | python -mjson.tool
```

Authentication and Security Model

Saturnring uses Django's authentication methods (https://docs.djangoproject.com/en/dev/topics/auth/). Integration with active directory over LDAP v3 is also supported in this release of Saturnring. For simple deployments default

The client's IQN and the servicename effectively serve as a authentication token because each iSCSI target has its initiator as its only authorized “security group”. Appending a random password string to the service name will effectively provide password security to the target. Current use cases should be covered by this level of basic security. Tougher security requirements may be addressed via iSCSI CHAP authentication etc., although this will require code changes to Saturnring.

Advanced Topics

Authentication Against Active Directory/LDAP

Saturnring can authenticate against an open-ldap or active-directory server. The implementation is based on python-ldap (in turn based on Open-ldap-2.x) and django-auth-ldap, which means it should work with almost 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"
```

Note: In order to setup a "test" Active-directory VM, setup an evaluation copy of Windows server in a virtualbox VM. For example see this blogpost http://stef.thewalter.net/2012/08/how-to-create-active-directory-domain.html

Saturnring configures authentication so that both local and LDAP authentication is possible. For example, the superuser 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.

Networking: Subnets and VLANs

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

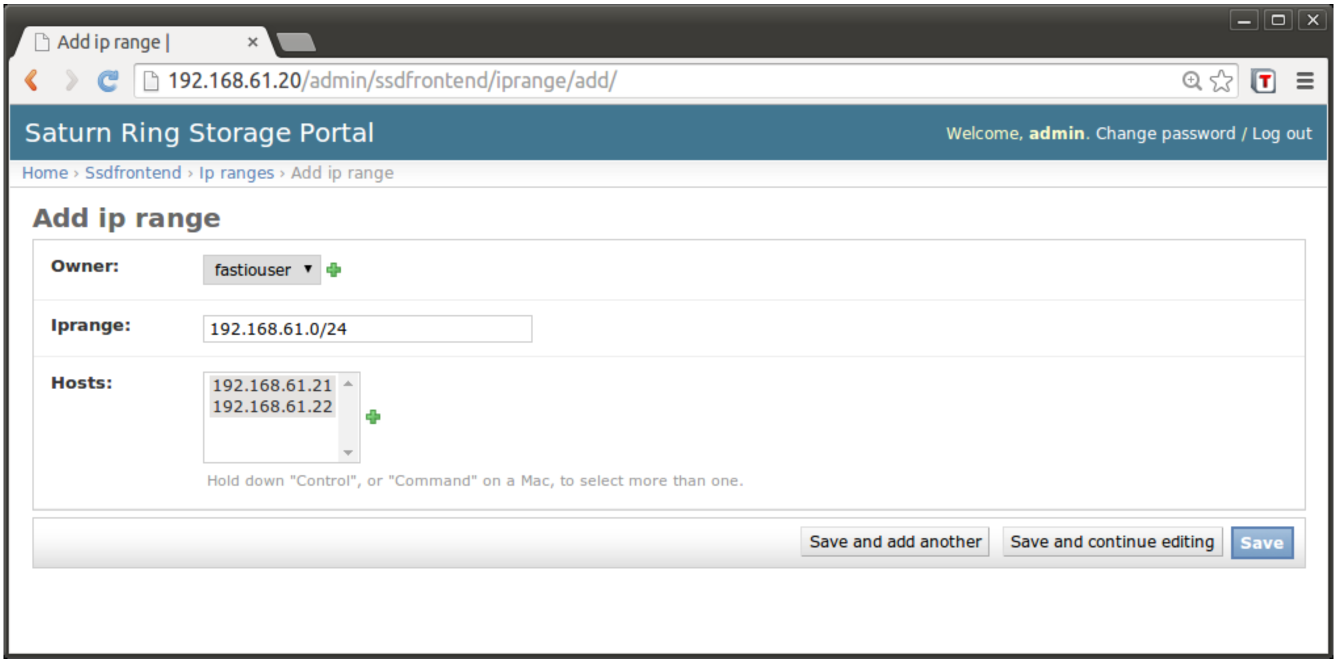

Illustration 16: Creating or modifying an IP range

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 (see Illustration 16). 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.

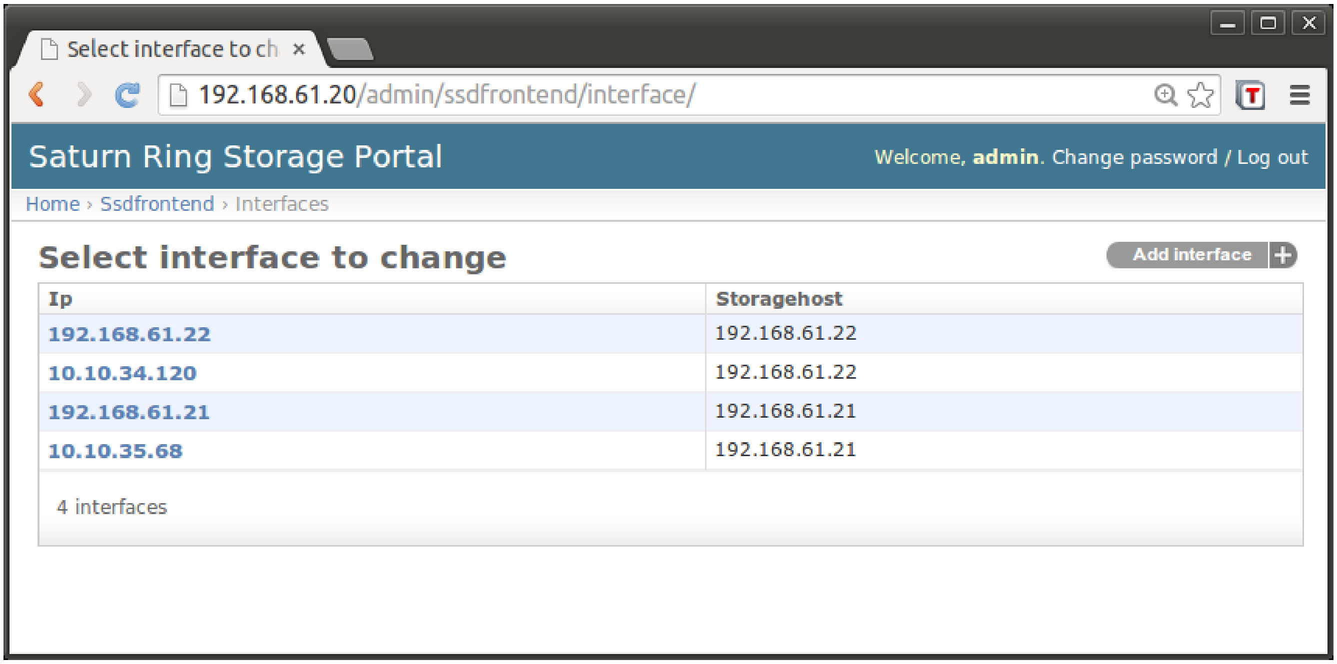

Illustration 17: List of interfaces on all storage hosts discovered by Saturnring via periodic scanning

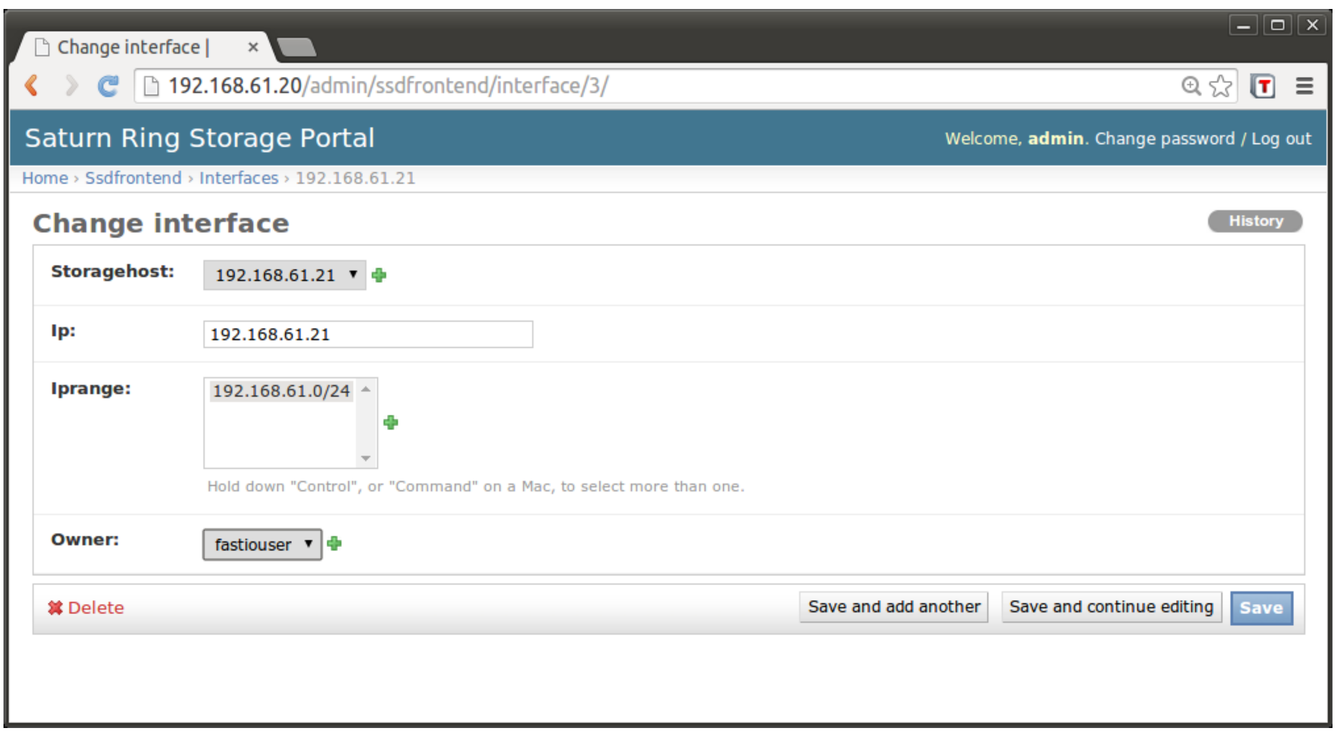

Illustration 18: Changing the owner of an interface

Here is a snippet that illustrates using the subnet feature to provision on the client side. For a complete script refer to the <

```
#User defines these variables
##################################################
SIZEINGB=1.0
SERVICENAME="vlan-service1"
SATURNRINGUSERNAME="fastiouser"
SATURNRINGPASSWORD="fastiopassword"
ANTI_AFFINITY_GROUP=${SATURNRINGUSERNAME}"unique-string"
SATURNRINGURL="http://192.168.61.20/api/provisioner/"
SUBNET="192.168.61.0/24"
##################################################

IQNINI=`cat /etc/iscsi/initiatorname.iscsi | grep ^InitiatorName=  | cut -d= -f2
`
RTNSTR=$( unset http_proxy && curl -s -X GET "${SATURNRINGURL}" --user "${SATURN
RINGUSERNAME}":"${SATURNRINGPASSWORD}" --data clientiqn="${IQNINI}"'&'sizeinGB="
${SIZEINGB}"'&'serviceName="${SERVICENAME}"'&'aagroup="${ANTI_AFFINITY_GROUP}"'&
'subnet="${SUBNET}" )
```

Saturn Settings

saturn.ini file

The saturn.ini file in the </home/vagrant/saturnring/ssddj> directory defines several Saturnring variables. The variables are described as comments in the file. The apache webserver has to be restarted everytime entries in this file are changed: in the saturnring VM, type

```
sudo service apache2 restart
```

Django settings.py file

Available at `/home/vagrant/saturnring/ssddj/ssddj` in the Vagrant example.

There are times when the Django settings file needs to be edited. Examples of such situations include

1. Choosing another database (instead of the default sqllite)

2. Changing logging preferences
3. Tweaks to make different authentication backends work (e.g. different settings for OpenLDAP or some unique active directory setup)

To understand the scope of options Django allows you to specify in the settings.py file, refer to the Django documentation on the settings file here:

https://docs.djangoproject.com/en/dev/ref/settings/

The apache2 service will need to be restarted whenever the settings file is changed.