Production Release

Introduction

This page and sub-pages contain technical information for deploying the USxS Production Releases.   The instructions here assume a basic familiarity with Docker.

Requirement

In order to successfully deploy the PR release, you will need:

- Minimum Server (VM) requirements:
    - Memory: 4GB per district + 1GB for docker and OS overhead
    - Disk space: 2GB per district. This allows for database, backups, logs, etc.
    - CPU: 4 cores (8 cores preferred)
- Host running:
    - Docker Community Edition - version 17.06-ce or higher: https://docs.docker.com/engine/installation/
    - Docker Compose - version 14.0 or higher: https://docs.docker.com/compose/install/

SSDT-Utils

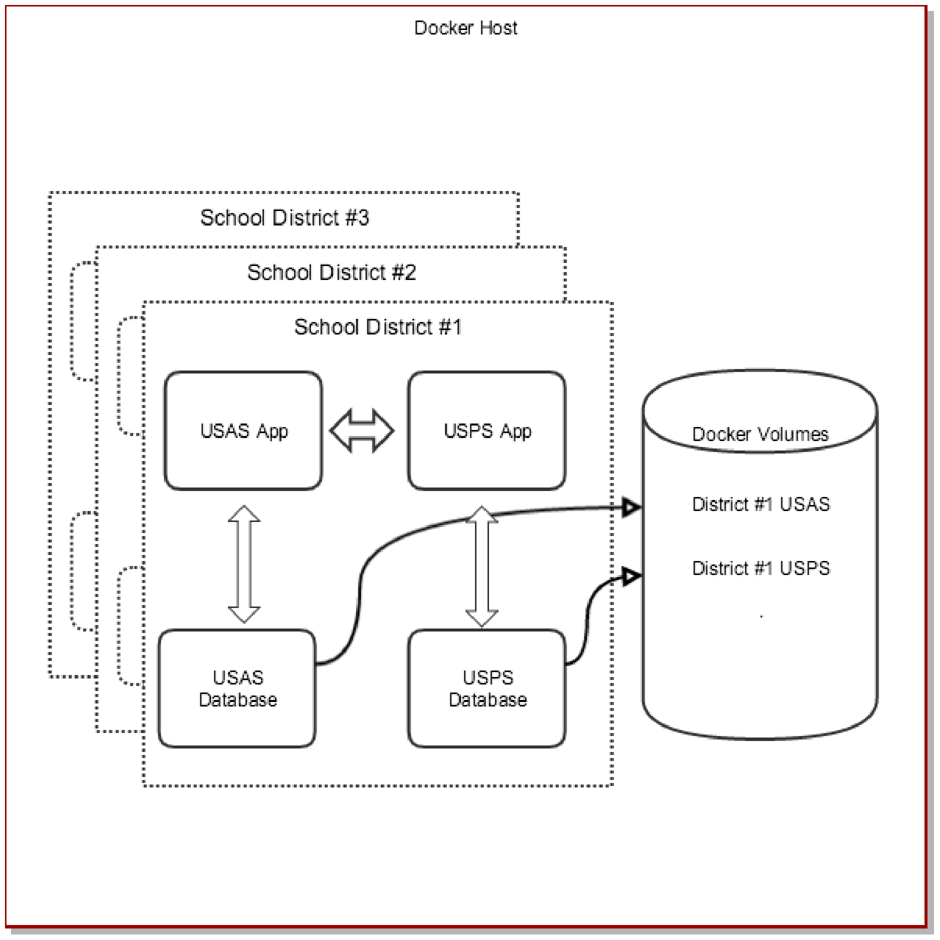
The SSDT has prepared a "utils" image which contains (or will contain):

- template compose files and setup scripts for SSDT applications
- Useful scripts for maintaining services and containers containing SSDT applications
- Dockerfile's for building the `ssdt-postgres` and `ssdt-tomcat` images.

Examples in this wiki will assume you have the SSDT-Utils installed on your docker host. See Install and Update SSDT Utils package for details.

USxS Architecture as Containers

The USxS Redesign systems are typical two-tier single tenant applications.   Each school district will have it's own instance including a Tomcat deployment and a Postgres database server. The convention for Docker is to have each "container" model a single process. Therefore, there will be at least two docker containers. One running Tomcat with the USxS application and a second container running the Postgres database server.   These two containers must be orchestrated to start and stop together.



Containers

The application and database containers are based on docker images provided by the SSDT. They contain the application code and database software. They also contain temporary data (such as log and work files). They do not contain any persistent district or user data. Therefore, they are "ephmeral" and can be deleted and recreated at any time without loss of information.

Volumes

Each district will have two docker volumes which contain the postgres database files. These are named: `{project}_usasdata` and `{project}_uspsdata`.   Because the volumes contain the actual databases, they are very important. The volumes can not be deleted or moved without lost of district data.   By default, the volumes will use docker's "local" volume driver. This means that the district's volumes will be mounted onto the docker hosts file system (under `/var/docker/volumes/` by default).

These docker volume directories can be included in the host's regular backup. However, if the database server is running, it is not possible to backup and restore the raw database files safely. If file system backup occurs while there are active transactions occuring in the database, then a restored database may be inconsistent or corrupt. Therefore, database backups should be performed with SSDT supplied scripts which use postgres to create a consistent snapshot backup. See Backup USxS Databases

Host Directory Structure

On your docker host, you will need to establish a directory structure for the individual district configurations. Each district will need a separate directory.

For example, here is a how the structure may look for some of NWOCA's districts:

```
- /data
   -> /prod
      -> hicksville
      -> holgate
      -> sampletown
      -> sylvania
        .
        .
        .
```

Each district directory will contain:

- `docker-compose.yml` file(s) defining the services (USAS , USPS or both) for the district
- a hidden `.env` file containing environment variables

The directory containing the `docker-compose.yml` is referred to as the "*project*" directory. The name of this directory is important because it will affect the default name of the containers and volumns created by the project.

Because these directories contain important configuration and database backup files, you should including them in the host's nightly backup.

See the specific application pages for how to configure each district's `docker-compose` files.

Setting up a District Instance

The SSDT Utils contains a directory called `/ssdt/prod` which contains template docker files and a `setup.sh` script to assist in configuring the docker compose configuration files. The `setup.sh` scripts performs the following:

- creates a `docker-compose.yml` which defines:
    - Service configurations for USAS, USPS or both
    - Networking between both applications and databases
    - Volume definitions for each database
    - API Keys for USAS USPS integration
    - Unique passwords for district database passwords
- creates a `.env` file to store passwords and API keys
- create a `.docker-compose.md5` file to allow verification that the `docker-compose.yml` has not been modified.

Each time the `setup.sh` script is executed, it creates a new `docker-compose.yml` file containing the latest mandatory configuration from the SSDT. Therefore, the ITC should never edit this file. Instead, the ITC should use a `docker-compose.override.yml` to provide additional configuration values or override SSDT provided values.

The `.env` file contains generated keys and passwords what will be unique for each district. It is very important that this file be preserved and the values not changed after the applications are launched. For example, when the databases are launched for the first time the database password will be set to the value in the `.env` file. If the database password is lost, then the applications will lose access to the database (requiring manual resetting of the database password).

However, the ITC may edit the `.env` file to add additional environment variables to override the SSDT default values.

Executing setup.sh

Execute the `setup.sh` script to create a new district instance as follows:

```
> cd /data/prod
> mkdir sampletown
> cd sampletown
> /ssdt/prod/setup.sh
Preparing 'sampletown' with default USxS configuration
Enter project name: <sampletown>
set project name in .env file
created docker-compose.yml
Generate USAS and USPS integration config? <Y/n>
Created integration API keys.  Enable integration modules after applications start
Review or create a docker-compose.override.yml for custom settings.
```

Please note the following:

- The project name defaults to the name of the current directory. The project name affects the names of volumes and networks when the project is started.
- By default, recommended, the script will automatically create API keys needed by the USAS/USPS Integration modules.
- You may execute the script multiple times. it will recreate the `docker-compose.yml` and leave any existing `.env` values unchanged.

Generated Sampletown Configuration

Below are the files created by the setup script for Sampletown:

docker-compose.yml

```
# **** DO NOT MODIFY THIS FILE. ****
# Place customizations in docker-compose.override.yml and .env
version: "3.3"
services:
  usasdb:
    restart: unless-stopped
    image: docker.ssdt.io/ssdt-postgres:5
    volumes:
      - usasdata:/var/lib/postgresql/data
    networks:
      - default
    environment:
      - DB_NAME=usasdb
      - DB_USER=usas
      - DB_PASS=${USAS_DB_PASSWORD:-usasdefault}
  usasapp:
    restart: unless-stopped
    image: docker.ssdt.io/usas-app:${USAS_TAG:-prod}
    depends_on:
      - usasdb
    networks:
      - default
    environment:
      - DB_HOST=usasdb
      - DB_NAME=usasdb
      - DB_USER=usas
      - DB_PASS=${USAS_DB_PASSWORD:-usasdefault}
      - USAS_MODULE_USPSINTEGRATION_CONFIGURATION_USPSCONFIGURATION_CLIENTHOST=uspsapp
      - USAS_MODULE_USPSINTEGRATION_CONFIGURATION_USPSCONFIGURATION_CLIENTDNSLOOKUP=true
      - USAS_MODULE_USPSINTEGRATION_CONFIGURATION_USPSCONFIGURATION_SERVERHOST=0.0.0.0
      - USAS_MODULE_USPSINTEGRATION_CONFIGURATION_USPSCONFIGURATION_SERVERHOSTDNSLOOKUP=false
      - USAS_MODULE_USPSINTEGRATION_CONFIGURATION_USPSCONFIGURATION_APPLICATIONID=${USAS_APPLICATIONID:- }
      - USAS_MODULE_USPSINTEGRATION_CONFIGURATION_USPSCONFIGURATION_APIKEY=${USAS_APIKEY:- }
      - USAS_MODULE_USPSINTEGRATION_CONFIGURATION_USPSCONFIGURATION_REMOTEAPPLICATIONID=${USPS_APPLICATIONID:- }
      - USAS_MODULE_USPSINTEGRATION_CONFIGURATION_USPSCONFIGURATION_REMOTEAPIKEY=${USPS_APIKEY:- }
  uspsdb:
    restart: unless-stopped
    image: docker.ssdt.io/ssdt-postgres:4
    volumes:
      - uspsdata:/var/lib/postgresql/data
    networks:
      - default
    environment:
      - DB_NAME=uspsdb
      - DB_USER=usps
      - DB_PASS=${USPS_DB_PASSWORD:-uspsdefault}
  uspsapp:
    restart: unless-stopped
    image: docker.ssdt.io/usps-app:${USPS_TAG:-prod}
    depends_on:
      - uspsdb
    networks:
      - default
    environment:
      - DB_HOST=uspsdb
      - DB_NAME=uspsdb
      - DB_USER=usps
      - DB_PASS=${USPS_DB_PASSWORD:-uspsdefault}
      - USPS_MODULE_USASINTEGRATION_CONFIGURATION_USASCONFIGURATION_CLIENTHOST=usasapp
      - USPS_MODULE_USASINTEGRATION_CONFIGURATION_USASCONFIGURATION_CLIENTDNSLOOKUP=true
      - USPS_MODULE_USASINTEGRATION_CONFIGURATION_USASCONFIGURATION_SERVERHOST=0.0.0.0
      - USPS_MODULE_USASINTEGRATION_CONFIGURATION_USASCONFIGURATION_SERVERHOSTDNSLOOKUP=false
      - USPS_MODULE_USASINTEGRATION_CONFIGURATION_USASCONFIGURATION_APPLICATIONID=${USPS_APPLICATIONID:- }
      - USPS_MODULE_USASINTEGRATION_CONFIGURATION_USASCONFIGURATION_APIKEY=${USPS_APIKEY:- }
      - USPS_MODULE_USASINTEGRATION_CONFIGURATION_USASCONFIGURATION_REMOTEAPPLICATIONID=${USAS_APPLICATIONID:- }
      - USPS_MODULE_USASINTEGRATION_CONFIGURATION_USASCONFIGURATION_REMOTEAPIKEY=${USAS_APIKEY:- }
volumes:
  usasdata:
  uspsdata:
```

.env

```
COMPOSE_PROJECT_NAME=sampletown
USAS_DB_PASSWORD=NjE3ZmI1ZDU3YjFiYTYxMjYzMjAyMTlm
USPS_DB_PASSWORD=MzRjNzY1MDJiZDVjMWFmNDIwZDRmY2Jm
USAS_APPLICATIONID=sampletown-usas
USAS_APIKEY=224fb9ededa136c0781ce93dce522fdbc486c20e80e8c5035ae29e7d816f4a6a
USPS_APPLICATIONID=sampletown-usps
USPS_APIKEY=38aa789299090baed480085af9331b9c1dcb0638d9d52768aa68cfaf4cef3262
```

Configuration Explanation

The generated configuration leverages docker-compose's default behavior to create valid instances of both applications with persistent database storage and inter-application communication. The configuration defines the following by default for the `sampletown` project:

- Unique database passwords for each database
- Unique API keys for integration for between USAS and USPS. These will be configured automatically in each application when the integration modules are enabled.
- On the docker engine (when the project is first deployed):
    - network named `sampletown_default`. This network will contain all four containers and allows the applications to communicate with the databases and each other.
    - two volumes named `sampletown_usasdata` and `sampletown_uspsdata`. This will contain the database files for each database. They will use the default 'local' driver (i.e. mounted on the hosts file system)

The configuration is structured to permit flexible customization by the ITC.   For example, the ITC may wish to use a different volume driver to store database data on a remote file share.

As mentioned above, ITC's **should not modify** the generated `docker-compose.yml` file directly. The SSDT may release improvements to the setup scripts which which re-create the file. Instead, the ITC should create a `docker-compose.override.yml` and place all local customizations there.

Additional Configuration

The generated configuration is sufficient to deploy and execute the USAS and USPS applications. However, each district will need some customization. At a minumum, the applications HTTP ports must be made accessible.

Exposing Ports

The generated configuration does not expose any ports and the applications do not provide encrypted ports. Each hosting site must decide how to expose the ports and provide an encrypted connection (HTTPS). The two general solutions are:

- Expose ports on the docker engine and use an external reverse-proxy
- Use an nginx-proxy to automatically reverse proxy for containers on the host

Using an External Reverse Proxy

If the ITC uses a standalone system for reverse proxy (e.g. a KEMP appliance), then the application ports can be exposed as unique ports on the docker engine and those ports configured in the reverse proxy. In this configuration, the ITC would create a `docker-compose.override.yml` like this to expose the application ports:

```
version: "3.3"
services:
  usasapp:
    ports:
      - "8100:8080"
  uspsapp:
    ports:
      - "8101:8080"
```

In the above example, USAS would be exposed on the docker host's port 8100 and USPS on port 8101. **It would be the ITC's responsibility to assign a unique port to each district and application**.

The ITC would then use the reverse proxy (likely with a wildcard certificate) to reverse proxy a domain name to these ports.

In this configuration, it's important not to expose the docker engine out side the firewall to the public internet. Users should only be able to access the application through the proxy's HTTPS port.

Auto Proxying with NGINX

See Using nginx-proxy for instructions on configuring an nginx web server with HTTPS to provide a reverse proxy on the docker engine. Once the proxy is established, then each district needs to be configured with environment variables to define the applications host name. This example configures sampletown to use nginx-proxy with LetsEncrypt for the certificate:

```
version: "3.3"
services:
  usasapp:
      networks:
        - proxy
      environment:
        - VIRTUAL_HOST=${COMPOSE_PROJECT_NAME}-usas.demo2.ssdt.io
        - VIRTUAL_PORT=8080
        - LETSENCRYPT_HOST=${COMPOSE_PROJECT_NAME}-usas.demo2.ssdt.io
        - LETSENCRYPT_EMAIL=hostmaster@example.org
  uspsapp:
      networks:
        - proxy
      environment:
        - VIRTUAL_HOST=${COMPOSE_PROJECT_NAME}-usps.demo2.ssdt.io
        - VIRTUAL_PORT=8080
        - LETSENCRYPT_HOST=${COMPOSE_PROJECT_NAME}-usps.demo2.ssdt.io
        - LETSENCRYPT_EMAIL=hostmaster@example.org
networks:
   proxy:
     external:
        name: proxy_default
```

This configuration does several things:

- It joins the applications to the "proxy" network. This allows the nginx container to access the application ports.
- Defines the `VIRTUAL_*` environment variables for nginx-proxy
- Defines the `LETSENCRYP_*` environment variables for LetsEncrypt (omit these if using a signed cert)
- Defines the "proxy" network as an external network (defined outside this project) as being the "`proxy_default`" network. The "`proxy_default`" is the network defined by the nginx-proxy project.

Notice that the file uses the COMPOSE_PROJECT_NAME environment variable to define the host names. If you follow the convention of using the project name in the hostname, then you can use this file as a template for each new district.

Application Configuration

Several other post-deployment configuration step may be necessary. These are detailed below.

Set Admin user password

By default, the special "*admin*" user profile will be created automatically the first time the application is started. **You must assign a unique admin password before any production data is entered or imported into the system.**

The initial password is assigned by defining the following environment variable in the usXsapp section of the `docker-compose.yml` or `docker-compose.override.yml`

Alternatively, the password can be set at application startup by adding these lines to the `environment` key of both `usasapp` and `uspsapp` services:

Once the application has been started, the environment variable above must be removed from the docker-compose.override file.

```
environment:
   - APPLICATION_ADMIN_PASSWORD=desiredpassword
   - APPLICATION_ADMIN_RESET=true
```

The above lines will cause the "admin" password to be reset each time the application is started. See Authentication and Authorization for more information. **If this property is used to recover access to the application, it must be removed immediately after the next startup.**

USAS/USPS Integration Setup

The docker compose files automatically define the configuration needed enable two-way integration between USPS-R and USAS-R. The final step to enable the integration is to login to each application and enable the appropriate integration module.

Login to each application and navigate to the **System->Modules** view . In USAS enable the "`USPS Integration Module`" and in USPS enable the "`USAS Integration Module`". Click the "Click here to reload" in each app to reload the menu. With the modules enabled, there will be an additional menu option to control the integration. In each application, use the integration menu to "Test Connection" option to verify connectivity.

A minor bug in the "Test Connection" option requires you to click the "Test Connection" button twice to see a result.

JVM Configuration

The memory for the jvm's in the containers may need to be adjusted for usasapp and/or uspsapp service(s). This is done via an environment variable in the docker-compose.override.yml file. This environment variable is set per service, so usasapp and uspsapp could have different settings. FYI, Xmx is the maximum heap size. Another setting is Xms, which is the initial java heap size.

The current recommendation is 2-4G, depending on the size of the district. This example shows how to set the maximum heap size to 2GB.

```
environment:
   - JAVA_OPTS=-Xmx2g
```

DB Pool Max Configuration

The default db connection pool max for both usasapp and uspsapp is 20. Connection pools are used for logging in, running reports, etc. A user logging in temporarily needs a connection to the db pool. If many users are running reports (consuming connection pools), and a different user tries to connect, if no pool is available there will be a delay (up to 60 seconds) in the connection. If the site is having issues, we recommend doubling the db pool max size to 40. This is done with an environment setting in the docker-compose.override.yml file. When active connections reaches max connections, the application has run out. See Using System Monitor for information on how to see information about the pool connections and current db pool max setting. This example shows how to set the maximum db connection pool size to 40. Note that like other application environment settings, this is done per application and the application needs to be restarted in order for the change to take effect.

```
environment:
   - DB_POOL_MAX=40
```