CONNECTING THE IT COMMUNITY

INTEROP®

Container Challenges: Know Before You Deploy

Scott Lowe

Blog: http://blog.scottlowe.org • Twitter: @scott_lowe

Github: https://github.com/lowescott • Life: Colossians 3:17

Before we begin

- **DO:** Ask questions! Audience participation is requested and encouraged.
- **DO:** Take pictures and post updates to the social media service of your choice.
- **DON'T:** Forget to silence your electronic devices.

About me

- An IT professional with 20+ years of experience in the industry
- An author or co-author on 7 books
- A lifelong geek with a passion for technology
- A blogger (visit my site at http://blog.scottlowe.org)
- A first-time Interop speaker and attendee! (Can you believe it?)
- A VMware employee (but not speaking for VMware)

Agenda

- Setting the stage
- Reviewing non-technical challenges
- Reviewing technical challenges
- Q&A

Setting the stage

Quick overview of Docker

- Leverages Linux kernel features (cgroups, namespaces)
- Depends on copy-on-write mechanisms for fast deployment and reduced resource usage
- Provides an developer-friendly CLI
- Uses a simple, easy-to-understand language
- Enables easy sharing of images to enable and encourage collaboration

Some common misconceptions

- “Run any app anywhere”
- “They’re like lightweight VMs”
- “Docker will replace <insert configuration management tool here>”
- “Docker will eliminate need for <insert hypervisor name here>”

Reviewing non-technical challenges

Managing project scope

- You may end up building (and maintaining) your own tools to make Docker work in your environment
  - Spotify had to build Helios
  - New Relic had to build Centurion and Shipright
  - Shopify had to build ejson
- Make sure you don't try to "boil the ocean," focus on an achievable goal

Staff readiness

- Deploying Docker means new skills are going to be needed
- Most of these are non-Docker skills:
  - Orchestration tools
  - Consensus mechanisms
  - Service registration and discovery
  - Application architectures

Organizational and/or operational changes

- Who will own Docker images?
- Who will own the Dockerfiles?
- How do you provide traceability from running container all the way back to Git commits in the source code?
- If you still have issues between Dev and Ops, Docker isn't going to magically fix them

Reviewing technical challenges

Persistent storage

- Container filesystem disappears when the container exits
- Docker volumes can help, but are not a first-class citizen in the ecosystem (refer to https://groups.google.com/forum/#!msg/docker-dev/5R9ag4UwUb8/WHnYy_L4Fy8J)
- Mounting paths on the host works, but containers are now tied to the host
- Using host paths shifts the problem, but doesn't fix it

Logging

- Docker (via `docker logs`) only captures STDOUT and STDERR
- Writing to log files could be an issue (refer to persistent storage concerns)
- Running a logging agent inside the container forces you to go the multi-process container route
- Alternately, you can re-architect your application to use STDOUT and STDERR

Single-process vs. multi-process

- Single-process containers are viewed as “the Docker way” of doing things
- You’ll need to go the multi-process route if you want logging agents, monitoring agents, etc. in the container
- Multi-process involves some sort of supervisor process (supervisord, runit, or others) managing the “real” application process(es) in the container

Security

- Containers (especially single-process containers) ***do*** reduce the attack surface
- Containers don't provide comprehensive isolation (SELinux and AppArmor can help)
- Consider using VMs as a security boundary
- None of the security challenges are insurmountable, but you do want to be sure to consider them

Deep application knowledge necessary

- How does the application handle logging? Can it be changed?
- How does the application handle signals?
- Does the application work as expected when PPID=1?
- What are the upstream and downstream dependencies for this application?

Container networking

- No good solution for networking across containers right now
  - Docker is working on pluggable network subsystem
  - OVN (Open Virtual Network) might be a good fit, but still in development
- A form of SDN or network virtualization is pretty much a given
- Some networking decisions/architectures will be driven by the orchestration tool

Orchestration

- You'll need an orchestration tool or you'll have to do static orchestration
- Orchestration tools are still relatively young (Mesos is at version 0.22, Swarm is at version 0.2.0, Kubernetes is at version 0.14.2)
- Orchestration tools require additional complexity (etcd, Consul, service registration and discovery)
- If static orchestration, how will you automate it?

Shared services

- Will you run shared services on the host (logging, monitoring, etc.), or in the containers?
- Running them in the containers increases image size, memory footprint, CPU usage, forces multi-process approach
- If running them outside the container, how will you manage them? (you're still going to need something to manage container hosts)

Questions & Answers

CONNECTING THE IT COMMUNITY

INTEROP®

Thank you! We're outta here!

Scott Lowe

Blog: http://blog.scottlowe.org • Twitter: @scott_lowe

Github: https://github.com/lowescott • Life: Colossians 3:17