

Containers for Cluster Management

XSEDE Service Provider Forum
10/13/17

Nathan Rini
Supercomputer Services Group

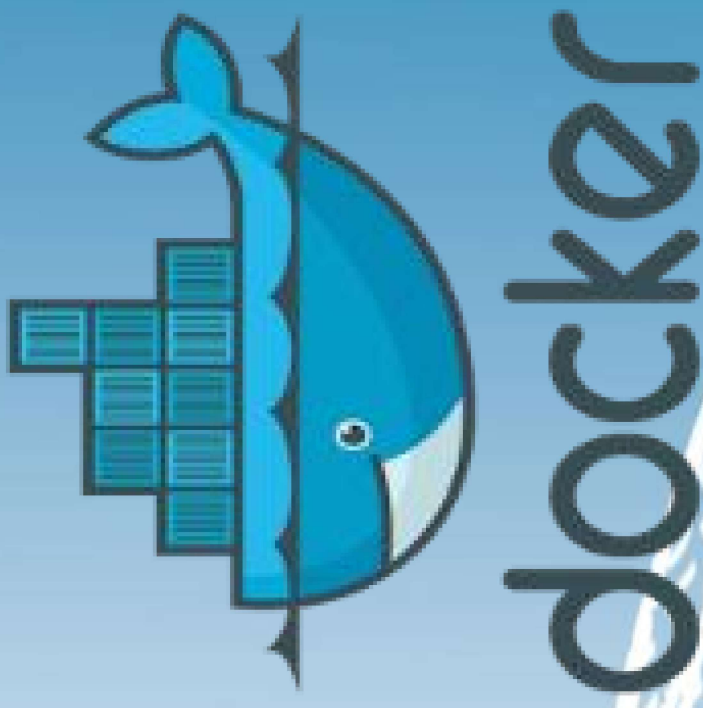


nate@ucar.edu

Why use containers for clusters?

- Virtualization is (relatively) slow:
 - Slow performance on cluster services can have multiplier effects for users.
- Existing cluster stacks (aka cluster management suites) are clunky:
 - Generally not designed, or have unreliable, HA functionality
 - Most cluster stacks are designed for specific hardware stacks
 - Several vendor stacks are very inflexible and break with different environments
- Containers come with a stack of features:
 - Provide needed environments for cluster stacks (e.g. exact paths to binaries and exact network interface names)
 - Easily moved between physical hosts with minimal downtime (usually in seconds)
 - Small performance penalty (around 3% on select syscalls)
 - Portable and built procedurally and reliably
 - Can usually be shared

Container Solutions





Inception

- <https://github.com/NCAR/Inception>

- Simple containers for users on HPC

- Suites usage in clusters:

- Infiniband support (no network namespace)
 - Nvidia driver support (only need nvidia devices mounted)
 - Seamless to users (mostly)
 - Users can call inception directly
 - Scheduler support not required but suggested
 - Admins setup images, not users
 - Can work with cluster filesystems (no user namespace)
- Users can call Inception inside of Inception

Inception

Config for a container

System administrators defined containers by setting up mounts (file or directory):

- Example mount setup: source mount -> mount in container
 - /gpfs/fs1/images/login -> /
 - /gpfs -> /gpfs
 - /etc/passwd -> /etc/passwd
 - /etc/shadow -> /etc/shadow
 - /etc/group -> /etc/group
 - /etc/hosts -> /etc/hosts
 - /tmp -> /tmp
 - /var/tmp -> /var/tmp
 - /dev -> /dev
 - /sys -> /sys
- Containers maintain security permissions of host and filesystems.
- Images are contained on shared filesystems to avoid wasting memory on batch nodes.



Inception

Swapping between user environments



- Default Login setup on Cheyenne (SLES12SP1 Cluster):**
 (nate@cheyenne6)~(0)\$ cat /etc/*release*
 NAME="SLES"
 VERSION="12-SP1"
 VERSION_ID="12.1"
 PRETTY_NAME="SUSE Linux Enterprise Server 12 SP1"
 Default Login setup on Cheyenne (SLES12SP1 Cluster):
- Switching to Yellowstone (RHEL6.4 Cluster) Login container:**
 (nate@cheyenne6)~(0)\$ inception -c yellowstone-login-rhel6.4 -x
 (nate@cheyenne6)/(0)\$ cat /etc/*release*
 LSB_VERSION=base-4.0-amd64:base-4.0-noarch:core-4.0-amd64:core-4.0-noar
 ch:graphics-4.0-amd64:graphics-4.0-noarch:printing-4.0-amd64:printing-4.0-noarc
 h
 Red Hat Enterprise Linux Server release 6.4 (Santiago)
 Red Hat Enterprise Linux Server release 6.4 (Santiago)
 cpe:/o:redhat:enterprise_linux:6server:ga:server



Inception

Use Case: Cluster Cron

Cron on clusters is messy.

- Cron jobs are not scheduled. (For better or worse.)
 - Jobs outside of scheduler don't get charged.
 - Users complain when their cron jobs run slow or their cronjobs are running while batch jobs are running.
 - Users like to use meta-schedulers
 - Users expect cron to be available and reliable.
 - Nodes crash on clusters all the time.
 - Users lose track of crontabs spread over clusters.
- Setting up a single cron server for users using inception solves these problems transparently to (most) users.
- Users with issues with central cron can worked with to use the scheduler and cron cooperatively.

Inception

Cluster Cron: System Setup (1 of 2)



1. Take image of login nodes and create inception container.
2. Create new crontab setup on shared filesystem.
3. Add access list for cron in pam on all nodes disabling users ability to change cron to force them to use cluster cron.
4. Setup additional mounts for cron in container to override system cron configuration:
 - a. source mount -> mount in container:
 - i. `/gpfs/fs1/cron/ -> /var/spool/cron`
 - ii. `/gpfs/fs1/cron/cron.d -> /etc/cron.d`
 - iii. `/gpfs/fs1/cron/cron.daily -> /etc/cron.daily`
 - iv. `/gpfs/fs1/cron/cron.hourly -> /etc/cron.hourly`
 - v. `/gpfs/fs1/cron/cron.monthly -> /etc/cron.monthly`
 - vi. `/gpfs/fs1/cron/cron.weekly -> /etc/cron.weekly`
 - vii. `/gpfs/fs1/cron/pam.cron.d -> /etc/pam.d/cron`
 - b. Create editable tabs directory to match cron's normal permissions:
 - i. `mkdir -m 2700 /gpfs/fs1/cron/tabs`



Inception

Cluster Cron: System Setup

1. Add systemd unit file to start cron inside of inception:
/usr/lib/systemd/system/cluster-cron.service
[Unit]
Description=Cluster Cron
After=cron.service
[Service]
ExecStart=/usr/local/bin/inception/ -c cron -- /usr/sbin/cron -s -n
-i
ExecReload=/usr/bin/kill -s SIGHUP \$MAINPID
Restart=on-failure
RestartSec=42s
[Install]
WantedBy=multi-user.target
2. Enable and Start the cluster-cron.service with systemctl
3. Setup crontab wrapper in PATH for users to use inception

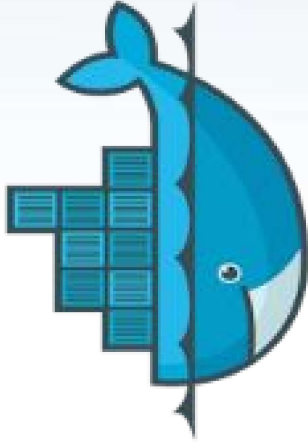
Inception

Cluster Cron: User Experience

Most users will be unaware that cron has been centralized. Other users who had been doing creative things with cron will likely notice.



- Viewing the wrapper:
user@login1:~> which crontab
/gpfs/fs1/bin/crontab
user@login1:~> cat \$(which crontab)
#!/bin/bash
/usr/local/bin/inception -x -c cron -- /usr/bin/crontab \$*
- What the user will see:
user@login1:~> crontab -l
no crontab for user
- What the user will see if they try to use the local system crontab:
user@login1:~> /usr/bin/crontab -l
You (user) are not allowed to access to (crontab) because of pam configuration.



docker

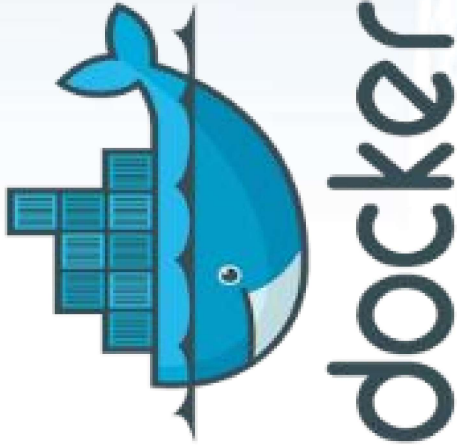


Docker

- Docker
 - De Facto standard in Containers
 - Designed for contained services
 - Ideally 1 service per container
 - Use docker-compose to combine multiple services
 - Network isolation of services
 - Security issues restrict user base
 - Recipe based container building and management
 - Automatic network configuration (usually)
- Any user with access to the docker CLI can take root of the machine.
- Services served by docker do not have this issue (unless misconfigured).
- Docker changes rapidly and new features can be buggy
- Clunky user interface and obscure configuration items
- Hard to keep track of best practices



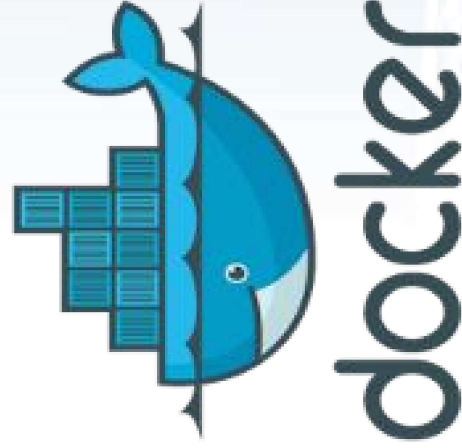
Docker Best Practices (2017)



- Use docker's build system to setup cluster services
- Use makefiles to simplify using docker
 - Never modify the contents of your docker images directly
 - Check reproducibility of all of your service builds
 - Make sure to clear caches and old images each build
 - Consider pulling source code from github in dockerfile to keep up with developers. Distro Package are usually pretty behind.
- Use a local docker pull through registry
- Use apt-cacher to avoid hammering the debian mirrors
- Use well defined container properties
 - Use logical names and set container host names
 - IP of services should always be set
- Use nginx (or another proxy) for web services to allow for HA
- Set all containers to use syslog for logging
- Use supervisor for services that may crash or fail
- Set containers to auto-restart
- Disable docker networking from changing your firewalls. Setup nat based bridges or use macvlan mode.

Example Docker Service

Nagios Dockerfile (partial)



```
FROM debian:stretch
```

```
RUN apt-get -y update && apt-get dist-upgrade -y
```

```
RUN apt-get -y install nsca monitoring-plugins-basic git make autoconf gcc unzip gzip libgd-dev  
apache2 supervisor cron php
```

```
RUN groupadd -r nagios
```

```
RUN useradd -g nagios -m -r -s /bin/bash nagios
```

```
RUN git clone https://github.com/NagiosEnterprises/nagioscore.git
```

```
WORKDIR /nagioscore
```

```
RUN ./configure
```

```
RUN make -j 120 all
```

```
RUN make fullinstall
```

```
COPY apache2.conf /etc/apache2/apache2.conf
```

```
RUN mkdir -m 0770 -p /var/run/apache2 && chown www-data:www-data /var/run/apache2
```

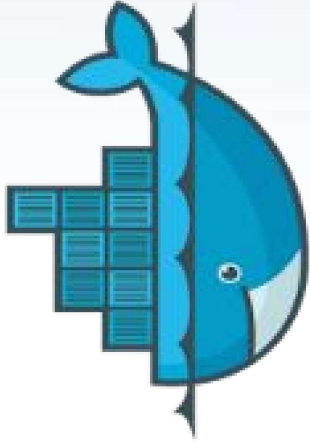
```
COPY nagios_config /etc/nagios
```

```
RUN mkdir -m 0770 -p /var/log/nagios/spool/checkresults /var/spool/nagios/cmd/ && chown
```

```
nagios:nagios -R /var/log/nagios /etc/nagios /var/spool/nagios
```

```
COPY supervisord.conf /etc/supervisord.conf
```

```
CMD /usr/bin/supervisord -c /etc/supervisord.conf
```



docker

Example Docker Service Nagios Makefile

```

DOCKER_TAG    ?= nagios
IP             ?= 192.168.0.22
RUN_ARGS      ?= docker run -it --network="bridge-nat" --ip "$(IP)" --log-driver=syslog -v /dev/log:/dev/log -h
$(DOCKER_TAG) --name $(DOCKER_TAG)
BUILD_ARGS    ?= --rm

default: run

build:
    docker build $(BUILD_ARGS) \
        --network=host -t $(DOCKER_TAG) .

stop:
    docker kill $(DOCKER_TAG) || true
    docker stop $(DOCKER_TAG) || true
    docker rm $(DOCKER_TAG) || true

nocache:
    $(eval BUILD_ARGS := $(BUILD_ARGS) --no-cache)

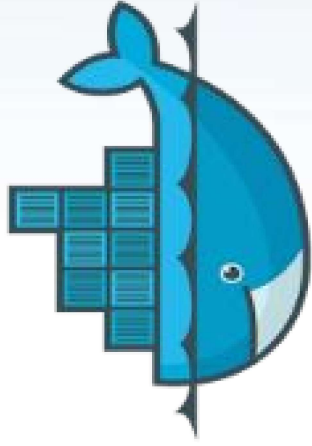
clean: nocache build

run: build stop
    $(RUN_ARGS) -d --restart unless-stopped $(DOCKER_TAG)

debug: build stop
    $(RUN_ARGS) --rm $(DOCKER_TAG)

bash: build stop
    $(RUN_ARGS) --rm $(DOCKER_TAG) /bin/bash
    
```


Example Docker Service Building and Running Instance



docker

```
$ make
docker build --rm --network=host -t nagios .
Sending build context to Docker daemon 903.2 kB
Step 1/27 : FROM debian:stretch
----> 2284106fa509

..... lots of logs here .....

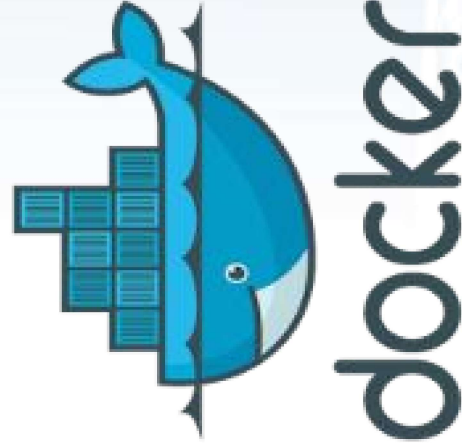
Removing intermediate container 2201bba12370
Step 27/27 : CMD /usr/bin/supervisord -c /etc/supervisord.conf
----> Running in 0807820e01b0
----> 21c2180de226
Removing intermediate container 0807820e01b0
Successfully built 21c2180de226
docker kill nagios || /bin/true
nagios
while docker stop nagios ; do /bin/true; done
Error response from daemon: No such container: nagios
docker run -d --rm -it --network="host" --log-driver=syslog -v /dev/log:/dev/log -h nagios --name
nagios nagios
d816948bd050b133cb1bcf299f531b10d0592e493d2afd48cbf6ad4c7f709120
```

Thank You

Backup Slides

Example Docker Service

Nagios Dockerfile (Full 1 of 2)



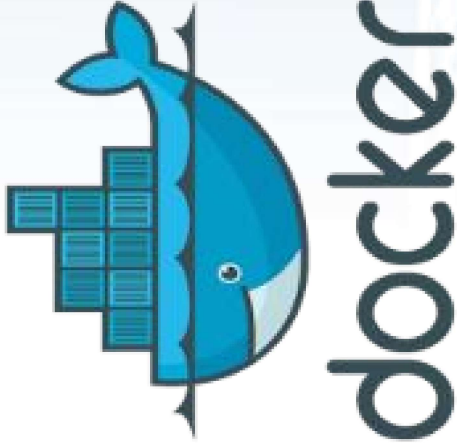
```
FROM debian:latest

RUN apt-get -y update && apt-get dist-upgrade -y
RUN apt-get -y install monitoring-plugins-basic git make autoconf gcc unzip gzip libgd-dev apache2 supervisor cron php libtool
libgettextpo-dev libgettextpo0 gettext m4 dnstools libmariadbclient-dev-compat libssl-dev wget libmcrypt-dev libmcrypt4 python-argh python
python-setuptools python-lxml
RUN a2enmod cgi.load
RUN groupadd -r nagios
RUN useradd -g nagios -m -r -s /bin/bash nagios

RUN git clone https://github.com/NagiosEnterprises/nagioscore.git
WORKDIR /nagioscore
RUN ./configure
RUN make -j 120 all
RUN make fullinstall

WORKDIR /
RUN git clone https://github.com/nagios-plugins/nagios-plugins.git
WORKDIR /nagios-plugins
RUN bash autogen.sh
RUN make -j 120 all
RUN make install

WORKDIR /
RUN git clone https://github.com/NagiosEnterprises/nsca.git
WORKDIR /nsca
RUN ./configure --host=ppc64le-unknown-linux-gnu --build=ppc64le-unknown-linux-gnu --with-nsca-user=nagios --with-nsca-grp=nagios
--with-nsca-port=5667
RUN make -j 120 all
RUN cp src/nsca /usr/sbin/
RUN cp src/send_nsca /usr/bin/
WORKDIR /
RUN rm -Rf /nsca
```



Example Docker Service

Nagios Dockerfile (Full 2 of 2)

```

WORKDIR /
RUN git clone https://github.com/NagiosEnterprises/nsca.git
WORKDIR /nsca
RUN git checkout nsca-2.7.2
RUN ./configure --host=ppc64le-unknown-linux-gnu --build=ppc64le-unknown-linux-gnu --with-nsca-user=nagios --with-nsca-grp=nagios
--with-nsca-port=5667
RUN make -j 120 all
RUN cp src/send_nsca /usr/bin/send_nsca-2.7.2
WORKDIR /
RUN rm -Rf /nsca

WORKDIR /
COPY apache2.conf /etc/apache2/apache2.conf
RUN mkdir -m 0770 -p /var/run/apache2 && chown www-data:www-data /var/run/apache2
COPY nagios_config /etc/nagios
RUN ln -s /etc/nagios /usr/local/nagios/etc
RUN mkdir -m 0770 -p /var/log/nagios/spool/checkresults /var/spool/nagios/cmd/ && chown nagios:nagios -R /var/log/nagios /etc/nagios
/var/spool/nagios
RUN htpasswd -c -b -s /etc/apache2/htpasswd.users nagiosadmin 'PASSWORD' #use a real password here!
RUN chown www-data:www-data /etc/apache2/htpasswd.users
RUN usermod -s /bin/bash www-data
RUN usermod -s /bin/bash nagios

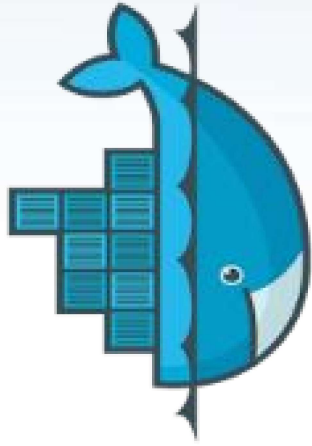
WORKDIR /
RUN git clone https://github.com/larsks/check_ganglia
WORKDIR /check_ganglia
RUN python setup.py install

#disable mail since nagios just spams
RUN ln -s /bin/true /bin/mail

COPY supervisor.conf /etc/supervisord.conf
CMD /usr/bin/supervisord -c /etc/supervisord.conf

```





docker

Example Docker Service

Nagios Makefile - Variables

```

DOCKER_TAG      ?= nagios
DEBRELEASE      ?= stretch
PKGCACHE        ?= 192.168.100.10:3142
IP              ?= 192.168.100.12
RUN_ARGS        ?= docker run -it
                --network="bridge-nat" --ip "$(IP)" --log-driver=syslog
                -v /dev/log:/dev/log -h $(DOCKER_TAG) --name
                $(DOCKER_TAG)
BUILD_ARGS      ?= --rm
    
```


Example Docker Service

Nagios Makefile - Build

build:

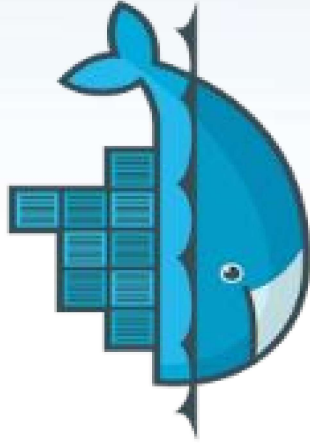
```
docker build $(BUILD_ARGS) \  
  --build-arg DEBRELEASE=$(DEBRELEASE) \  
  --build-arg PKGCACHE=$(PKGCACHE) \  
  --network=host -t $(DOCKER_TAG) .
```

stop:

```
docker kill $(DOCKER_TAG) || true  
docker stop $(DOCKER_TAG) || true  
docker rm $(DOCKER_TAG) || true
```

nocache:

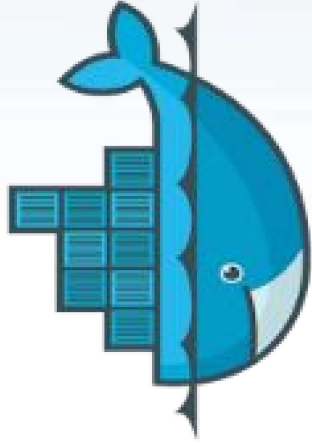
```
$(eval BUILD_ARGS := $(BUILD_ARGS))  
--no-cache)
```



docker

Example Docker Service

Nagios Makefile - Run



docker

run: build stop
`$(RUN_ARGS) -d --restart unless-stopped
 $(DOCKER_TAG)`

debug: build stop
`$(RUN_ARGS) --rm $(DOCKER_TAG)`

bash: build stop
`$(RUN_ARGS) --rm $(DOCKER_TAG) /bin/bash`