A CIDA Docker Proof of Concept

Jordan Walker

2015-01-06

1 Abstract

There are many options out there in the worlds of containerization and configuration management of services. As a group, CIDA [1] has several types of projects that can benefit from the different configuration management solutions that are out there. In particular applications that require particular package dependencies in order to operate correctly can make use of a configuration management solution to ensure that those dependencies are available to the application.

Docker [8] is one such option available, and has been growing in popularity of late. I propose a proof of concept Docker setup on CIDA infrastructure in order to determine whether it should be considered for general applicability across many applications as a containerization and configuration management mechanism.

The application chosen for this stage of our use of docker will be a simple service that is non-stateful and particularly fits the Docker use-case well. The result of this proof of concept will be a better understanding of the security implications of using Docker in our infrastructure and some guidance on what applications and services fit the Docker model of containerization and configuration management.

2 Rationale

While other configuration management approaches are being used and pursued within our group [6], [5], [7] Docker provides a unique approach to the problem that benefits certain applications in ways that we do not have answers to in any current system. A proof of concept will go a long way to show the benefits and challenges of Docker or similar systems.

The following features of Docker are particularly of interest at this point:

- Running multiple sandbox-ed containers within a single Virtual Machine
- Having an artifact that represents all dependencies of a service
- Inheritance of configuration so that many containers can be based on a single base setup
- Ability to have developer determined commands run for setup that would otherwise be performed manually by admin staff

Some of these features are available in other approaches that have been suggested, but Docker has simple mechanisms for all these points. If there is was a best-practice for this type of task at CIDA it would not be needed to come up with another solution, but to date nothing is clear in its use for this particular need.

3 Service Containerization

The goal in making containers for services is to have everything the service needs to run be pre-baked into a container. Then the container is deployed and able to run the service with minimal adjustments. Such a container can be achieved through virtualization, but recent changes have made Linux Containers [2] an appealing alternative. Docker fits into the latter group and offers lower overhead when compared to virtual machines.

When using containers to avoid dependencies conflicts in system dependencies, system level containers are required. Alternate approaches to sandbox-ed environments are worth discussing for the benefit they bring, but none solve the system dependency problem. The Java classloader allows for different services to use different versions of libraries, python virtualenv [4] installs isolated python environments to achieve the same end, and packrat [3] is an R package that offers similar functionality.

4 Configuration Management

Configuration of a service involves configuration files as well as installation of all the dependencies. This can be accomplished through a combination of virtualization and automation scripts or by putting all the configuration in the artifact itself. The latter approach is that taken by Docker and fits well into a continuous integration setup where the result of a build is an artifact that can work its way to production.

A point should be made that Docker isn't configuration management in the sense of other configuration management tools, but rather by specifying a build file for what should be included in the container. For some projects this will not be sufficient, but for our proof of concept use-case with a stateless service this will do just fine.

Lastly, when applying configuration to servers, often times root access is necessary. The distinction of what is run with root privileges should clearly be made and reviewed in conjunction with the security team to avoid possible breaches. Any solution we come up with will likely have this feature, and Docker in particular provides a good mechanism for managing this.

5 Requirements

Having made the point that this is a worthwhile exercise, here are the requirements from the author for such a solution.

- A single machine with Docker host installed
- A registry (remote or local to host) that images can be pushed to or pulled from
- Ability for devs (or eventually Jenkins) to run docker commands
  - Some commands may be restricted for security reasons, in particular
  - docker run should not allow volumes to be created as there is no easy mechanism for preventing root volumes from being mounted
  - docker run must allow at least flags -d -P for devs to get use from this
- A process for approving Dockerfiles for building
- A means for approved images to be uploaded to registry
- Ports in the range 49153 and 65535 should be opened for other services to be able to communicate with the docker container services
- The following guidelines will be followed by developers of Dockerfiles
  - The de-escalation from root shall occur as soon as possible
  - A non-root user shall run all services inside the container
  - Best practices [9] shall be followed
    - Exception: recommendations to use official repositories should be ignored because we are going to have our own registry of trusted images

6 Conclusion

A proof of concept Docker setup will give valuable insight into how this tool might fit into CIDA's infrastructure. Benefits such as lower overhead and improved consistency of services promise to improve our ability to deliver quality software. Lessons learned from this initial exploration will inform us as we look to future work in developing microservices.

Generally it is wise to avoid the “next big thing” in technology as it often leads to fragmented stacks and products that have not withstood the test of time. Docker is being touted as the “next big thing”, so we should tread softly, but the promise it brings for many areas of our development make it too hard to avoid it much longer.

References

[1] Center for integrated data analytics. URL: `http://cida.usgs.gov`.

[2] Linux containers. URL: `https://linuxcontainers.org/`.

[3] packrat. URL: `https://github.com/rstudio/packrat`.

[4] Virtualenv. URL: `https://virtualenv.pypa.io/en/latest/`.

[5] Inc. Ansible. Simple it automation. URL: `http://www.ansible.com/home`.

[6] CFEngine AS. Cfengine. URL: `http://cfengine.com/`.

[7] Inc Chef Software. It automation for speed and awesomeness. URL: `https://www.chef.io/chef/`.

[8] Inc. Docker. Docker introduction. URL: `http://www.docker.com/whatisdocker/`.

[9] Inc. Docker. Dockerfile best practices. URL: `https://docs.docker.com/articles/dockerfile_best-practices/`.