Testing with Docker

Experiences from the development of Auristor

Marc Dionne
Your File System Inc.

Context

- Extensive unit testing in place
  - Command suite
  - RPCs
  - Libraries
- .. but limited testing of larger setups
  - Ubik database behaviour
  - Volume operations

Context

- Distributed systems are difficult to test
  - Multiple servers
  - Complex configuration requirements
- Hard to answer some theoretical questions
  - Scaling
    - how many fileservers can really be in a cell
    - How many database servers could be used if some code limits were removed
  - Verifying that database servers reach (or regain) quorum
- Typical solution: VMs
  - Requires a lot of resources, hard to scale to larger numbers of servers
  - Requires time to build images, boot servers
  - Complexity of configuring everything correctly

Goals

- Speed
  - Quick cycle for repeatedly testing code changes
- Scaling
  - Want to be able to test/reach some limits, ex: run the maximum number of fileservers
- Flexibility
  - Mix of database and fileservers
  - Various cell sizes
- Simplicity
  - Fit in TAP framework
  - Fully scripted/automated

Docker

- Docker looked promising as a base tool
  - Isolates the host from the testing
  - Eases cleanup – files and leftover processes
  - Lightweight compared to a full VM
- Recent version available in Fedora
- RHEL not too far behind

Building blocks

- TAP framework
- Single server cell utility, used in many tests
  - Builds a centralized temporary config, including fileserver data
  - Start a bos server, creates instances

Docker - Network

- Bridge interface created by docker daemon
  - Can also specify your own
- When created, containers get an IP address assigned within that subnet
  - Released when container is deleted
  - No DHCP, can't predict address
- Host gets a fixed IP (ex: 172.17.42.1) on the same subnet
- Host and containers can talk to each other without further config
  - Communication between containers and outside world beyond host requires config

Docker - Files

- By default only a few host files are visible inside containers
  - Container only sees files in the image
- Modifications are stored in docker's internal storage
  - /var/lib/docker on fedora
  - Stores only differences from base image
  - Removed when container is deleted
- Can use volumes (-v|--volume) to map host files into the container's namespace, ro or rw
- Volumes can also be named and used to share data between containers
  - Removed when the last container referencing it is deleted

Docker - Users

- Docker doesn't currently namespace user IDs
  - Work in progress
- Containers run as root, even if started by a regular user
  - The process can switch to another user ID
  - A default user can be specified when building the image
- File access is done as that user
  - If exposing host files, may need to cleanup inside container or as root afterwards
  - If running as a different user, may need to adjust permissions of exposed host files

Docker - Image

- Containers are created from a base image
- A custom image can be built from a published image
  - Dockerfile describes what to add/modify
    - Base image – will be fetched if not available locally
    - Commands to run, typically yum/dnf install
    - User ID
    - Local files to add to the image
    - Volume definitions
  - “docker build” creates and labels the image
    - Fairly fast once base image is available
    - Only needs to be rebuilt if there's a need for updated components – for testing, rarely

Strategy

- Use centos as base image
  - Same image can be used on different platforms
  - But may need per-platform base image at some point
- Build the build tree on host
  - Use a volume to map it into the containers
    - Same path in all containers
  - Use executables directly from the build tree
    - Building rpms is very slow
    - “make install” also takes a little while
    - Shortens the code/compile/test cycle
- Basic structure
  - Single host script to setup config, start/coordinate containers and stop/delete them
  - Single “slave” script that starts everything needed inside the container
  - Test scripts call host script with parameters – number db and fileservers

Strategy

- Build on top of existing cell testing infrastructure
  - Already have a command to setup config and start a single server cell
  - Add option to start just db or fs server processes, or both
- Files
  - All config files and server data consolidated under a single directory
  - Separate areas on host for each container
    - In particular, logs and data need to be separate, and some runtime data (sysid, etc.)
    - Mapped at the same location in all containers
    - Mapping from the host is mainly to make debugging easier
  - Ex: host sees /var/tmp/slavefs0.. /var/tmp/slavedb0.., slaves use /var/tmp/yfs_docker_tests

Network

- Let docker assign IPs as each container starts
- Host script uses “docker inspect” to discover container IP addresses
  - Builds config file for server and client processes
  - Places resulting file in a build tree location accessible in the containers
  - Containers poll for that file as a signal to start
    - File moved into final location
- Check that cell has started before returning
  - Use “vos listfs” to verify that all fileservers have registered in the vldb
  - Use “vos listvol” to verify that all volservers have started and are connected to their fileservers

Tests

- Use docker pause/unpause to simulate a server crashing or going away and coming back
- Various in-tree test scenarios are built on top of basic host script
  - Test of volume operations, addsite, release, move
  - Ubik tests
    - Reaching quorum in the expected time frame
    - Regaining quorum after losing the sync site
    - Avoiding DB corruption
- Host script can be used directly for ad-hoc testing

Observations

- Can start a large number of servers in a short time frame
  - Ex: ~1m for 50 servers
  - Very quick for single DB server cells (no quorum delay), about 30s extra for multiple DB servers
- Needed many tweaks to the cell startup sequence, for consistent (shorter) timings
  - All servers start simultaneously
    - Uncommon situation for a real cell
  - Many dependencies
    - DB servers need to reach quorum
    - FS servers need to register with the DB servers
    - fileserver and volserver on each container establish communication with each other
  - Start with pre-exisiting empty databases
  - Single server cell startup is now ~3s

Summary

- docker has its limitations, but for testing it is an excellent fit
- The framework has allowed us to easily exercise scenarios that would be difficult/impossible to test otherwise

Future - WIP

- Incorporate client module testing
  - Run lightweight VMs with qemu/kvm
  - Build VM image
  - Share files with host
  - Load client kernel module in VM
- Run tests that combine a set of servers and a set of clients
- Add more complex scenarios
- Stress tests
- Benchmarks

Questions

Quick demo? (thechnology permitting)