AURISTOR

YOUR DATA · YOUR FILE SYSTEM

THE AFS NAMESPACE AND CONTAINERS

A SECURE DISTRIBUTED FILESYSTEM APPROACH TO SECURELY PROVIDE PERSISTENCE TO LINUX CONTAINERS

LINUX AF_RXRPC AND KERNEL AFS BY DAVID HOWELLS
(DHOWELLS@REDHAT.COM)

DEMOS BY MARC DIONNE
(MARC.DIONNE@AURISTOR.COM)

PRESENTED BY JEFFREY ALTMAN
(JALTMAN@AURISTOR.COM)

CREDITS

The demonstration you are about to see is the product of David Howells, Marc Dionne, the rest of the AuriStor team, and all of those who led the way at IBM, Carnegie Mellon University, and Transarc.

David Howells began development of Linux Kernel AFS in October 2002 (linux-2.5.40). Today, AF_RXRPC is a high-performance RX implementation integrated with the Linux networking stack that can be used to implement custom services. Keyrings and FS-Cache are core components leveraged by many Linux kernel and userland systems. AFS provides access to the /afs file namespace whether hosted by AuriStorFS, OpenAFS or IBM AFS 3.6.

The AuriStorFS Linux kernel module is a monolithic combination of a robust high performance RX implementation, multiple RX security classes, multiple data caching models, and protocol support for AuriStorFS, OpenAFS and IBM AFS 3.6. AuriStorFS sustains high concurrency workloads on large scale multiprocessor systems even when mixing cached and direct I/O.

The demonstration was constructed by Marc Dionne.

SOFTWARE DEPLOYMENT USING /AFS

- The /afs file namespace has long been used for distribution of software and configuration
- MIT Athena Environment and Morgan Stanley Aurora were inspirations for hundreds of other homegrown models
- /afs has unique capabilities designed to foster cross-platform software deployments via a global file namespace
  - RO snapshots provide a stable public view while the RW instance provides a private canvas to develop the next release
  - RW to RO volume "release" operations are atomic from the perspective of the clients
  - RO volume replication permits distribution of updates to sites that potentially span the globe
    - Internal data centers
    - DMZs
    - Cloud services
  - A globally canonical file namespace, /afs, permits software to be distributed anywhere in the world to any device connected to the internet
  - The @sys path component substitution variable permits a path to be shared by multiple architectures while evaluating to architecture and operating system specific binaries or configuration

BENEFITS OF /AFS

- Deploy once - execute anywhere and anytime
- Hundreds of thousands or millions of clients can share the same deployment
- Minimizes the cost (time and effort) of deploying new systems or replacing existing systems
- The file namespace has been around for more than 30 years
- Transitioning to new operating systems and processor architectures is simplified
- Local caching provides the performance benefits of local installation
- New software revisions can be installed side-by-side older releases
- Linked AFS cells permit test releases to be overlaid on top of the live production environment for validation

THE /AFS NAMESPACE MODEL

- What the application sees:
  /afs/<cell>/software/<vendor>/<product>/<version>/bin/<executable>
- “bin”is a symlink to “.@sys/bin”
- @sys is replaced by a locally defined value (e.g., “amd64_linux26”)
- “.amd64_linux26” is a mount point to a volume containing the amd64 linux version of the software
  (e.g. s.<vendor_code>.<product_code>.<amd64_linux_code>.<version> or “s.ad.ac.x64lx.3_14_01”

THE LOCAL NAMESPACE MODEL

- What the application sees: /usr/local/<product>/bin/<executable>
- Where <product> is either:
  - A symlinks to /afs/<cell>/software/<vendor>/<product>/<version>
  - Or a local “mount” to the AFS volume that the AFS mount point <version> refers to
- In either case, “bin” is a symlink to “.@sys/bin”
- @sys is replaced by a locally defined value (e.g., “amd64_linux26”)
- “.amd64_linux26” is a mount point to a volume containing the amd64 linux version of the software (e.g. s.<vendor_code>.<product_code>.<amd64_linux_code>.<version> or “s.ad.ac.x64lx.3_14_01”

LINUX CONTAINERS
SOLVING STORAGE PROBLEMS WITH /AFS

- The future of software deployment but has challenges when it comes to data access
  - No common file system namespace hampers portability
  - No shared file system across containers
    - No work around when executing in multiple hosts or data centers
  - Network file system access requires exposing secrets to the container

- /afs is a perfect file system namespace for use from within containers
  - Requires the availability of /afs in the Linux distributions
  - Requires the ability to start containers with a predefined network identity and secret
    - Will work with AuriStorFS Extended Authz model to restrict data access to container instance executing on a trusted host

LINUX CONTAINERS
IMAGE CONSTRUCTION: THEORY VS PRACTICE

- Theory:
  - One image per executable or product
- Practice:
  - One image that contains all executables
    - Containers are deployed as lighter weight VM images

- When a single binary must be updated “patch anyone?” a new Container image must be built and deployed

- What if Container images only referenced AFS volume mounts?
  - Then software versions could be updated without rebuilding and publishing the container images.

KUBERNETES INTEGRATION
USING AFS VOLUMES FOR SCRATCH OR PERSISTENCE

- /afs cells can be deployed both outside and inside the cluster
- AFS volumes mounted in a /afs namespace can be used for shared persistent storage
  - Volumes can be replicated to local nodes
- AFS volumes can be created on demand for scratch or persistent storage
  - Volumes can be migrated or replicated between nodes within a cluster
- Long term goal – overflow compute model in Kubernetes clusters that span internal data centers and external cloud infrastructure (AWS, Azure, GCP, …)

KUBERNETES INTEGRATION
HOSTING AN AURISTOR FILE SYSTEM CELL AS PODS

- AuriStorFS servers deployment
  - Each AuriStorFS service (location, protection, file, vol, key management) is deployed as a pod
  - Dedicated storage nodes can manage storage for the cluster, or
  - Each node within the cluster can host a fileserver pod

DEMO 1
AURISTORFS AND LINUX KERNEL AFS

- https://asciinema.org/a/181731

- This shows a single machine where the kafs and AuriStorFS clients are used in parallel.
- Note the interesting case of mounting a particular cell's root volume with kafs over the path where it is available from AuriStorFS. All access to that particular cell would be handled by kafs, while access to all other cells under /afs would be handled by AuriStorFS.
- It also shows kafs' use of separate mounts for each volume.

DEMO 2
AURISTORFS AND LINUX KERNEL AFS - DOCKER

- https://asciinema.org/a/181871

- This demonstrates making volumes available to a container with direct docker commands.
- Note that in the case of kafs, crossing mountpoints in the container requires that the target volume already be mounted on host. At the moment new kafs mounts can only be created in the root network namespace and can't be created in containers with other network namespaces. This restriction will be eliminated in a future revision.

DEMO 3
AURISTORFS – KUBERNETES VOLUME PLUGIN

- https://asciinema.org/a/181733

- This demonstrates mounting an AuriStorFS volume into a pod through the use of a FlexVolume plugin.
- It requires that the cluster nodes have the AuriStorFS client installed and running, and the AuriStorFS FlexVolume plugin installed. The volume is made available through a bind mount of `/afs/.:mount/:` to the target location
- The volume does not need to have mountpoints anywhere in the distributed namespace.

DEMO 4
LINUX KERNEL AFS – KUBERNETES VOLUME PLUGIN

- https://asciinema.org/a/181732

- This demonstrates making a volume available in a kubernetes pod through the use of a FlexVolume plugin and the kernel's kafs.
- It requires that the cluster nodes have a kernel built with kafs support, and have the kafs FlexVolume installed. The volume does not need to be mounted when the pod is created, and does not need to have a mountpoint anywhere in the distributed namespace. The requested volume will be mounted with a single volume mount, and unmounted when the pod is deleted.

DEMO 5
AURISTORFS – KUBERNETES SCRATCH VOLUME PLUGIN

- https://asciinema.org/a/181729

- This demonstrates a simple "scratch" volume kubernetes plugin.
- It is implemented here as a separate plugin, but it could also have been combined with the general plugin with the use of an additional property. The plugin is deployed to all the cluster nodes, along with stashed credentials (in the form of a keytab file) for a principal that is authorized to perform the necessary operations on a target server. The volume property specifies a 'mktemp' template used to build the final volume name. The volume is created and mounted into the pod. When the pod is deleted, the volume is unmounted and deleted.

AURISTOR

YOUR DATA · YOUR FILE SYSTEM

THE AFS NAMESPACE AND CONTAINERS

STATUS REPORTS: AURISTOR FILE SYSTEM, KAFS, AND OPENAFS USE CASES