|  |
| --- |
|  |
| Saturnring Architecture and User Guide |
| v0.02 |
|  |
| **Sachin Agarwal** |
| **4/17/2015** |

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

Contents

[List of FIGURES 3](#_Toc417908989)

[The Mile High View 4](#_Toc417908990)

[Introduction 4](#_Toc417908991)

[Design Principle 4](#_Toc417908992)

[Key Considerations 5](#_Toc417908993)

[Terminology 7](#_Toc417908994)

[Saturnring Architecture 8](#_Toc417908995)

[Installation Guide 11](#_Toc417908996)

[Vagrant Installation – for development and testing 11](#_Toc417908997)

[Guide for Administrators 13](#_Toc417908998)

[Introduction 13](#_Toc417908999)

[Login Screen and Overview 13](#_Toc417909000)

[User and Quota Management 15](#_Toc417909001)

[iSCSI Server and Storage Management 18](#_Toc417909002)

[Targets and Logical Volumes 19](#_Toc417909003)

[Troubleshooting & locks 23](#_Toc417909004)

[24](#_Toc417909005)

[Networking 24](#_Toc417909006)

[Guide for Users 27](#_Toc417909007)

[Saturnring API 30](#_Toc417909008)

[Provisioning new storage and Querying pre-existing iSCSI target metadata 30](#_Toc417909009)

[Example 1: Setup a simple iSCSI target 30](#_Toc417909010)

[Clump Groups 32](#_Toc417909011)

[iSCSI target Deletion 34](#_Toc417909012)

[Security 37](#_Toc417909013)

[Advanced Configuration Topics 38](#_Toc417909014)

# List of FIGURES

[Figure 1: iSCSI storage configuration and data/control plane 8](#_Toc417889467)

[Figure 2: High Level Software Architecture of Saturnring 10](#_Toc417889468)

[Figure 3: Login Screen 14](#_Toc417889469)

[Figure 4: post-Login screen for admin user 15](#_Toc417889470)

# 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. More

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** 
  + 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. 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.
  + 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).
  + 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 Architecture

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.

The shared state in the Saturnring system is implemented via the SQL database. Any SQL database supported by the Django ORM can be used, although sqllite should be avoided due to known locking issues. The database holds information about users, iSCSI targets, volume groups, logical volumes, IP interfaces, quotas etc. It may be prudent 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.

# Installation Guide

## 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 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 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/

1. Navigate to /saturnring/deployments/vagrant

cd ~/DIRROOT/saturnring/deployments/vagrant

**STAGE 1:** Bringing up Saturnring portal/API server (192.168.56.20)

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 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

1. 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)

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

vagrant up iscsiserver1

1. 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)

1. 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.
2. 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

1. 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)

# Guide for Administrators

## Introduction

## 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.

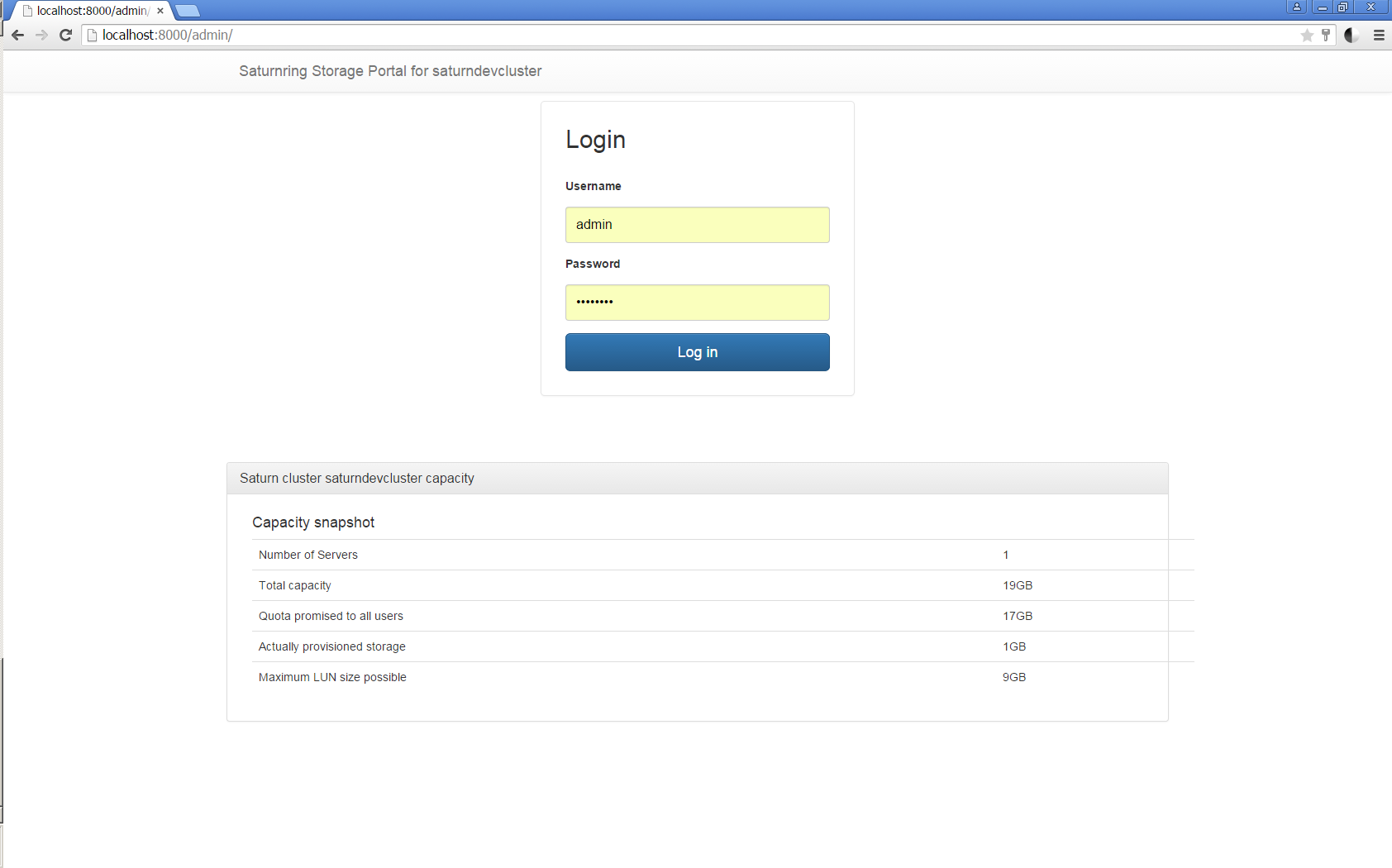


Figure 3: Login Screen

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. It should be noted that the adaptation retains the look and feel of the classic Django admin that comes out-of-the-box with Django web framework; therefore some widgets/workflows seen in the interface may not be applicable or enabled. This document will clarify these inconsistencies.

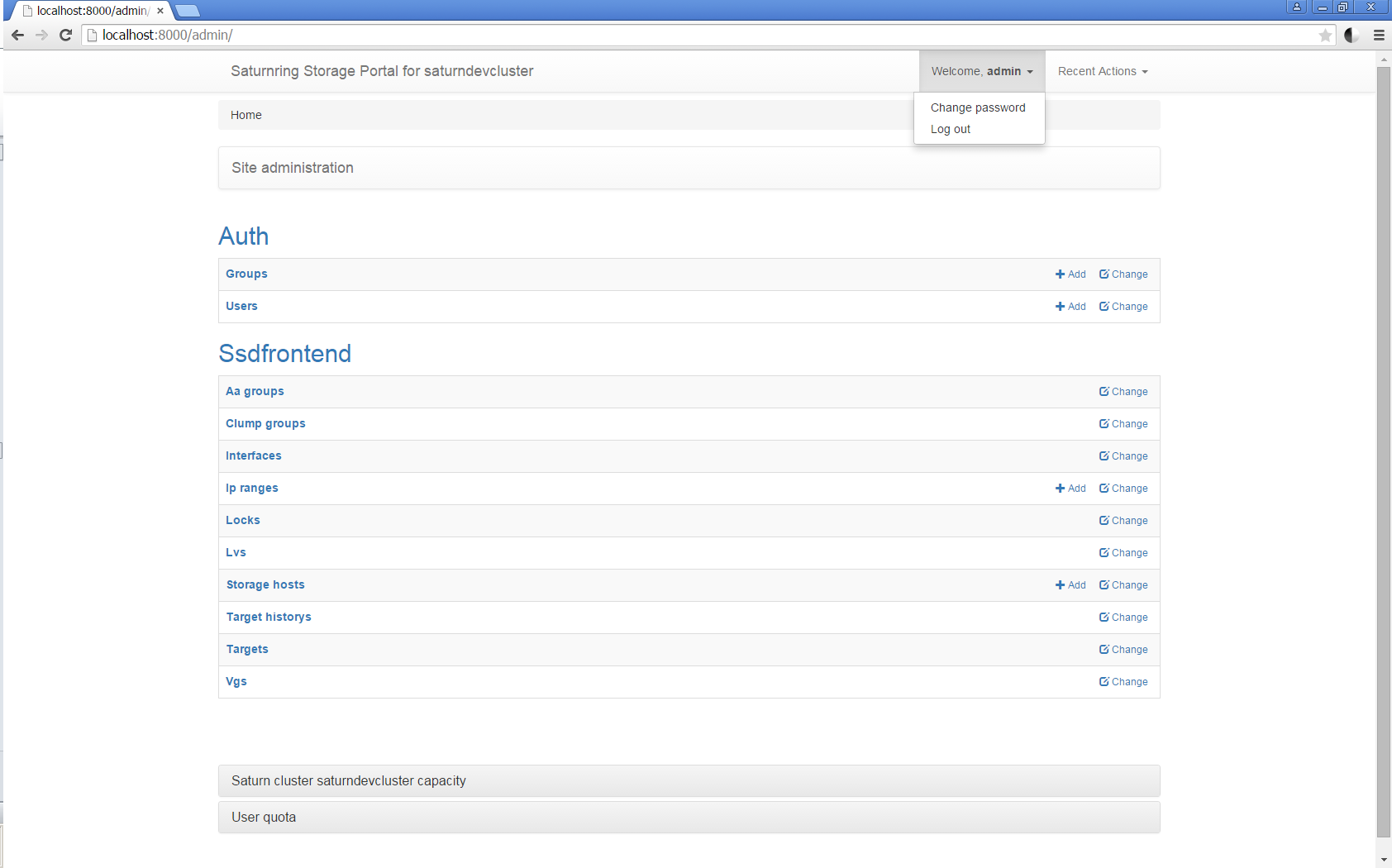
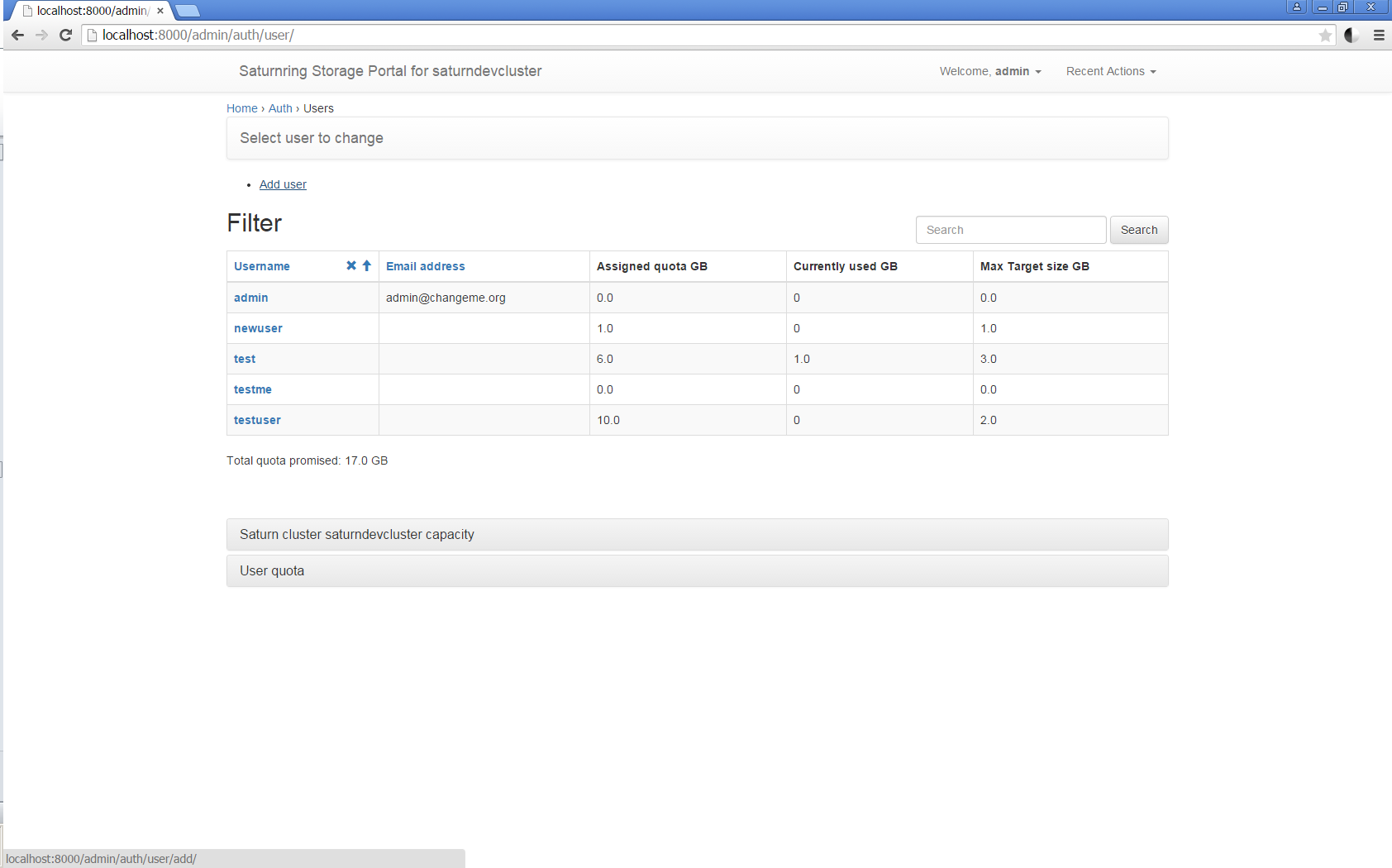


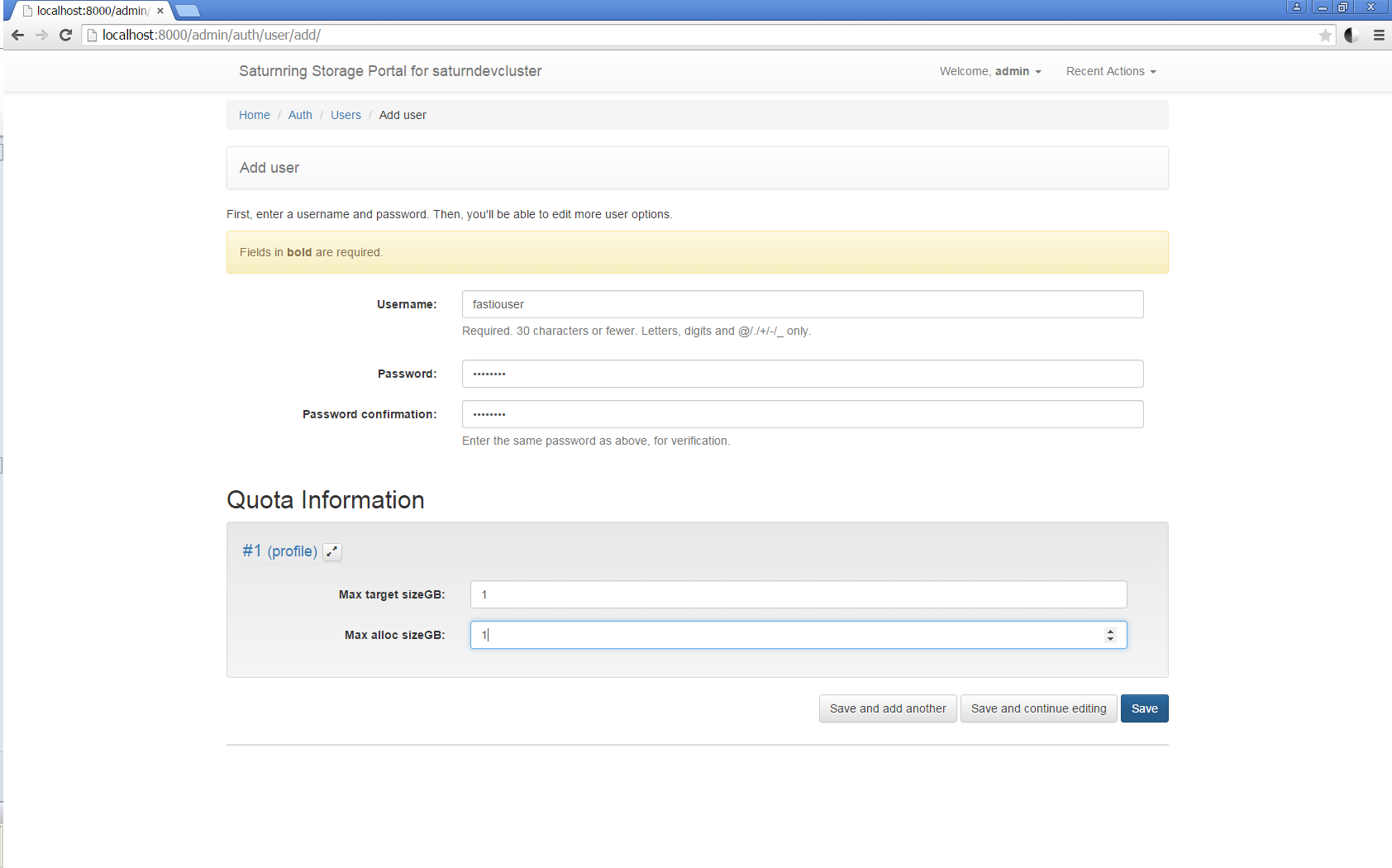
Figure 4: post-Login screen for admin user

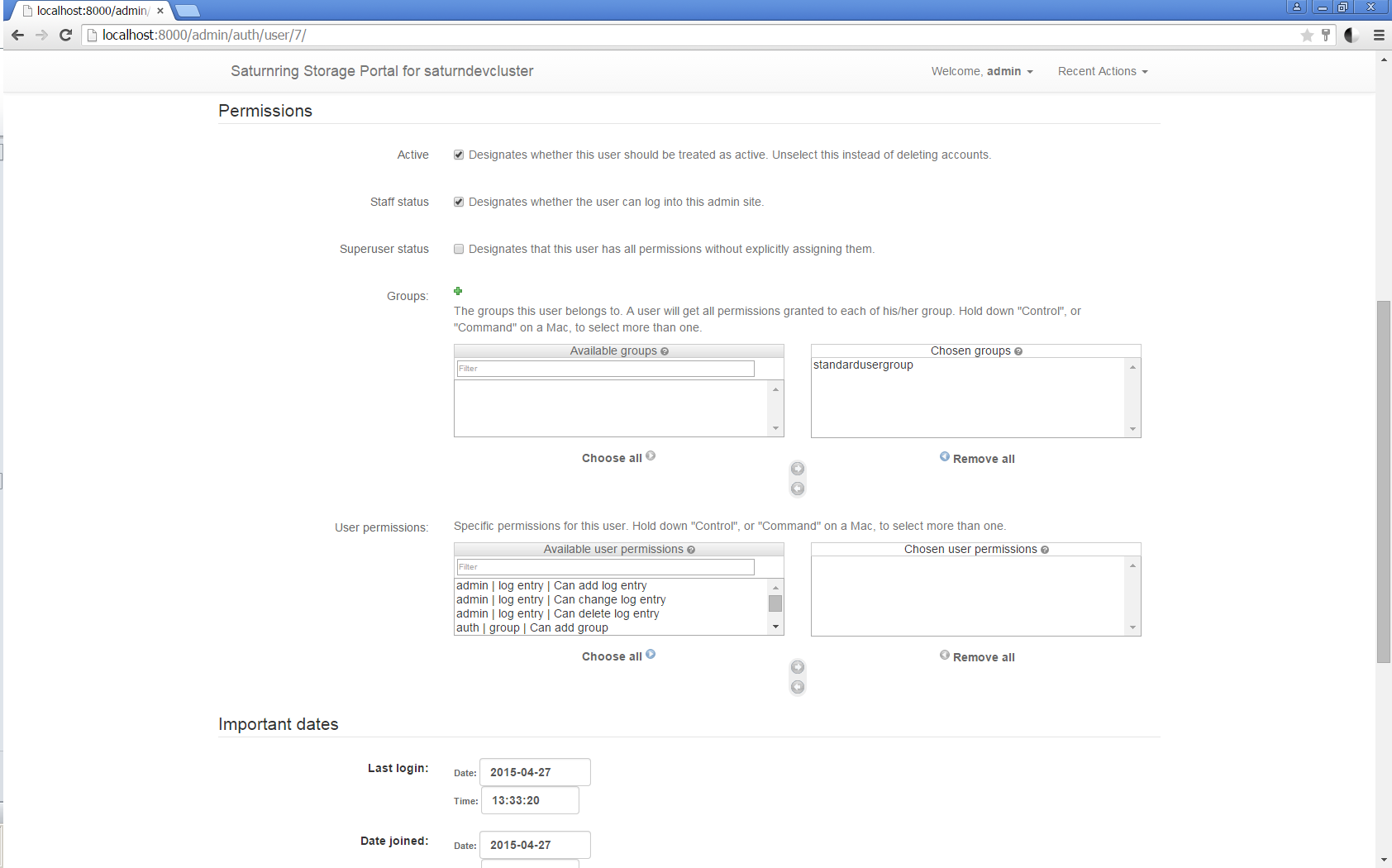
Figure 4 shows the post-login screen for the Admin user. The top of the screen shows cluster and user statistics The admin user is also permitted to provision iscsi storage; so the user quota and currently used quota is shown, along with the maximum allowed iSCSI target size.

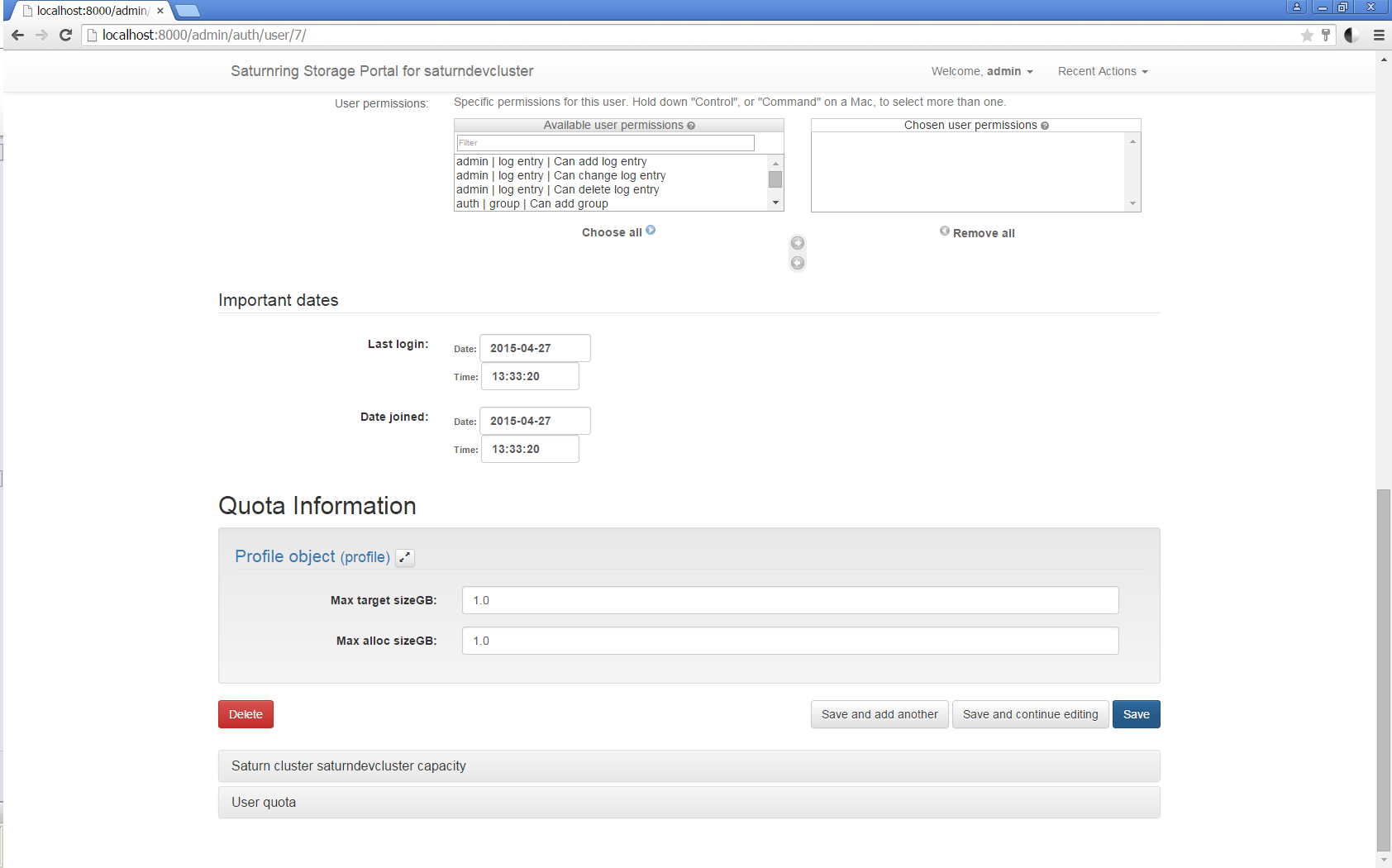
There are two broad categories of controls. The “Auth” controls with the ability to define users and assign them to groups with permissions to specific tables in the back-end Saturnring database and the “SSDfrontend” controls that defines various aspects of the Saturnring system. Eac

## User and Quota Management

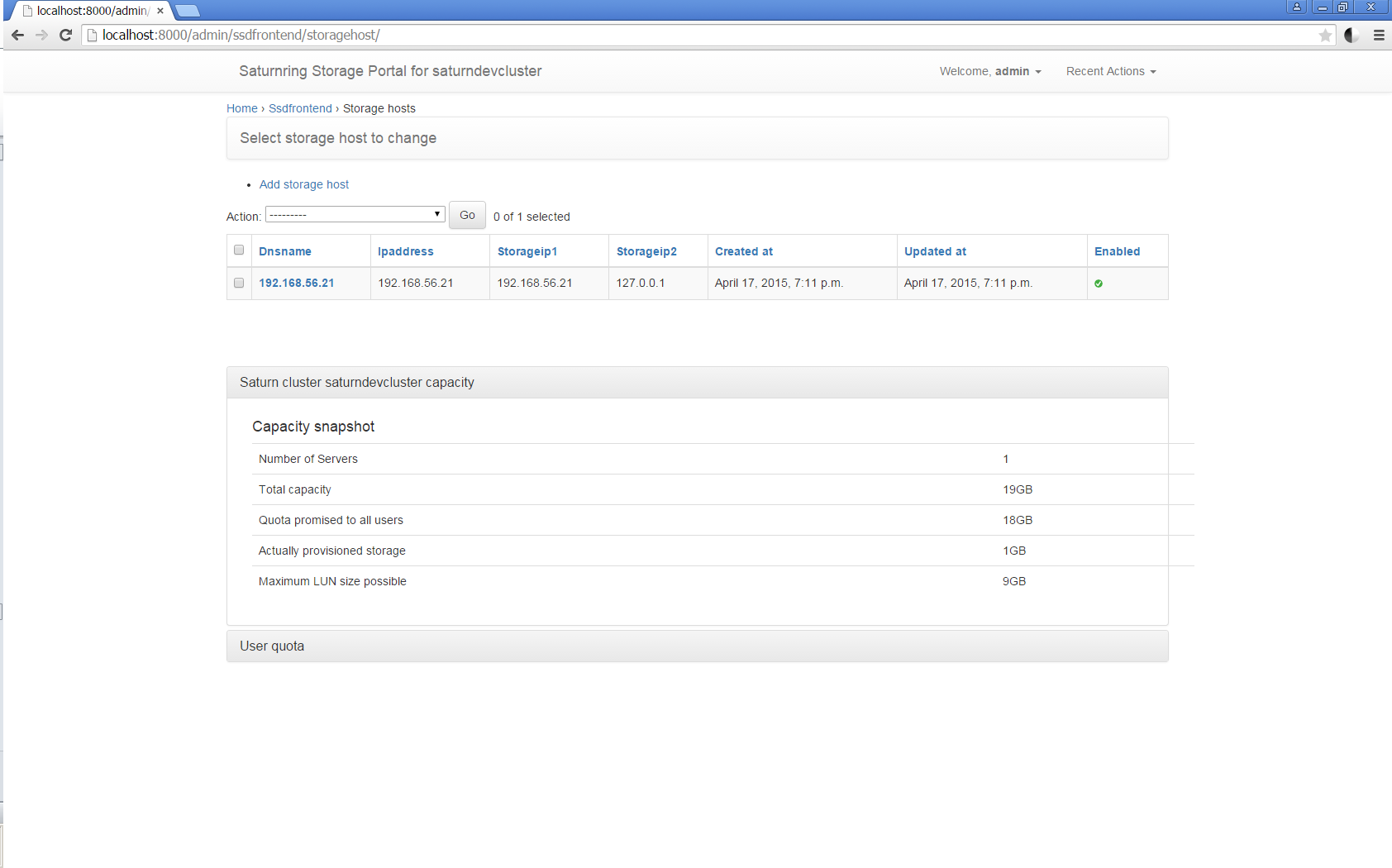


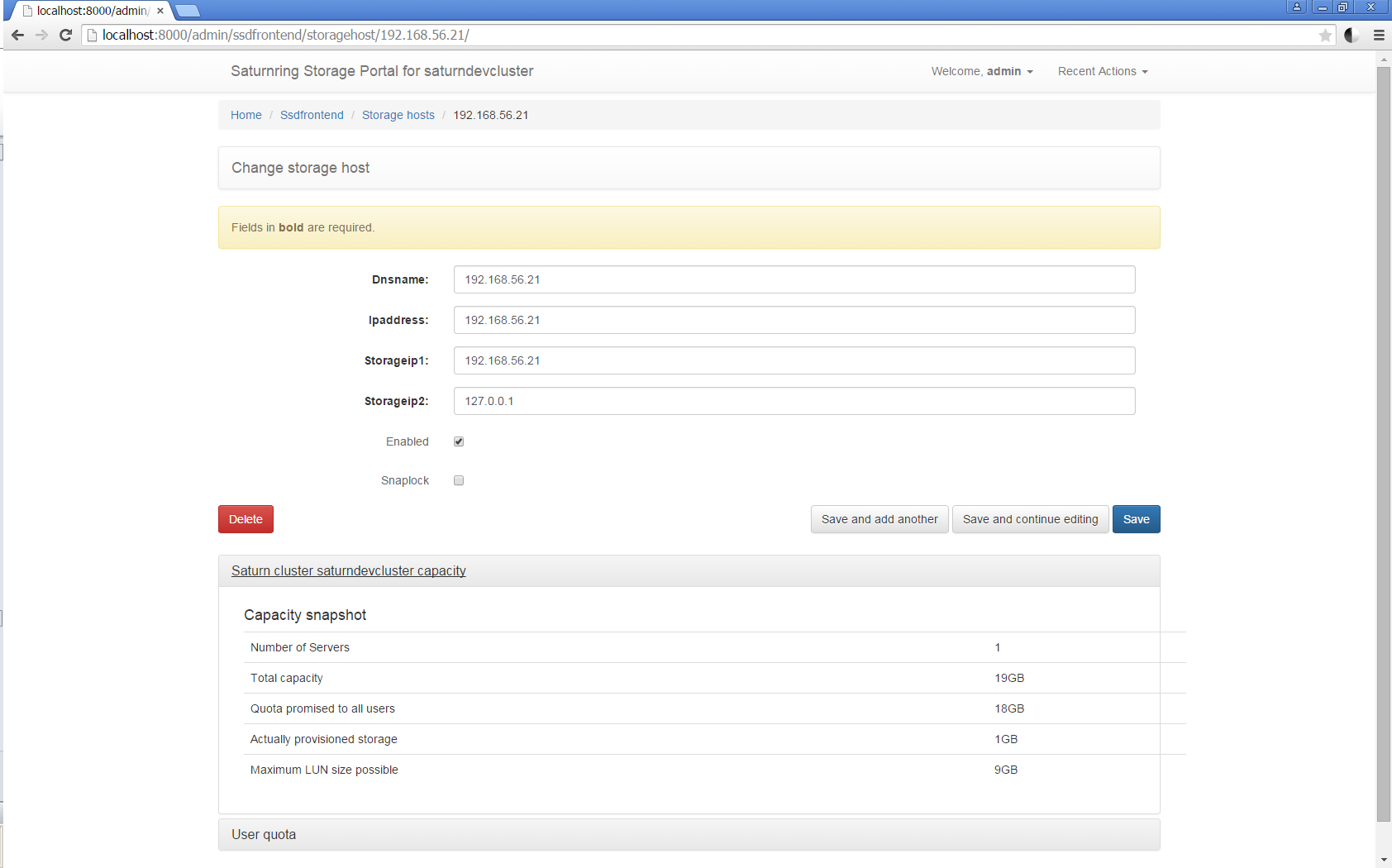




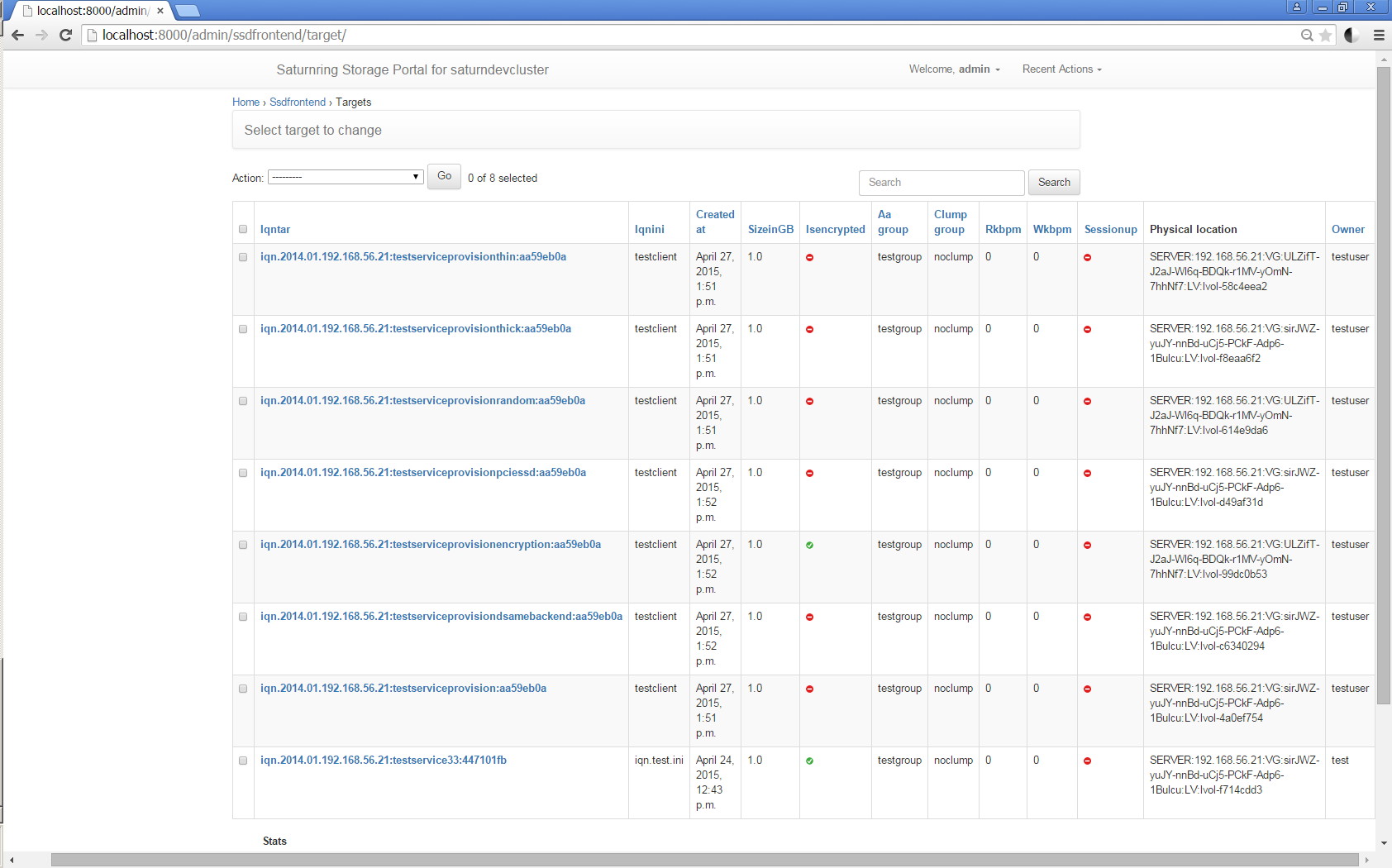


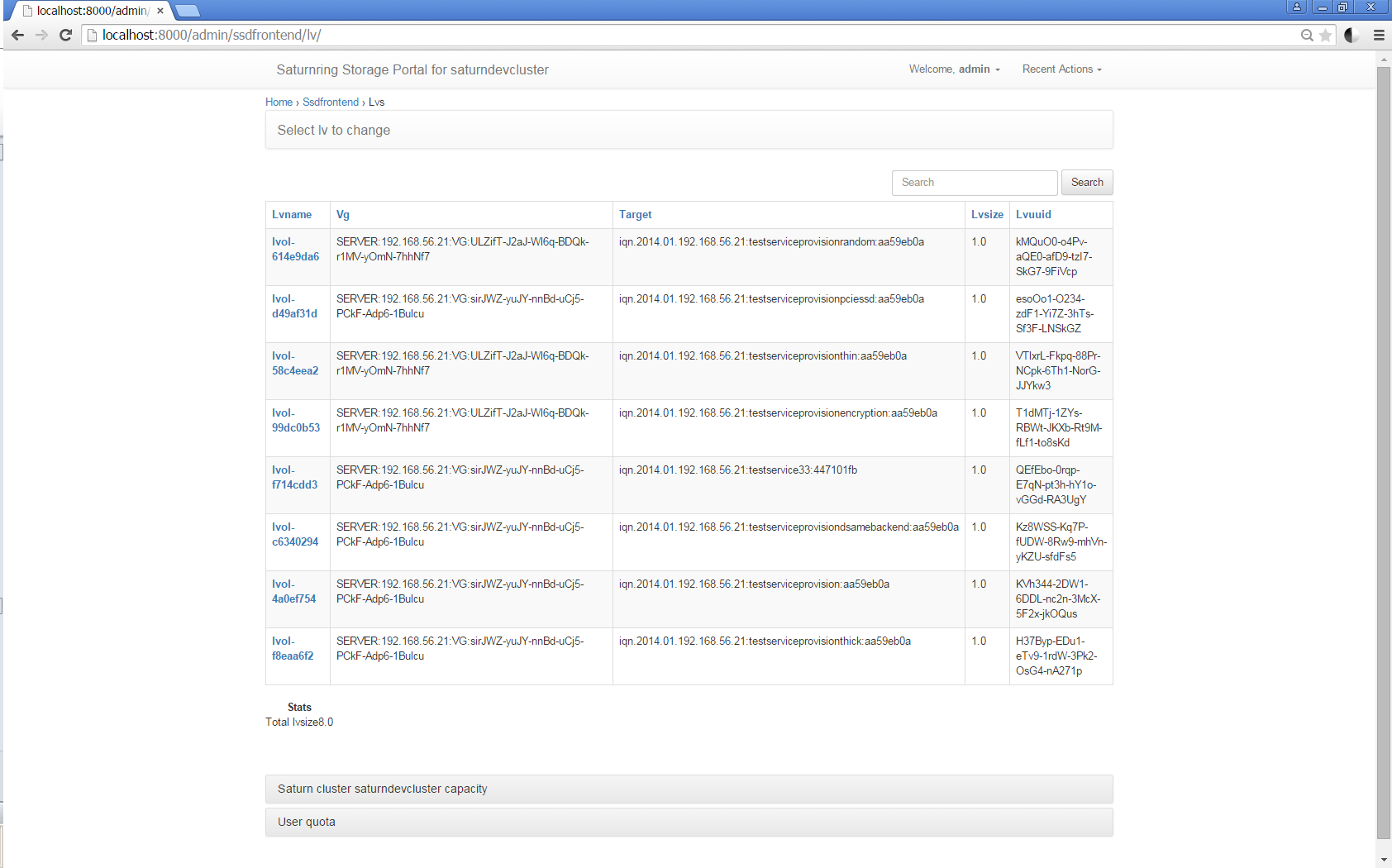
## iSCSI Server and Storage Management

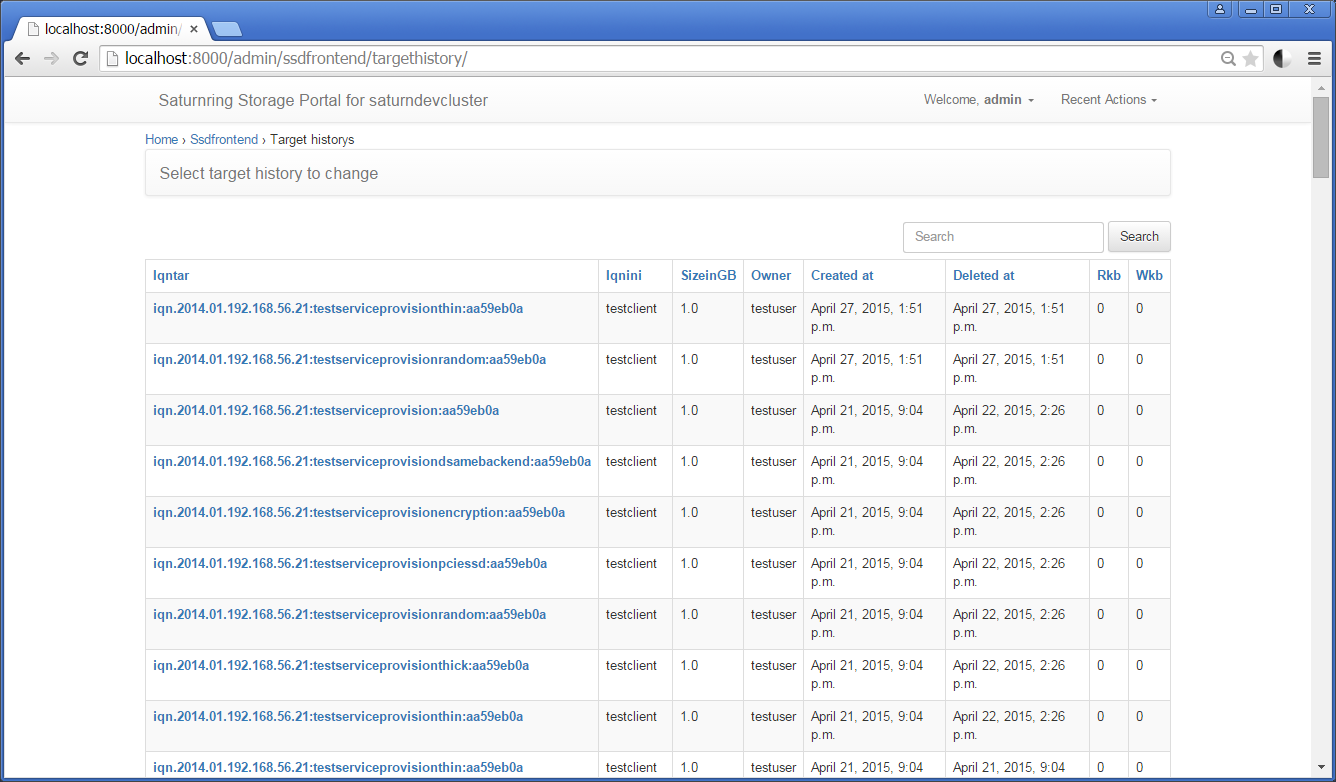
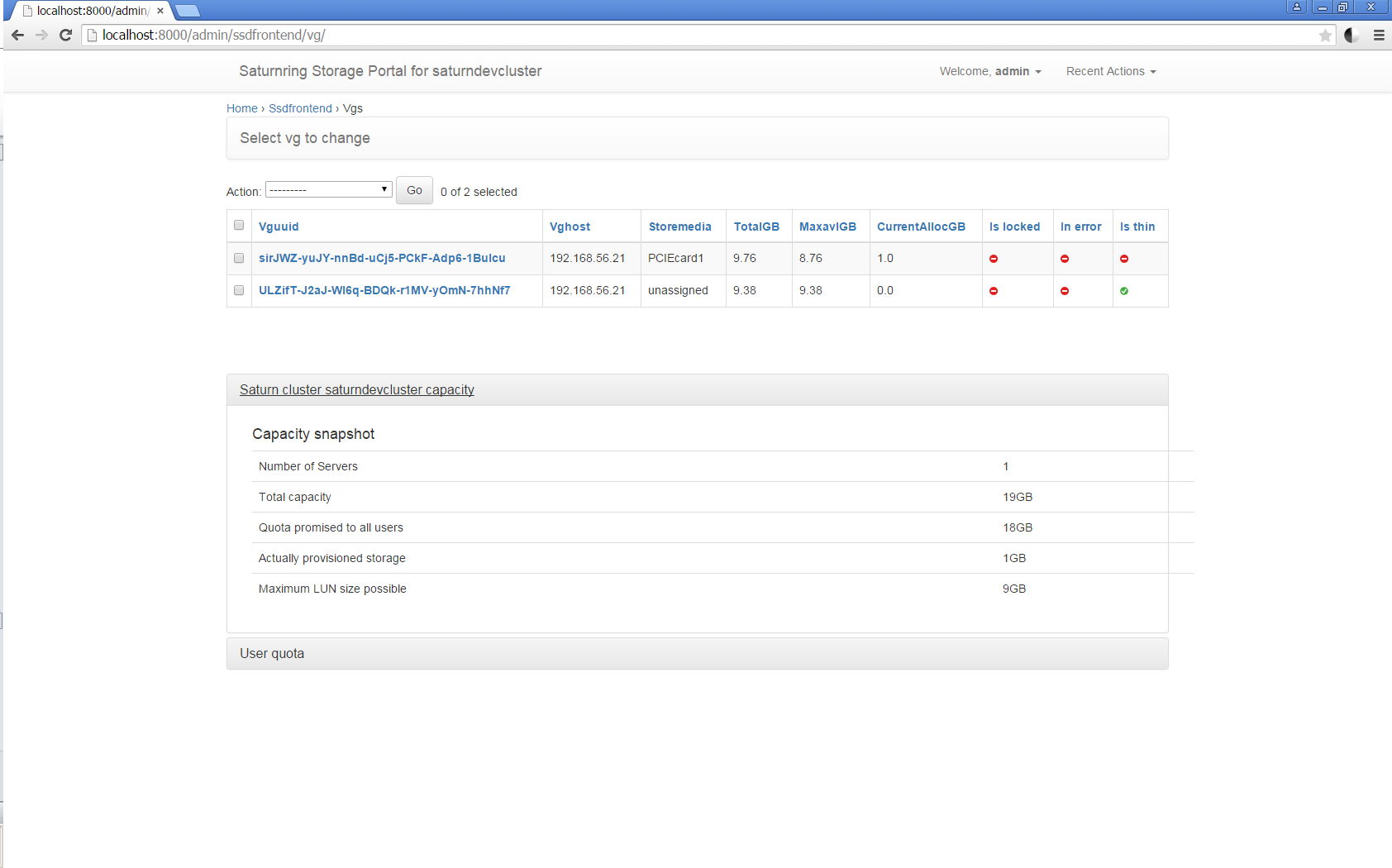


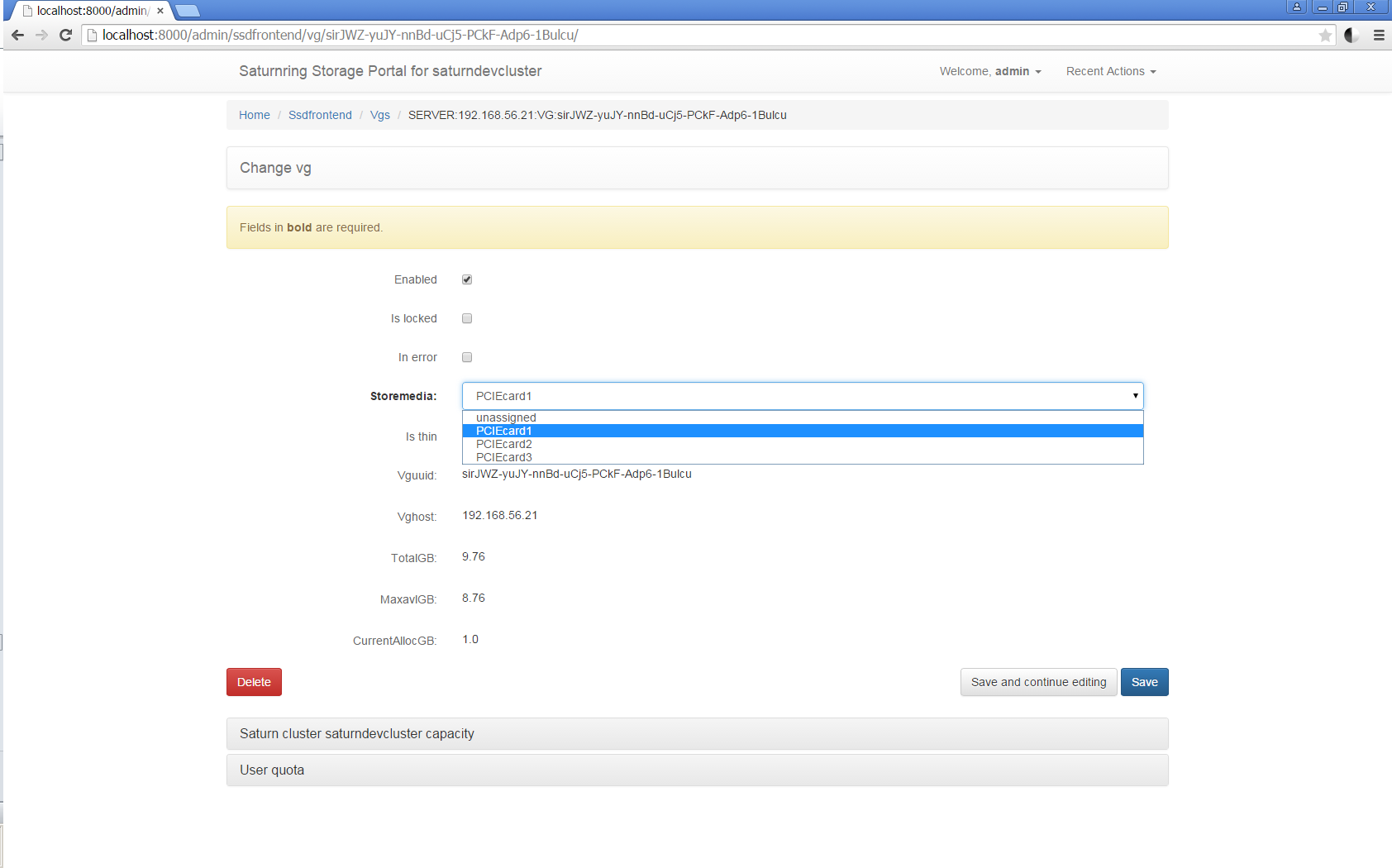


Targets and Logical Volumes

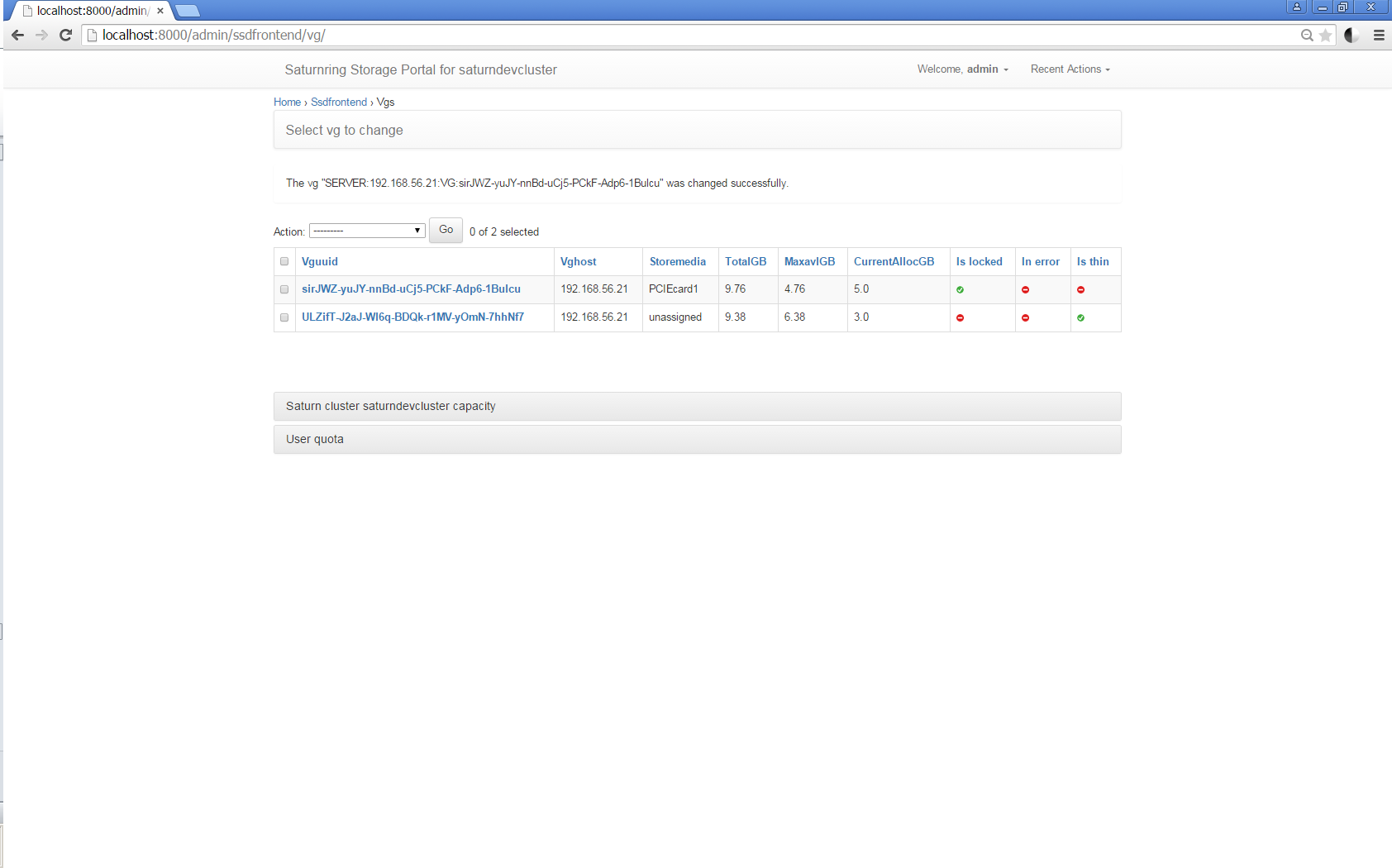


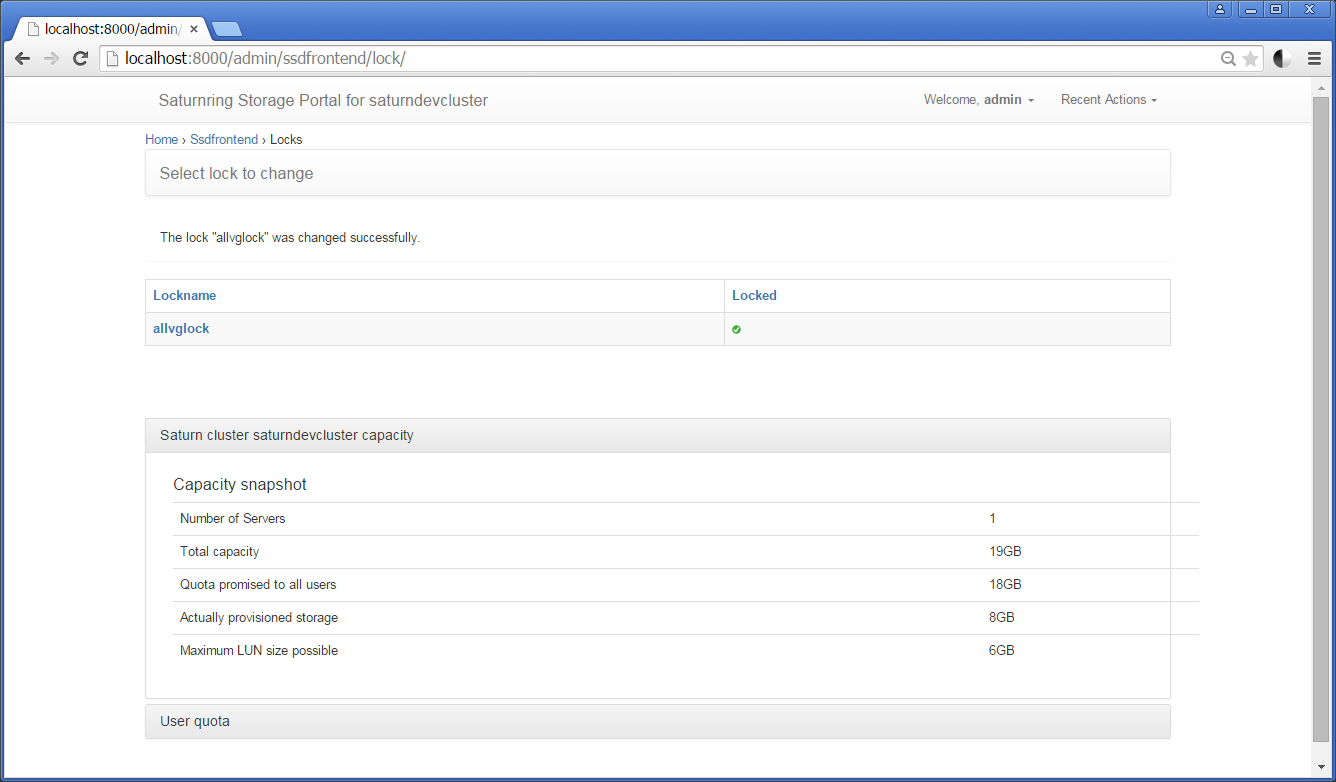




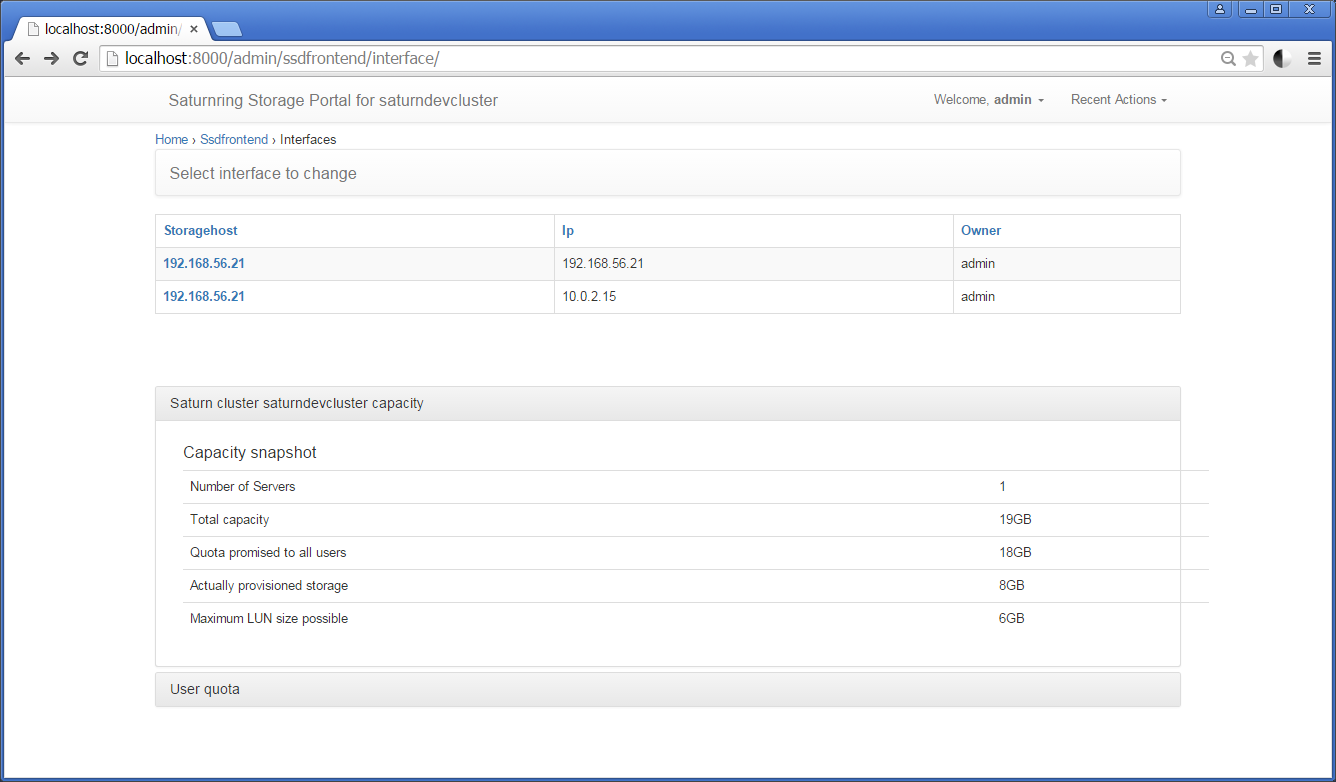
## Troubleshooting & locks

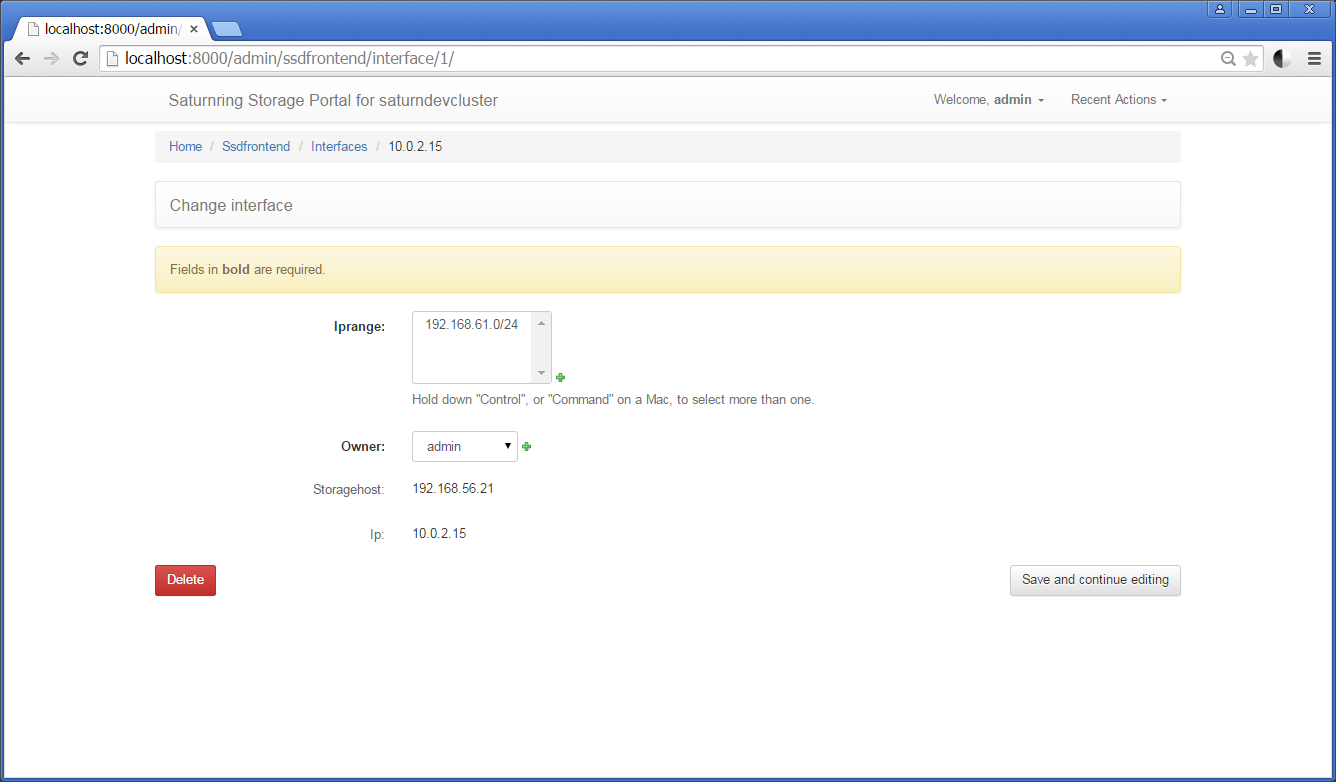


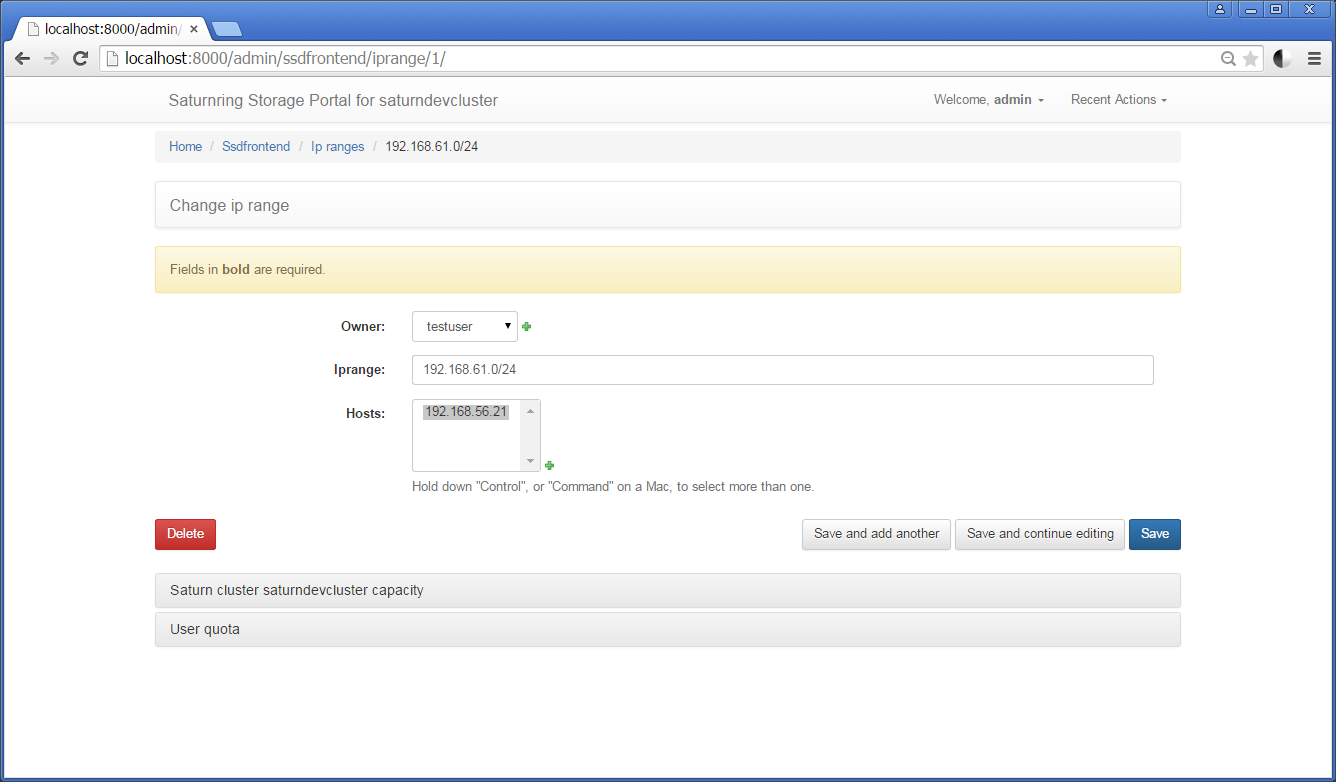


# 

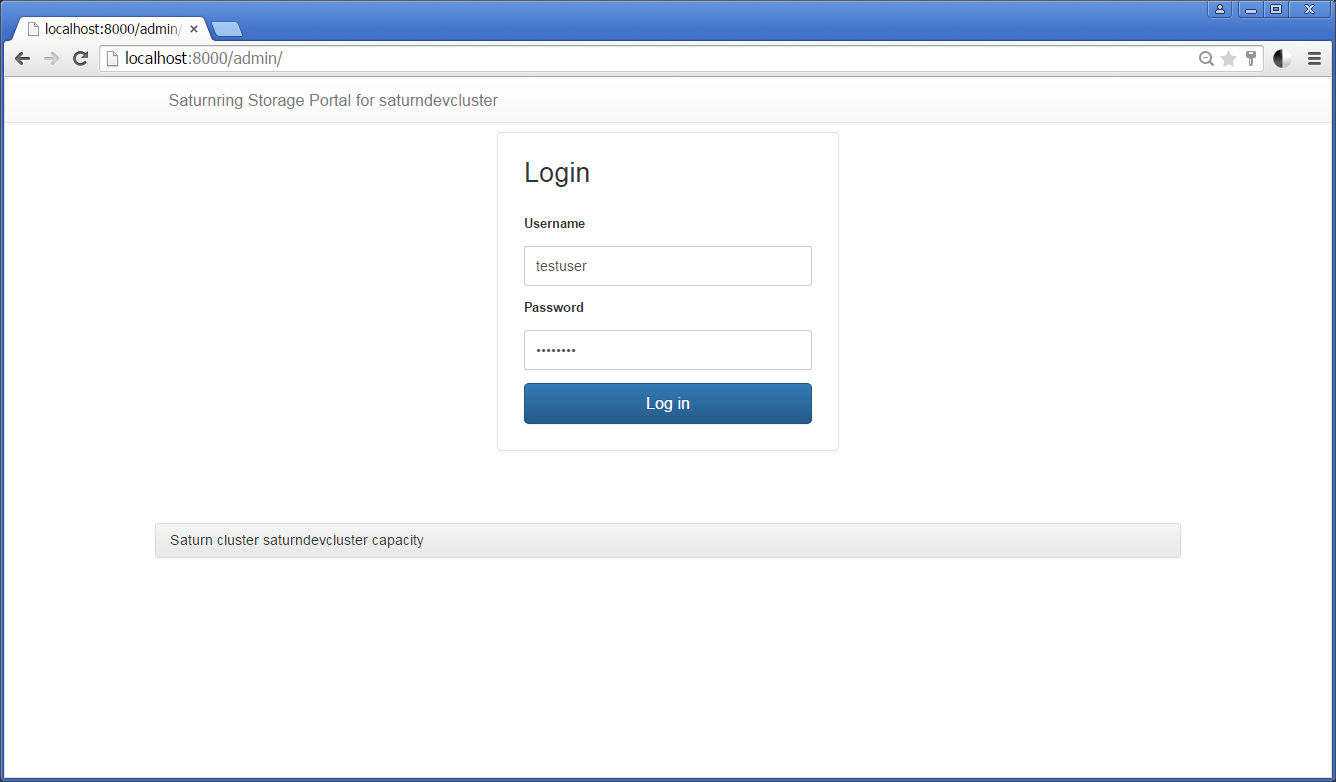
## Networking

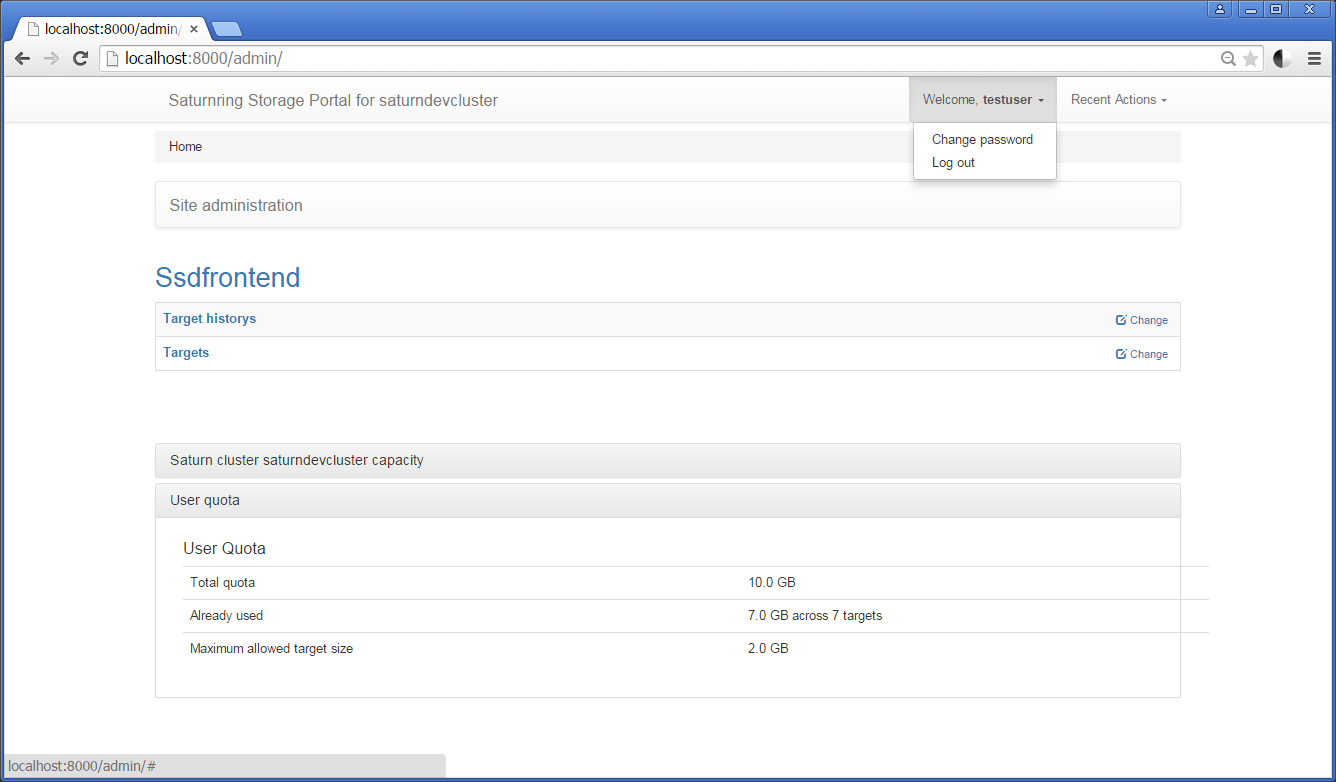


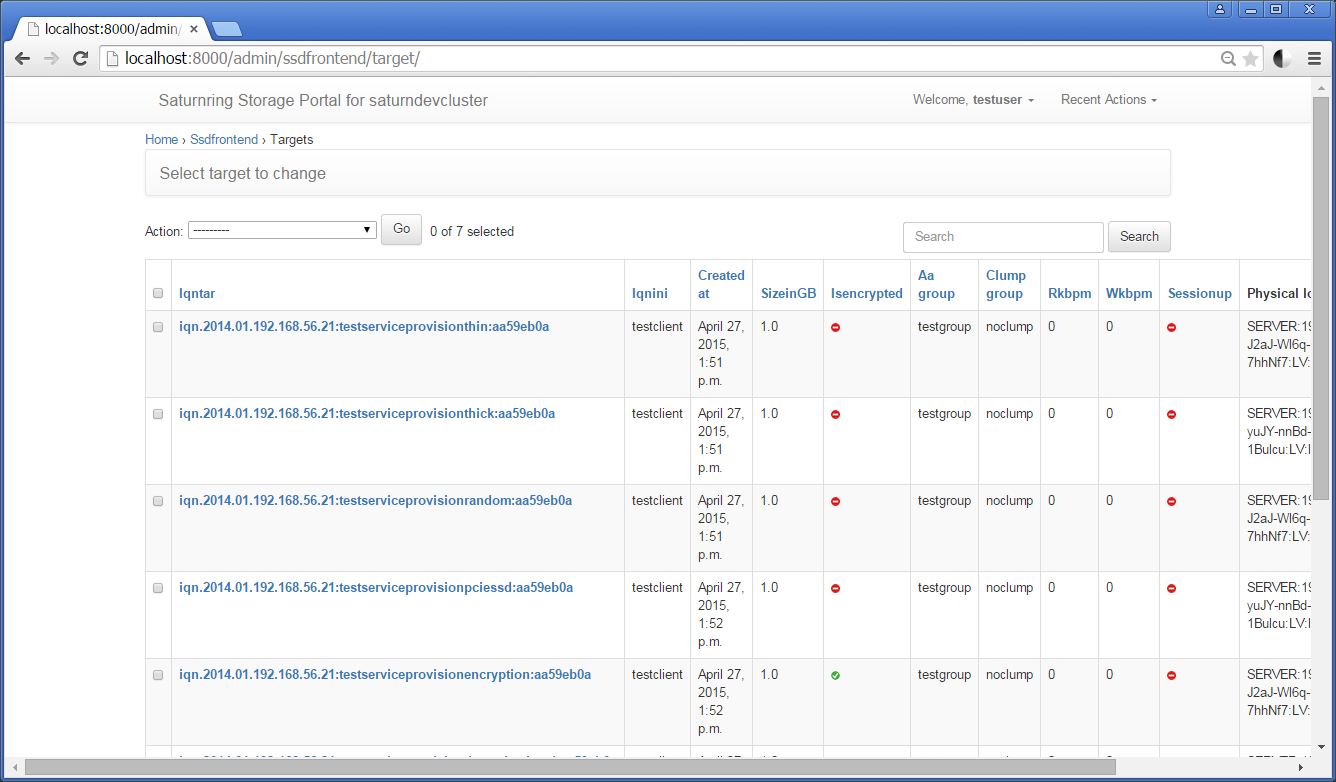


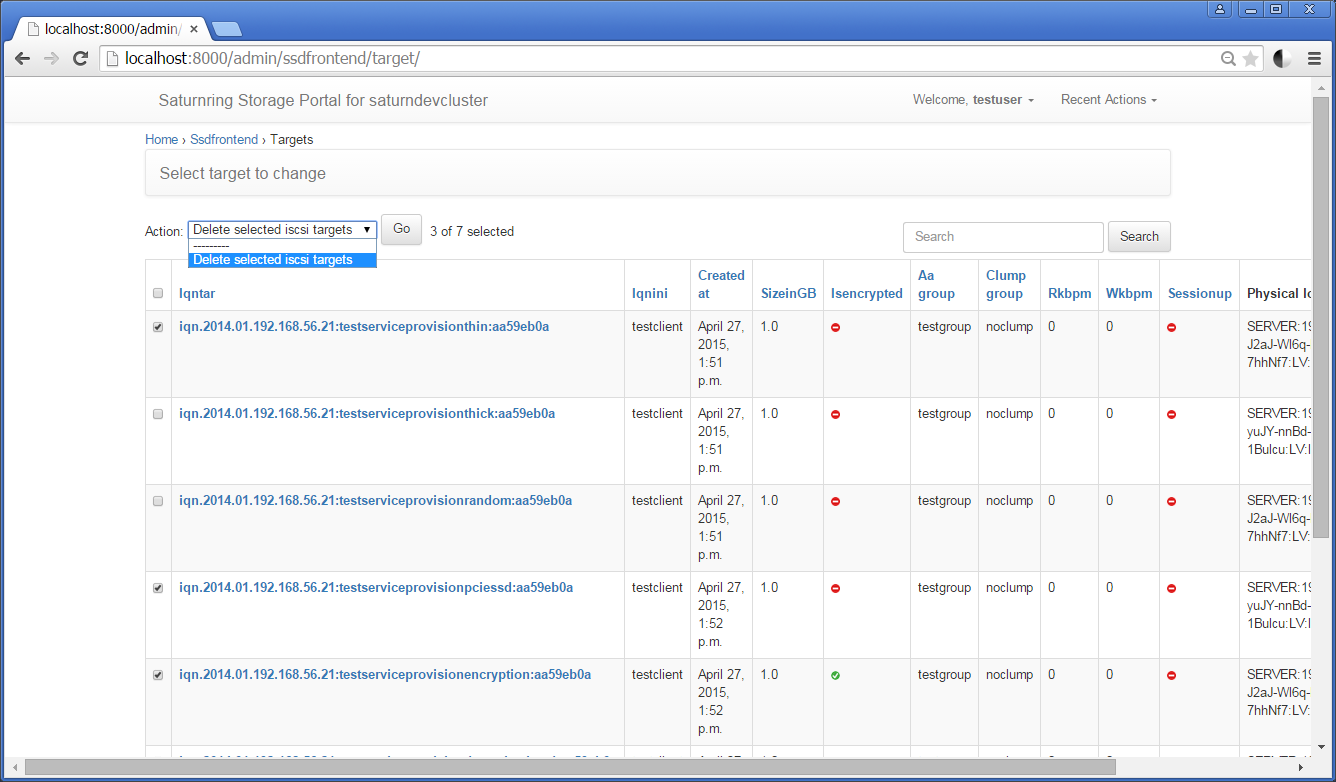


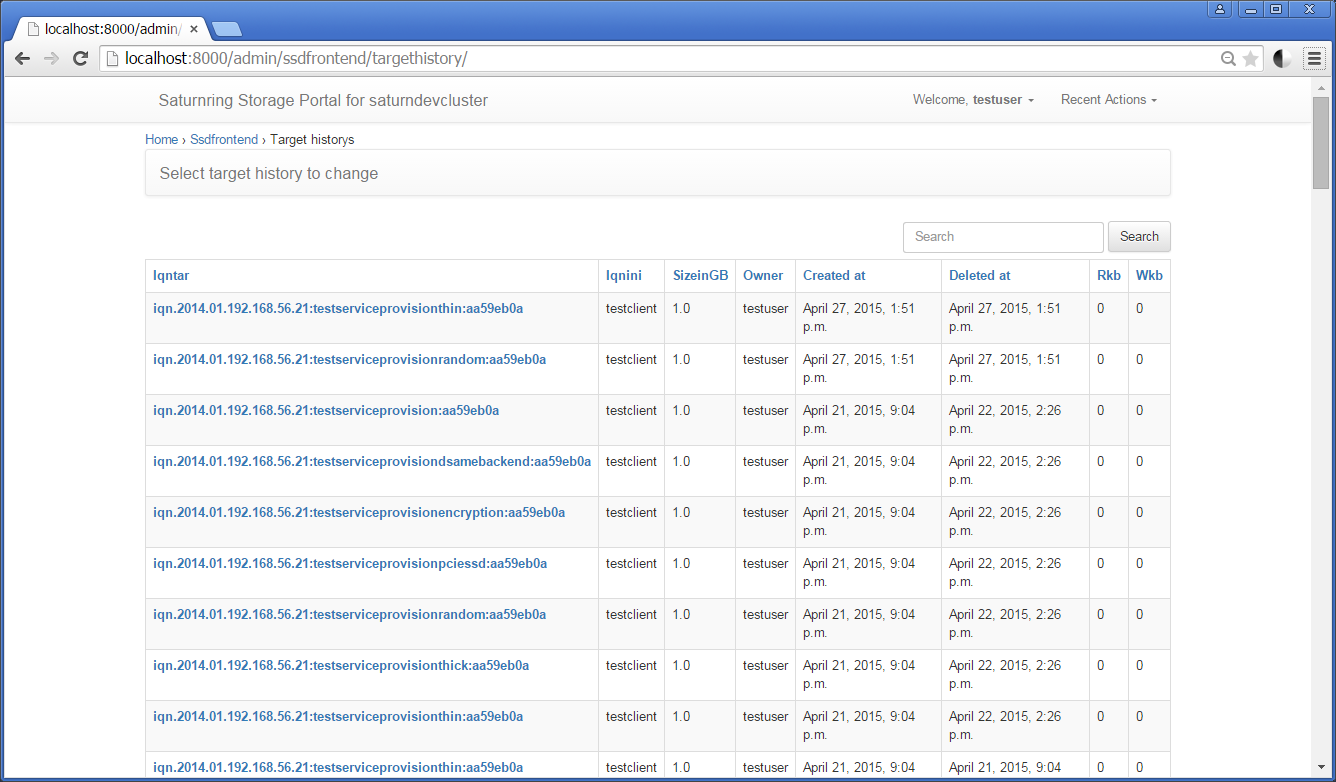
# Guide for Users











# Saturnring API

The primary purpose of the API is to control (provision/delete) storage via a HTTP request. 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" \

-d "isencrypted=1" \

-u 'testuser:password' \

| python -mjson.tool

This 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 iqnini>

1. 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). Targets with sessionup indicating true cannot be deleted.
2. sizeinGB: This is the requested storage size in GB – this is the size of the underlying LV backing the target.
3. 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.
4. targethost\_storageip1: The targethost storageip is the IP address to be used for the iSCSI connection.
5. 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.

### 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"

}

## 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.

# Security

# Advanced Configuration Topics