A container model for resource provision at a WLCG Tier-2

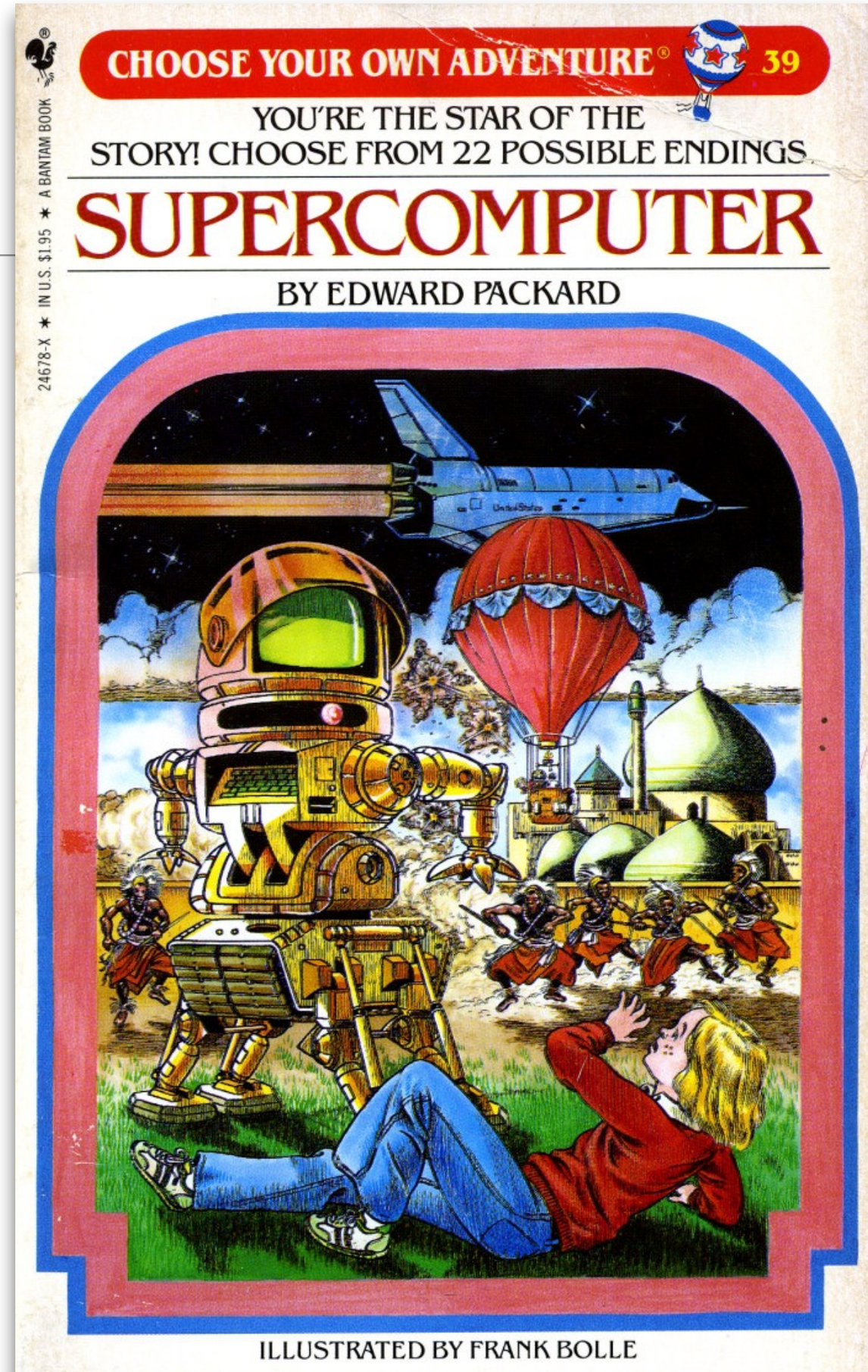
Gareth Roy, Gordon Stewart, David Crooks, Sam Skipsey & David Britton.
ACAT 2017: 21-25 August, University of Washington

University of Glasgow

GridPP
UK Computing for Particle Physics

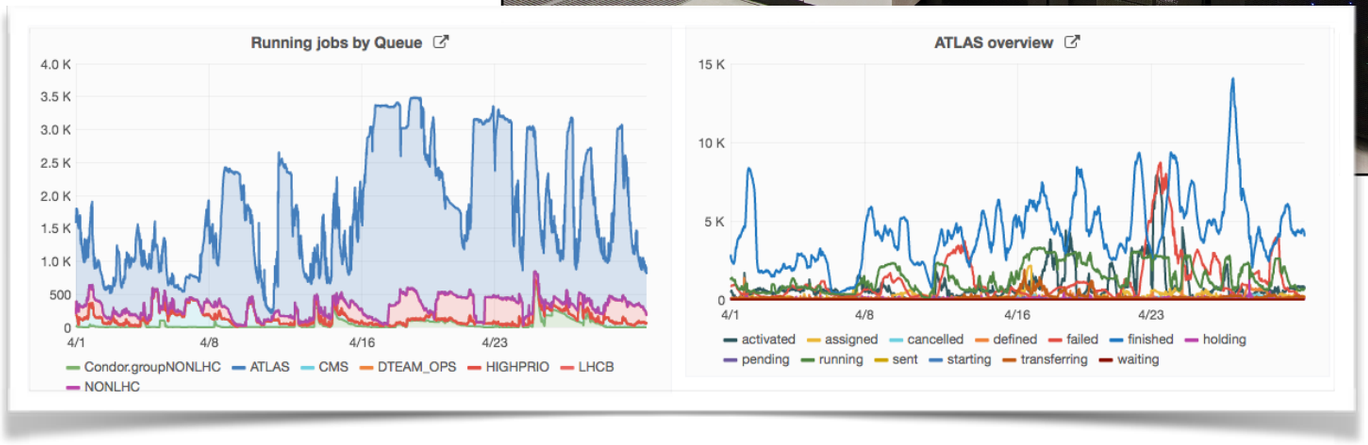
Overview

- Motivation
- Requirements from a Tier-2 perspective
- A worker node container
- Monitoring & Logging
- CI/CD
- Conclusion

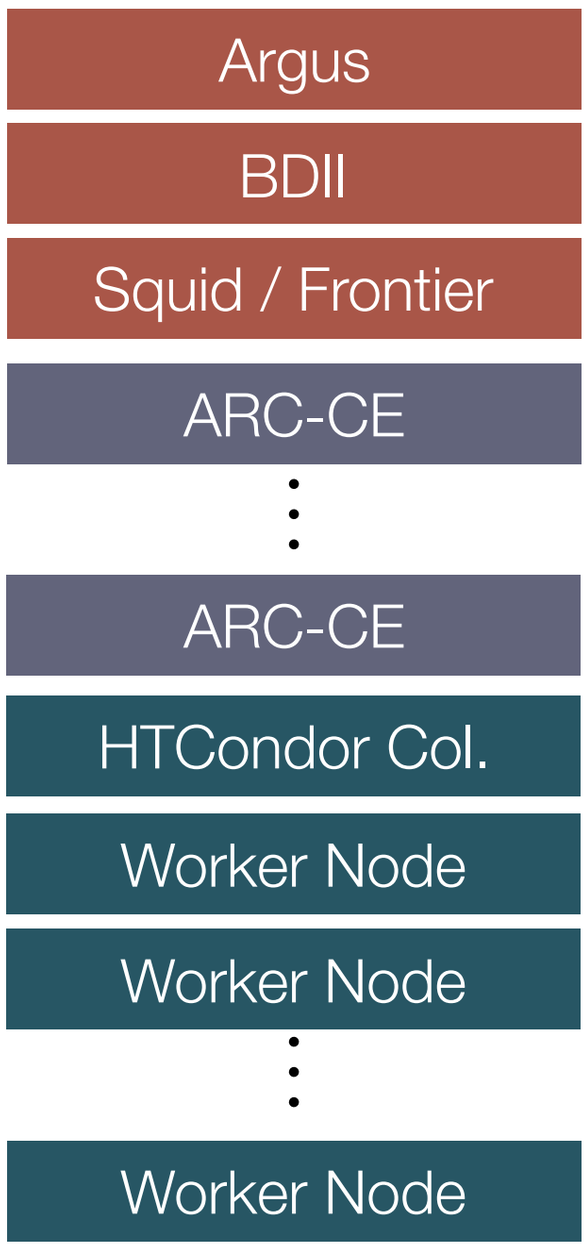


UKI-SCOTGRID-GLASGOW

- At present consist of:
  - 6200 CPU cores (with 1760 of that running VAC)
  - 63643 HEPSPEC.
  - 3.8 PB of Storage.
  - 160 Gb/s internal network bandwidth.
  - Primarily supports ATLAS and LHCb, acts as a CMS Tier-3 along with other smaller experiments and local user groups.

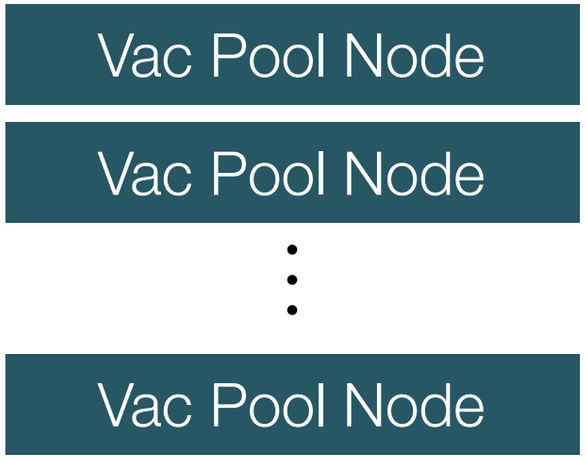


UKI-SCOTGRID-GLASGOW



Traditional Grid Site

- Glasgow examining methods to reduce overall manpower for running a Tier-2.
- At present run both traditional Grid resources as well as a Vac pool (25% or resources)
- Vac pros:
  - Simple.
  - Lightweight.
  - Auto-updating.
- Vac cons:
  - Slow to start VMs.
  - Inflexible resource requirements.
  - Opaque to local monitoring.
  - Scheduling is naive.
- Our goal is to take the good ideas from Vac and bring them into our standard site through the use of containers.

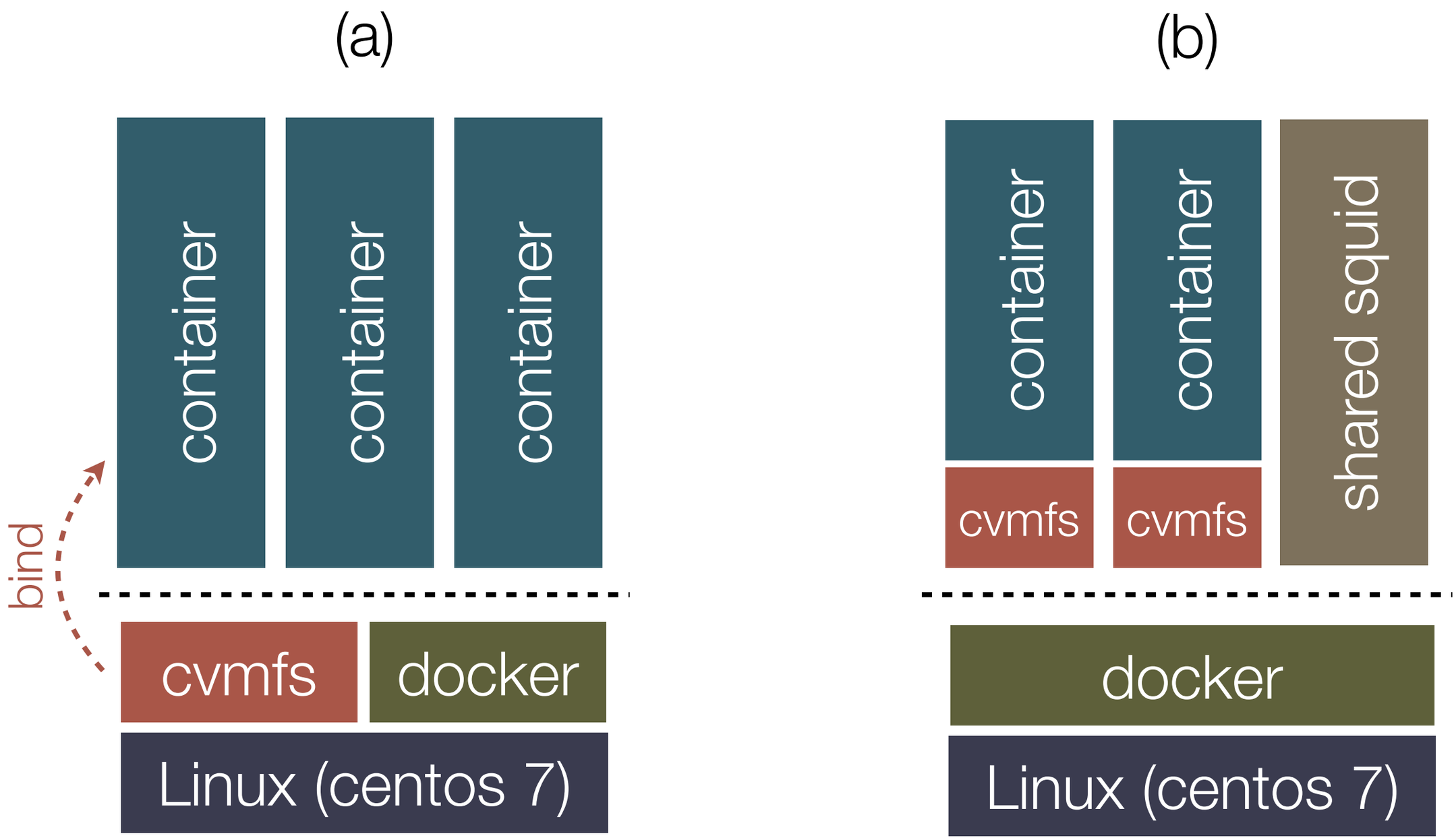


Vac Pool

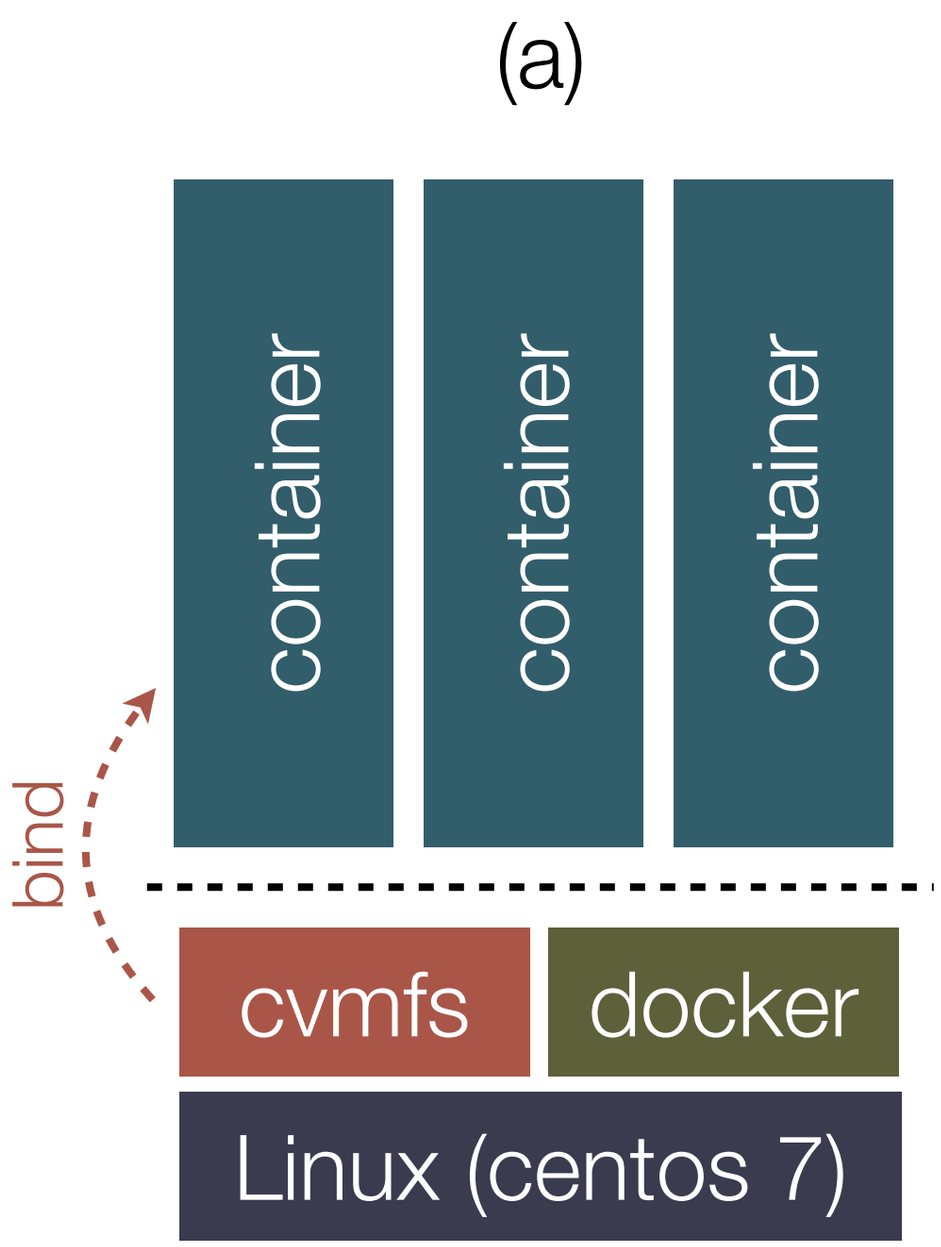
Requirements

- WLCG Requirements:
  - CVMFS, needed for access to experiments software stacks.
  - HEP_OSlibs, for many experiments a standard baseline of installed packages (provides 32-bit compatibility as well as many other required libraries).
  - ca-policy-egi-core/ca-policy-lcg, needed to meet WLCG trust chain requirements.
- Our Requirements:
  - Lightweight resource provision, quick to start and preferably ephemeral.
  - Connects to our existing HTCondor pool to leverage existing infrastructure.
  - Good monitoring that integrates with existing local tools.
  - Use “standard” tools, with broad support base.

Integrating CVMFS with a container (Docker)



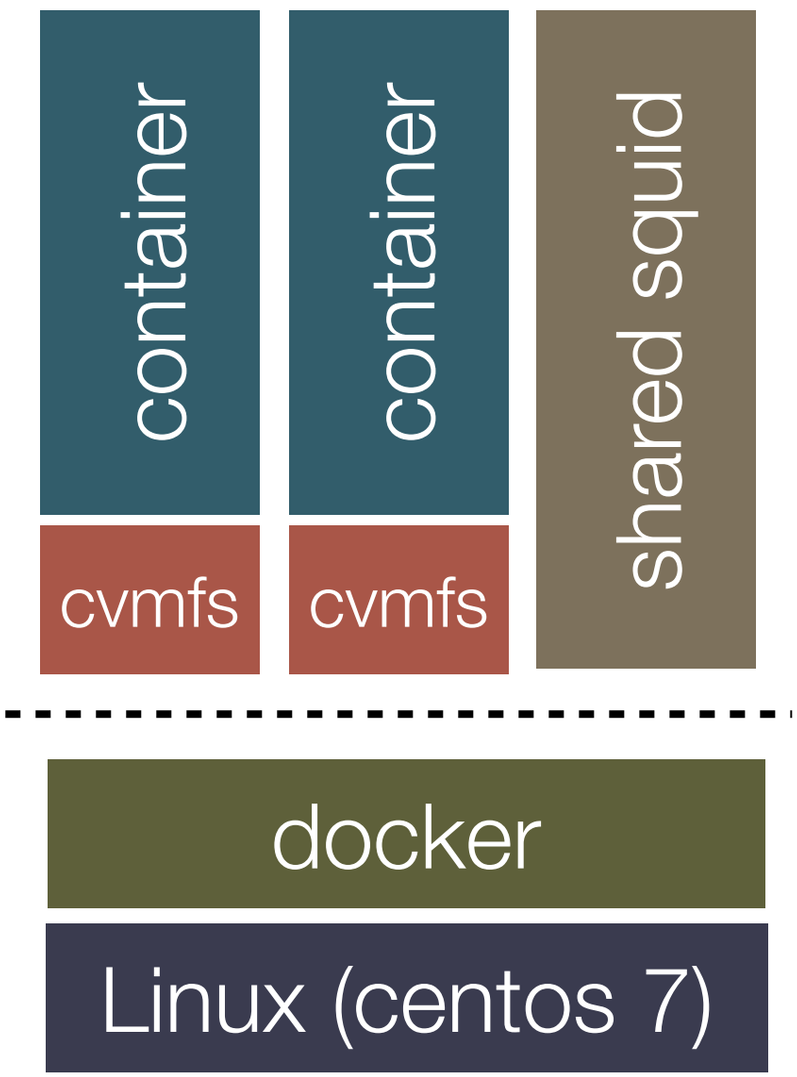
Integrating CVMFS



- Statically mount CVMFS filesystems to /cvmfs via /etc/fstab
- Bind mount into container with volume mounts (-v /cvmfs:/cvmfs)
- Pros:
  - shared cache.
  - simple bind semantics.
- Cons:
  - all required repositories need to be statically mounted.
  - breaks reboot (at least on vanilla centos7).
- Promising alternative CVMFS volume driver (https://gitlab.cern.ch/cloud-infrastructure/docker-volume-cvmfs/)

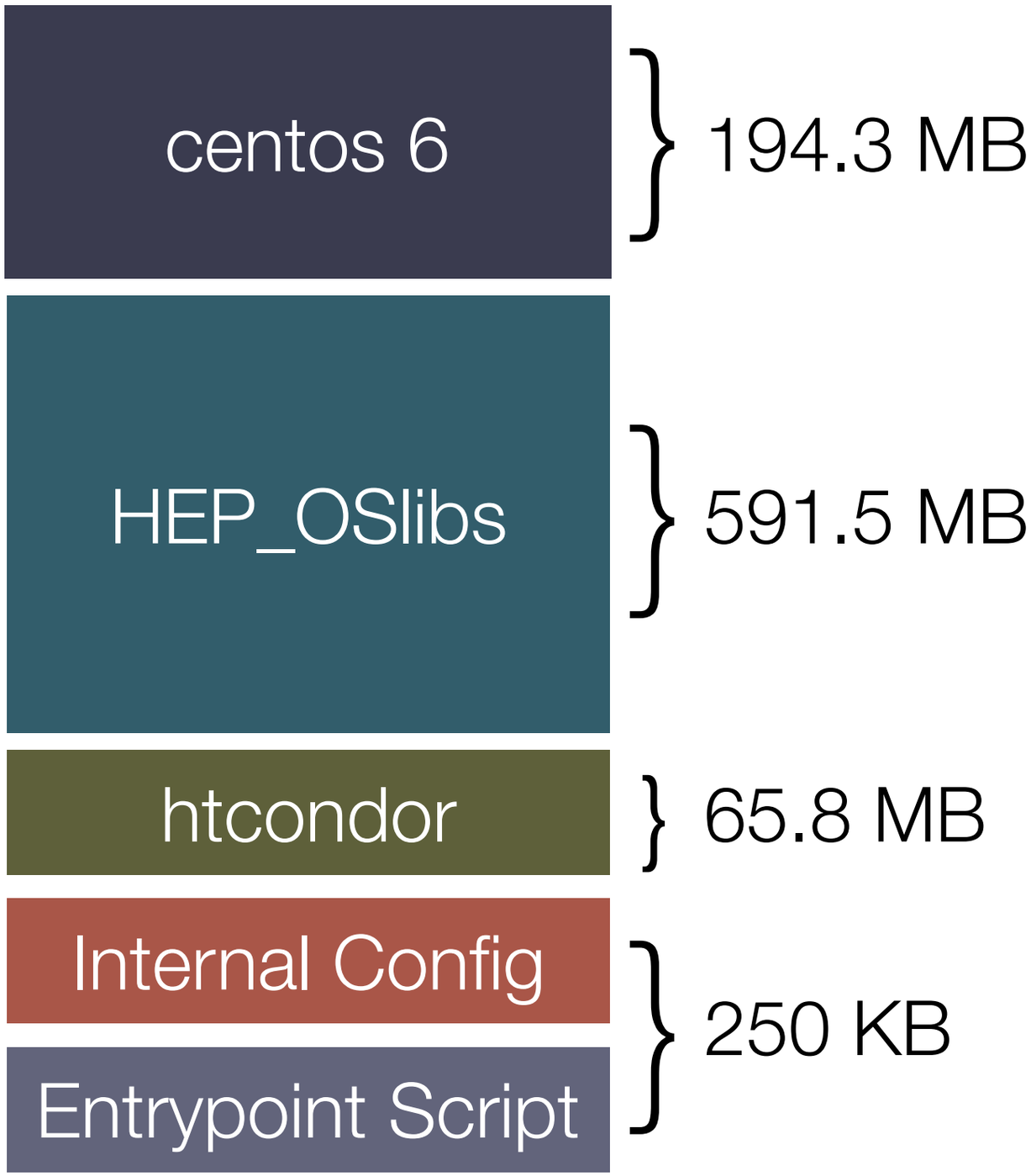
Integrating CVMFS

(b)



- Install autofs within container and run as a service.
- Container needs escalated privileges and fuse device to be mounted (--cap-add MKNOD --cap-add SYS_ADMIN --device /dev/fuse)
- Pros:
  - no CVMFS install required on “hypervisor”.
  - allows use of read-only linux distributions (coreos, rancheros).
- Cons:
  - no shared cache, approximate with shared local squid.
  - container needs to be run with elevated privileges.

WN Container (with bind mounted CVMFS)



- Total size is 851.9MB.
- Based on centos 6 for compatibility (centos 7 planned).
- Includes HEP_OSlibs as a common baseline for WLCG experiment code.
- Distributed to work nodes via an internal docker registry.
- In future replace HEP_OSlibs with experiment provided **singularity container** to reduce size.

Caveats

- Uses standard “grid style” pool accounts - would like to move to HTCondor per slot accounts.
- HTCondor configured to use CCB due to private networking between collector and per container startd's.
- In a non-privileged container no access to cgroups or other privileged information so the following parameters need to be unset in our condor configuration:
  - BASE_CGROUP=
  - DISCARD_SESSION_KEYRING_ON_STARTUP=FALSE
- Resource constraints applied to container at run time, not via HTCondor interaction with groups (can cause issues with eviction constraints).

```
[root@kube001 ~]# docker history kube001.beowulf.cluster:5000/cntwn
IMAGE               CREATED             CREATED BY                                      SIZE
c36974a7d4b9        3 days ago          /bin/sh -c #(nop)  ENTRYPOINT ["/cntwn.sh"]     0 B
6f383137eca3        3 days ago          /bin/sh -c #(nop) COPY file:0f2808c48e9b3eb20   723 B
a5845f68b5e4        3 days ago          /bin/sh -c #(nop) COPY file:248d1e69515bc4fc4   754 B
7dc81c18b6df        3 days ago          /bin/sh -c #(nop) COPY file:42c0d6c2e80b92b25   4.798 kB
90e219f2c0e0        5 days ago          /bin/sh -c rpm --import http://research.cs.wi   65.8 MB
ed7f2016b043        5 days ago          /bin/sh -c #(nop) ADD tarsum.v1+sha256:562103   167 B
dd1f9738d74c        5 days ago          /bin/sh -c #(nop) COPY file:3a1f1cc9294faa13a   10.25 kB
30f771feb031        5 days ago          /bin/sh -c #(nop) COPY file:04ba3676fc5724f70   238.9 kB
5fd8a5ef64d4        5 days ago          /bin/sh -c yum -y update &&     yum -y instal   591.5 MB
3e32556ae4ba        4 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]            0 B
<missing>           4 weeks ago         /bin/sh -c #(nop)  LABEL name=CentOS Base Ima   0 B
<missing>           4 weeks ago         /bin/sh -c #(nop) ADD file:c64182001a98ff64ec   194.3 MB
[root@kube001 ~]# _
```

Grid environment via CVMFS

```
host# docker exec --rm -it -v /cvmfs:/cvmfs centos:6 /bin/bash

cntr# . /cvmfs/grid.cern.ch/emi3wn-latest/etc/profile.d/a1_grid_env.sh
cntr# . /cvmfs/grid.cern.ch/emi3wn-latest/etc/profile.d/setup-en-example.sh

cntr# lcg-cp ….
```

- Using the same method as ATLAS VAC VM's we load a grid environment from CVMFS, along with all required CA certs and CRLs.
- Reduces image size, and decouples grid middleware from container image.
- In practice the grid environment is loaded via HTCondor at job run time with a custom user job wrapper (c.f. ATLAS VAC payloads)
- Using this process it is possible to create VAC-like containers that can contact experiment pilot factories directly.

A running container

```
Aug18   0:08  |   \_ /usr/bin/docker-containerd-shim-current a8d6581f1facdd718bb8349af7c95b4771056a3d42b27
Aug18   0:00  |   |   \_ /bin/bash /cntwn.sh
Aug18   0:03  |   |       \_ condor_master -pidfile /var/run/condor/condor_master.pid
Aug18   0:28  |   |       |   \_ condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R
Aug18   0:12  |   |       |   \_ condor_shared_port -f -p 9621
Aug18   3:40  |   |       |   \_ condor_startd -f
14:53   0:01  |   |       |       \_ condor_starter -f -a slot1_1 svr019.beowulf.cluster
14:53   0:00  |   |       |           \_ /bin/bash -l /var/lib/condor/execute/dir_30397/condor_exec.exe
14:53   0:00  |   |       |               \_ /usr/bin/time -o /var/lib/condor/execute/dir_30397/cP4MDm21S3
14:53   0:00  |   |       |                   \_ /bin/bash ./runpilot3-wrapper.sh -s UKI-SCOTGRID-GLASGOW_
14:53   0:07  |   |       |                       \_ /usr/bin/python pilot.py -d /var/lib/condor/execute/d
14:53   0:06  |   |       |                           \_ /usr/bin/python RunJob.py -a /cvmfs/atlas.cern.ch
14:56   0:00  |   |       |                           \_ /bin/sh -c export ATHENA_PROC_NUMBER=8;export
14:56   0:29  |   |       |                           |   \_ python /cvmfs/atlas.cern.ch/repo/sw/softw
16:17   0:00  |   |       |                           |       \_ /bin/sh ./runwrapper.ESDtoAOD.sh
16:17   0:47  |   |       |                           |       |   \_ /cvmfs/atlas.cern.ch/repo/sw/soft
16:19   6:16  |   |       |                           |       |       \_ /cvmfs/atlas.cern.ch/repo/sw/
16:19   6:17  |   |       |                           |       |       \_ /cvmfs/atlas.cern.ch/repo/sw/
16:19   6:14  |   |       |                           |       |       \_ /cvmfs/atlas.cern.ch/repo/sw/
16:19   6:14  |   |       |                           |       |       \_ /cvmfs/atlas.cern.ch/repo/sw/
16:19   6:13  |   |       |                           |       |       \_ /cvmfs/atlas.cern.ch/repo/sw/
16:19   6:15  |   |       |                           |       |       \_ /cvmfs/atlas.cern.ch/repo/sw/
16:19   6:18  |   |       |                           |       |       \_ /cvmfs/atlas.cern.ch/repo/sw/
16:19   6:13  |   |       |                           |       |       \_ /cvmfs/atlas.cern.ch/repo/sw/
16:17   0:07  |   |       |                           |       \_ MemoryMonitor --pid 16780 --filename
14:56   0:00  |   |       |                           \_ /bin/sh -c cd /var/lib/condor/execute/dir_303
14:56   0:49  |   |       |                               \_ MemoryMonitor --pid 12170 --filename memo
Aug18   0:02  |   |       \_ tail -f /var/log/condor/KernelTuning.log /var/log/condor/MasterLog /var/log/c
```

HTCondor

Pilot

Payload

As observed from the “Hypervisor”

Monitoring & Logging

- Local monitoring is essential to enable sites to identify issues and optimise workloads.
- Identified standard tools which can be deployed as containers:
  - For metric gathering uses cAdvisor (stand alone container here, built in when using Kubernetes)
  - For aggregation, gather metrics from each instance via Prometheus.
  - Forward logs from each container via LogSpout
    - Logging is not perfect as stdout grabbed by condor and pilot payloads and not “tee’d” back to stdout.
    - Currently only logging HTCondor logs (as well service container logs e.g. prometheus).
  - Log aggregation using oklog (will be moved to Elasticsearch once site instance becomes available).

cAdviser (via container or built into Kubernetes)

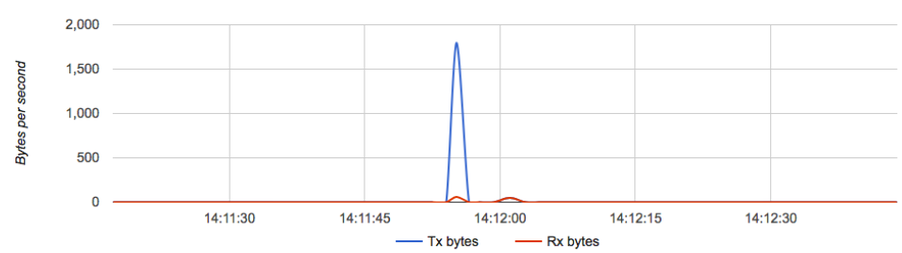
Processes

User	PID	PPID	Start Time	CPU %	MEM %	RSS	Virtual Size	Status	Running Time	Command
201806	18,555	9,737	12:01	98.60	2.80	1.79 GiB	2.54 GiB	RN	02:06:50	athena.py
201806	18,556	9,737	12:01	98.60	2.80	1.79 GiB	2.54 GiB	RN	02:06:47	athena.py
201806	18,561	9,737	12:01	98.60	2.80	1.79 GiB	2.54 GiB	RN	02:06:52	athena.py
201806	18,557	9,737	12:01	98.50	2.80	1.79 GiB	2.54 GiB	RN	02:06:42	athena.py
201806	18,558	9,737	12:01	98.50	2.80	1.79 GiB	2.54 GiB	RN	02:06:46	athena.py
201806	18,559	9,737	12:01	98.50	2.80	1.79 GiB	2.54 GiB	RN	02:06:46	athena.py
201806	18,560	9,737	12:01	98.50	2.80	1.79 GiB	2.54 GiB	RN	02:06:43	athena.py
201806	18,562	9,737	12:01	98.50	2.80	1.79 GiB	2.54 GiB	RN	02:06:41	athena.py
201806	9,736	9,710	11:58	1.80	0.00	1.04 MiB	15.94 MiB	SN	00:02:28	MemoryMonitor
201806	9,737	9,735	11:58	1.50	2.70	1.69 GiB	2.54 GiB	SNl	00:02:02	athena.py
201806	9,648	8,336	11:58	0.90	0.00	1.05 MiB	16.50 MiB	SN	00:01:18	MemoryMonitor
201806	5,455	3,532	11:57	0.10	0.00	29.33 MiB	216.21 MiB	SNl	00:00:08	python
499	2,308	24,009	11:57	0.00	0.00	7.73 MiB	46.29 MiB	Ss	00:00:01	condor_starter
201806	2,317	2,308	11:57	0.00	0.00	1.53 MiB	12.56 MiB	SNs	00:00:00	condor_exec.exe
201806	3,531	2,317	11:57	0.00	0.00	328.00 KiB	4.02 MiB	SN	00:00:00	time
201806	3,532	3,531	11:57	0.00	0.00	1.87 MiB	12.93 MiB	SN	00:00:00	runpilot3-wrapp
201806	5,649	5,455	11:57	0.00	0.10	119.45 MiB	299.21 MiB	SN	00:00:04	python
201806	7,768	5,649	11:58	0.00	0.00	2.40 MiB	13.39 MiB	SN	00:00:01	sh
201806	8,336	5,649	11:58	0.00	0.00	1.92 MiB	12.95 MiB	SN	00:00:00	sh
201806	9,710	7,768	11:58	0.00	0.10	94.45 MiB	604.61 MiB	SN	00:00:03	python
201806	9,735	9,710	11:58	0.00	0.00	1.52 MiB	11.27 MiB	SN	00:00:00	runwrapper.EVNT
root	23,924	23,909	Aug02	0.00	0.00	1.12 MiB	11.09 MiB	Ss	00:00:00	cntwn.sh
499	23,967	23,924	Aug02	0.00	0.00	4.25 MiB	44.93 MiB	Ss	00:00:03	condor_master
root	24,005	23,967	Aug02	0.00	0.00	2.44 MiB	24.36 MiB	S	00:00:21	condor_procd
499	24,007	23,967	Aug02	0.00	0.00	3.64 MiB	43.11 MiB	Ss	00:00:03	condor_shared_p

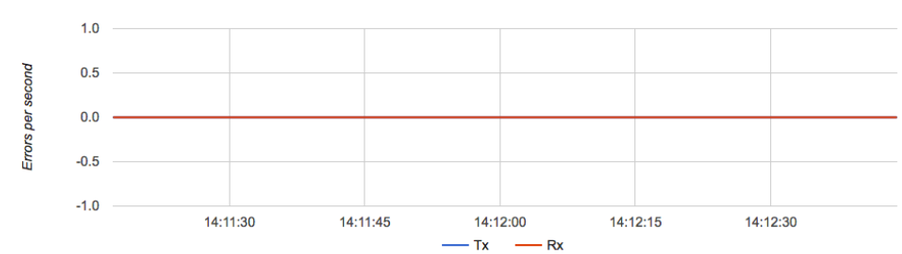
Network

Interface: **eth0**

Throughput

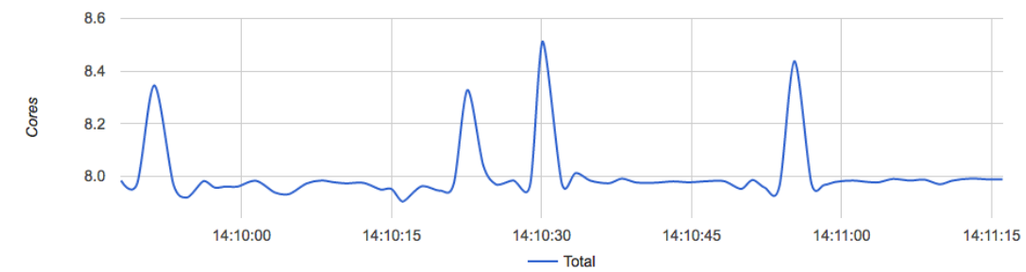


Errors

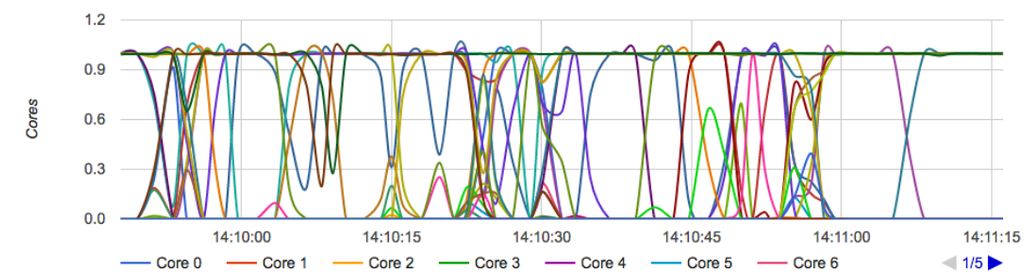


CPU

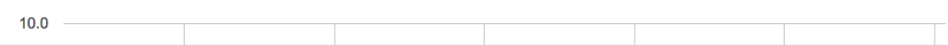
Total Usage



Usage per Core

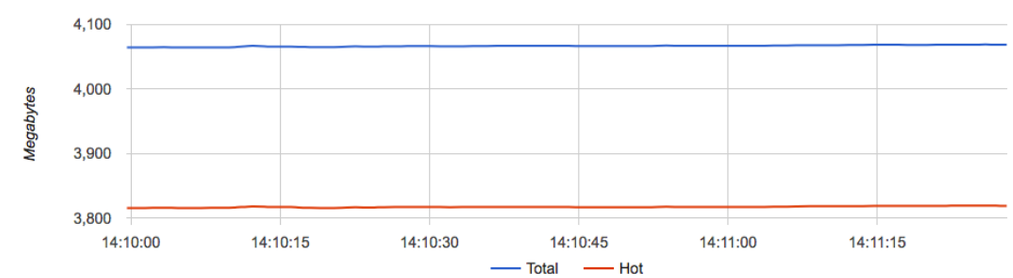


Usage Breakdown

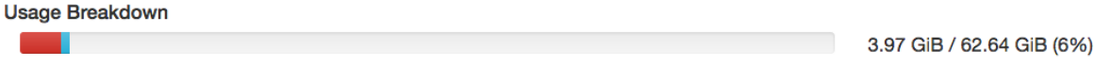


Memory

Total Usage



Usage Breakdown



Aggregate Metrics via Prometheus

Prometheus Alerts Graph Status Help

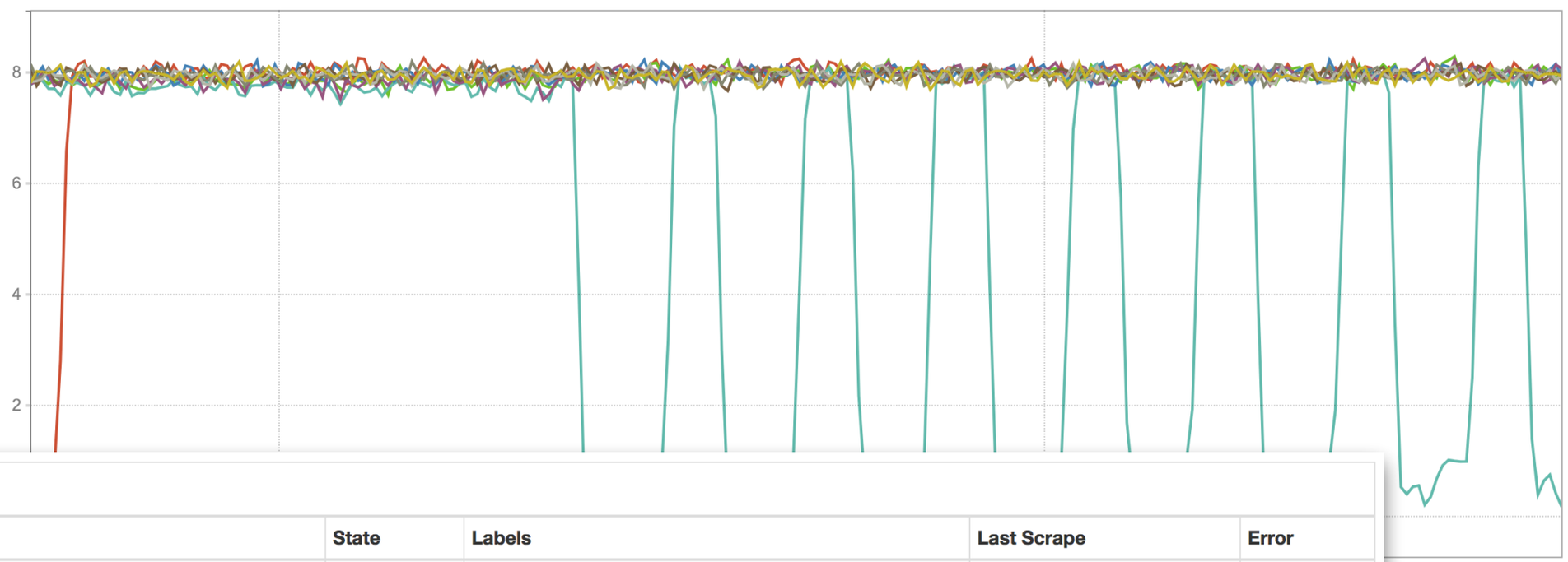
sum(rate(container_cpu_usage_seconds_total{name=~"kube.*"}[1m])) by (name)

Load time: 589ms
Resolution: 28s
Total time series: 9

Execute container_cpu_usage_seconc

Graph Console

2h Until Res. (s) stacked



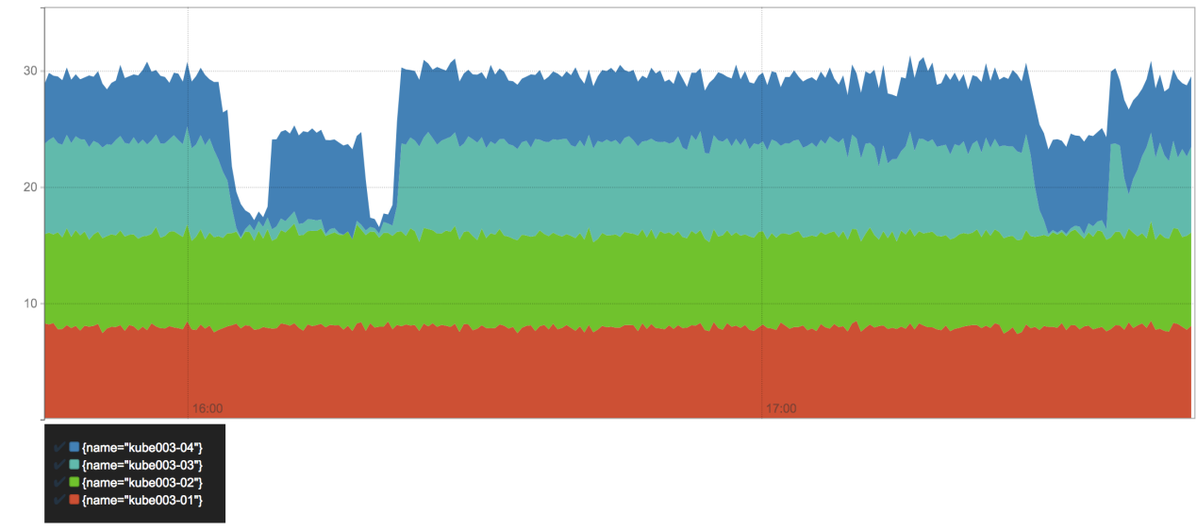
docker_hosts

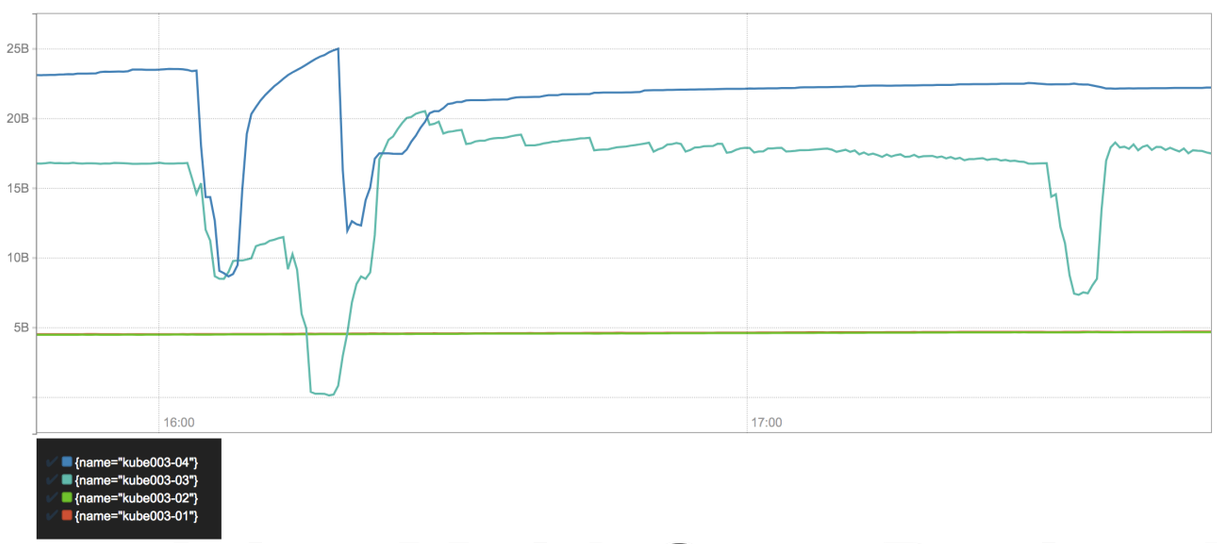
Endpoint	State	Labels	Last Scrape	Error
http://10.141.200.1:8080/metrics	UP	instance="10.141.200.1:8080"	2.265s ago	
http://10.141.200.2:8080/metrics	UP	instance="10.141.200.2:8080"	2.43s ago	
http://10.141.200.3:8080/metrics	UP	instance="10.141.200.3:8080"	4.944s ago	

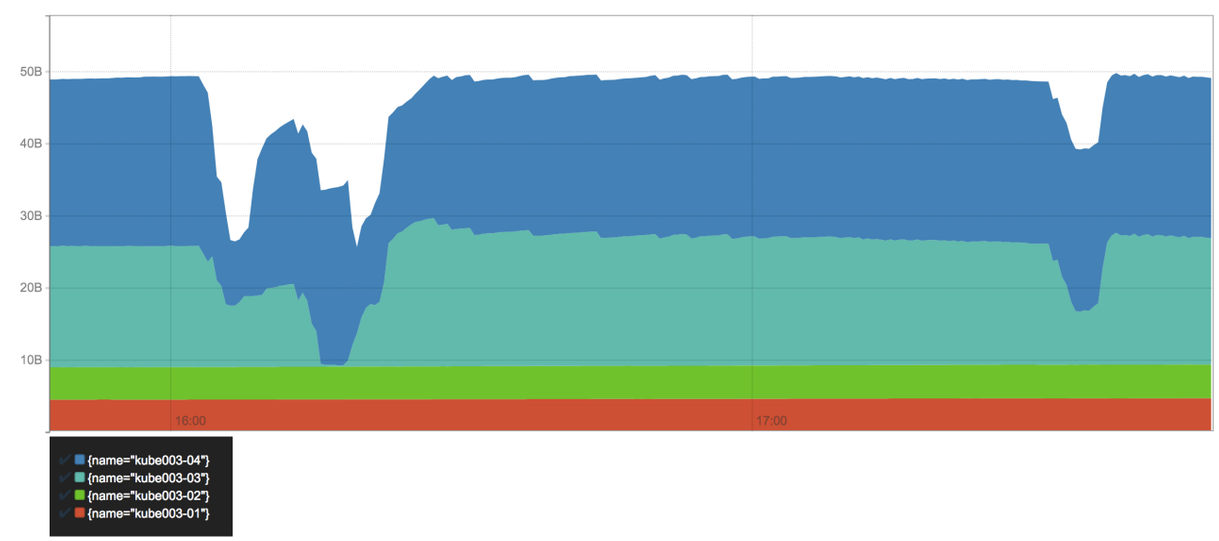
prometheus

Endpoint	State	Labels	Last Scrape	Error
http://localhost:9090/metrics	UP	instance="localhost:9090"	3.249s ago	

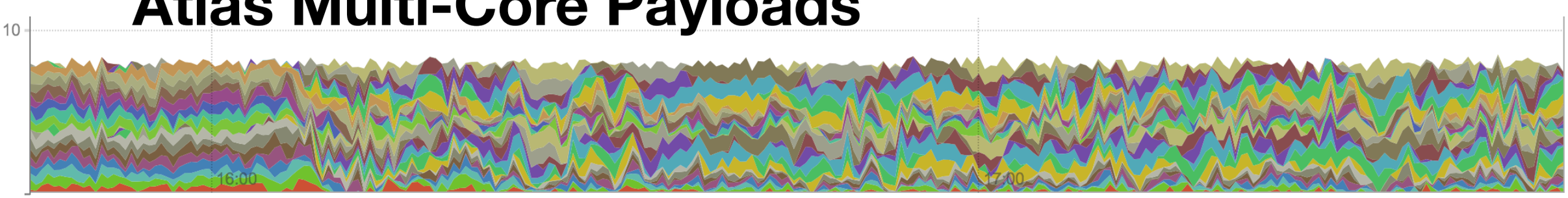
Aggregate Metrics via Prometheus



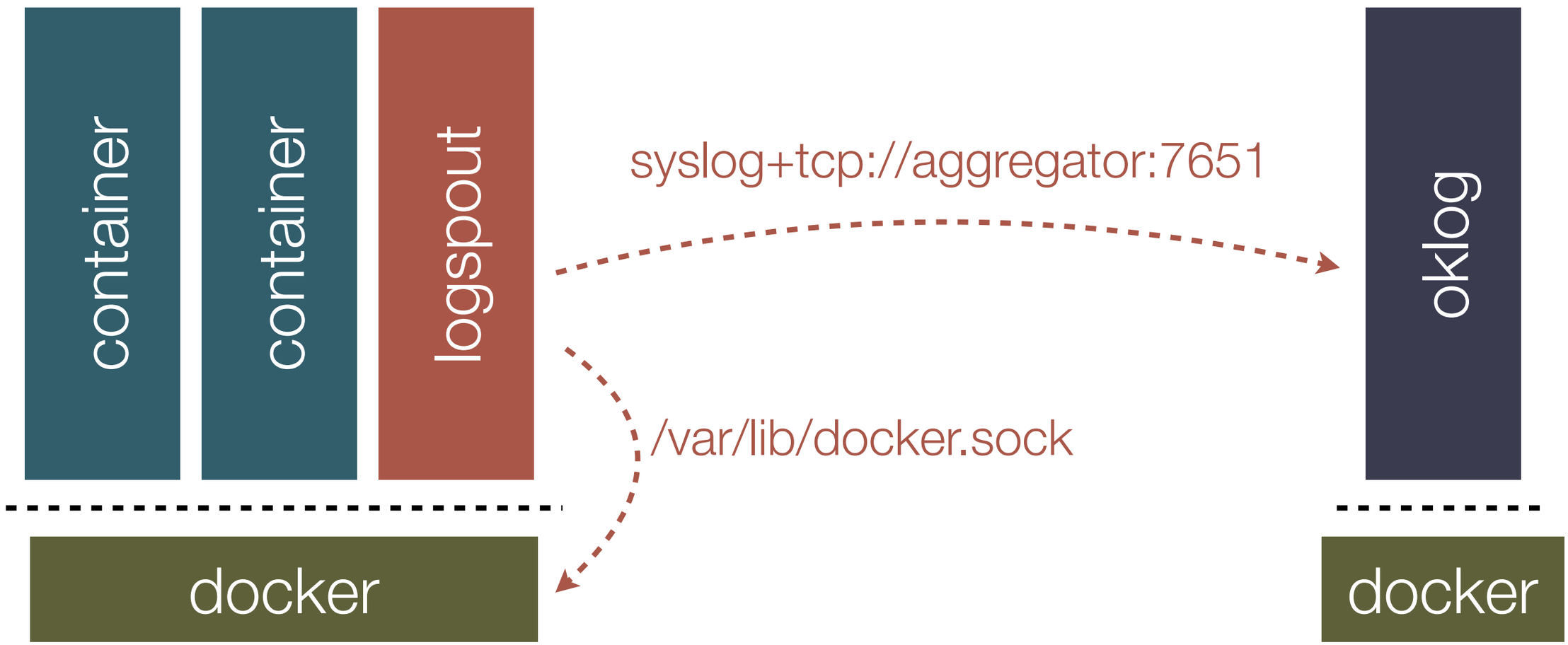




Atlas Multi-Core Payloads



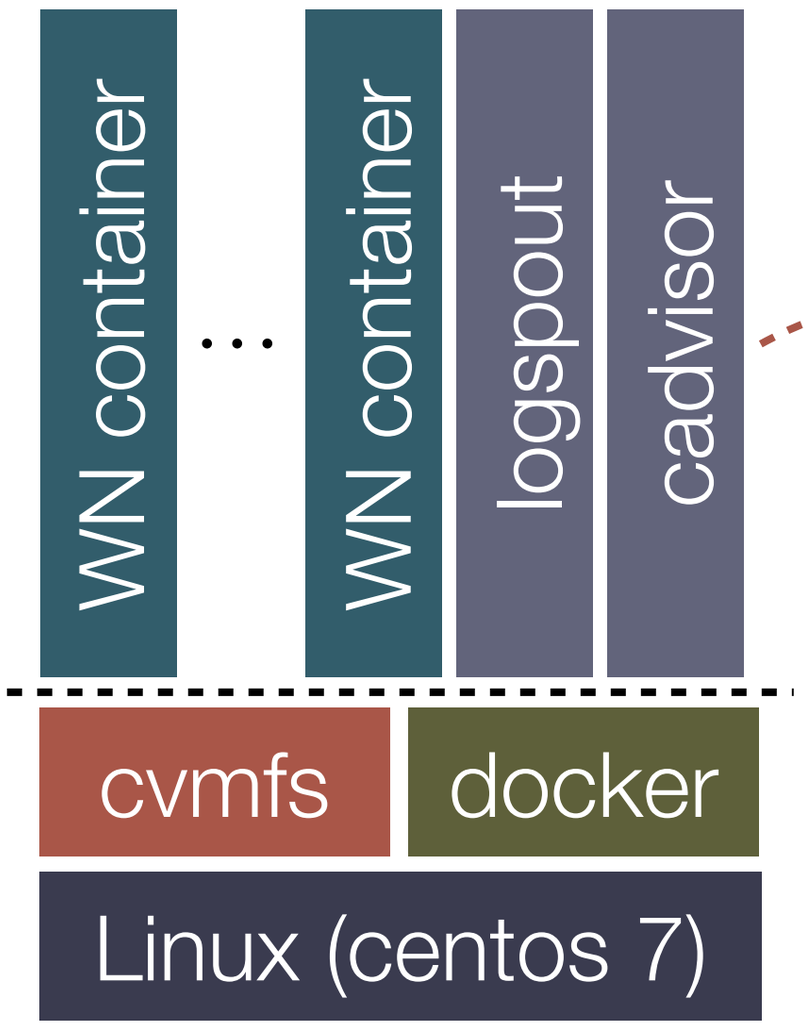
Log aggregation with logspout & oklog

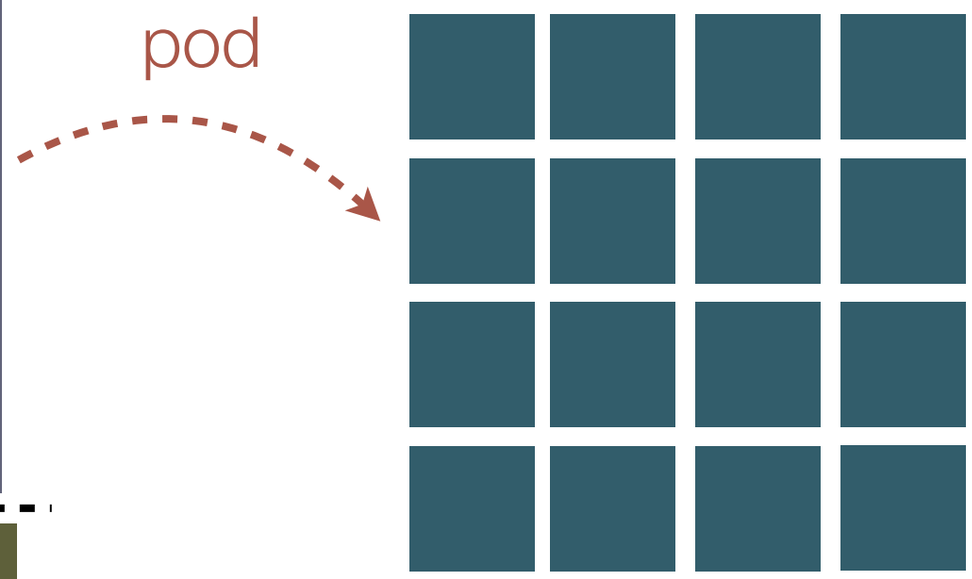



```
/ # ./oklog query -from 5m -q "prometheus"
<11>1 2017-08-04T13:51:01Z e7c84cebe4b3 prometheus 22185 - - time="2017-08-04T13:51:01Z" level=info msg="Checkpointing in-memory metrics and chunks..." source
="persistence.go:633"
<11>1 2017-08-04T13:51:01Z e7c84cebe4b3 prometheus 22185 - - time="2017-08-04T13:51:01Z" level=info msg="Done checkpointing in-memory metrics and chunks in 38
2.354532ms." source="persistence.go:665"
/ # _
```

Lightweight log aggregation, with “grep-like” semantics. To scale up oklog could be replaced with an ELK stack.

Future Deployment & Orchestration



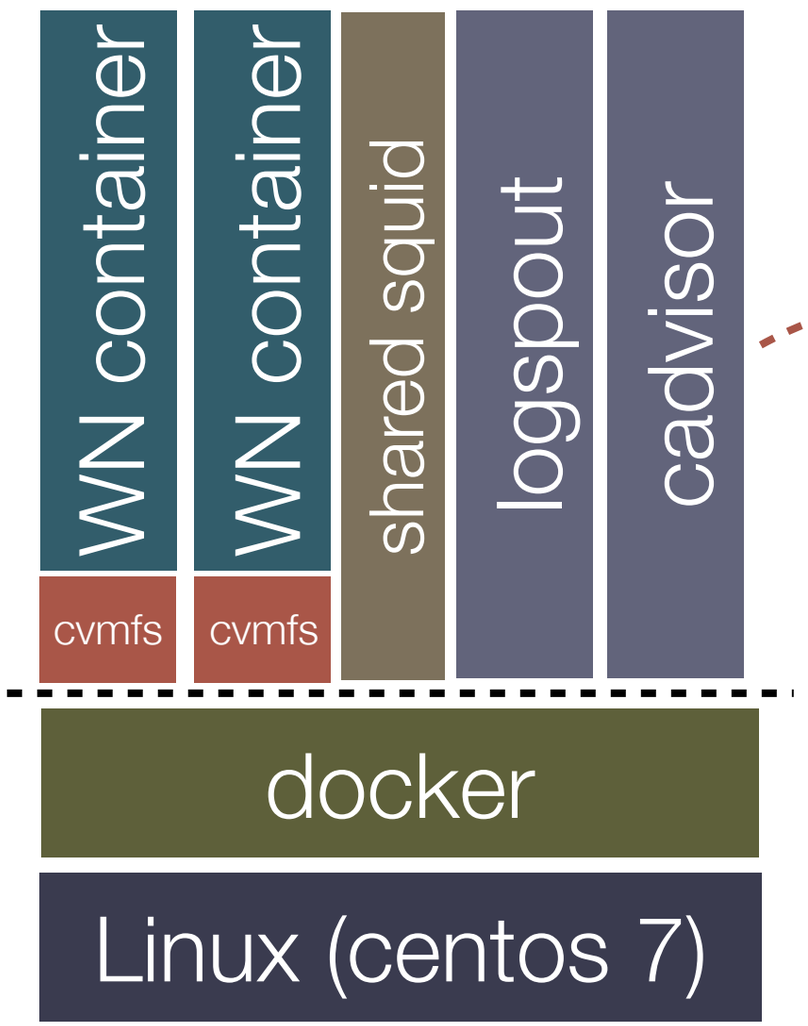


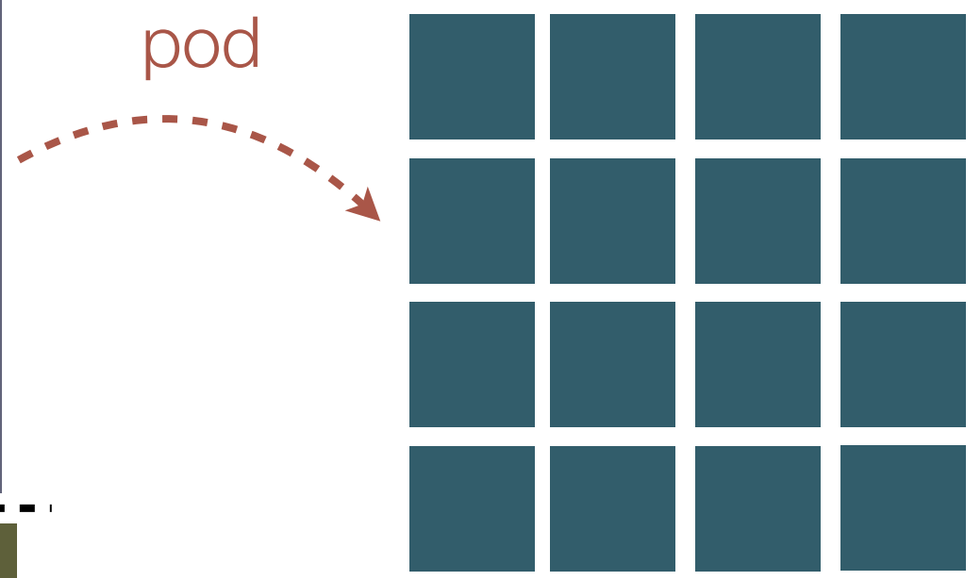
kubernetes

docker SWARM

```
[root@kube002 ~]# docker ps -a
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
9e75acb6d825        scotgrid/cntwn                 "/cntwn.sh"              7 days ago          Up 7 days           0.0.0.0:9621->9621/tcp   kube002-04
255e86f7a0fc        scotgrid/cntwn                 "/cntwn.sh"              7 days ago          Up 7 days           0.0.0.0:9620->9620/tcp   kube002-03
16d90fbbdb4b        scotgrid/cntwn                 "/cntwn.sh"              7 days ago          Up 7 days           0.0.0.0:9619->9619/tcp   kube002-02
68ff46f29364        scotgrid/cntwn                 "/cntwn.sh"              7 days ago          Up 7 days           0.0.0.0:9618->9618/tcp   kube002-01
b222002f02ba        gliderlabs/logspout:latest     "/bin/logspout syslog"   13 days ago         Up 13 days          80/tcp                   logspout
83e3531217af        google/cadvisor:latest         "/usr/bin/cadvisor -l"   2 weeks ago         Up 2 weeks          0.0.0.0:8080->8080/tcp   cadvisor
[root@kube002 ~]# _
```

Future Deployment & Orchestration





kubernetes

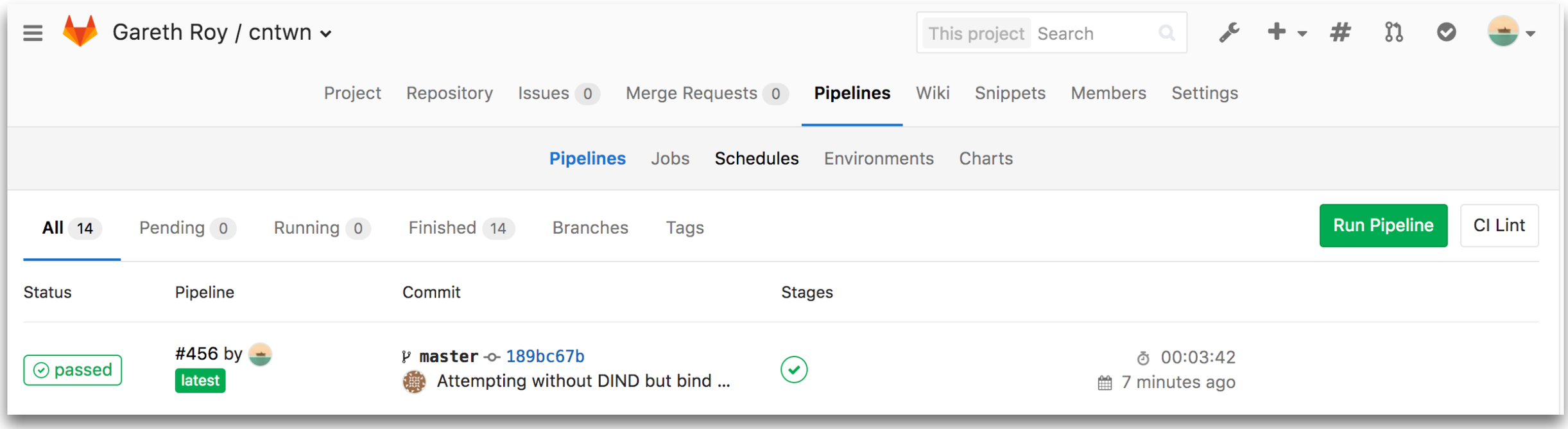
docker SWARM

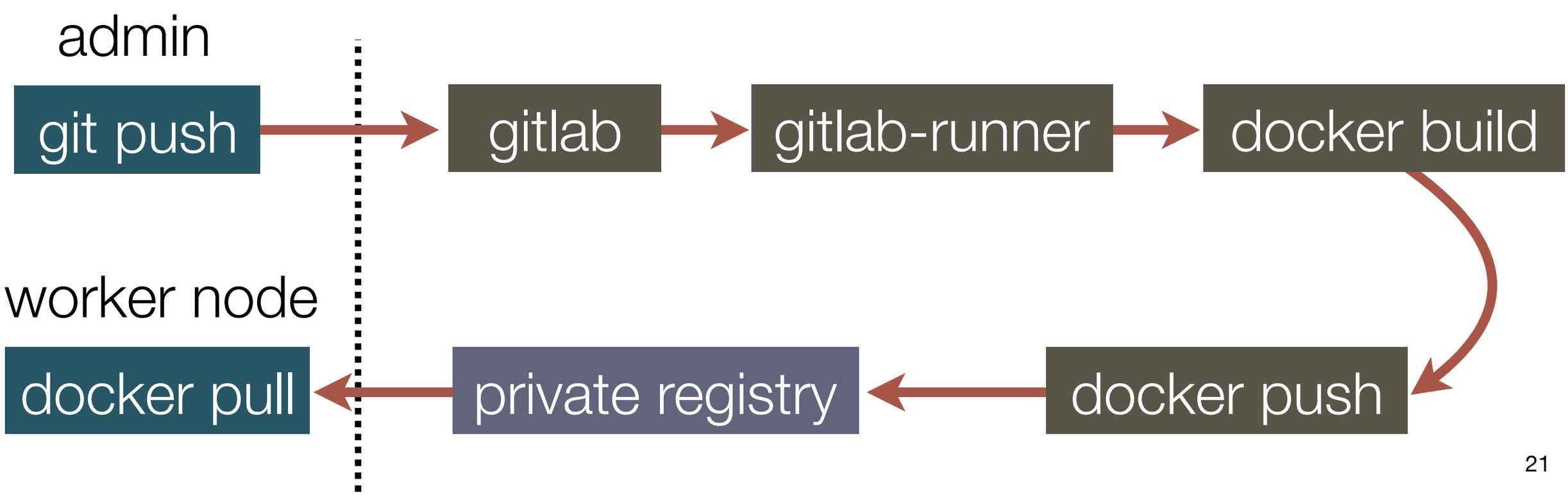
```
[root@kube002 ~]# docker ps -a
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
9e75acb6d825        scotgrid/cntwn                 "/cntwn.sh"              7 days ago          Up 7 days           0.0.0.0:9621->9621/tcp   kube002-04
255e86f7a0fc        scotgrid/cntwn                 "/cntwn.sh"              7 days ago          Up 7 days           0.0.0.0:9620->9620/tcp   kube002-03
16d90fbbdb4b        scotgrid/cntwn                 "/cntwn.sh"              7 days ago          Up 7 days           0.0.0.0:9619->9619/tcp   kube002-02
68ff46f29364        scotgrid/cntwn                 "/cntwn.sh"              7 days ago          Up 7 days           0.0.0.0:9618->9618/tcp   kube002-01
b222002f02ba        gliderlabs/logspout:latest     "/bin/logspout syslog"   13 days ago         Up 13 days          80/tcp                   logspout
83e3531217af        google/cadvisor:latest         "/usr/bin/cadvisor -l"   2 weeks ago         Up 2 weeks          0.0.0.0:8080->8080/tcp   cadvisor
[root@kube002 ~]# _
```

CI/CD

- The goal is to have our compute resources be as ephemeral as possible.
  - Ideally a new container for each job run, to isolate and allow continuous deployment of upgrades.
  - This simplifies security patches etc., as they will be pushed on each container restart.
  - Allows “canary” builds to be deployed to test upgrades and also allows easy rollback by starting previous container
- To enable this use standard continuous integration tools:
  - GitLab (for revision control).
  - gitlab-runner (for building and pushing containers).
  - private docker registry for distributing images
  - simple cron for pulling updated images.

CI/CD


Gareth Roy / cntwn
This project
Search
Project
Repository
Issues 0
Merge Requests 0
Pipelines
Wiki
Snippets
Members
Settings
Pipelines
Jobs
Schedules
Environments
Charts
All 14
Pending 0
Running 0
Finished 14
Branches
Tags
Run Pipeline
CI Lint
Status
Pipeline
Commit
Stages
passed
#456 by
latest
master 189bc67b
Attempting without DIND but bind ...
00:03:42
7 minutes ago


admin
git push
gitlab
gitlab-runner
docker build
worker node
docker pull
private registry
docker push

Conclusion

- Examined the requirements of a container to run WLCG payloads.
- Created a nominal container and established PoC by running production ATLAS multi-core payloads.
- Integrated monitoring, logging and constructed a CI/CD pipeline to automate updates to containers.
- Next Steps:
  - Give each container a “lifespan” so they auto-expire.
  - Integrate with Kubernetes (or other mechanism) to get auto-container redeploys (via services).