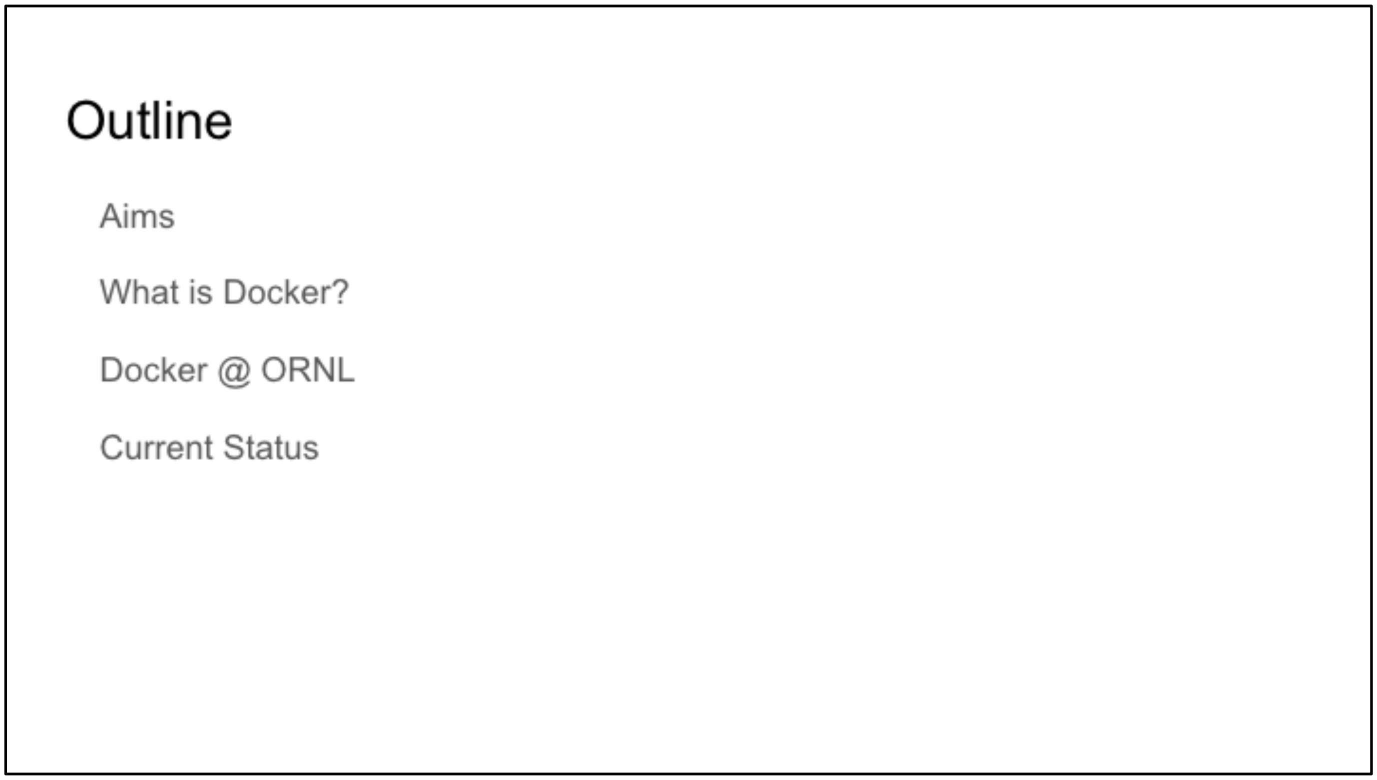
Docker + ICAT @ ORNL

Peter Parker
Software Developer, SNS and HFIR
Oak Ridge National Laboratory, TN, USA



Outline

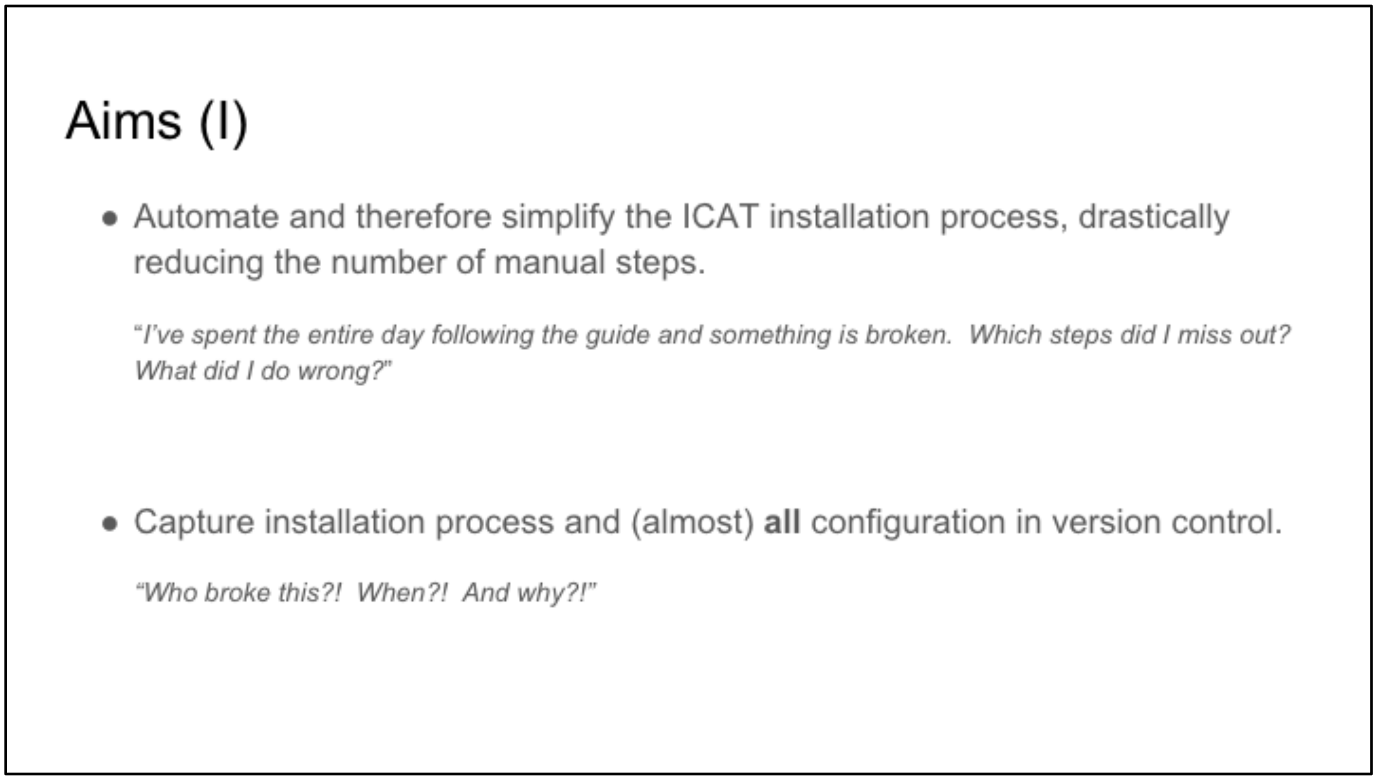
Aims

What is Docker?

Docker @ ORNL

Current Status

- Overview:
  - what specific aims we had at ORNL
  - revisit what Docker is and reiterate a couple of important points after Rolf's talk
  - a summary of how we've set up Docker at ORNL
  - a current status report
- Some cross-over with my poster, so hopefully this won't be too redundant



Aims (I)

- Automate and therefore simplify the ICAT installation process, drastically reducing the number of manual steps.

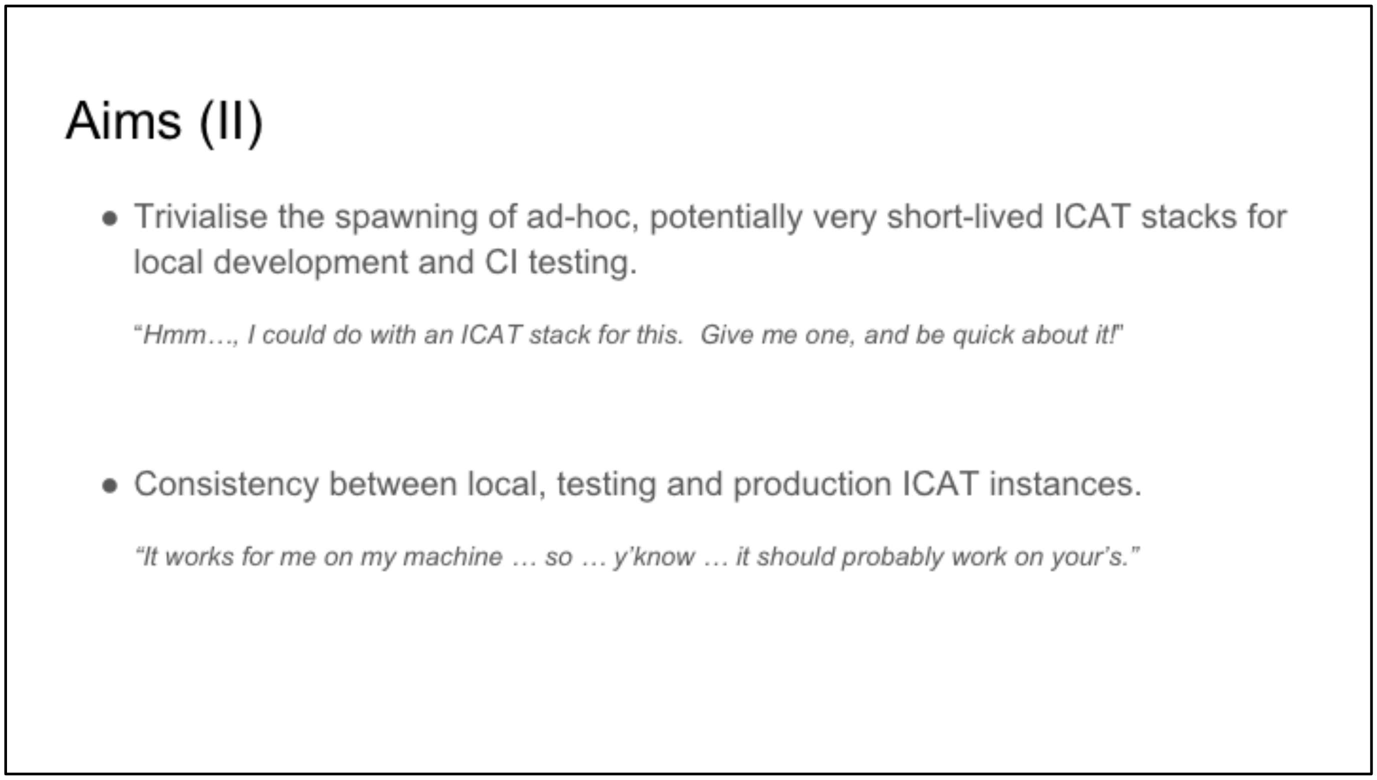
  "*I've spent the entire day following the guide and something is broken. Which steps did I miss out? What did I do wrong?*"

- Capture installation process and (almost) **all** configuration in version control.

  "Who broke this?! When?! And why?!"

1. Automate the installation of ICAT and cut down on manual steps since manual steps are error prone, and it's easy to get lost or lose track of where you are when there are lots of steps.

2. Expand the coverage of version control by capturing ICAT configuration in Git, but also the actual installation process of ICAT as well. This is a big deal, since I'm a software dev who writes code, and I accept nothing less than full version control with code so why accept anything less for configuration and installation?



Aims (II)

- Trivialise the spawning of ad-hoc, potentially very short-lived ICAT stacks for local development and CI testing.

  "*Hmm…, I could do with an ICAT stack for this. Give me one, and be quick about it!*"

- Consistency between local, testing and production ICAT instances.

  "It works for me on my machine … so … y'know … it should probably work on your's."

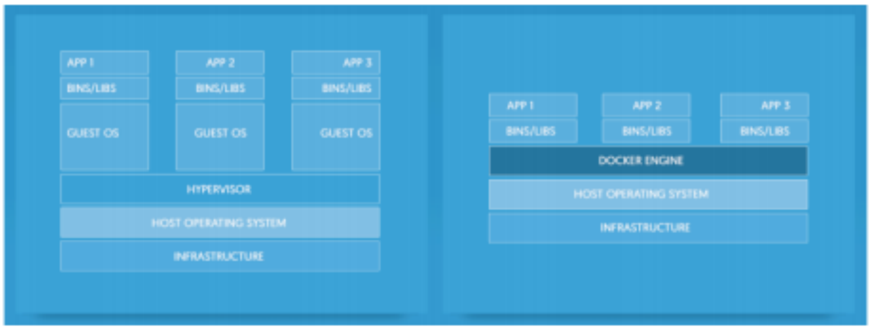
1. Make it (really) easy to spin up an instance of ICAT, so If I'm developing an application that uses ICAT, it would be nice to have:
    a. A local instance of ICAT sitting on my own machine so that I can run tests against it and have something to "program against" directly
    b. Disposable, short-lived instances of ICAT for CI.
2. Finally I'd like to have all instances of ICAT consistent with each other, **as far as possible**
    a. If I've spent time getting something configured locally, or on a test server, I want that work to count everywhere.

What is Docker?

(Roughly) analogous to Vagrant.

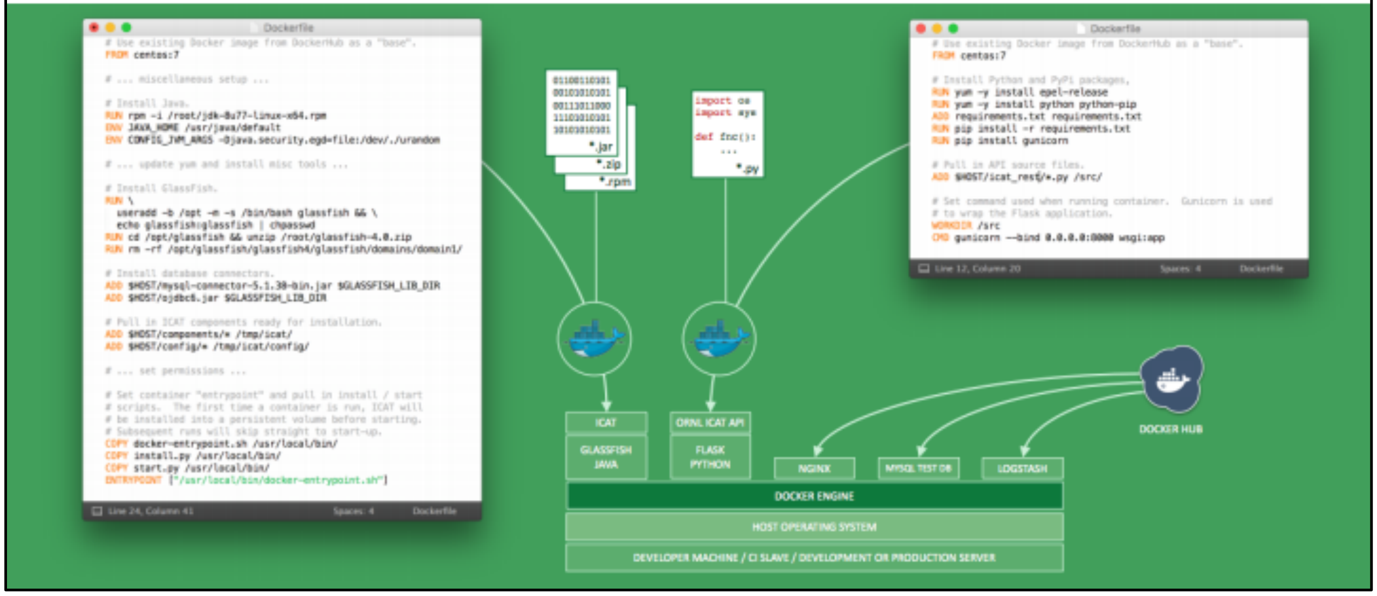
Uses *containers* instead of VMs.

Containers are *lightweight*.



1. Rolf covered this, but I want to reiterate the key point that...
    a. ...Docker containers are roughly analogous to virtual machines in that they allow applications to run inside isolated environments that you have complete control over
    b. **But** the difference is that they are extremely lightweight, so
        i. Can be spun up in fractions of a second
        ii. Don’t impact performance
2. Image from poster shows VMs on the left, containers on the right:
    a. VMs have to carry an entire OS around with them, which is a lot of baggage
    b. Containers share a lot of their OS with the host, which is already running

Docker @ ORNL (I)



```
# Use existing Docker image from DockerHub as a "base".
FROM centos:7

# ... miscellaneous setup ...

# Install Java.
RUN rpm -i /root/jdk-8u77-linux-x64.rpm
ENV JAVA_HOME /usr/java/default
ENV CONFIG_JVM_ARGS -Djava.security.egd=file:/dev/./urandom

# ... update yum and install misc tools ...

# Install GlassFish.
RUN \
  useradd -b /opt -m -s /bin/bash glassfish && \
  echo glassfish:glassfish | chpasswd
RUN cd /opt/glassfish && unzip /root/glassfish-4.0.zip
RUN rm -rf /opt/glassfish/glassfish4/glassfish/domains/domain1/

# Install database connectors.
ADD $HOST/mysql-connector-5.1.38-bin.jar $GLASSFISH_LIB_DIR
ADD $HOST/ojdbc6.jar $GLASSFISH_LIB_DIR

# Pull in ICAT components ready for installation.
ADD $HOST/components/* /tmp/icat/
ADD $HOST/config/* /tmp/icat/config/

# ... set permissions ...

# Set container "entrypoint" and pull in install / start
# scripts.  The first time a container is run, ICAT will
# be installed into a persistent volume before starting.
# Subsequent runs will skip straight to start-up.
COPY docker-entrypoint.sh /usr/local/bin/
COPY install.py /usr/local/bin/
COPY start.py /usr/local/bin/
ENTRYPOINT ["/usr/local/bin/docker-entrypoint.sh"]
```

```
# Use existing Docker image from DockerHub as a "base".
FROM centos:7

# Install Python and PyPi packages.
RUN yum -y install epel-release
RUN yum -y install python python-pip
ADD requirements.txt requirements.txt
RUN pip install -r requirements.txt
RUN pip install gunicorn

# Pull in API source files.
ADD $HOST/icat_rest/*.py /src/

# Set command used when running container.  Gunicorn is used
# to wrap the Flask application.
WORKDIR /src
CMD gunicorn --bind 0.0.0.0:8000 wsgi:app
```

Another image taken from my poster that shows the basic components of our current ICAT stack and what goes into producing it.

At the bottom is the general layout, with containers for:

- The ICAT server
- An API we've set up that wraps the ICAT REST API (more about that later in the site report)
- NGINX
- MySQL database used for test instances of ICAT stacks
- Logstash for parsing logs and sending them to graylog.

And they are a product or function of all the stuff at the top:

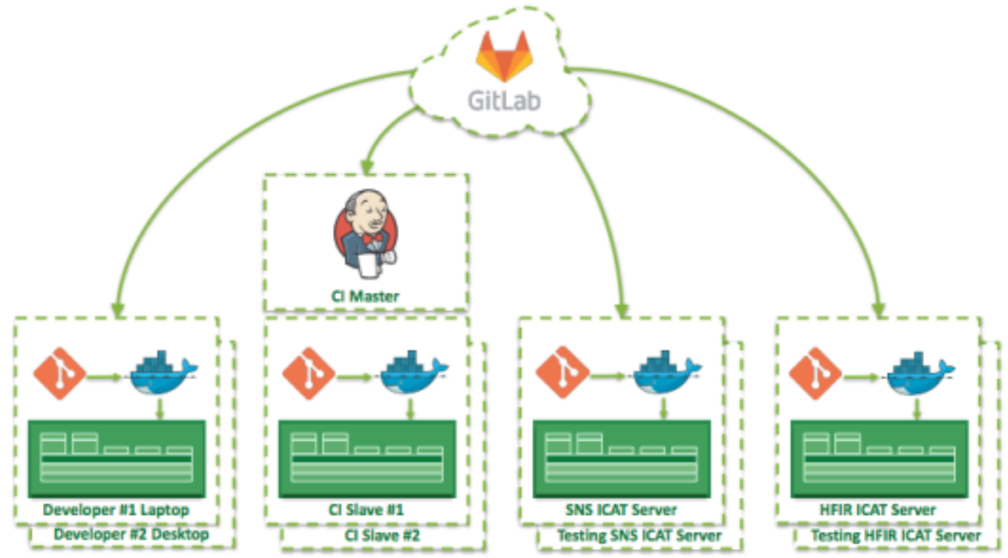
- Requirements and dependencies:
  - Java
  - Glassfish
  - DB connectors
- Zipped-up ICAT components
- Location-specific (situation-specific) configuration
- Credentials

… Which are all pulled together by the Dockerfile to create an **image** which is run inside of a **container**.

One thing I hope is clear is ICAT stacks are now completely disposable -- they're a function of everything in version control!

No more worrying about whether or not you can easily reproduce what you have once you’ve gotten it working.

Docker @ ORNL (II)



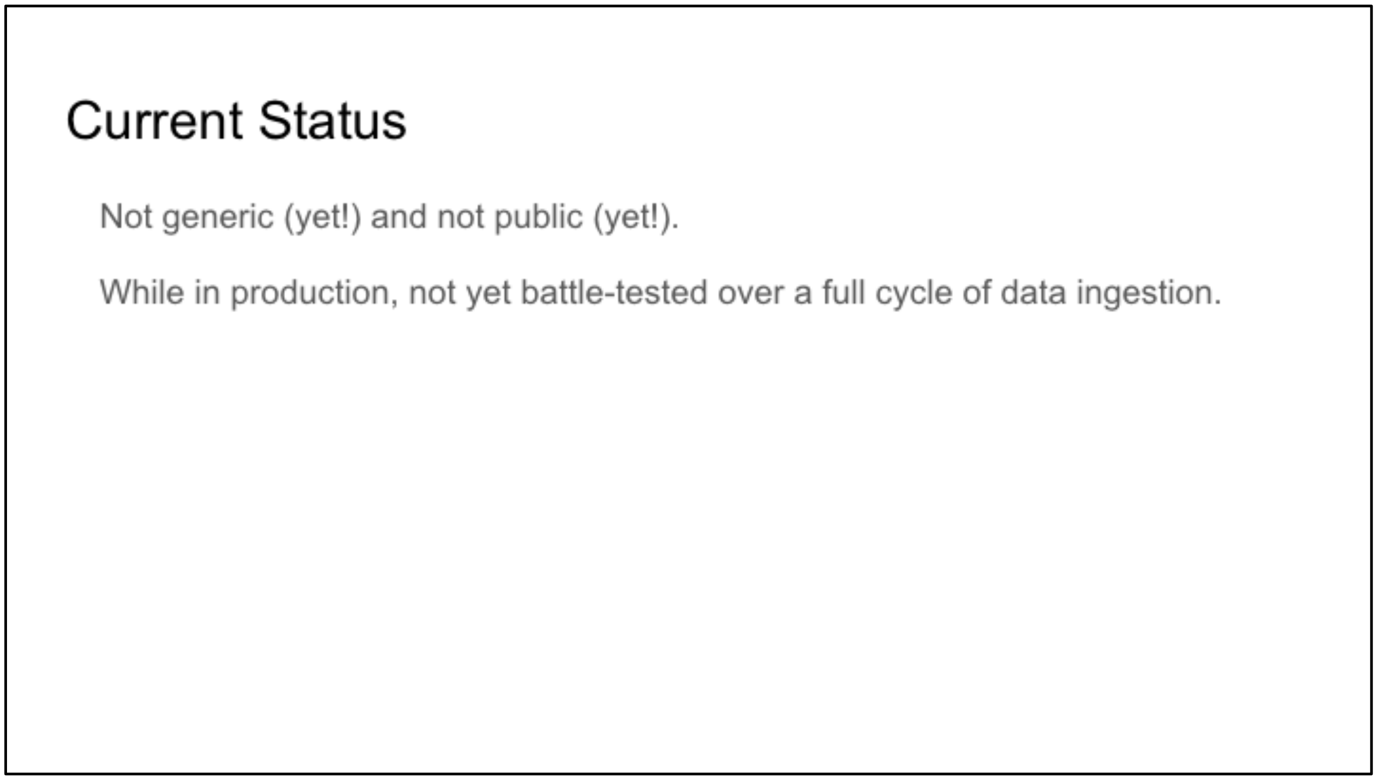
Yet another image from my poster showing an overview of our **workflow** for ICAT-related development / deployment

Starts on the left where a developer wants a local instance of ICAT:
- They pull down the code for the first time using git
  - Then one or two commands away from building and deploying an ICAT stack
  - A trivial step!
- They can make any changes they want
  - Changes can be things like updated dependencies or ICAT modules, changed configuration, a different setup
  - and crucially, test them locally and see the result of their changes quickly
- Changes are commited to Git and then pushed up to GitLab

Changes are then pulled down automatically by Jenkins and then all automated tests are run:
- Note that these tests cover everything! They cover installation, configuration, make sure everything is working together correctly.
- Made possible by the complete automation process.

Once we’re happy with everything and ready to deploy, we can then SSH onto the production boxes, then:
- Completely delete the ICAT stack.
- And then deploy a brand new ICAT stack.

This is stress-free and should -- at least in theory -- be very straight forward since the process is automated.



Current Status

Not generic (yet!) and not public (yet!).

While in production, not yet battle-tested over a full cycle of data ingestion.

- At the moment our Docker setup is for SNS and HFIR only.
    - In principle this could be made more generic, but would take some work.
        - We'd have to extract the ICAT server part of things (not logstash, nginx, etc)
        - Perhaps a collaboration with Rolf would be a good idea
- The code is not public yet. It's hosted on a local GitLab instance at the lab.
    - Credentials aren't in the repo, but ORNL-specific stuff is.
- Disclaimer for the sake of honesty:
    - While we do have this setup deployed to production, it is not currently doing anything and has not been battle tested.
    - It **has**, however got a test suite to make sure things are fitting together as we expect.

Thanks!