Grouper Packaging and Versioning

Grouper 2.5 requires running in the InCommon Trusted Access Platform (ITAP) container.

- Grouper has a consistent directory structure, the correct version of libraries, successful and correct upgrades
- Run your container deployment with orchestration however you like etc.
  - However, we do want people to run things in a high maturity level, but its not required for use of the container
- Institutions running Grouper will install/deploy/upgrade with the ITAP container, not from the installer. **Patches do not exist anymore for 2.5**
  - The installer has a command to get started with the container
- No more tarballs for Grouper
  - Jars for Grouper are in Maven. The same jars in the container are from Maven.

Scroll down on this page for details on container maturity levels

Child pages:

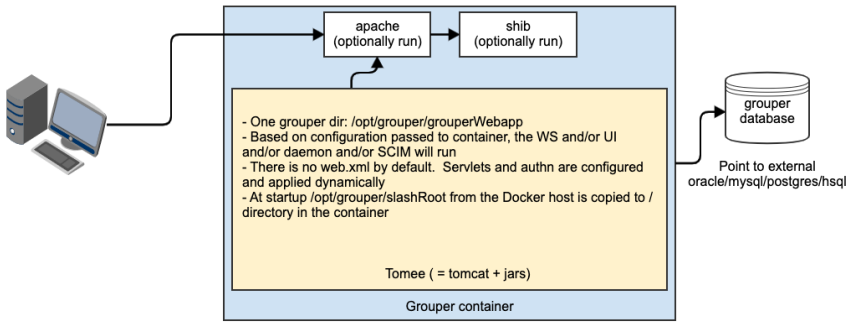
Versioning

The version of Grouper is 2.5.X (X is a number. not patch level for API/UI/etc). Grouper jars are built and released in maven central, and the container is built from those jars and released to dockerhub. Once some testing has occurred it is announced on slack and on the release notes page as "released". Do not use a new container until it is announced as "released". The codebase and container are immutable after announced as released. After a couple institutions are using the container version in production, or after a few days have passed and no show-stopping errors have been reported, the container version will be marked as "stable" on the release notes wiki. The latest stable release is highlighted and is the one people should upgrade to if upgrading. If you use a release that is not marked as "stable", please let the Grouper team know how your experience is so we have data points on when things are stable.

Container

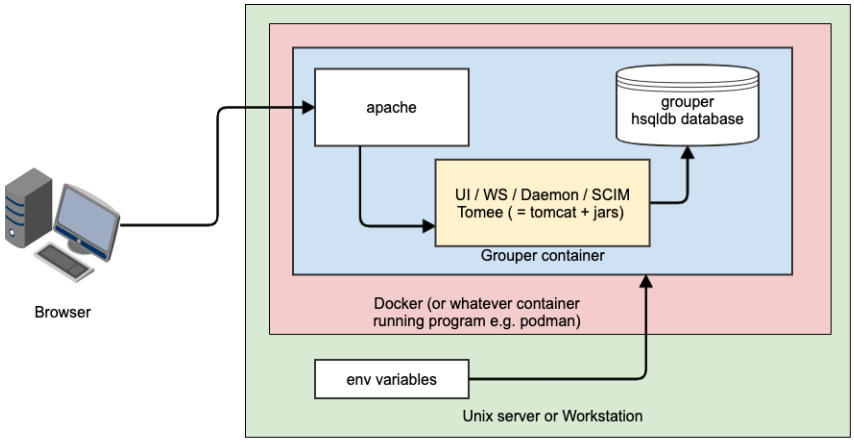
The ITAP container for 2.5 has some changes compared to 2.4.

Everything in Grouper is now in one directory. The daemon runs from the webapp. You should run either UI or WS or daemon or GSH or SCIM, but you can run more than one or all at once if you like



Container maturity level -1

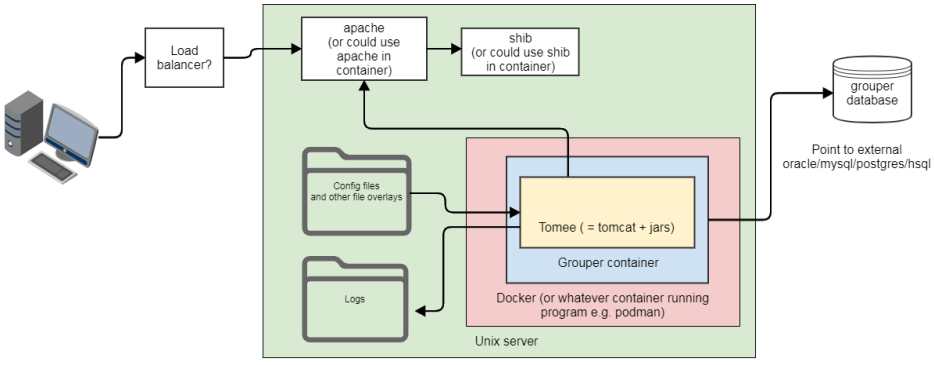
To get Grouper up and running to be able to log in and create groups, you can use the "quickstart" maturity level -1. This is using the container in way that just starts and runs. It prioritizes convenience over security or the proper way to run things. This should never be run in an environment with sensitive data. You can evolve from this setup to a higher maturity level.



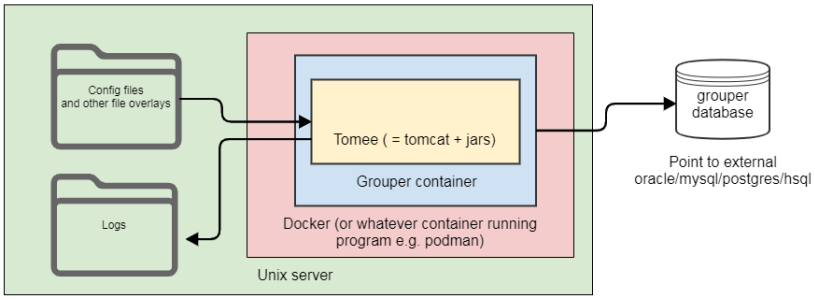
Container maturity level 0

If an institution does not have experience with running containers, Grouper can easily be run without orchestration or other best practices of using containers. The institution can run the Grouper container before developing their container standards and practices

Note, in production they would run this setup (or multiple) for the UI, run another instance (or multiple) for WS. This is an example webapp



This is an example daemon maturity level 0. In each maturity level, the daemon is the same as the webapp (ui or ws), just doesnt need web requests. Note there is no HTTP here, though you could use it with the status servlet if you like....

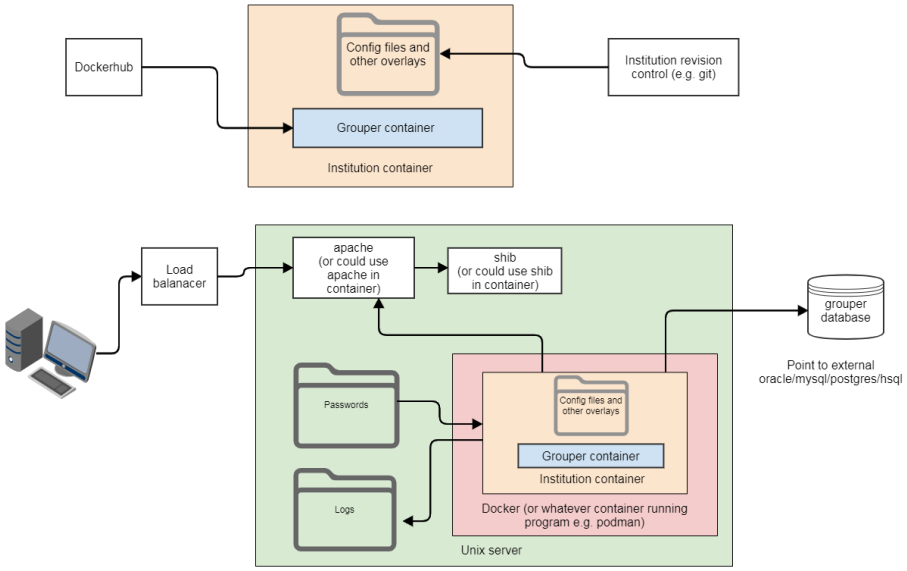


Getting started with Grouper, use the installer: task: installContainer

1. Walks through setting up the database
2. Sets up config files on server
3. Configures logging on server
4. Verifies and advises on installing docker / podman
5. Creates a start script
6. Ensures there is a service script so Grouper starts on server startup
7. Helps with networking and ports
8. Documentation on how to SSH into container
9. Advises about containers for WS/UI/daemon/GSH

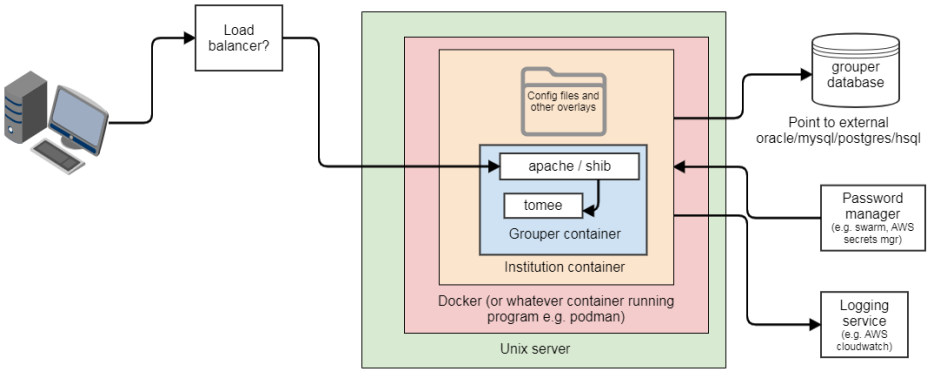
Container maturity level 1

Keep your configs in source control, have an institution Dockerfile and sub-container



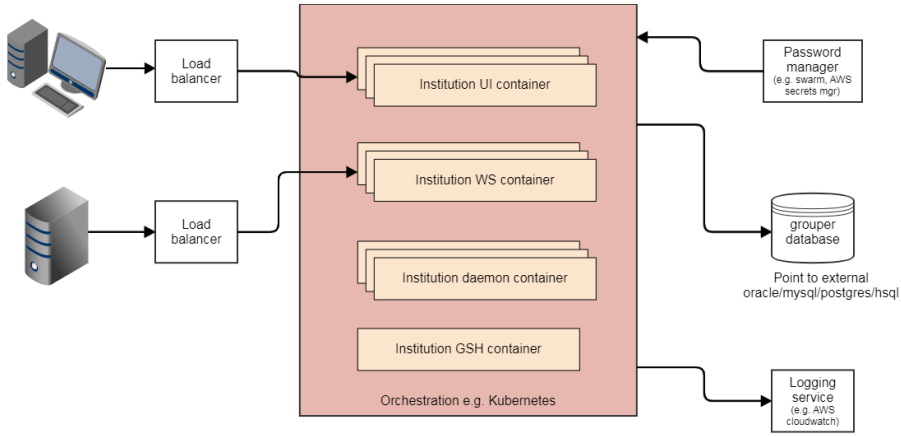
Container maturity level 2

Make the container not dependent on server that runs it. In this case we need a solution for passwords and logs



Container maturity level 3

Use container orchestration



Architecture

Grouper runs in tomcat in a container. The UI, or WS, or daemon run in tomcat in the container. If the daemon is the only process running, then there are no servlets accepting traffic but tomcat still runs the daemon. It is important that only one process is running whether it is the UI, WS, and/or daemon (or a combination of them, though not recommended in production).

Grouper determines what is running by environment variables in the container. Based on that servlets will be dynamically registered in CommonServletContainerInitialization.java. In the UI or WS the URLs map to logic through one servlet.

Design

Design goals

- Support various levels of maturity for containers (including institutions with no experience)
- Give guidance on how to evolve containerization so best practices are followed
- Support a quick start
- Share experience / configuration with various
    - Add deployment recipes for various public cloud infra
- Provide a very easy path to go from tomcat or tomcat/apache to ITAP container

Changes

- Only support containerized deployments
- Grouper in 2.5 will live in one directory in one container: /opt/grouper/grouperWebapp
    - In Grouper 2.4 there were 4 directories: ws, ui, daemon, scim
    - All code and libraries for all features of grouper will be in this one place
- A switch by command line or grouper-hibernate.properties will tell the container if it is a UI/WS/daemon/GSH, or it can be multiple
    - You can fire up GSH from any container
- The grouper hibernate and morph params can be passed from env variable or params to container

        - db url, db user, db pass, morph pass
        - the passes can be paths of external files
        - the db pass can be encrypted
        - the entire "conf" directory can be mounted externally
        - the log directory can me mounted outside the container
        - any overlay can be attached to the webapp from /opt/grouper/shashRoot
            - for instance, to overlay the grouper.properties, put a file in /opt/grouper/slashRoot/opt/grouper/grouperWebapp/WEB-INF/classes/grouper.properties
- So... to convert from tomcat to container:
    - Install docker
    - Run the ITAP base container with no changes and no dockerfile
    - Pass in if WS/UI/daemon
    - Either put configs in the database or external mount the conf dir
    - Configure logging to log to mounted external file. Do this by mounting the conf dir, and configure the log4j.properties to log to external
    - For the bootstrap (DB config), pass in or keep in mounted external files the db and morph creds (passwords should be in files)
    - Map the ports so everything works
        - Could be the apache port or tomcat port
- After that, the institution can at their own pace
    - Use a password manager
    - Put logs somewhere else besides host server
    - Github/gitlab
    - Continuous integration
    - Use orchestration
    - Run in cloud
    - Whatever else we recommend

Tarballs

We wont have tarballs anymore, just building the TAP image in ant.

The downloads site will need some logic to coordinate the latest version of the client/installer