incubator-sdap-nexus Documentation

Release 1.0.0-SNAPSHOT

Apache SDAP

Oct 10, 2019

Contents:

Warning: Apache incubator-sdap-nexus is an effort undergoing incubation at The Apache Software Foundation (ASF), sponsored by the name of Apache TLP sponsor. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.

Warning: Apache incubator-sdap-nexus is an effort undergoing incubation at The Apache Software Foundation (ASF), sponsored by the name of Apache TLP sponsor. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.

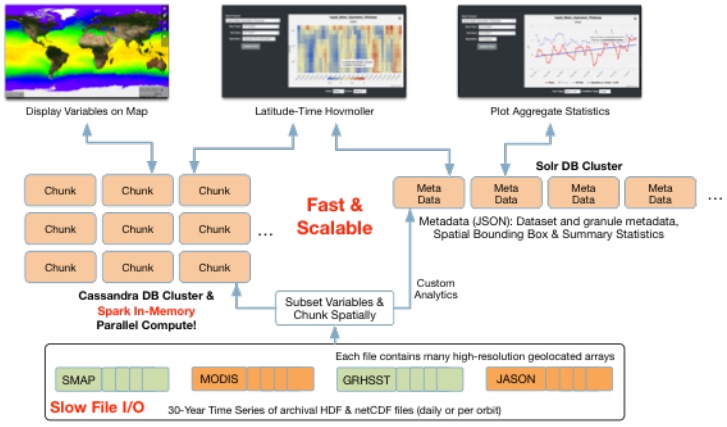
CHAPTER 1

About NEXUS

NEXUS is a data-intensive analysis solution using a new approach for handling science data to enable large-scale data analysis.

It supports a number of algorithms out of the box:

- Time Series
- Latitude/Time Hovmöller
- Longitude/Time Hovmöller
- Latitude/Longitude Time Average
- Area Averaged Time Series
- Time Averaged Map
- Climatological Map
- Correlation Map
- Daily Difference Average



Check out the *Quickstart Guide*.

CHAPTER 2

Quickstart Guide

This quickstart will take approximately 45 minutes to complete.

2.1 Introduction

NEXUS is a collection of software that enables the analysis of scientific data. In order to achieve fast analysis, NEXUS takes the approach of breaking apart, or "tiling", the original data into smaller tiles for storage. Metadata about each tile is stored in a fast searchable index with a pointer to the original data array. When an analysis is requested, the necessary tiles are looked up in the index and then the data for only those tiles is loaded for processing.

This quickstart guide will walk you through how to install and run NEXUS on your laptop. By the end of this quickstart, you should be able to run a time series analysis for one month of sea surface temperature data and plot the result.

2.2 Prerequisites

- Docker (tested on v18.03.1-ce)
- Internet Connection
- bash
- cURL
- 500 MB of disk space

2.3 Prepare

Start downloading the Docker images and data files.

2.3.1 Pull Docker Images

Pull the necessary Docker images from the SDAP repository on Docker Hub. Please check the repository for the latest version tag.

```
export VERSION=1.0.0-SNAPSHOT
```

```
docker pull sdap/ningester:${VERSION}
docker pull sdap/solr-singlenode:${VERSION}
docker pull sdap/cassandra:${VERSION}
docker pull sdap/nexus-webapp:standalone.${VERSION}
```

2.3.2 Create a new Docker Bridge Network

This quickstart constsists of launching several Docker containers that need to communicate with one another. To facilitate this communication, we want to be able to reference containers via hostname instead of IP address. The default bridge network used by Docker only supports this by using the `--link` option wich is now considered to be deprecated.

The currently recommended way to acheive what we want is to use a user defined bridge network and launch all of the containers into that network.

The network we will be using for this quickstart will be called `sdap-net`. Create it using the following command:

```
docker network create sdap-net
```

2.3.3 Download Sample Data

The data we will be downloading is part of the AVHRR OI dataset which measures sea surface temperature. We will download 1 month of data and ingest it into a local Solr and Cassandra instance.

Choose a location that is mountable by Docker (typically needs to be under the User's home directory) to download the data files to.

```
export DATA_DIRECTORY=~/nexus-quickstart/data/avhrr-granules
mkdir -p ${DATA_DIRECTORY}
```

Then go ahead and download 1 month worth of AVHRR netCDF files.

```
cd $DATA_DIRECTORY

export URL_LIST="https://podaac-opendap.jpl.nasa.gov:443/opendap/allData/ghrsst/data/
↪GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/2015/305/20151101120000-NCEI-L4_GHRSST-SSTblend-AVHRR_
↪OI-GLOB-v02.0-fv02.0.nc https://podaac-opendap.jpl.nasa.gov:443/opendap/allData/
↪ghrsst/data/GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/2015/306/20151102120000-NCEI-L4_GHRSST-
↪SSTblend-AVHRR_OI-GLOB-v02.0-fv02.0.nc https://podaac-opendap.jpl.nasa.gov:443/
↪opendap/allData/ghrsst/data/GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/2015/307/20151103120000-
↪NCEI-L4_GHRSST-SSTblend-AVHRR_OI-GLOB-v02.0-fv02.0.nc https://podaac-opendap.jpl.
↪nasa.gov:443/opendap/allData/ghrsst/data/GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/2015/308/
↪20151104120000-NCEI-L4_GHRSST-SSTblend-AVHRR_OI-GLOB-v02.0-fv02.0.nc https://podaac-
↪opendap.jpl.nasa.gov:443/opendap/allData/ghrsst/data/GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/
↪2015/309/20151105120000-NCEI-L4_GHRSST-SSTblend-AVHRR_OI-GLOB-v02.0-fv02.0.nc␣
↪https://podaac-opendap.jpl.nasa.gov:443/opendap/allData/ghrsst/data/GDS2/L4/GLOB/
↪NCEI/AVHRR_OI/v2/2015/310/20151106120000-NCEI-L4_GHRSST-SSTblend-AVHRR_OI-GLOB-v02.
↪0-fv02.0.nc https://podaac-opendap.jpl.nasa.gov:443/opendap/allData/ghrsst/data/
↪GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/2015/311/20151107120000-NCEI-L4_GHRSST-SSTblend-AVHRR_
↪OI-GLOB-v02.0-fv02.0.nc https://podaac-opendap.jpl.nasa.gov:443/opendap/allData/
↪ghrsst/data/GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/2015/312/20151108120000-NCEI-L4_GHRSST-
↪SSTblend-AVHRR_OI-GLOB-v02.0-fv02.0.nc https://podaac-opendap.jpl.nasa.gov:443/
↪opendap/allData/ghrsst/data/GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/2015/313/20151109120000-
↪NCEI-L4_GHRSST-SSTblend-AVHRR_OI-GLOB-v02.0-fv02.0.nc https://podaac-opendap.jpl.
↪nasa.gov:443/opendap/allData/ghrsst/data/GDS2/L4/GLOB/NCEI/AVHRR_OI/v2/2015/314/
```

(continues on next page)

(continued from previous page)

```
for url in ${URL_LIST}; do
  curl -O "${url}"
done
```

You should now have 30 files downloaded to your data directory, one for each day in November 2015.

2.4 Start Data Storage Containers

We will use Solr and Cassandra to store the tile metadata and data respectively.

2.4.1 Start Solr

SDAP is tested with Solr version 7.x with the JTS topology suite add-on installed. The SDAP docker image is based off of the official Solr image and simply adds the JTS topology suite and the nexustiles core.

Note: Mounting a volume is optional but if you choose to do it, you can start and stop the Solr container without having to reingest your data every time. If you do not mount a volume, every time you stop your Solr container the data will be lost.

To start Solr using a volume mount and expose the admin webapp on port 8983:

```
export SOLR_DATA=~/nexus-quickstart/solr
docker run --name solr --network sdap-net -v ${SOLR_DATA}:/opt/solr/server/solr/
↪nexustiles/data -p 8983:8983 -d sdap/solr-singlenode:${VERSION}
```

If you don't want to use a volume, leave off the `-v` option.

2.4.2 Start Cassandra

SDAP is tested with Cassandra version 2.2.x. The SDAP docker image is based off of the official Cassandra image and simply mounts the schema DDL script into the container for easy initialization.

Note: Similar to the Solr container, using a volume is recommended but not required.

To start cassandra using a volume mount and expose the connection port 9042:

```
export CASSANDRA_DATA=~/nexus-quickstart/cassandra
docker run --name cassandra --network sdap-net -p 9042:9042 -v ${CASSANDRA_DATA}:/var/
↪lib/cassandra -d sdap/cassandra:${VERSION}
```

2.5 Ingest Data

Now that Solr and Cassandra have both been started and configured, we can ingest some data. NEXUS ingests data using the ningester docker image. This image is designed to read configuration and data from volume mounts and then tile the data and save it to the datastores. More information can be found in the *Ningester* section.

Ningester needs 3 things to run:

1. Tiling configuration. How should the dataset be tiled? What is the dataset called? Are there any transformations that need to happen (e.g. kelvin to celsius conversion)? etc...
2. Connection configuration. What should be used for metadata storage and where can it be found? What should be used for data storage and where can it be found?
3. Data files. The data that will be ingested.

2.5.1 Tiling configuration

For this quickstart we will use the AVHRR tiling configuration from the test job in the Apache project. It can be found here: AvhrrJobTest.yml. Download that file into a temporary location on your laptop that can be mounted by Docker.

```
export NINGESTER_CONFIG=~/nexus-quickstart/ningester/config
mkdir -p ${NINGESTER_CONFIG}
cd ${NINGESTER_CONFIG}
curl -O https://raw.githubusercontent.com/apache/incubator-sdap-ningester/
↪bc596c2749a7a2b44a01558b60428f6d008f4f45/src/testJobs/resources/testjobs/
↪AvhrrJobTest.yml
```

2.5.2 Connection configuration

We want ningester to use Solr for its metadata store and Cassandra for its data store. We also want it to connect to the Solr and Cassandra instances we started earlier. In order to do this we need a connection configuration file that specifies how the application should connect to Solr and Cassandra. It looks like this:

```
# Tile writer configuration
ningester:
  tile_writer:
    data_store: cassandraStore
    metadata_store: solrStore
---
# Connection settings for the docker profile
spring:
    profiles:
      - docker
    data:
      cassandra:
        keyspaceName: nexustiles
        contactPoints: cassandra
      solr:
        host: http://solr:8983/solr/

datasource:
  solrStore:
    collection: nexustiles
```

Save this configuration to a file on your local laptop that can be mounted into a Docker container:

```
touch ${NINGESTER_CONFIG}/connectionsettings.yml
cat << EOF >> ${NINGESTER_CONFIG}/connectionsettings.yml
# Tile writer configuration
ningester:
  tile_writer:
```

(continues on next page)

(continued from previous page)

```
    data_store: cassandraStore
    metadata_store: solrStore
---
# Connection settings for the docker profile
spring:
    profiles:
      - docker
    data:
      cassandra:
        keyspaceName: nexustiles
        contactPoints: cassandra
      solr:
        host: http://solr:8983/solr/

datasource:
  solrStore:
    collection: nexustiles
EOF
```

2.5.3 Data files

We already downloaded the datafiles to `${DATA_DIRECTORY}` in *Create a new Docker Bridge Network* so we are ready to start ingesting.

2.5.4 Launch Ningester

The ningester docker image runs a batch job that will ingest one granule. Here, we do a quick for loop to cycle through each data file and run ingestion on it.

Note: Ingestion takes about 60 seconds per file. Depending on how powerful your laptop is and what other programs you have running, you can choose to ingest more than one file at a time. If you use this example, we will be ingesting 1 file at a time. So, for 30 files this will take roughly 30 minutes. You can speed this up by reducing the time spent sleeping by changing `sleep 60` to something like `sleep 30`.

```
for g in `ls ${DATA_DIRECTORY} | awk "{print $1}"`
do
  docker run -d --name $(echo avhrr_$g | cut -d'-' -f 1) --network sdap-net -v $
↪{NINGESTER_CONFIG}:/home/ningester/config/ -v ${DATA_DIRECTORY}/${g}:/home/
↪ningester/data/${g} sdap/ningester:${VERSION} docker,solr,cassandra
  sleep 60
done
```

Each container will be launched with a name of `avhrr_<date>` where `<date>` is the date from the filename of the granule being ingested. You can use `docker ps` to watch the containers launch and you can use `docker logs <container name>` to view the logs for any one container as the data is ingested.

You can move on to the next section while the data ingests.

Note: After the container finishes ingesting the file, the container will exit (with a `0` exit code) indicating completion. However, the containers will **not** automatically be removed for you. This is simply to allow you to inspect the

containers even after they have exited if you want to. A useful command to clean up all of the stopped containers that we started is `docker rm $(docker ps -a | grep avhrr | awk '{print $1}')`.

2.6 Start the Webapp

Now that the data is being (has been) ingested, we need to start the webapp that provides the HTTP interface to the analysis capabilities. This is currently a python webapp running Tornado and is contained in the nexus-webapp Docker image. To start the webapp and expose port 8083 use the following command:

```
docker run -d --name nexus-webapp --network sdap-net -p 8083:8083 -e SPARK_LOCAL_
↪IP=127.0.0.1 -e MASTER=local[4] -e CASSANDRA_CONTACT_POINTS=cassandra -e SOLR_URL_
↪PORT=solr:8983 sdap/nexus-webapp:${VERSION}
```

Note: If you see a messasge like `docker: invalid reference format` it likely means you need to re-export the `VERSION` environment variable again. This can happen when you open a new terminal window or tab.

This command starts the nexus webservice and connects it to the Solr and Cassandra containers. It also sets the configuration for Spark to use local mode with 4 executors.

After running this command you should be able to access the NEXUS webservice by sending requests to http://localhost:8083. A good test is to query the `/list` endpoint which lists all of the datasets currently available to that instance of NEXUS. For example:

```
curl -X GET http://localhost:8083/list
```

2.7 Launch Jupyter

At this point NEXUS is running and you can interact with the different API endpoints. However, there is a python client library called `nexuscli` which facilitates interacting with the webservice through the Python programming language. The easiest way to use this library is to start the Jupyter notebook docker image from the SDAP repository. This image is based off of the `jupyter/scipy-notebook` docker image but comes pre-installed with the `nexuscli` module and an example notebook.

To launch the Jupyter notebook use the following command:

```
docker run -it --rm --name jupyter --network sdap-net -p 8888:8888 sdap/jupyter:$
↪{VERSION} start-notebook.sh --NotebookApp.password=
↪'sha1:a0d7f85e5fc4:0c173bb35c7dc0445b13865a38d25263db592938'
```

This command launches a Juypter container and exposes it on port 8888.

Note: The password for the Jupyter instance is `quickstart`

Once the container starts, navigate to http://localhost:8888/. You will be prompted for a password, use `quickstart`. After entering the password, you will be presented with a directory structure that looks something like this:

Click on the `Quickstart` directory to open it. You should see a notebook called `Time Series Example`:

Click on the `Time Series Example` notebook to start it. This will open the notebook and allow you to run the two cells and execute a Time Series command against your local instance of NEXUS.

2.8 Finished!

Congratulations you have completed the quickstart! In this example you:

1. Learned how to ingest data into NEXUS datastores
2. Learned how to start the NEXUS webservice
3. Learned how to start a Jupyter Notebook
4. Ran a time series analysis on 1 month of AVHRR OI data and plotted the result

CHAPTER 3

Ningester

3.1 How to run this image

The basic command is:

```
docker run -it --rm -v <absolute path to config directory on host>:/config/ -v
↪<absolute path to granule on host>:/data/<granule name> sdap/ningester <profiles to␣
↪activate>
```

Replacing the following:

- `<absolute path to config directory on host>` should be the absolute path on the host to the configuration for the job
- `<absolute path to granule on host>` should be the absolute path on the host to the granule intended for ingestion
- `<granule name>` should be the filename of the granule
- `<profiles to activate>` is a comma-separated list of profiles to activate

The ExampleJob.yml file shows an example Job configuration that would ingest an AVHRR granule.

3.2 Configuration

Upon running the image, the ningester job will scan the `/config` directory for any files that end with the `.yml` extension. Specifically it uses find:

```
find /config -name "*.yml" | awk -vORS=, '{ print $1 }'
```

Therefore, to configure the job, mount your configuration files into `/config` using a Docker volume. Alternatively, configuration is loaded via Spring Boot's relaxed binding rules. So, you can also configure the job through environment variables where each 'level' of the yaml file gets replaced by an '_'.

For example, given a configuration option in yaml that looks like:

```
ningester:
    tile_slicer: sliceFileByTilesDesired
    sliceFileByTilesDesired:
      tilesDesired: 1296
      timeDimension: time
      dimensions:
        - lat
        - lon
```

These could be replaced with the following corresponding Environment variables:

```
NINGESTER_TILE_SLICER=sliceFileByTilesDesired
NINGESTER_SLICE_FILE_BY_TILES_DESIRED_TILES_DESIRED=1296
NINGESTER_SLICE_FILE_BY_TILES_DESIRED_TIME_DIMENSION=time
NINGESTER_SLICE_FILE_BY_TILES_DESIRED_DIMENSIONS[0]=lat
NINGESTER_SLICE_FILE_BY_TILES_DESIRED_DIMENSIONS[1]=lon
```

However, because ningester has a lot of configuration options, it is recommended to use the yaml option.

3.3 Data

Ningester is designed to ingest 1 granule per run. It looks for the granule to ingest in the `/data` directory of the container image. Use a Docker volume to mount your data into `/data`.

The image relies on this command to find the first file in `/data` and it will use that file for ingestion:

```
find /data -type f -print -quit
```

3.4 Examples

A few example commands are shared here.

3.5 Docker for Mac

The ConnectionSettings-DockerForMac.yml file shows an example of how to configure the connection settings when running this job under Docker for Mac with Solr and Cassandra running on your host Mac.

Replace `<path to ningester>` with the path on your local workstation to the ningester github project.

```
docker run -it --rm -v <path to ningester>/docker/example_config/:/config/ -v <path␣
↪to ningester>/src/test/resources/granules/20050101120000-NCEI-L4_GHRSST-SSTblend-
↪AVHRR_OI-GLOB-v02.0-fv02.0.nc:/data/20050101120000-NCEI-L4_GHRSST-SSTblend-AVHRR_OI-
↪GLOB-v02.0-fv02.0.nc.nc sdap/ningester dockermachost,solr,cassandra
```

CHAPTER 4

Docker Images

incubator-sdap-nexus contains a number of different Docker images for download. All images are available from the SDAP organization on DockerHub.

4.1 Solr Images

All docker builds for the Solr images should happen from this directory. For copy/paste ability, first export the environment variable `BUILD_VERSION` to the version number you would like to tag images as.

4.1.1 Common Environment Variables

Any environment variable that can be passed to solr.in.sh and be passed as an environment variable to the docker container and it will be utilized. A few options are called out here:

`SOLR_HEAP` *default: 512m*

Increase Java Heap as needed to support your indexing / query needs

`SOLR_HOME` *default /opt/solr/server/solr*

Path to a directory for Solr to store cores and their data. This directory is exposed as a `VOLUME` that can be mounted.

If you want to mount the `SOLR_HOME` directory to a directory on the host machine, you need to provide the container path to the docker run `-v` option. Doing this allows you to retain the index between start/stop of this container.

4.1.2 sdap/solr

This is the base image used by both singlenode and cloud versions of the Solr image.

How To Build

This image can be built by:

```
docker build -t sdap/solr:${BUILD_VERSION} .
```

How to Run

This image is not intended to be run directly.

4.1.3 sdap/solr-singlenode

This is the singlenode version of Solr.

How To Build

This image can be built from the incubator/sdap/solr directory:

```
docker build -t sdap/solr-singlenode:${BUILD_VERSION} -f singlenode/Dockerfile --
↪build-arg tag_version=${BUILD_VERSION} .
```

How to Run

This Docker container runs Apache Solr v7.4 as a single node with the nexustiles collection. The main decision when running this image is wether or not you want data to persist when the container is stopped or if the data should be discarded.

Persist Data

To persist the data in the `nexustiles` collection, we need to provide a volume mount from the host machine to the container path where the collection data is stored. By default, collection data is stored in the location indicated by the `$SOLR_HOME` environment variable. If you do not provide a custom `SOLR_HOME` location, the default is `/opt/solr/server/solr`. Therefore, the easiest way to run this image and persist data to a location on the host machine is:

```
docker run --name solr -v ${PWD}/solrhome/nexustiles:/opt/solr/server/solr/nexustiles␣
↪-p 8083:8083 -d sdap/solr-singlenode:${BUILD_VERSION}
```

`${PWD}/solrhome/nexustiles` is the directory on host machine where the `nexustiles` collection will be created if it does not already exist. If you have run this container before and `${PWD}/solrhome/nexustiles` already contains files, those files will *not* be overwritten. In this way, it is possible to retain data on the host machine between runs of this docker image.

Don't Persist Data

If you do not need to persist data between runs of this image, just simply run the image without a volume mount.

```
docker run --name solr -p 8083:8083 -d sdap/solr-singlenode:${BUILD_VERSION}
```

When the container is removed, the data will be lost.

4.1.4 sdap/solr-cloud

This image runs SolrCloud.

How To Build

This image can be built from the incubator/sdap/solr directory:

```
docker build -t sdap/solr-cloud:${BUILD_VERSION} -f cloud/Dockerfile --build-arg tag_
↪version=${BUILD_VERSION} .
```

How to Run

This Docker container runs Apache Solr v7.4 in cloud mode with the nexustiles collection. It requires a running Zookeeper service in order to work. It will automatically bootstrap Zookeeper by uploading configuration and core properties to Zookeeper when it starts.

It is necessary to decide wether or not you want data to persist when the container is stopped or if the data should be discarded.

Note: There are multiple times that `host.docker.internal` is used in the example `docker run` commands provided below. This is a special DNS name that is known to work on Docker for Mac for connecting from a container to a service on the host. If you are not launching the container with Docker for Mac, there is no guarantee that this DNS name will be resolvable inside the container.

Cloud Specific Environment Variables

`SDAP_ZK_SERVICE_HOST` *default: localhost*

This is the hostname of the Zookeeper service that Solr should use to connect.

`SDAP_ZK_SERVICE_PORT` *default: 2181*

The port Solr should try to connect to Zookeeper with.

`SDAP_ZK_SOLR_CHROOT` *default: solr*

The Zookeeper chroot under which Solr configuration will be accessed.

`SOLR_HOST` *default: localhost*

The hostname of the Solr instance that will be recored in Zookeeper.

Zookeeper

Zookeeper can be running on the host machine or anywhere that docker can access (e.g. a bridge network). Take note of the host where Zookeeper is running and use that value for the `SDAP_ZK_SERVICE_HOST` environment variable.

Persist Data

To persist the data, we need to provide a volume mount from the host machine to the container path where the collection data is stored. By default, collection data is stored in the location indicated by the `$SOLR_HOME` environment variable. If you do not provide a custom `SOLR_HOME` location, the default is `/opt/solr/server/solr`.

Assuming Zookeeper is running on the host machine port 2181, the easiest way to run this image and persist data to a location on the host machine is:

```
docker run --name solr -v ${PWD}/solrhome:/opt/solr/server/solr -p 8983:8983 -d -e␣
↪SDAP_ZK_SERVICE_HOST="host.docker.internal" -e SOLR_HOST="host.docker.internal"␣
↪sdap/solr-cloud:${VERSION}
```

`${PWD}/solrhome` is the directory on host machine where `SOLR_HOME` will be created if it does not already exist.

Don't Persist Data

If you do not need to persist data between runs of this image, just simply run the image without a volume mount.

Assuming Zookeeper is running on the host machine port 2181, the easiest way to run this image without persisting data is:

```
docker run --name solr -p 8983:8983 -d -e SDAP_ZK_SERVICE_HOST="host.docker.internal"␣
↪-e SOLR_HOST="host.docker.internal" sdap/solr-cloud:${VERSION}
```

When the container is removed, the data will be lost.

Collection Initialization

Solr Collections must be created after at least one SolrCloud node is live. When a collection is created, by default Solr will attempt to spread the shards across all of the live nodes at the time of creation. This poses two problems

1) The nexustiles collection can not be created during a "bootstrapping" process in this image.
2) The nexustiles collection should not be created until an appropriate amount of nodes are live.

A helper container has been created to deal with these issues. See *sdap/solr-cloud-init* for more details.

The other option is to create the collection manually after starting as many SolrCloud nodes as desired. This can be done through the Solr Admin UI or by utilizing the admin collections API.

4.1.5 sdap/solr-cloud-init

This image can be used to automatically create the `nexustiles` collection in SolrCloud.

How To Build

This image can be built from the incubator/sdap/solr directory:

```
docker build -t sdap/solr-cloud-init:${BUILD_VERSION} -f cloud-init/Dockerfile .
```

How to Run

This image is designed to run in a container alongside the *sdap/solr-cloud* container. The purpose is to detect if there are at least `MINIMUM_NODES` live nodes in the cluster. If there are, then detect if the `nexustiles` collection exists or not. If it does not, this script will create it using the parameters defined by the `CREATE_COLLECTION_PARAMS` environment variable. See the reference documents for the create function for the Solr collections API for valid parameters.

Note: The `action=CREATE` parameter is already passed for you and should not be part of `CREATE_COLLECTION_PARAMS`

Note: This image was designed to be long running. It will only exit if there was an error detecting or creating the `nexustiles` collection.

Environment Variables

`MINIMUM_NODES` *default: 1*

The minimum number of nodes that must be 'live' before the collection is created.

`SDAP_ZK_SOLR` *default: localhost:2181/solr*

The host:port/chroot of the zookeeper being used by SolrCloud.

`SDAP_SOLR_URL` *default: http://localhost:8983/solr/*

The URL that should be polled to check if a SolrCloud node is running. This should be the URL of the *sdap/solr-cloud* container that is being started alongside this container.

`ZK_LOCK_GUID` *default: c4d193b1-7e47-4b32-a169-a596463da0f5*

A GUID that is used to create a lock in zookeeper so that if more than one of these init containers are started at the same time, only one will attempt to create the collection. This GUID should be the same across all containers that are trying to create the same collection.

`MAX_RETRIES` *default: 30*

The number of times we will try to connect to SolrCloud at `SDAP_SOLR_URL`. This is roughly equivalent to how many seconds we will wait for the node at `SDAP_SOLR_URL` to become available. If `MAX_RETRIES` is exceeded, the container will exit with an error.

`CREATE_COLLECTION_PARAMS` *default: name=nexustiles&collection.configName=nexustiles&numShards=1*

The parameters sent to the collection create function. See the reference documents for the create function for the Solr collections API for valid parameters.

Example Run

Assuming Zookeeper is running on the host machine port 2181, and a *sdap/solr-cloud* container is also running with port 8983 mapped to the host machine, the easiest way to run this image is:

```
docker run -it --rm --name init -e SDAP_ZK_SOLR="host.docker.internal:2181/solr" -e␣
↪SDAP_SOLR_URL="http://host.docker.internal:8983/solr/" sdap/solr-cloud-init:${BUILD_
↪VERSION}
```

After running this image, the `nexustiles` collection should be available on the SolrCloud installation. Check the logs for the container to see details.

Check out the *Quickstart Guide*.