# Saturnring User Guide

**Beta document – work in progress.**

## Table of Contents

## *Synopsis*

This document illustrates the usage of the Saturnring portal as an administrator and as a user.

## *Portal Operations: Admin*

Navigating to the portal address (usually http://saturnringipaddress_or_dnsname/admin) will display the following page.
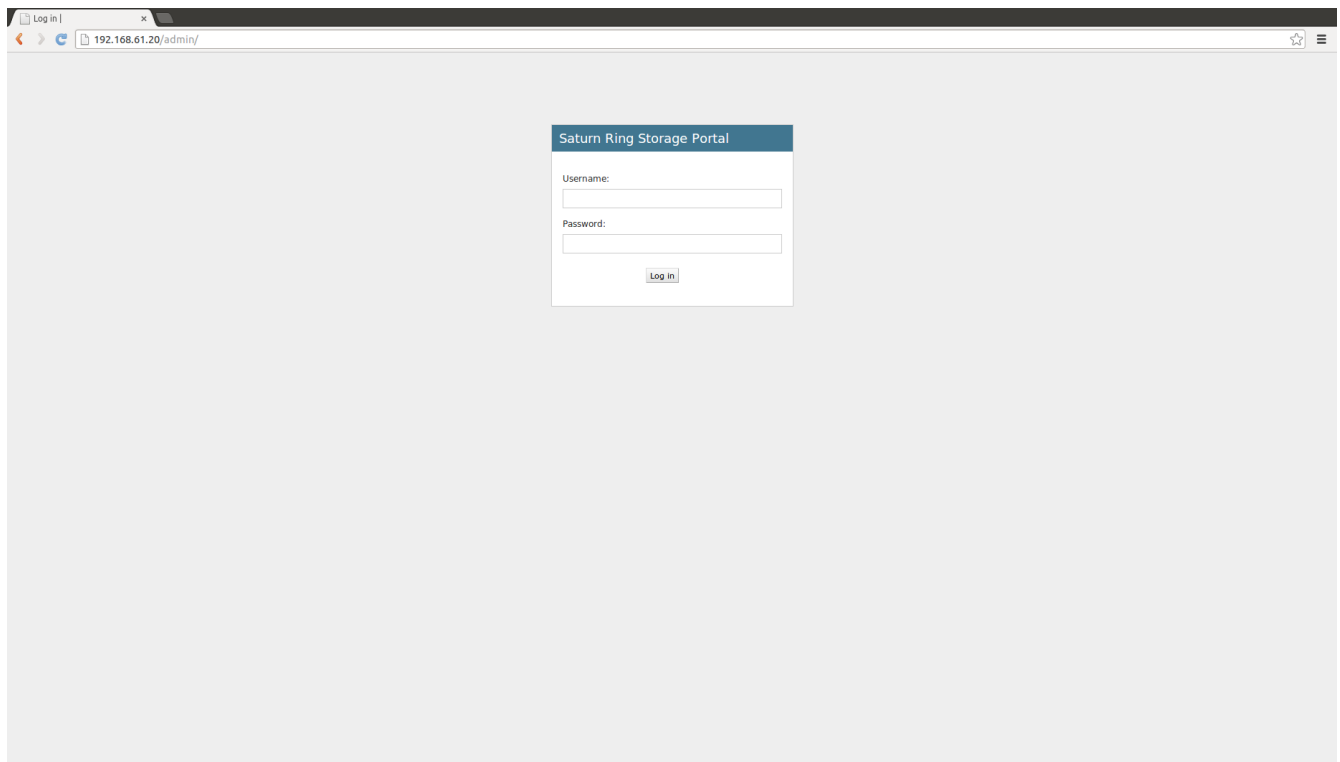


Fig 1: Login Screen

After supplying admin/superuser credentails the screen shown in Fig. 2 should show.
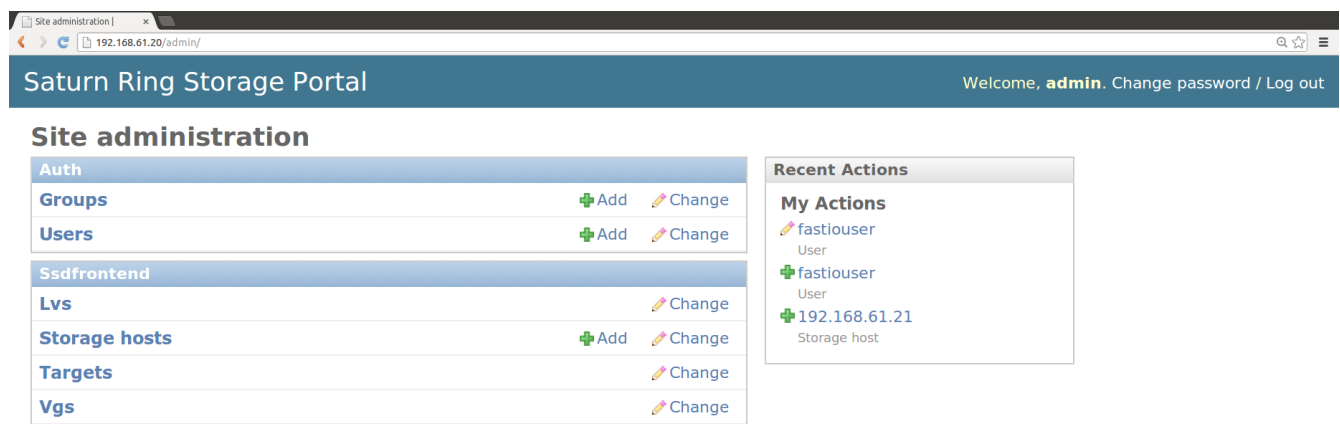
Fig. 2 is the Django admin interface.

There are 2 sub-categories of links. The Auth category is for user management (Groups,users) and the SSDfrontend (Lvs-logical volume information, Storage hosts, Targets and Vgs – volume groups) is the category to manage storage. There is also a recent actions pane.

Fig.3: Adding a new iSCSI saturnring server

Home > Ssdfrontend > Storage hosts > Add storage host

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=192.168.61.22. 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 can be used to specify iSCSI portals over different VLANs, 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 -s -X 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

Fig 4: Changing Volume group properties

There are 2 very 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 can get you into trouble (what if the overprovisioned targets need more than the available actual storage?) . For the safe non-over provisioned storage, set Opf to 1.0, and thinusedmaxpercent to ~95%.

Fig. 5: Targets (admin view)

By navigating to Home>Ssdfrontend>Targets the admin can get a bird's eye view of all targets across all iSCSI servers provisioned in the iSCSI system. The admin can also delete targets from this view (provided the Sessionup property is false, i.e., there is no active iSCSI session on that target).

Clicking on any of the target IQNs will show the properties of that target.

Note that the "sessionup" property may take a minute or two to refresh.
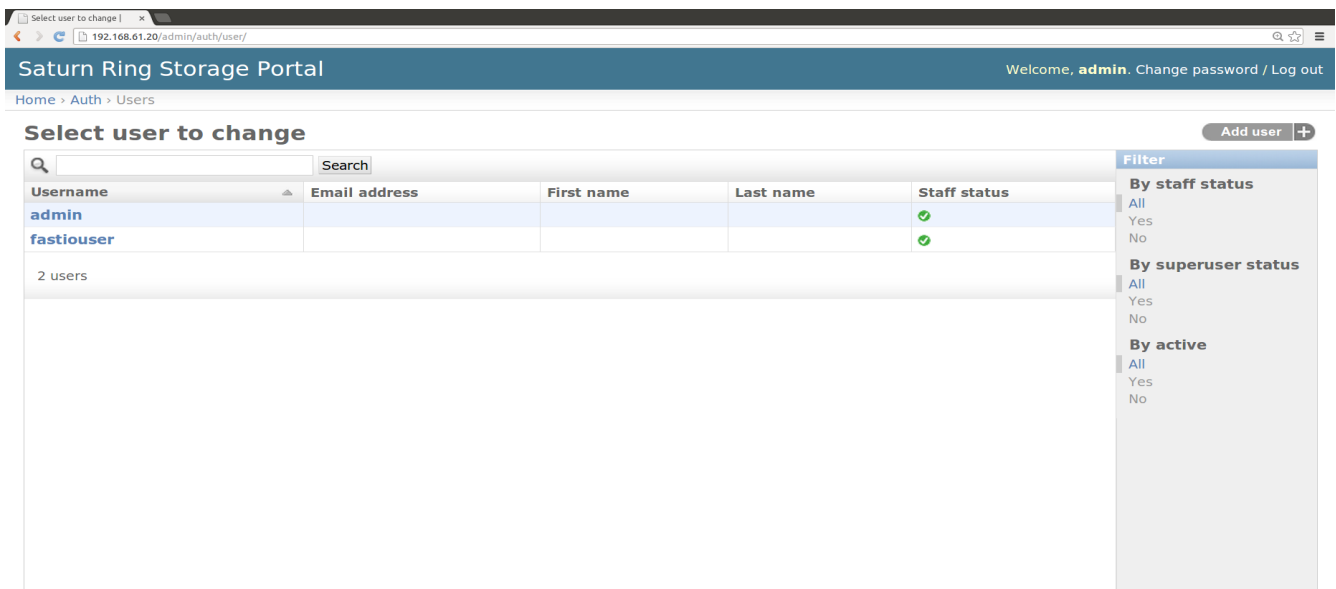
Fig. 6: Users (admin view)

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)



Fig. 7: Adding a new user

Saturn Ring Storage Portal

Home › Auth › Users › fastiouser

## Change user

**Username:** `fastiouser`
Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

**Password:** **algorithm**: pbkdf2_sha256 **iterations**: 12000 **salt**: vscRf6****** **hash**: KJjpkB*****************************************
Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

**Personal info**

**First name:**

**Last name:**

**Email addr...**

> **Staff Status: Enable this checkbox**

**Permissions**

☑ Active
Designates whether th... ...uld be treated as active. Unselect this instead of deleting accounts.

☑ Staff status
Designates whether the user can log into this admin site.

☐ Superuser status
Designates that this user has all permissions without explicitly assigning them.

**Groups:** The groups this user belongs to. A user will get all permissions granted to each of his/her group. Hold down "Control", or "Command" on a Mac, to select more than one.

Available groups 🔍
🔍 Filter

Chosen groups 🔍                                                      ➕

Remove all

> **User permissions**
> ssdfrontend|target|Can add target
> ssdfrontend|target|Can change target
> ssdfrontend|target|Can delete target

**User permissions:** Specific permissions for this user. Hold down "Control", or "Comm... ...elect more than one.

Available user permissions 🔍
🔍 Filter

admin | log entry | Can add log entry
admin | log entry | Can change log entry
admin | log entry | Can delete log entry
auth | group | Can add group
auth | group | Can change group
auth | group | Can delete group
auth | permission | Can add permission
auth | permission | Can change permission
auth | permission | Can delete permission
auth | user | Can add user
auth | user | Can change user
auth | user | Can delete user
contenttypes | content type | Can add content type
contenttypes | content type | Can change content type

Chosen user permissions 🔍

ssdfrontend | target | Can add target
ssdfrontend | target | Can change target
ssdfrontend | target | Can delete target

Choose all 🔘                          Remove all

**Important dates**

**Last login:** Date: `2014-05-26`  Today | 📅
Time: `20:06:28`  Now | (

**Date joined:** Date: `2014-05-26`  Today |
Time: `20:06:28`  Now | (

> **Quotas**

**Profile**
Profile: Profile object

**Max target sizeGB:** `4.0`

**Max alloc sizeGB:** `10.0`

Fig 8: Setting user properties

8

Fig 9: Setting quotas

Fig 9. shows a zoomed-in version of Fig 8. 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).

## *Portal Operations: User*

A user may log into the portal using her credentials. The initial screen just has a single link to Targets provisioned and owned by the user:

Fig 10: User logged-in view

Fig 11: Deleting a target

The user can delete any of her targets by checking the box against the target and then clicking on Action->Delete iscsi target. Deletion results in the irreversible removal of the LVM logical volume backing the storage, so please be sure before issuing the command.

Note that the "sessionup" property may take a minute or two to refresh.

## *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.

There is no "Delete" API call: this is by design to prevent script-errors causing mass target deletions and data loss. Targets can be deleted via the portal (Fig . 11). Even dozens of targets will only take a minute or two to delete via the portal (multiple selection of targets to delete is possible).

## Authentication and Security Model

Saturnring uses Django's authentication methods (https://docs.djangoproject.com/en/dev/topics/auth/). There are relatively simple ways of extending authentication to use LDAP, active directory etc.

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.

## *Under the hood: 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.

For ideas on how a high-availability can be created using Saturnring anti-affinity-groups labd client-side software RAID 1, look at the example script saturnring/deployments/vagrant/clientscripts/high_availability_storage.sh

## 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. For example:

iSCSI initiator

Unique identifier of an iSCSI client (an Altus 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.
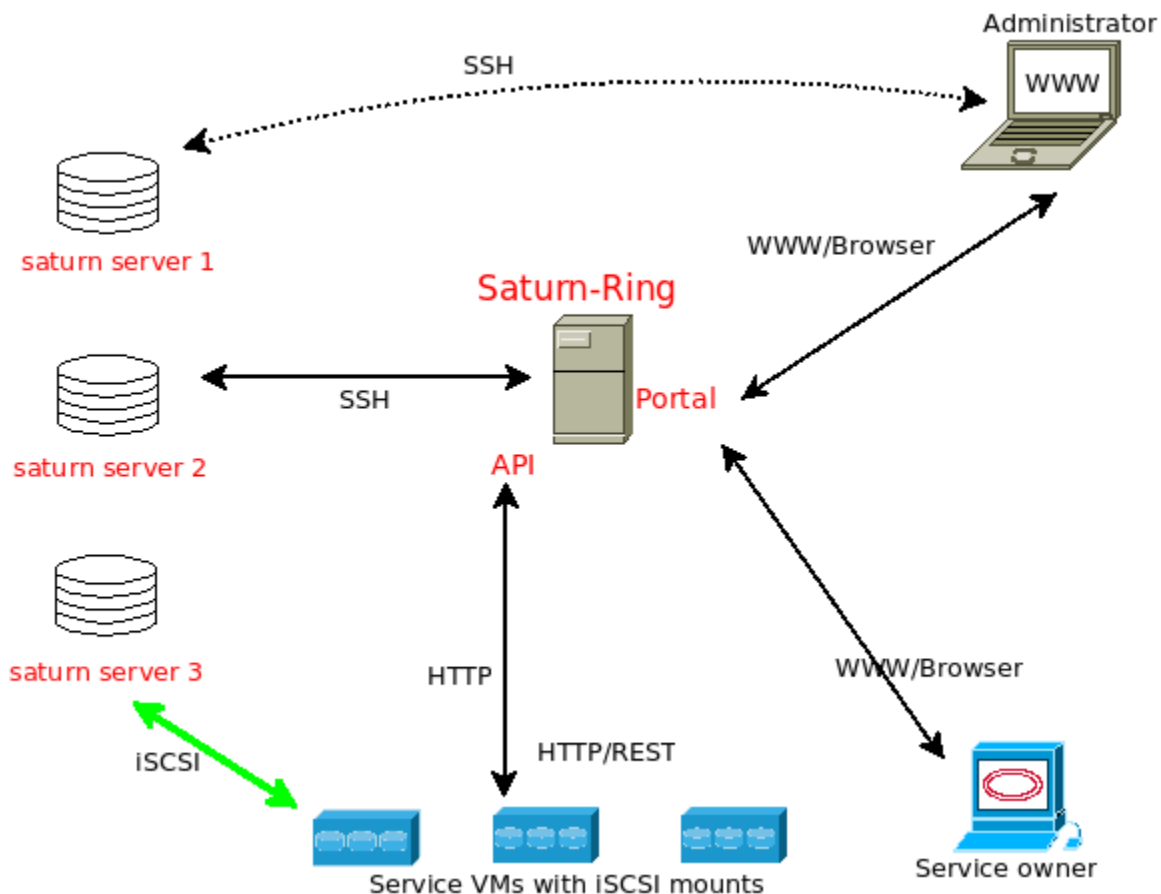
Saturnring

Management layer (website, Restful-API) to provision and manage Saturn iSCSI storage

## 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.

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 ssddj/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.