TrueNAS SCALE: A “Datacenter-in-a-box"

Exploring Security Concepts and Backup Immutability – Part 1
Revision 1.1 - NickF

What is a “Datacenter-in-a-box"?

Datcenters are large, sprawling rooms lined with cabinets that are filled with servers, switches, routers and firewalls. They require substantial investment in real estate, redundant electrical power, robust cooling systems, and multiple WAN connections.

In recent years, many companies and individuals have shied away from capital investments in the infrastructure required to maintain or build datacenters. The market has shifted towards the “cloud”, where large capital investments are replaced with recurring monthly costs for subscription services. Software-as-a-service (Saas), Hardware-as-a-service (Haas).

Google, Amazon and Microsoft, just to name a few, have had record growth and profits in their cloud computing businesses. Why is that? If we recognize that “the cloud is just someone else's computer”, why is it that the capital investment of these tech giants and their massive datacenters is any more valid than our own?

I believe the answer is two-fold. The first is that sometimes it’s easier to offload the risk and resources involved off to someone else. There’s not a lot of reasons for the average computer geek to host their own email server, and for many companies, there aren’t a lot of compelling reasons to host their own Exchange servers any more. Quite simply, it can be a headache and there’s other work to be done – many people feel that there is no reason to reinvent the wheel.

The second is a lack of knowledge and expertise. For decades now VMWare has dominated the on-premises datacenter market. Software like Proxmox, and KVM in general, have been eating up the “cloud” market. Neither of these solutions are easy to use and require technical expertise and finesse to manage. VMWare certifications and training can cost thousands of dollars and hundreds of hours of studying.

Enter TrueNAS SCALE. Unlike Proxmox and VMWare, it's relatively simple to setup and manage. Unlike proprietary software like Unraid, it’s free and open source. IXSystems, the developer of TrueNAS often touts the term “True Data Freedom”. If you have a disaster situation, why should you have to pay Amazon buckets of money in egress fees to

download your backups? If you are a YouTube content creator, backing up all of your videos can be a nightmare. Let's try and solve this problem together.

Why is this called "Datacenter-**in-a-box?"** With a little bit of elbow grease, we can turn something as small as an old corporate desktop, or as big as an IXSystems TrueNAS M50 into a powerhouse that securely holds all of our data in one place. After we get through the initial setup, we will have an easy-to-use and secure place to store all of our data and run whatever services we need inside of a single server. This guide is meant to follow my "Getting Started with Virtualization on TrueNAS SCALE" resource. If you haven't checked that out yet, please give it a read before we get started here.

The Security Onion

Cyber security is a complex topic. You can spend years studying security concepts, there are doctorate degree programs, there are certification programs and every day there are people and companies who get attacked by crypto malware. For us 'normal' computer geeks and systems administrators, we're never going to know it all. Thankfully, however, we don't *have to* know everything. For most use cases, understanding that security is implemented in layers, following best practices, and accepting the wisdom of the experts is a *good enough* starting point. Perfection is the enemy of good, and decision paralysis will likely result in an epic failure before getting a solution implemented 80% of the way there.

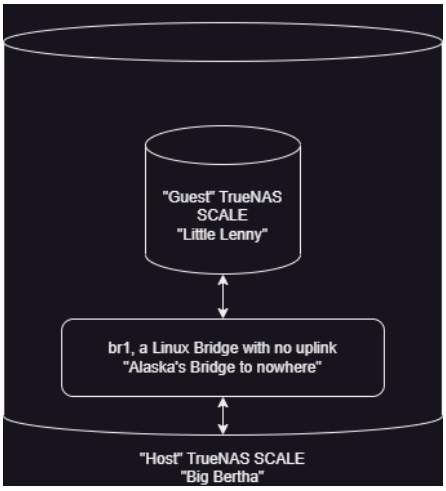
The question we are searching to answer is: How do we "air-gap" data and applications, and how do we ensure that our backups are "immutable"? In my sysadmin life I was presented with a challenge that forced me to answer these questions. We were tasked with taking a huge application back on-prem, a student information system. It was comprised of three primary components, a Web and Application server, a file storage server, and a database server. This system was previously hosted and maintained by the vendor. We had "outsourced" the risk of holding all of this restricted (DCL3) information, and now I was going to be responsible for keeping it safe.

The purpose of this article is to provide some guidance into implementing security at different individual layers of the stack. There is no one-size-fits-all solution to security, but knowing and understanding "what you can do where" makes decision making easier.

Laying the Groundwork: The "Air-gapped and immutable" backup

The "3-2-1" strategy for backup is often cited as a general rule of thumb, or a best practice for securing your data. That is: 2 local copies of the data and 1 remote copy. With a bit of virtualization magic sauce, we can accomplish having 2 local copies of our data inside one physical server. For this exercise, I am going to virtualize an instance of TrueNAS SCALE running on-top of our bare-metal TrueNAS SCALE installation. This configuration will require two host-bus adapters, or HBAs. One HBA will control the disks on the host operating system, while the other will control the disks on the guest.

When talking about layers, this is accomplished at a hardware level. Intel VTd/IOMMU are the technologies that allow PCI-E Passthrough. Since these technologies are on the silicon, it is the lowest possible layer of the onion we can use to build two systems inside of one. The next thing we will need is a secure way for the two systems to communicate with each other. We can accomplish this fairly simply, using a Linux Bridge interface that doesn't have an uplink to a physical network card. A logical design diagram of this might look something like this:



System Specifications

For this article, I am utilizing an iXSystems TrueNAS M50. This system has an *external* HBA that connects to a JBOD, or "Just-a-Bunch-Of-Disks". However, this same methodology will work with two *internal* HBA's connected to different disks inside of the same chassis.

In order to pass through the PCI-E device from the host operating system of "Big Bertha" and allow "Little Lenny" to use it, we have to determine what the (bus\#/slot\#/fcn\#) information is of the card.

By running the command:

```
lspci | grep "Serial Attached"
```

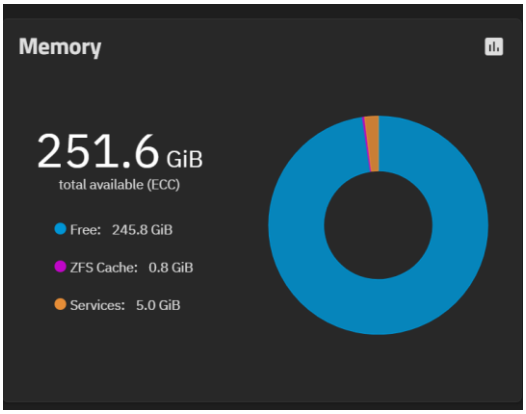
I can see that there are indeed two controllers in my system. In my case, af:00.0 controls the internal disks and **af:00.0** controls the external disk shelf.

18:00.0 Serial Attached SCSI controller: Broadcom / LSI SAS3008 PCI-Express Fusion-MPT SAS-3 (rev 02)

af:00.0 Serial Attached SCSI controller: Broadcom / LSI SAS3216 PCI-Express Fusion-MPT SAS-3 (rev 01)

DANGER, WILL ROBINSON, DANGER! If you select the wrong one, and your existing pool is present on the controller you select, you will have a very bad time! The easiest (non technical) way to determine which is which is to simply remove one from the server and run the command again.

"Big Bertha" currently has 256GB of RAM:



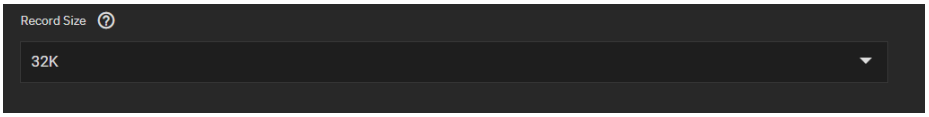
Since ZFS on Linux will only use ½ of the total system memory for ARC caching (see: [Module Parameters — OpenZFS documentation),](#) We can safely give 64 GiB to “Little Lenny” and have plenty left over for some future use.

“Big Bertha” has a 12 wide mirror arrangement for its pool, and we can store the “boot drive” for “Little Lenny” on that pool. “Little Lennie” will have a 12-drive RAIDZ3 inside of the 12-bay JBOD.

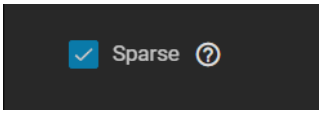
This is a high-end example using Enterprise grade hardware from iXSystems. Reviewing the minimum system requirements and recommendations I had published in the virtualization guide, it’s certainly possible to scale this example down. Realistically the SCALE VM under the host only needs to meet the bare minimum requirements [SCALE Hardware Guide | (truenas.com).](#)

Getting Started

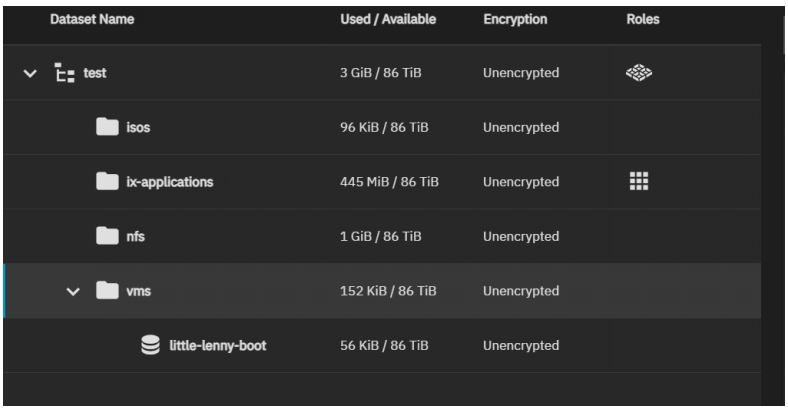
On “Big Bertha” we are going to go to Storage -> Manage Datasets -> Add Dataset. I generally like to make a parent dataset that holds all of my ZVols, and we’ll call this dataset ‘vms’. Set the record size to 32k.



Under that new dataset, we’ll create a Zvol called “little-lenny-boot", give it 100 GiB and mark it as “Sparse”.



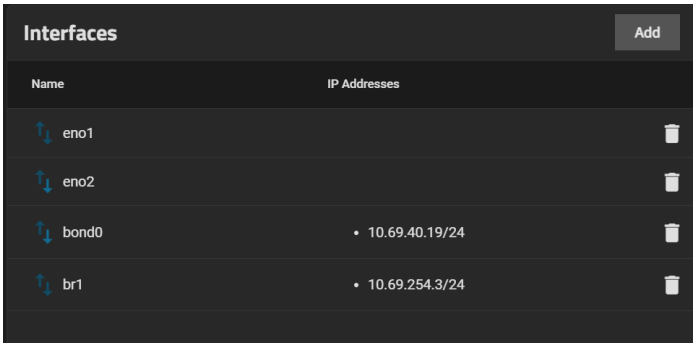
When you are done, it should look like this



The next step is building our 'Bridge to nowhere' network interface. Go over to Virtualization -> press Add

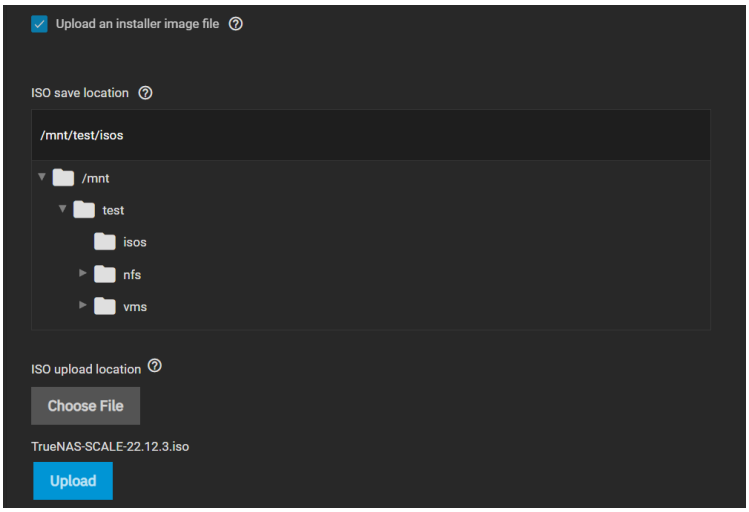
- Type should be bridge
- Name should be br1
- Description should be 'secure data'
- Do not select any tick boxes or bridge members
- Add an alias using RFC1918 address space that isn't in use elsewhere on your network. Leave a little space at the beginning of the range. I'll use 10.69.254.3/24.

When you're done, it should look something like this:



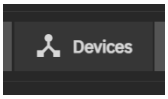
Now we are going to go over to Virtualization-> Add Virtual Machines

- Guest Operating system should be set to Linux
- Our name is going to be 'littlelenny'
- Description should be 'Little-Lenny TrueNAS SCALE Backup'
- We'll give him 1 CPU, 8 cores and 1 thread for now.
- I typically select 'Host Passthrough' for CPU Mode
- Assigning 64 GiB of RAM as discussed above
- It's also a good idea set the minimum to 64 GiB of RAM, so we always have that RAM reserved for this VM.
- Next, we'll change the Disk type to VirtIO, select 'Use existing disk image' and select the ZVol we created earlier.
- For networking, we'll sect VirtIO as the type again, and attach to the network we'll use for 'Management' and access to the WebUI, in my case bond0.
- Installation media will allow you to upload the latest TrueNAS SCALE ISO and mount it as a CD.

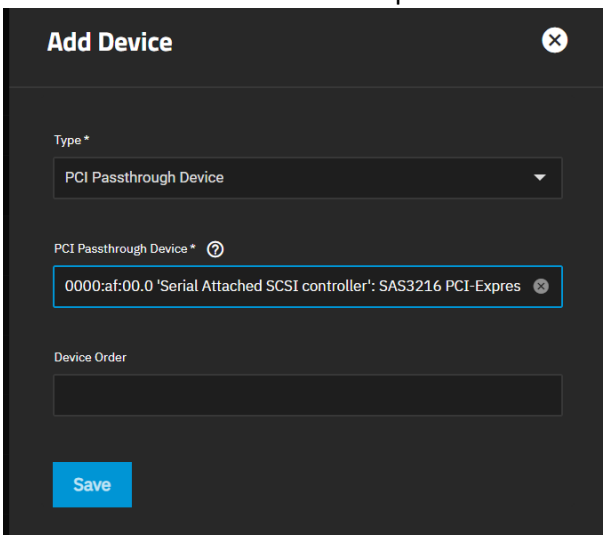


- Skip the GPU section and confirm.

With basic outline of our VM created, we'll have to now go through and do a few extra steps in the devices section.



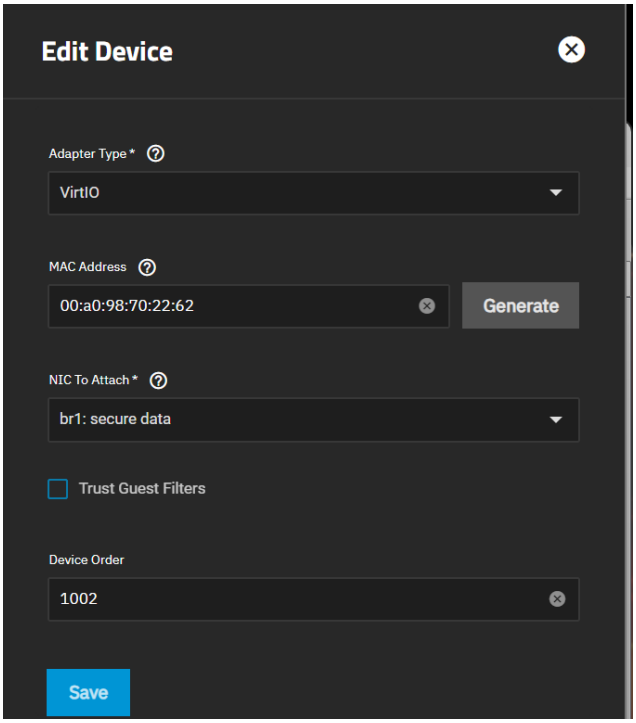
- Once there press 'Add' on the top and select 'PCI Passthrough Device'. Earlier we ran lspci and found that our HBA is **af:00.0** Serial Attached SCSI controller: Broadcom / LSI SAS3216 PCI-Express Fusion-MPT SAS-3 (rev 01) Select the correct one in the drop down menu and press save.



DANGER, WILL ROBINSON, DANGER! If you select the wrong one, your pool will suspend. If the VM automatically starts on boot, you basically are stuck in a "can't go forward can't go back" situation.

- Now we are going to add our back-end storage network which will live on the br1 interface we created earlier. Press Add Device ->Adapter type should be VirtIO->Attach to br1 and press save. Take a note of the MAC

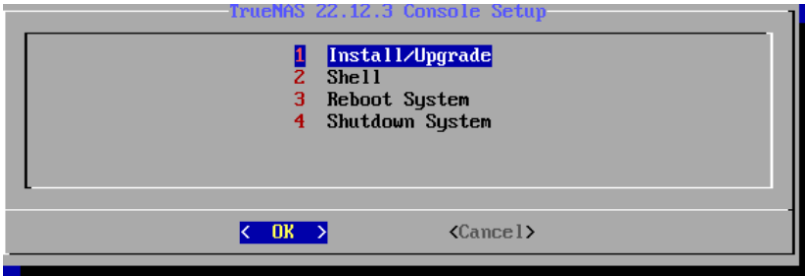
address of this interface so we can easily distinguish it later. For me it's 00:a0:98:70:22:62



Installing SCALE – Little Lenny's Legacy begins

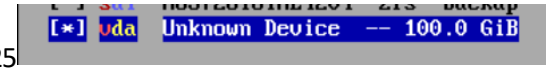
Let's now turn on the VM we have created and get the installation wizard going. The VM will take some time to start up, longer than typical, because we have reserved all 64 GiB of memory above – just be patient.

One it turns on you will be greeted with the typical TrueNAS SCALE installer.



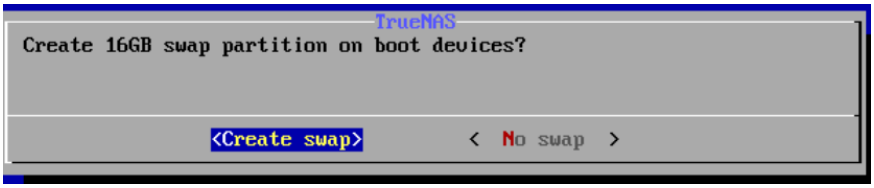
I'm not going to go through this step-by-step but the key thing here is to install SCALE to the boot-drive we created a ZVol for on "Big Bertha" and to give our network interfaces the correct IPs for their respective networks.

For the storage, A VirtIO device will show up as "vda" so it will be pretty easy to identify.



25

I generally like to give the boot drive a swap partition, and we have provisioned enough space for it.

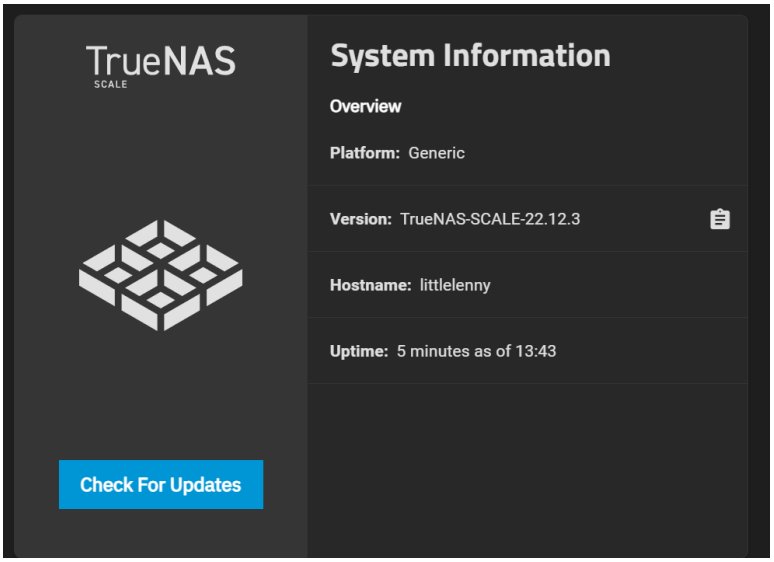


When the installation completes, we see the typical TrueNAS shell menu. If you go directly to the "configure network interfaces" option directly, you will have a difficult time determining which interface is ens3 and which interface is ens4. Drop down to the Linux Shell, option 7. Now type "ip a".

```
root@truenas[~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:a0:98:08:6b:3f brd ff:ff:ff:ff:ff:ff
    altname enp0s3
    inet6 fe80::2a0:98ff:fe08:6b3f/64 scope link
       valid_lft forever preferred_lft forever
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:a0:98:70:22:62 brd ff:ff:ff:ff:ff:ff
    altname enp0s4
    inet6 fe80::2a0:98ff:fe70:2262/64 scope link
       valid_lft forever preferred_lft forever
root@truenas[~]# _
```

With that, we can now determine with certainty that ens4 is the second network interface we configured and mapped to the internal bridge network. Now we can type "exit" to drop back to the shell menu and configure the IP address alias information for both of our adapters.

At this point we should have two SCALE systems running on the same server! Hooray!



Big Bertha: Building our pools and datasets

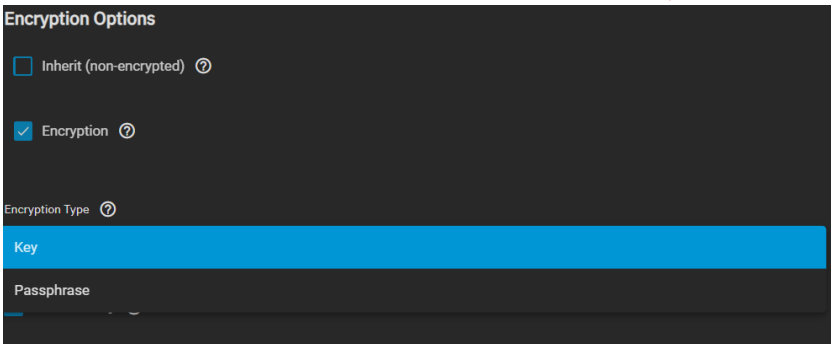
There are many ways to go about structuring your data, and there is no "right answer". For the sake of this example, I am going to start the journey on 'Big Bertha'. This is the only server that our clients will be using. When I built the pool on that server, I made the decision not to encrypt the whole pool.

This decision was made in part because of this change: [NAS-121871] Make sure unencrypted dataset cannot be created inside an encrypted dataset - iXsystems Jira (atlassian.net). You can no longer create an unencrypted dataset inside of an encrypted one. For more information on this topic, the docs are very good here: Storage Encryption | (truenas.com)

That being said, I will be making encrypted *datasets* on 'Big Bertha' to hold all of our most sensitive information. For the sake of our example here, I am going to make two datasets that will have SMB shares associated with them. One will be a repository for VEEAM backups, and the other will hold all of our secure corporate/personal documents and pictures.

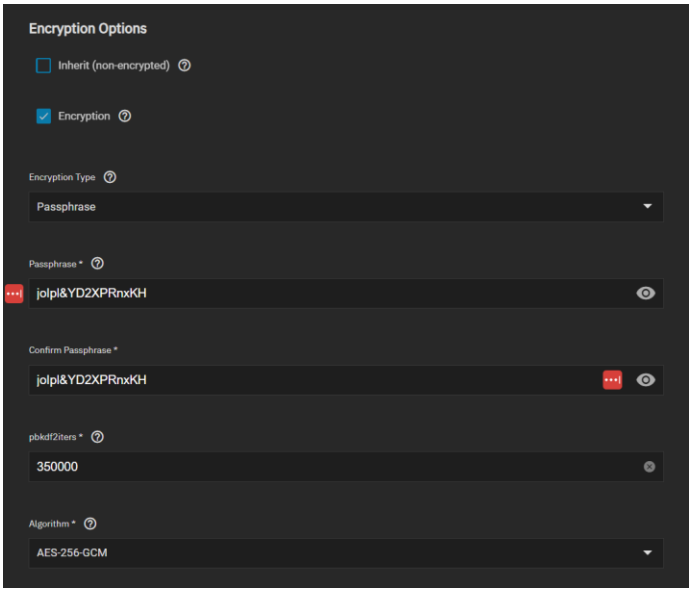
For simplicities sake, I have opted to make a single parent dataset that is encrypted, and nest those two under it, sharing the encryption keys. When creating the dataset, we get a couple of options:

IF YOU LOSE YOUR KEY OR FORGET YOUR PASS PHRASE, YOU WILL LOSE ACCESS TO YOUR DATA!

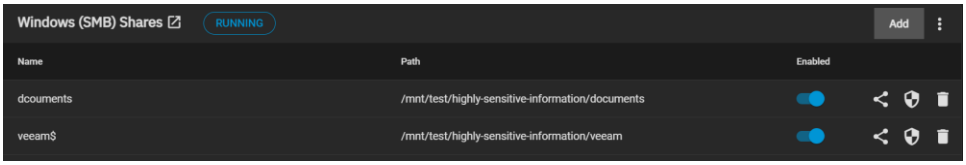


An encryption 'key' is a file that will open the proverbial encryption lock, whereas a passphrase is like a code you need to speak to the guy behind the door at the speak-easy. While I always fantasized about putting a thumb drive with encryption keys on a thumb drive behind a literal "break glass" box, I personally prefer passphrases I can easily store in a password manager. Although, KeePass can probably store both formats.

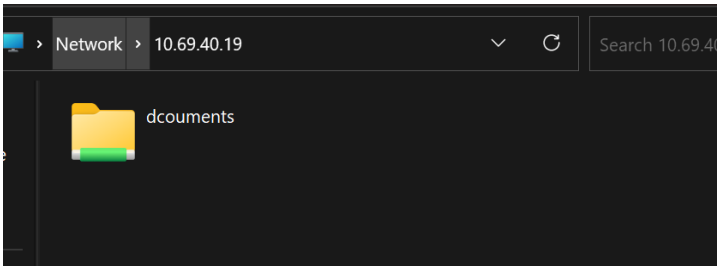
For now I am going to use a strong passphrase which I'll share in cleartext for your entertainment and disgust. For my documents dataset, I will leave the default 128k record size (conservative), but for my Veeam dataset, I am going to use 1MB. More info here: Tuning recordsize in OpenZFS | Klara Inc (klarasystems.com). Thanks Allan Jude!



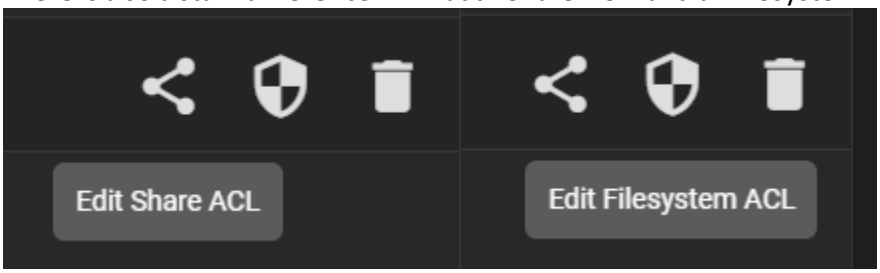
I'm not going to dive too deep here into SMB shares and ACLs here, but it's important to touch on some key points.



Putting a dollar sign at the end of a share name makes it hidden from network discovery, and will require the user to know the full path. This is a behavior native to SMB in general and something that Windows sysadmins have been leveraging for decades. Notice I do not see the veeam share from my workstation.

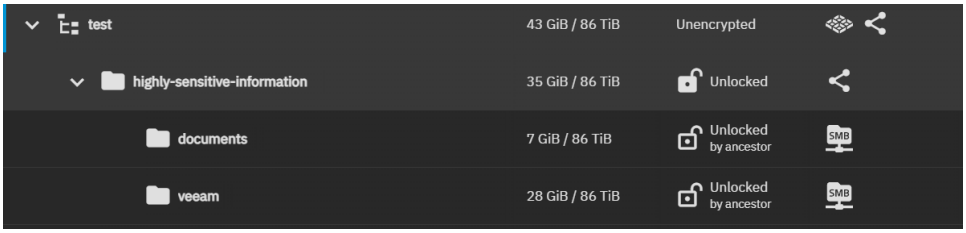


There is also a stark difference in what a "Share ACL" and a "Filesystem ACL" are used for.



On TrueNAS by default there are no restrictions for anyone to navigate to, and access, a share. If the "Share ACL" is left at the default, it is possible for *any user* to *open a connection* to a share, even if they are unable to see, read, write or modify it's contents. What they can do once they are connected, in terms of traverse, read, write, execute, modify, etc are all controlled by the "Filesystem ACL". In part 2 of this series, I will be going through and further discussing how to harden our deployment. For now, this article may be helpful [Setting Up Permissions | (truenas.com)]

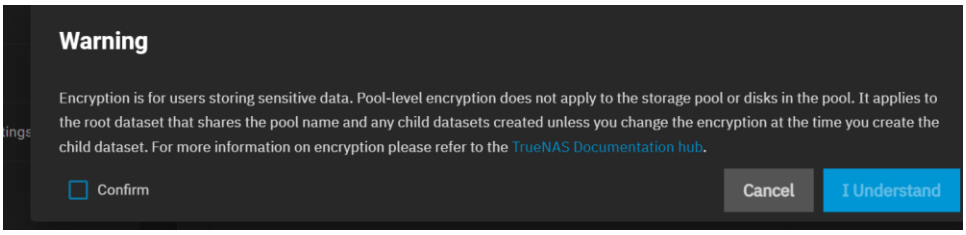
With that, my datasets are created and encrypted, and I have data seeded to them from our backups and clients.



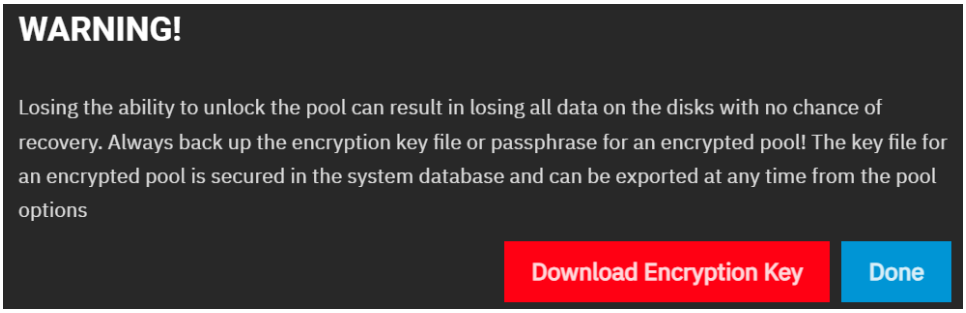
As a note from the Veeam perspective: all of what we have done is totally transparent at the file system layer. You can stack application layer encryption from Veeam itself. [Backup Job Encryption - User Guide for VMware vSphere (veeam.com).] Please just use a different passphrase 😜.

Little Lennie: Building our Pools and Datasets

The pool topology and encryption strategy for "Little Lenny" are going to be a bit more conservative. We are using a RAIDZ3 because we are looking for maximum redundancy here at the cost of performance. We are also going to implement encryption *at the pool level*.

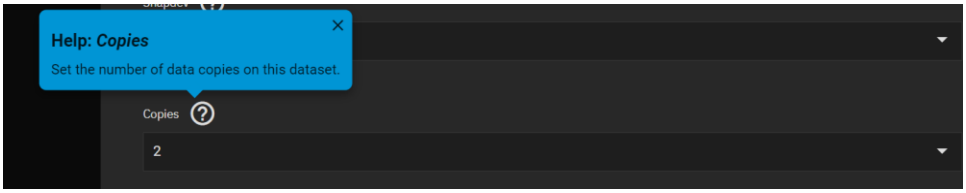


There is no option here to use a passphrase, only a key is supported. It will be automatically generated for you when you create the pool. Download the key (stored in a JSON file) and put it somewhere safe and preferably offline.

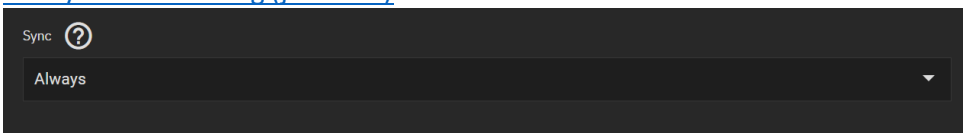


With our pool created, we can now increase our resiliency a bit further. In ZFS, by default, *metadata* is stored at least 2 times across your pool. Data on the other hand, is stored only ONCE (this is separate from parity) and for obvious reasons. ZFS scrubs are *designed* to prevent bit-rot and in 99.9% of the cases, this is enough.

However, I am being extremely conservative here. I have a JBOD connected to my NAS with a single SAS cable to a single HBA. If that cable or my HBA go wonko, we may be in a situation where there are *many thousands* of checksum errors. In order to be absolutely certain that my data doesn't get corrupted in such an event, I'm going to set copies of my data to 2 for the entire pool. Please note: This is not a replacement for RAIDZ, it is a layer ON TOP. [zfs: copies=n is not a substitute for device redundancy! – JRS Systems: the blog (jrs-s.net)](jrs-s.net) Furthermore, in defense of copies: [On ZFS Copies (richardelling.com)](richardelling.com)



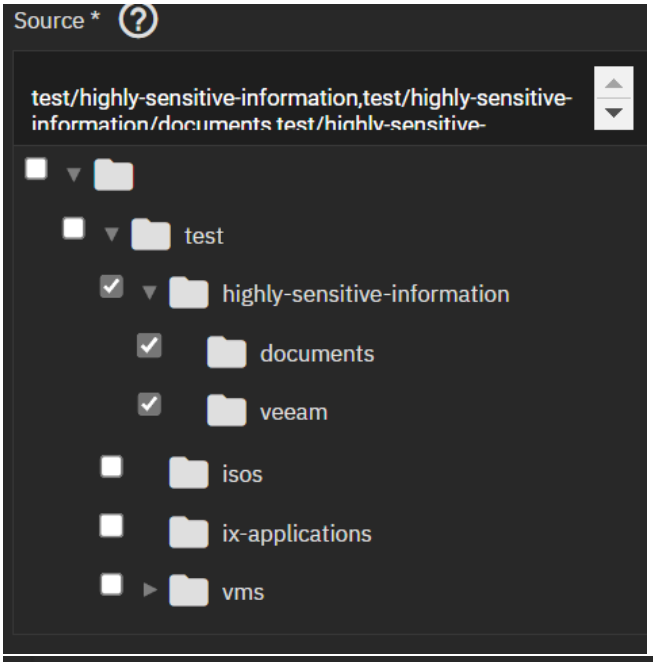
Next I am going to create myself a dataset in order to act as the parent for the receiving end of my backups. There I am going to set Sync=Always. Even though the data in flight is only going to traverse over a virtual bridge (rather than a physical network port), I want to be absolutely certain all data is written to disk. [ZFS sync/async + ZIL/SLOG, explained – JRS Systems: the blog (jrs-s.net)](jrs-s.net)

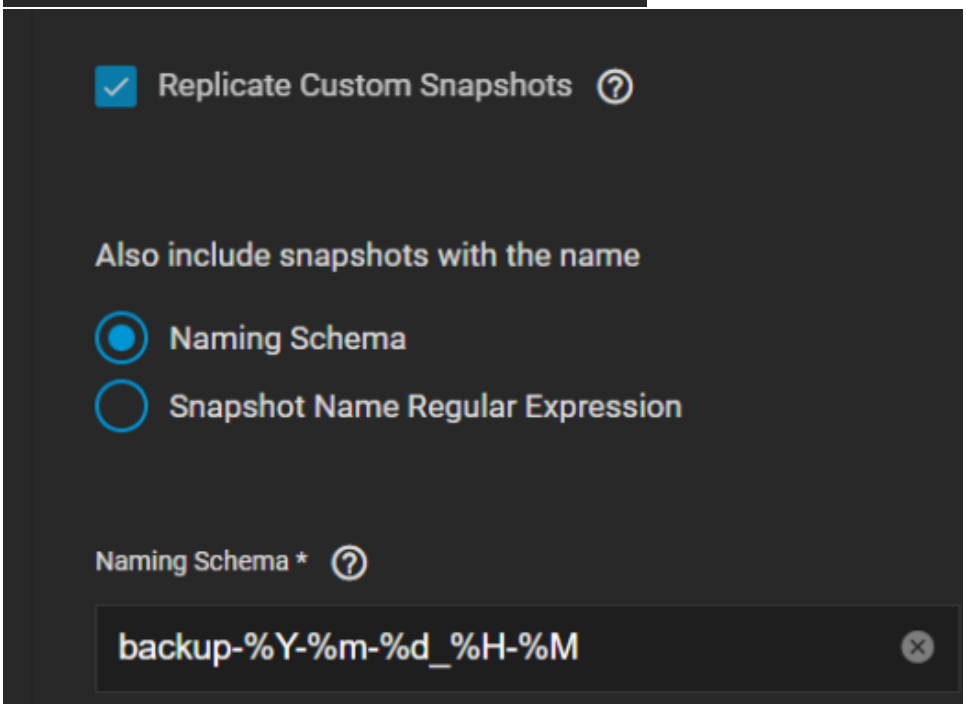


Let the replication BEGIN!

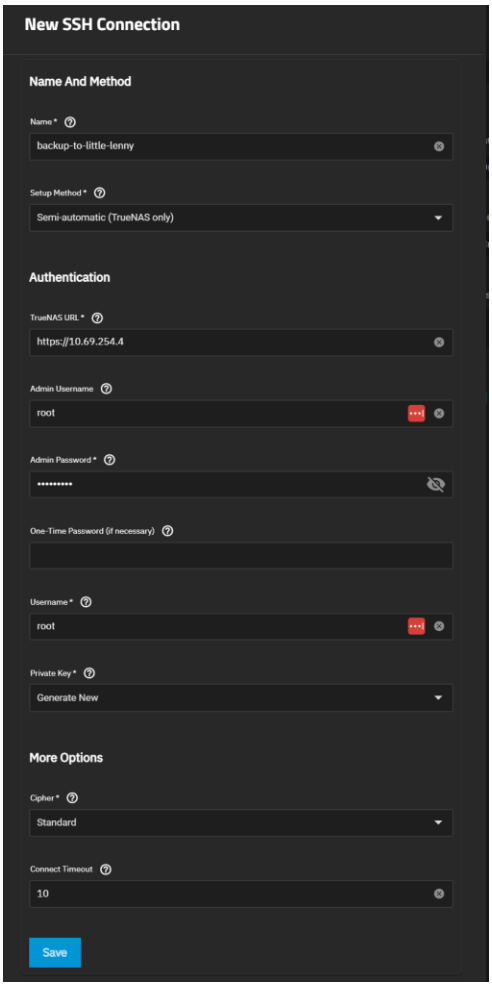
We will circle back to doing more with Little Lenny in Part 2 when we talk a bit more about hardening. For now, Let's flip back over to Big Bertha. If we navigate to Data Protection -> Replication Tasks -> Add, we can begin the backup process.

We're going to set the source location to be "On This System". I'm going to select the parent "highly-sensative-information" dataset, and I won't tick the "Recursive" box so it will not replicate any other snapshot tasks we make later. Next I am going to select "Replicate Custom Snapshots" and change the naming schema to have the prefix "backup". This will allow us to differentiate retention policies for different snapshot tasks on the source system later if we choose.

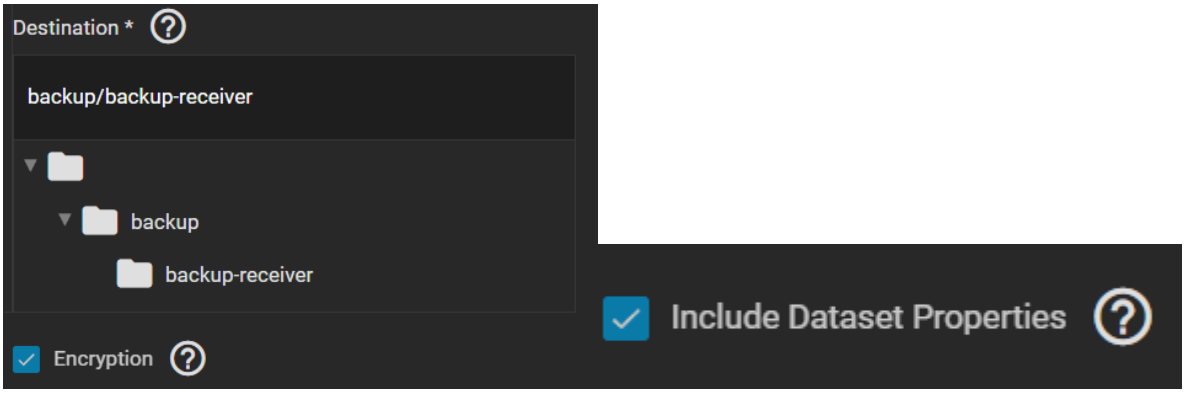




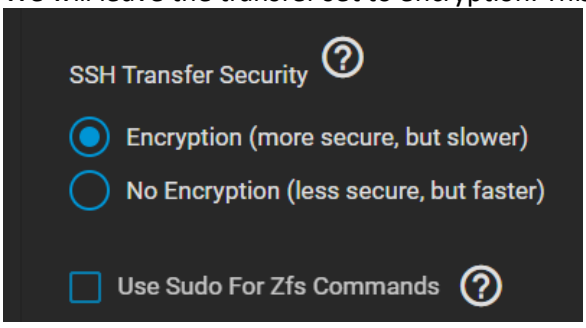
The destination location should be set to "On a Different System" and we'll follow the "semi-automatic" workflow. I am going to name it "backup-to-little-lenny" and in the URL section, I am going to use the IP address of our internal bridge network – NOT the management one. I will tell the system to generate a new private key, and press save.



Next we'll set the destination to the dataset we created earlier with sync=always. You do **not** need to tick the encryption box below, and you should not tick it. The dataset properties, including encryption, will be replicated over automatically.

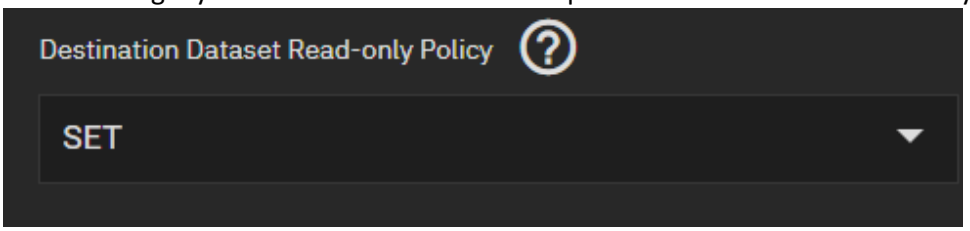


We will leave the transfer set to encryption. This way, both data in flight and data at rest are encrypted.

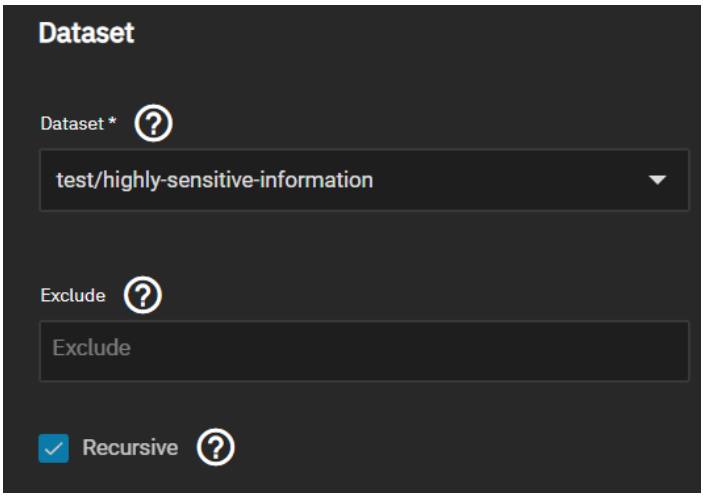


Then Press Next, and select the schedule you want to use. I'm going to use the default "Daily Schedule" but I will change the Destination Snapshot Lifetime to 4 Weeks (double the 2 week default we will hold on the source system). We can then press start replication.

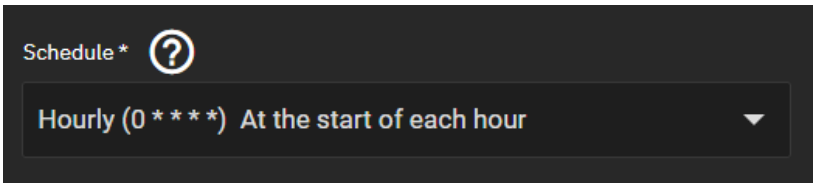
Worth noting: By Default – the destination snapshots are marked as "Read-Only"



Now, lets go over to the Periodic Snapshot Task that was created, and tick the recursive box:

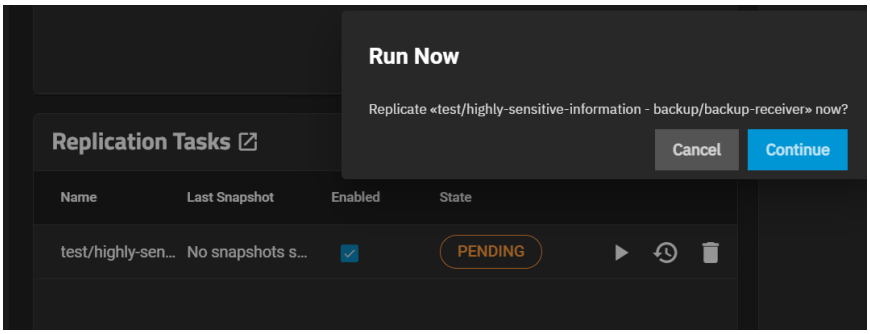


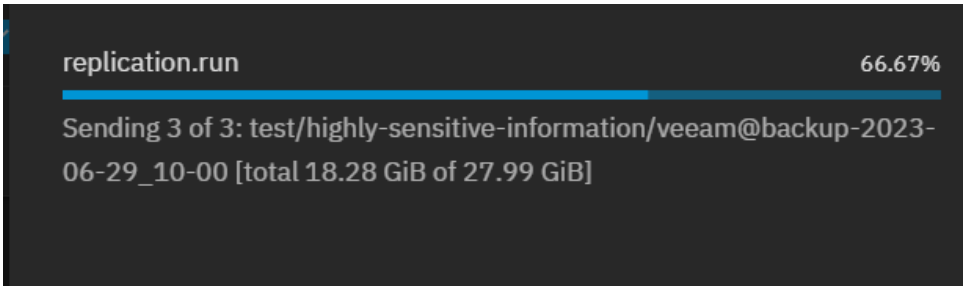
And I am also going to change the Snapshot task to run Hourly.



What this means is, even though I am only shipping the snapshots to Little Lenny at Midnight Each night, I am snapshotting the source datasets every single hour. We are sending all of those hourly snapshots nightly. Since by default, we are allowing the system to take empty snapshots, there isn't really a cost here.

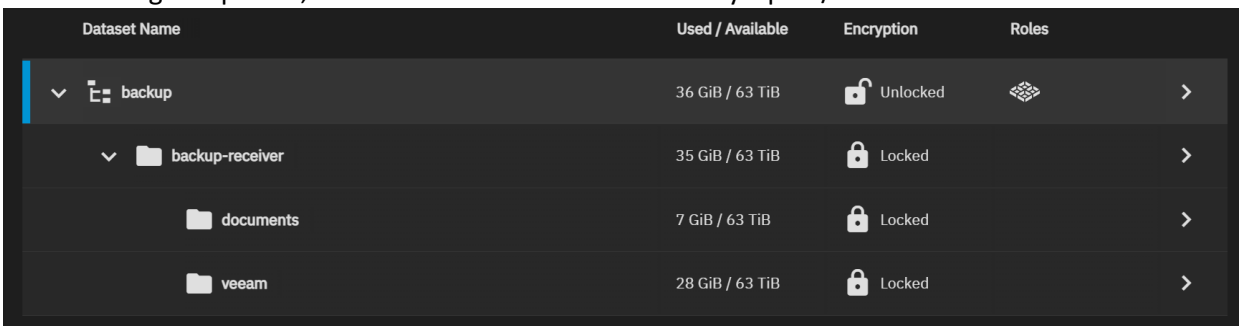
After the initial first snapshot is created, You can manually make one if you are impatient, as long as you follow the naming schema we defined earlier. Press the Play button to start the initial seeding.



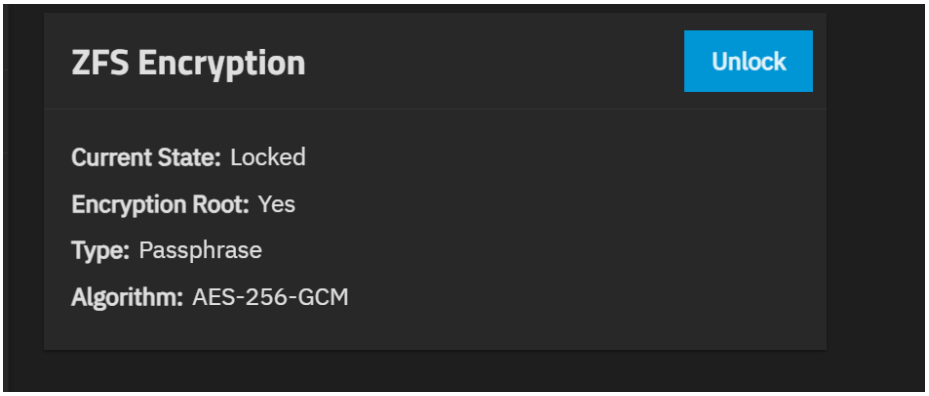


Reviewing the Deployment Model

With seeding completed, let's now take a look at Little Lenny's pool/datasets.



Notice that the "backup-receiver" dataset, and it's children are all locked. This is because "Little Lenny" does NOT have the keys we created on "Big Bertha" (or passphrase in this example) to those datasets. So not only are they read only like we had set earlier, we cannot even read the data without entering our passphrase!



In brief summary form, lets list what we have accomplished.

- We created encrypted datasets on our host system that hold all of our most sensitive information. Only the host system has those encryption keys.
- We created a back-end sandboxed virtual machine that has an encrypted pool. Only the VM has those encryption keys.
    - The backend pool here is highly redundant *and* resilient, using both a RaidZ3 and Copies=2.
- We created a virtual network inside of the host system. We are sending an encrypted data stream from our host to our guest through this network – never touching our production netowork.
- We sent the encrypted datasets (and marked them read only!) from the host system to the guest – but without giving it the keys

In Part 2 of this series we will discuss further hardening our deployment model. Let's get whacko together soon.