# RHEL 8 Update

Sander van Vugt

# RHEL 8 Update

Installation

# Installation Specifics (beta)

- Select NVMe Disk Driver
- Select Workstation to install a server with GUI

# Working without Subscription

- Loop mount the installation disk
- Make repositories for BaseOS and the AppStream directory
- Ensure that an empty file with the name redhat.repo exists in /etc/yum.repos.d

# RHEL 8 Update

## Application Streams

# Understanding Application Streams

- An application stream is used to separate user space packages from core kernel operations
- The goal is to make it easier to update user packages without requiring major revision updates on the OS itself
- Multiple packages can be made available at the same time
- Base packages are provided through the BaseOS repository
- AppStream is provided as a separate repository, make sure to include it when setting up repositories manually!

# Understanding Modules

- AppStream is delivered in two ways formats
  - Traditional RPMs
  - New Modules
- In a Module multiple software versions can be offered. Each software version is added as a stream. Each module does have a default stream
  - Use **yum install package-name** to install the package using the deault module and stream
  - Install specific module streams using **yum install @module:version**, as in **yum install @postgresql:9.6**

# Managing Modules

- **yum module list** shows available modules
- **yum module info *module-name*** shows info about modules, including streams, packages etc
- **yum module info --profile php:7.2** shows information about a specific stream
- **yum module @php:7.2** will install a specific stream of the PHP module

# RHEL 8 Update

Storage: File Systems

# Understanding Stratis

- Stratis is a new volume management file system that is faster and easier to manage than its predecessors
- It is Red Hat's answer to Btrfs, ZFS and LVM and enables advanced storage features
  - Thin provisioning
  - Snapshosts
  - Cache tire
  - Programmatic API
  - Monitoring and Repair

# Understanding Stratis - 2

- Stratis is not a file system, it's a solution that helps organizing storage into pools from which multiple independent file systems can be created
- Stratis is user-space and monitors underlying kernel space storage components such as device mapper and XFS
- Being user-space makes it easier to develop, provide an API as well as integrate with other non-kernel storage APIs such as Ceph and Kubernetes CSI
- Stratis can be used on top of any storage device (including LVM)

# Understanding Stratis - 3

- The stratis Storage Pool is created from one or more storage devices and volumes are created from the pool
- The file system is put on top of the volume and is an integrated part of it
  - that means that by resizing the volume you'll automatically resize the FS as well

# Using Stratis

- **yum install stratis-cli stratisd**
- **systemctl enable –now stratisd**
- **stratis pool create mypool /dev/nvme0n2p1**
  - Add new block devices later using **stratis blockdev add-data**
  - Note that the block device must be at least 1 GiB
- **stratis fs create mypool myfs1**
  - Note this will create an XFS file system!
- **mkdir /myfs1**
- **mount /dev/stratis/mypool/myfs1 /myfs1**
- **stratis pool list**
- **stratis filesystem list**

# Using Stratis Snapshots

- **stratis fs snapshot mypool myfs1 myfs1-snapshot**
  - Changes to the original FS will not be reflected in the snapshot
- Revert the original volume to the state in the snapshot
  - **umount /myfs1**
  - **stratis fs destroy mypool myfs1**
  - **stratis fs snapshot mypool myfs1-snap myfs1**
- Note that this approach wouldn't work on LVM!

# Getting Information

- **stratis pool list**
- **stratis fs list mypool**
- **stratis blockdev list mypool**

# RHEL 8 Update

Storage: VDO

# Understanding VDO

- VDO (Virtual Data Optimizer) is the Virtual Disk Optimizer
- It is used as a separate volume manager on top of which file systems will be created
- Provides thin-provisioned storage
  - Use a logical size 10 times the physical size for VMs and containers
  - Use a logical size 3 times the physical size for object storage
- Used in Cloud/Container environments
- It manages deduplicated and compressed storage pools in RHEL 8

# Setting up VDO

- **yum install vdo kmod-kvdo**
- **vdo create --name=vdo1 --device=/dev/nvme0np2 --vdoLogicalSize=1T**
- **mkfs.xfs –K /dev/mapper/vdo1**
- **udevadm settle** will wait for the system to register the new device name
- In /etc/fstab, include the **x-systemd.requires=vdo.service** mount option
- Monitor using **vdostats --human-readable**

# RHEL 8 Update

Networking

# Changes in Networking

- NetworkManager is still the default solution to manage network configurations
- IPVLAN connects containers nested in virtual machines to networking hosts
- A new TCP/IP stack is provided that provides bandwidth and round trip propagation time (BBR)
  - BBR is a new TCP delay-controlled control algorithm
  - BBR brings better network quality with less packet loss and minimized latency

# Understanding IPVLAN

- IPVLAN is a driver for a virtual network device that can be used in a container environment to access the host network
- IPVLAN exposes a single MAC address to the external network, regardless the number of IPVLAN devices inside the host network
- So even with multiple IPVLAN devices in multiple containers, the local switch will see one single MAC address only.

# Understanding IPVLAN Modes

- L2 mode: virtual devices receive and respond to ARP requests and no netfilter chains are executed on the default namespace on the containerized traffic
- L3 mode: ARP-related neighbor entries must be configured manually, L3 mode virtual devices process only L3 traffic and above. Egress traffic of a container is processed by the netfilter postrouting and output chains
- L3S mode: like L3 mode, but also Ingress traffic is processed by a netfilter chain in the default namespace

# RHEL 8 Update

System Management

# System Management

- RHEL Web Console is provided as a graphical management utility for local and remote systems
- Use **systemctl enable --now cockpit.socket** to enable and start

# RHEL 8 Update

Security

# RHEL 8 Update

Security: Cryptographic Policies

# Cryptographic Policies

- RHEL 8  uses cryptographical policies
- The policy defines the minimal standard for cryptographic algorithms and policies that must be met. Applications that don't meet the standard will be released
- Four different policy levels are supported
  - DEFAULT
  - LEGACY
  - FUTURE
  - FIPS: use **fips-mode-setup** to switch RHEL into FIPS140-2 compliance
- Use **update-crypto-policies --show** to show current
- Use **update-crypto-policies --set POLICY** to change

# RHEL 8 Update

Security: Policy-based Decryption

# Understanding Policy Based Decryption

- Policy-Based Decryption (PBD) is a collection of technologies that enable unlocking encrypted root volumes in an automated way
- In PBD, different unlocking methods can be combined
  - user passwords
  - Trusted Platform Module (TPM) devices
  - PKCS#11 devices like smartcard readers
  - Network servers
- Different unlocking mechanisms can be combined in a policy, which makes it possible to unlock the same volume in different ways

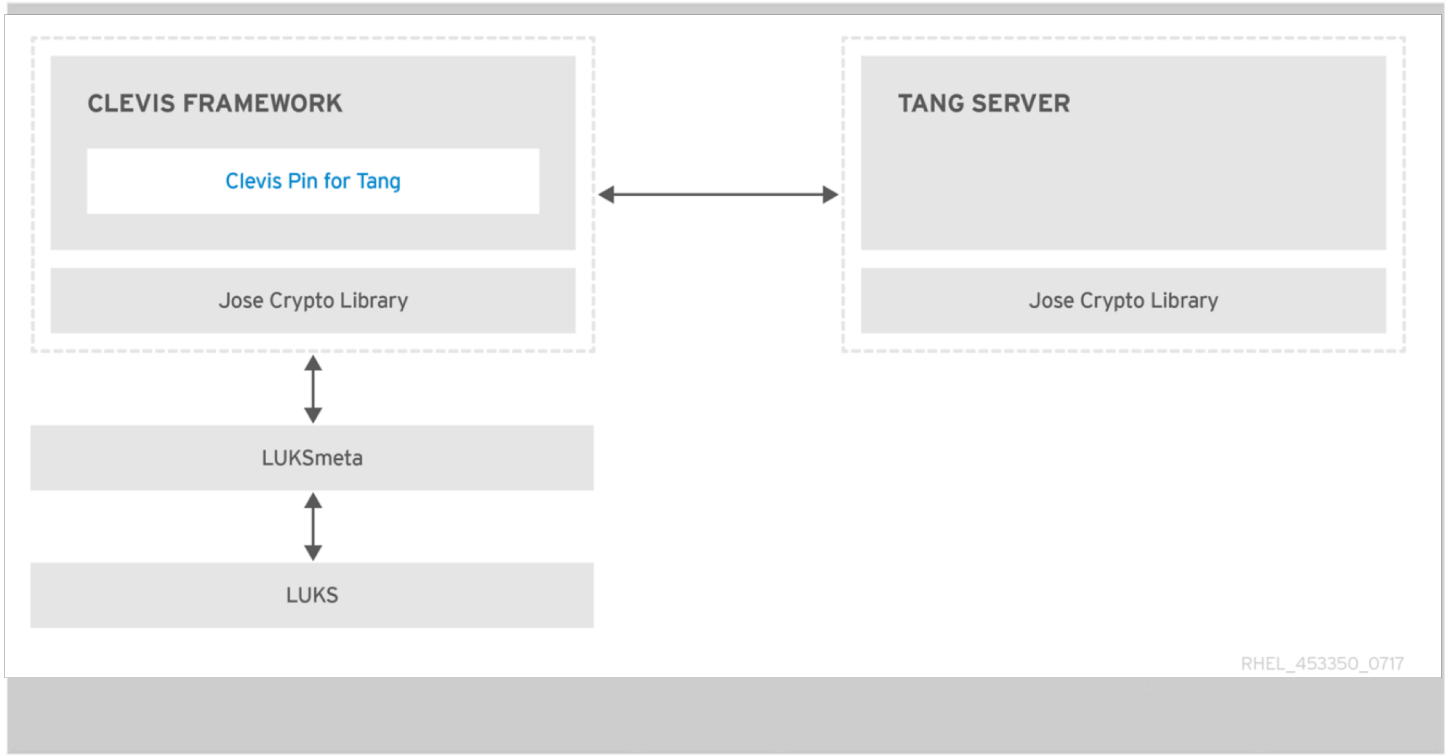
# Understanding RHEL 8 PBD

- RHEL 8 implements PBD sugin the Clevis framework and pins (plug-ins)
- Each pin provides a separate unlocking capability
  - tang: allows volumes to be unlocked using a network server
  - tpm2: allows volumes to be  unlocked using a TPM2 policy

# Understanding NBDE and PBD

- NBDE is Network Bound Disc Encryption, a subcategory of PBD
- It allows binding encrypted volumes to a network server
- The current implementation includes a Clevis pin for Tang server and the Tang server itself

# NBDE overview



# Understanding Tang

- Tang is a network server that is used for decrypting LUKS devices
- Clevis is the framework that allows for decrypting LUKS volumes and connects to the Tang server to do so
- When provisioning NBDE, the Clevis pin for Tang gets a list of Tang server's advertised asymmetric keys
- Alternatively, a list of Tang public keys can be distributed out-of-band so that clients don't require direct access to the Tang server

# Getting Started

- Install the clevis package
- **man clevis** for more details
- Install the **tang** server
- Enable **tangd.socket**
- Use **jose** to manually generate keys

# Rough Procedure  Overview: tang

- **yum install tang**
- **systemctl enable tangd.socket --now**

# Rough procedure overview: Clevis

- **yum install clevis-luks**
- **clevis luks bind –d /dev/sdb tang '{"url":"http://tang.srv"}'**
- **cryptsetup luksDump /dev/sdb** will show the LUKS crypti info in the LUKS2 header token
- **yum install clevis-dracut**
- **dracut -f**

# RHEL 8 Update

Security: nftables

# Understanding nftables

- **nftables** is the successor of iptables
- In RHEL8, nftables is used as the back-end to firewalld
- Architecture is very similar to **iptables** syntax, syntax is not
- Many improvements:
  - Support for lookup tables, which means that rules no longer a scanned in a linear way
  - Updates to chains are atomic and don't require a reload of the entire table
  - Kernel can update applications about rule changes
  - Using the "inet" protocol family, rules can be applied for IPv4 and IPv6 simultaneously

# Getting Started

- **nft list tables**
- **nft list table firewalld** shows what's inside
- Best practice: add new rules to the tables using the **firewall-cmd** interface
- More info: https://developers.redhat.com/blog/2016/10/28/what-comes-after-iptables-its-successor-of-course-nftables

# RHEL 8 Update

Containers

# RHEL 8 Update

## Containers: Understanding RHEL8 Containers

# Understanding the Landscape

- RHEL 8 provides tools to work with containers to run directly on top of RHEL 8 in single-node use cases
- That means that NO docker or any other container engine is required! RHEL8 containers run daemon-less
- For running containers in enterprise environments, OpenShift is required

# Managing Containers in RHEL8

- **podman**: direct management of pods and container images
- **buildah**: for building, pushing and signing container images
- **skopeo**: for copying, inspecting and signing images
- **runc**: for providing container run and build features to podman and buildah
- These tools are compatible with the Open Container Initiative and therefore can be used to manage Linux containers on top of any OCI-compatible container engine
- Install using **yum module install --container-tools**
- Optional: **yum install podman-docker** providers docker like syntax in the podman  command

# Configuring Container Settings

- /etc/containers/registries.conf is used to tell RHEL where to go
- By default all is setup to start working!
- Add your own insecure registries in the [registries.insecure] section
- Note that registry.redhat.io requires authentication, registry.access.redhat.com doesn't need any authentication but is deprecated
  - Registries are processed in order, so use the most successful registry first in your list
  - Include the registry in the pull command to ensure where the image is fetched from: **podman pull registry.redhat.io/rhel8-beta/rhel**

# RHEL 8 Update

Containers: Working with Images

# Working with podman

- **podman search rhel8** will look up the RHEL8 image
- **podman pull rhel8-beta/rhel** will download it from the first regisgtry that answers
- **podman images** will show all images stored on the local system
- **podman inspect <imagename>** will inspect image contents: it will show you what exactly will happen when you run the container
- **podman run  <imagename>** will run it
- **podman ps** gives a list of currently running containers
- **podman mount <id>** allows you to mount a container on a specific location so that you can further inspect it
- **podman rmi** is used to remove images

# Inspecting Remote Containers

- **podman** is for local images, use **skopeo** to inspect remote images
- **skopeo inspect docker://access.registry.redhat.io/rhel8-beta/rhel-init** to inspect the image before pulling it.

# RHEL 8 Update

## Containers: Working with Containers

# Running Containers

- **podman run** will run a container from an image. The container will execute its task and next stop
- If no task is defined as the default task, the container will immediately stop again
- Use different arguments with the **podman run** command to run tasks within the container

# Examples

- **podman run --rm ubuntu:latest cat /etc/os-release** will start the container, run the command and exit and delete the container afterwards
- **podman run –name=mycontainer –it ubuntu:latest /bin/bash** will start the container, run the bash shell and keep it open
  - **-i** is interactive
  - **-t** opens a terminal
- **podman run --name=mycontainer -v /dev/log:/dev/log --rm ubuntu:latest logger helloe** will start the container and map the local /dev/log device to the container /dev/log device. Next, logs a message an exists

# Inspecting containers

- **podman ps** shows all containers actually running
- **podman inspect <id>** shows properties on a container based on its id
  - Note that some properties only apply to containers that are actually running
  - Use **--format** to request values of specific properties: **podman inspect --format='{{.NetworkSettings.IPAddress}}' <id>**
- **podman exec -it <id> /bin/bash** allows you to run a bash shell within a container
  - Note that the command you want to run needs to exist within the container

# Starting Containers

- If a previous container wasn't removed with the --rm option, it's still on your system
- Use **podman start <name>** to start it
- Use **podman stop <id>** to stop it using SIGTERM
- Use **podman kill <id>** to stop it using SIGKILL

# RHEL 8 Update

Containers: Buildah

# Understanding Buildah

- Buildah is a tool that helps in building images
- It is NOT the preferred tool for actually running containers
- Doesn't need a container runtime daemon
- Build an image build on another container or from scratch
- Build tools are external to the image, which reduces the image size
- The official documentation is here: https://www.projectatomic.io/blog/2017/11/getting-started-with-buildah/

# Buildah main activities

- **buildah bud** will build a container from a Dockerfile
- **buildah from <imagename>** builds an image from another image
- **buildah from scratch** allows you to build from scratch
- **buildah inspect** shows container or image metadata
- **buildah mount** mounts the container root FS
- **buildah commit** uses updated contents of a container root FS as a FS layer to commit content to a new image
- **buildah rmi/rm** remove image or container
- **buildah unmount** unmounts container

# Getting Images with Buildah

- Use **buildah from rhel8-beta/rhel** to start a container based on the rhel8-beta image
- Use **buildah images** to see that no new image was created
- Use **buildah containers** to see that a new buildah-based container was started
- Use **podman ps** and see that the container is not currently running, you didn't tell it what to do!
- Use **buildah run rhel-working-container cat /etc/redhat-release** to run a command
  - Note that you should rather be running containers with **podman**

# Building an Image from Dockerfile

- Use **buildah bud -t myscript .** to build an image from a Dockerfile
  - This will look for Dockerfile in the current directory
  - And run "myscript" on the container that is specified in the Dockerfile
- The result is a new image, not a new container
  - Use **buildah images** for a list of current images
  - Use **buildah containers** to see that no container was started
- If desired, start the container yourself
  - Use **buildah from myscript** to create a working container from the image
  - Use **buildah run myscript-working-container** to run the container you've just created

# Modifying Containers with Buildah

- There are two ways to modify a container
  - Mount the container and copy files to it
  - Use **buildah copy** and **buildah config** to modify the container
- After making the modifications, use **buildah commit** to commit the changes to a new image

# Using **buildah mount**

- Mount a working container and commit changes
  - Mount, setting a variable: **mymount=$(buildah mount myscript-working-container)**
  - Edit content anywhere in the script
  - Use **buildah commit myscript-working-container containers-storage:myscript2**
- Verify the new image and run the container
  - **buildah images**
  - **buildah from docker.io/library/myscript2:latest**
  - **buildah run myecho2-working-container**
- Unmount the container: **buildah umount myscript2-working-container**

# Using **buildah copy**

- **buildah copy** allows you to copy an arbitrary file to anywhere in the container
  - **buildah copy myscript-working-container-3 newscript /usr/local/bin**
- If you copy a script, you should set the command to run
  - **buildah config myscript-working-container-3 --entrypoint "/bin/sh -c /usr/local/bin/newscript"**
- After which you can run the new script
  - **buildah run myscript-working-container-3**
- And when all looks good, commit to a new image:
  - **buildah commit myecho-working-container-3 containers-storage:mynewscript**

# Creating Images from scratch with Buildah

- A scratch container is a minimal container that only holds a small amount of container metadata
- Scratch containers are different
  - They are minimal, so can just contain some binaries without all of their dependencies
  - The scratch container must have an RPM database and a release package
  - Using a minimal container such as centos-minimal might be more convenient as it contains all that is required to run

# RHEL 8 Update

Containers: Composer

# Understanding Composer

- Composer is created to make it easy to build and deploy custom images in cloud environments
- It also allows for easy assembling of packages into deployable images for virtual, cloud-based and bare-metal systems