The Rapid Rise of Containers in Production

DANIEL NURMI | 2017

Contents

1.

Container History (well…OS virtualization)

Brief History of Containers

1979

Chroot!

- Ability to execute UNIX processes with a view of alternate root filesystem

2000-2005

Jails/Zones

- FreeBSD/Solaris/Linux
- Adding process/IPC/networking isolation in addition to root FS
- Security focused use-cases popular

2006-2012

Control Groups, LXC, Namespaces and Clustering

- Adding resource access controls and more isolation options
- Systems built to manage groups of environments across physical systems

Brief History of Containers

2013-2014

Docker, Rocket

- De-facto standardization around container creation/execution/format
- Centralized hubs of shared container ‘images’

2014-2017

Kubernetes/Mesos/Swarm

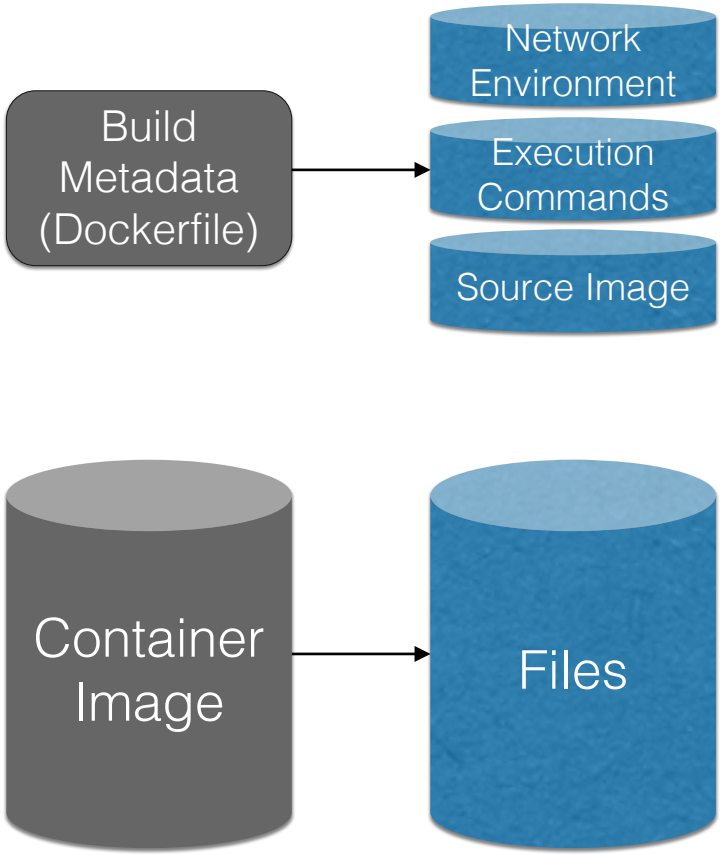
- Large scale container cluster orchestration and management
- Rapid adoption in data-center/large scale system deployments

What is a Container Today?

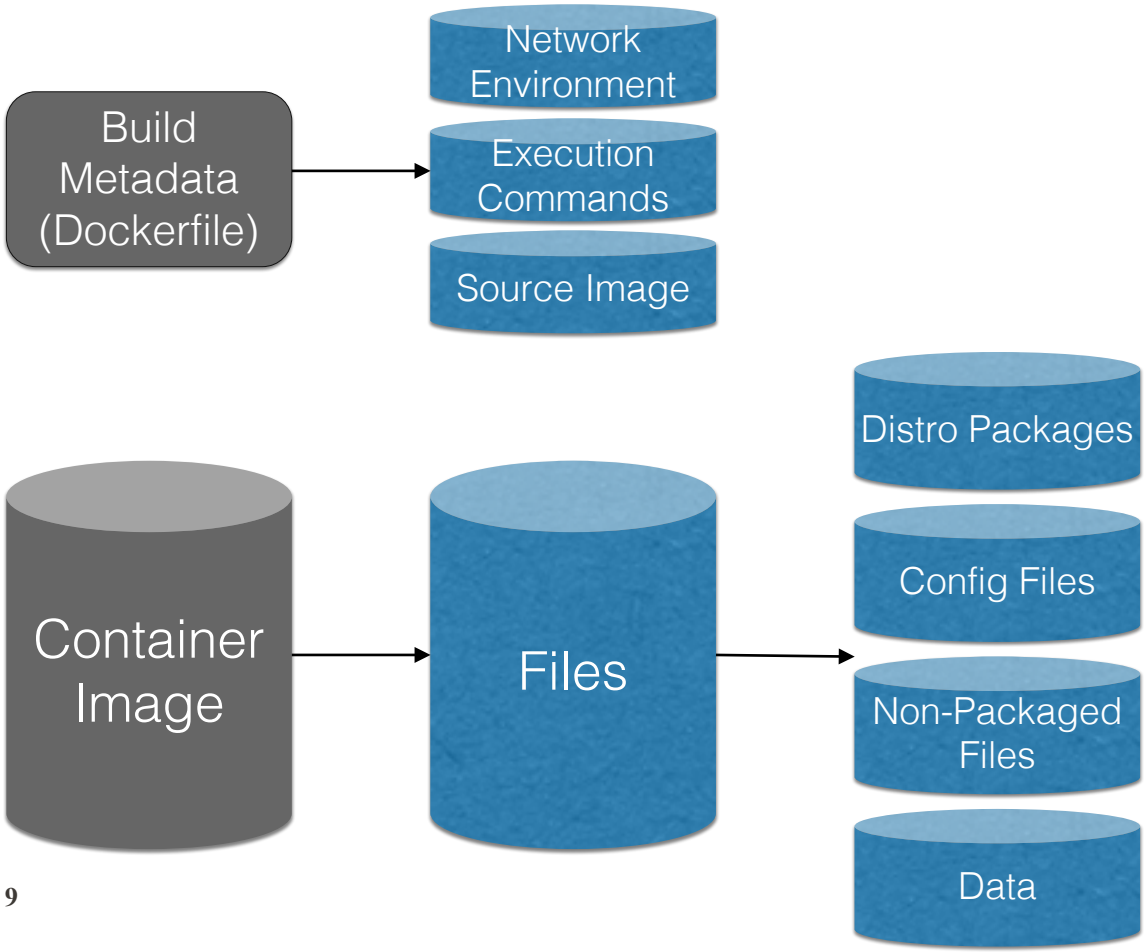
- Minimal root filesystem (plus your application)
- Small set of metadata describing environment
- Executed by an ‘engine’ that is capable of interacting with OS namespace and resource isolation subsystems to create isolated runtime environment (ex: Docker)
- Sometimes described “like a very light-weight VM”
- NOTE: different clustering systems define units composed of multiple containers that together form a service application

Anatomy of a Linux Container

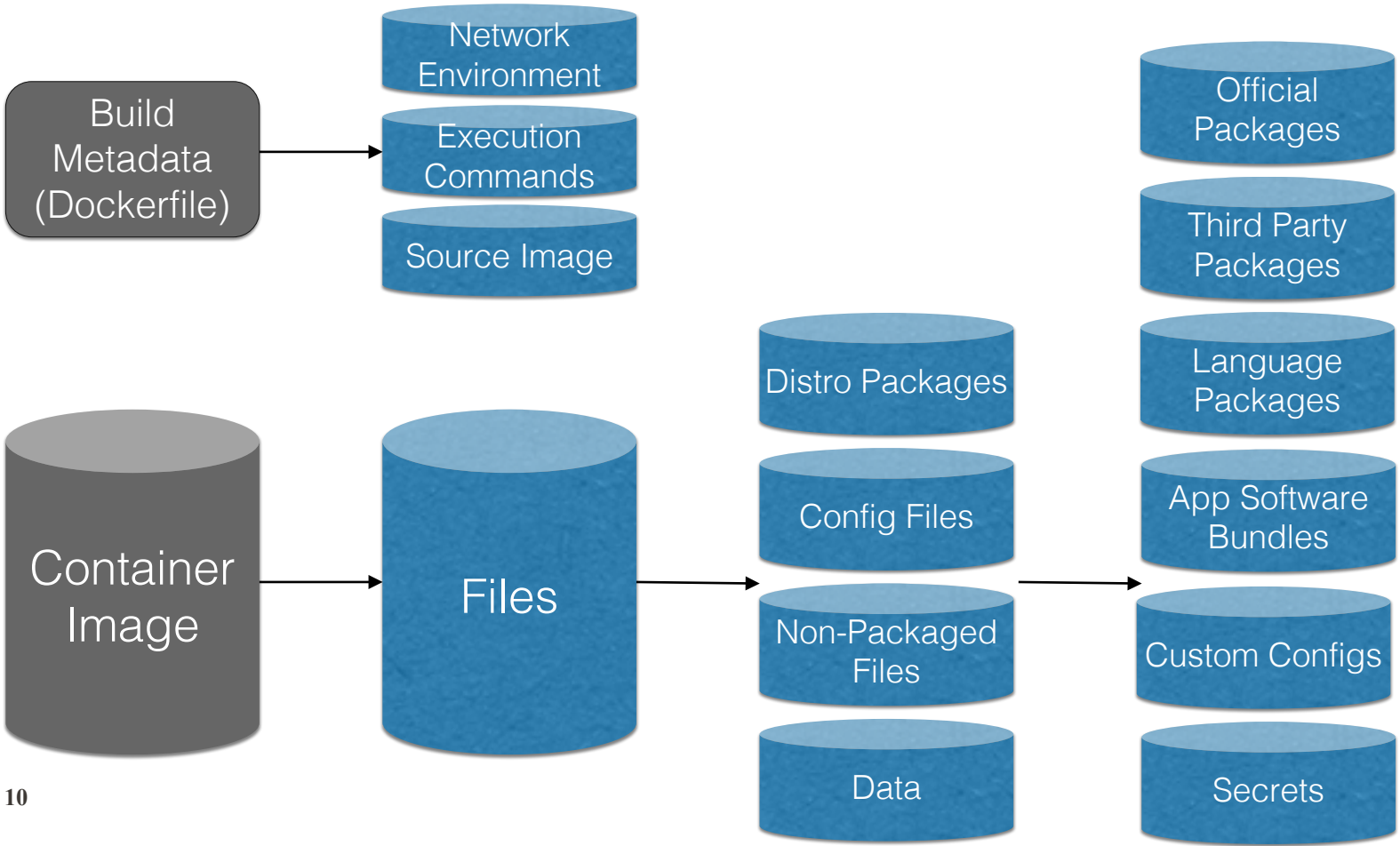
Anatomy of a Linux Container



Anatomy of a Linux Container



Anatomy of a Linux Container



“like a light-weight VM”

- …but not exactly.
- Not OS agnostic – shares kernel with host
- Not full system virtualization (just OS) – no hardware resource abstractions
- Very portable (between like hosts WRT OS version)
- Very fast iterations to build/deploy
- Built by applying changes to previous container images

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sched.h>
#include <sys/mount.h>

char cstack[1024*1000];

static int child_init() {
  printf("HELLO FROM A \"CONTAINER\" %d\n", getpid());
  printf("---\n");
  chroot("/tmp/rootfs");
  system("mount -t proc proc /proc");
  system("ps -aux");
  printf("---\n");
  system("ip link");
  printf("---\n");
  return(0);
}

int main(int argc, char **argv) {
  pid_t cpid;

  cpid = clone(child_init, cstack + (1024*1000), CLONE_NEWPID |
CLONE_NEWNET | CLONE_NEWNS | SIGCHLD, NULL);
  waitpid(cpid, NULL, 0);

  exit(0);
}
```



```
[root@tele ~]# tar zxf rootfs.tgz
[root@tele ~]# gcc container.c && ./a.out
HELLO FROM A "CONTAINER" 1
---
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START
TIME COMMAND
root         1  0.0  0.0   5164    88 ?        S+   17:37
0:00 ./a.out
root         3  0.0  0.0  49020  1820 ?        R+   17:37
0:00 ps -aux
---
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
---
[root@tele ~]#
```

The Container as a “unit”

- ***Extremely flexible*** abstraction – many are trying to determine which ‘unit’ containers are best suited
- **“Application”:** purely as a way to bundle an application along with a set of imbedded dependencies
- **“Service”:** ready-to-run service components that can be composed/deployed in support of applications
- **“Package”:** bundled software ready to be build upon/extended
- **“Process”:** everything is a container, including single binaries per container

Adoption

Developers

- Can develop application/application set in isolation on a single laptop
- Isolation allows flexible library/configuration choices to be made
- Easy to share/leverage other developers' containers

Fast and Easy

- Engines run on plain Linux/Windows machines, whether bare-metal of virtualized
- If you can deploy Linux/Windows somewhere, you can deploy containers right away.

Healthy Ecosystem of Cluster Orchestration systems

- Open-source (and free) orchestration systems, well engineered
- Proof is in the pudding – some very large-scale systems using tech that is freely available
- VMs and Clouds have paved the way, now we're seeing broader adoption of the idea of a 'Datacenter OS'

Healthy Ecosystem of Supporting Systems

- Monitoring, storage, CI/CD, security…

Industry Participants

Adoption Leads to Engagement

- Adoption has exceeded the industry's ability to move to container-based systems immediately
- Every corner of the Data-center is being inspected for production container suitability
- Many assert that next generation data-centers will be accessed via primarily container-based infrastructure

2.

Deployments and Benefits

Typical Container Workflow

PHASE 1

Developer Develops

- Creates an application container either from scratch or ‘FROM’ an existing container image
- Rapidly iterates on local system, pulling in libraries/configurations/etc. as needed
- Can grab existing containers that run pre-configured services (DBs, etc.) easily

PHASE 2

App Container is Committed and Tested

- Enters a build/test pipeline
- Testing (functional, unit)
- Security Analysis(?)

PHASE 3

Deployed to Production

- Orchestration system rolls out/deploys new container
- System-dependent, lots of variety
- Ultimately executed by a container engine very similar to what was running on the dev’s local system

Deployment Options

DIY

- Native support in Linux and Windows kernels for clone()ing a process into various isolated namespaces
- Major distros all support some form of container tools

Mesos/Kubernetes

- Mesos: cluster scheduling/resource management, containers and more
- Kubernetes: container focused, application design assertions/constraints

Docker

- Dockerhub container repository with lots of content
- Build images, share images, deploy into one-medium number of servers quickly

Clouds

- AWS ECS: containers in AWS instances
- Google Container Engine: using Kubernetes
- Microsoft Azure: Mesos, Kubernetes and Docker cluster hosting

Benefits

Speed

- Very light-weight equals fast development iterations
- Also fast to move unit through a dev->prod pipeline

Isolation

- Multiple environments running side-by-side on small number of phys. servers

Portability

- High degree of assurance that ‘if it works on my machine, it will work in production’

Encapsulation

- Proven to be useful mechanism to bundle useful services and make available for others to use easily

Tensions

Speed

- So easy and fast – single developer can generate many containers in a short amount of time
- Realize speed through a pipeline requires solid automation and testing

Isolation

- Isolated can mean opaque – limited insight into the containers themselves
- Secure?

Portability

- Many competing systems available that can all execute same set of containers – how and when to make a big decision on technology

Encapsulation

- Easy to bring in containers developed elsewhere
- Control over software shifted from OPS/SEC to DEV

How are tensions being handled?

- Some have replaced VM/Clouds with containers: at the expense of container benefits
- Some have relied on best practices: we trust you developers, don't do the wrong thing
- Some have built custom tools and infrastructure: it's mostly there, but some parts are missing
- Some are deploying at small scale and dev/test: waiting for maturity before moving to production
- Anchore! Analyze, inspect and control containers based on user-defined certification/validation policies.

Anchore Approach: get the data, expose the data, use the data

Tension: trusted, certified base containers

- Engine downloads, analyzes, makes available to user set of curated container base images

Tension: speed

- Assist in building automated pipelines by adding control points for policy application
- Anchore CI/CD integration with tools like Jenkins

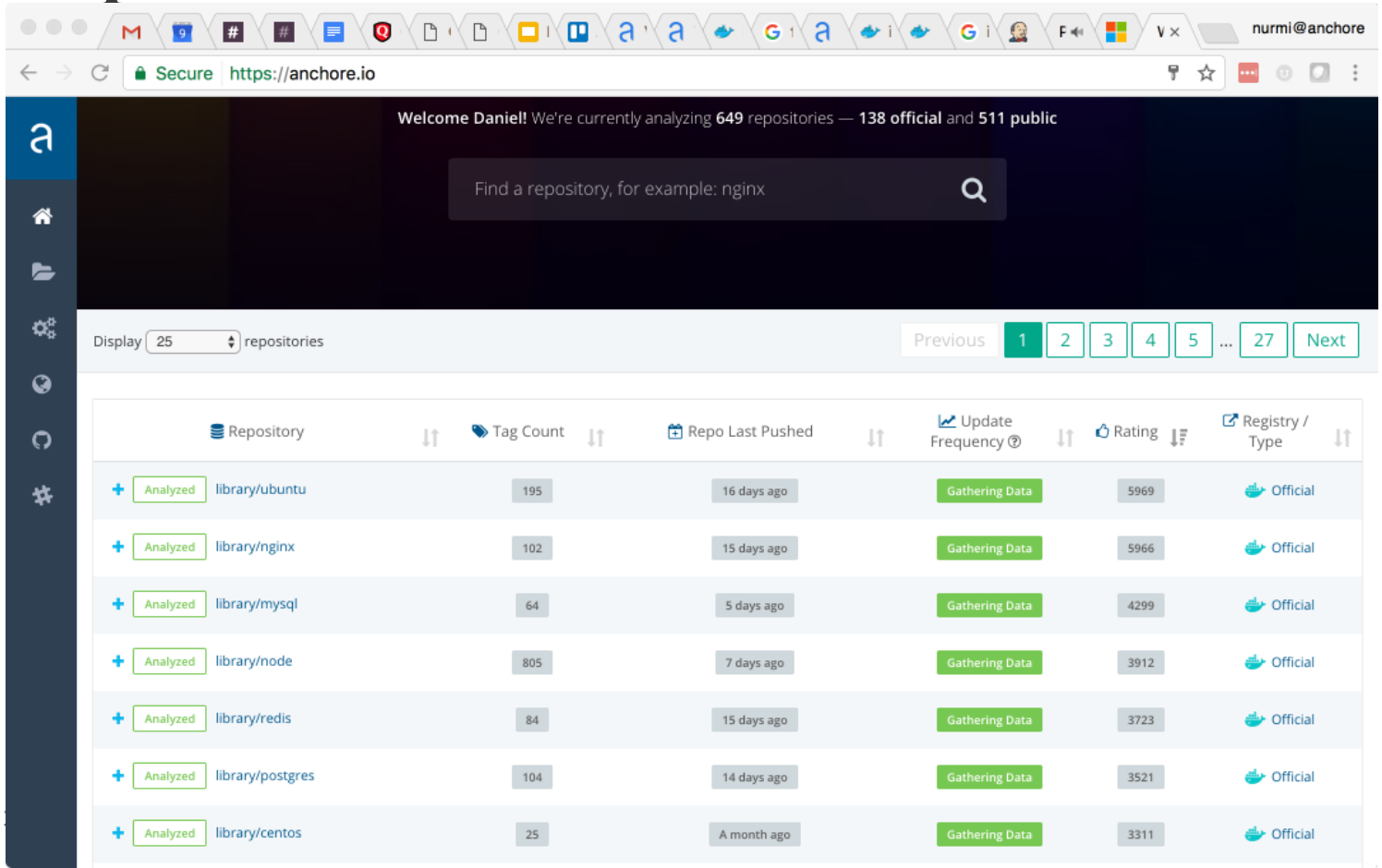
Tension: opaque containers

- Tools for inspecting, reporting, navigating container images
- All the way down to file contents

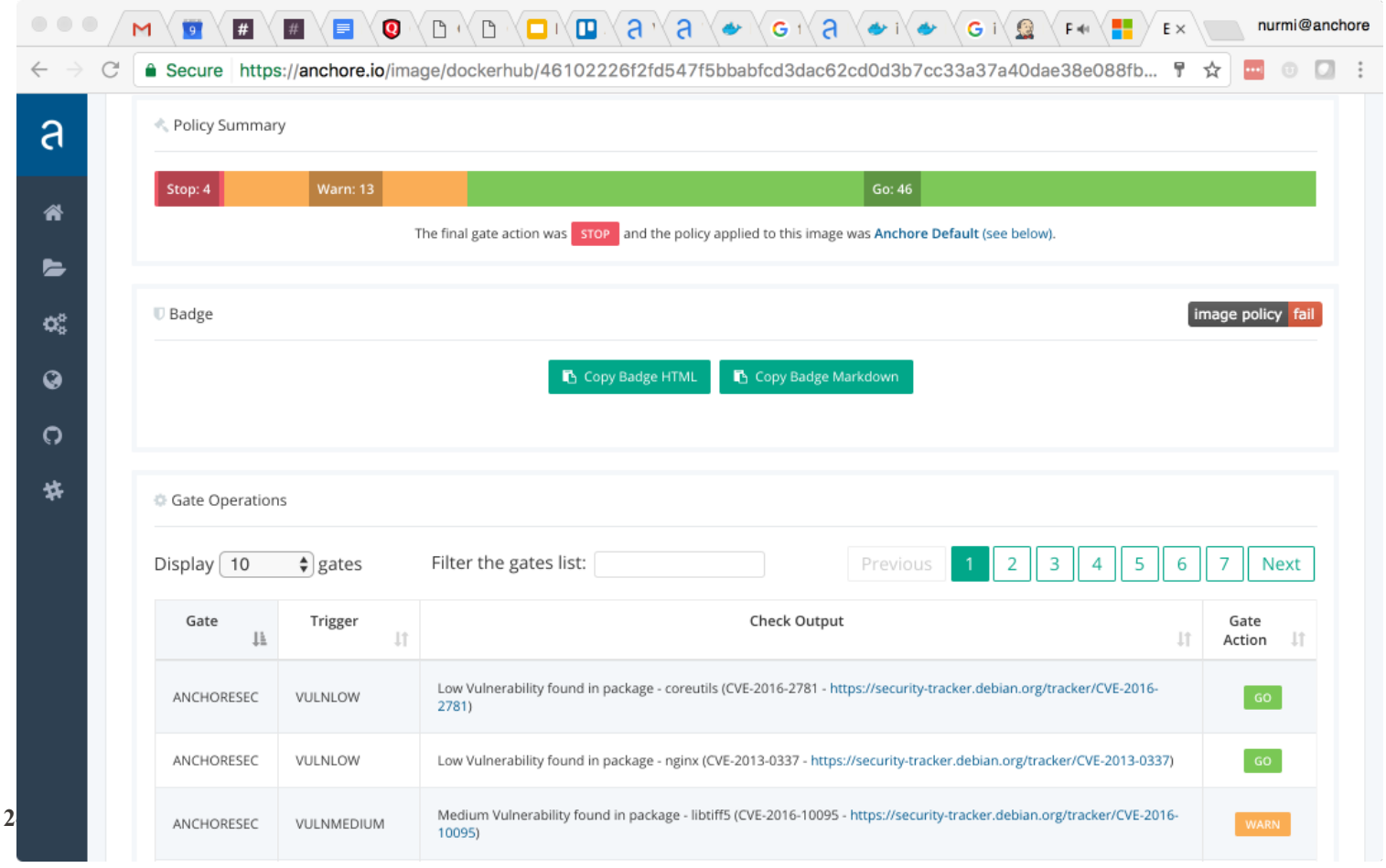
Tension: sprawl

- Detailed analysis allows for interesting queries
- Next Heartbleed just came out – see vulnerability surface immediately (no scan) and get instruction on how to remediate quickly

https://anchore.io



https://anchore.io



Anchore Tech

Anchore.io

- Scanning dockerhub (more soon!) public and private container images - Security, policy, contents
- ~20TB of analyzed data, 30k images and counting
- https://anchore.io

Jenkins CI/CD Plugin

- Include anchore analysis/policy application (and gate) into your container CI/CD process
- Official jenkins plugin from Jenkins UI

Anchore Scanner

- Open-source scanner itself
- Linux CLI for analyzing any container image and applying policy/security scan
- https://github.com/anchore/anchore

On-prem services

- Kubernetes webhook admission control
- On-prem stateful scanning, policy application, notification service

Challenges and Discussion

- Bare metal -> VM -> OS containers -> regular processes -> PaaS frameworks: is there ’one system’ or will the spectrum continue to broaden?
- Who has the control over what software is eventually actually deployed (dev, ops, dev/ops, security?
- Storage (well…state in general)!?
- Will containers eventually become just as heavy as VMs, as more standardization, security, and OS agnostic functionality are added to the mix?
- How will containers impact the increasing ubiquity of mobile environments/OSes?
- Micro-services architectures – lots of discussion about how containers are ushering in microservices – is the container abstraction right for microservices or just convenient?
- Will containers become the ‘process’, and if so what does the OS look like (and what do OS distributions look like?)

Thank you!

nurmi@anchore.com

HTTP://WWW.ANCHORE.COM

Typical Container Deployment