Ingest

The Ingest Server coordinates the distribution of packages throughout the Chronopolis network, and can perform additional services if needed (examples below). This is done through a HTTP API which both Intake and Replication services can interact with and receive updates about data they are processing.

Links

- Gitlab: Chronopolis-Core
- API/Developer Documentation

Installation

Prerequisites

- Postgresql - The Ingest Server connects to a postgresql database to store information about transfers, bags, and tokens
- Token Staging Area - The Ingest Server needs an area to store tokens during transfer
- SSH key exchange - Replications are done through rsync, meaning public keys should be shared between nodes in order to authorize and authenticate access

RPM

Download the rpm for your operating system

Running

The ingest server runs as an executable jar. Using the init script allows for starting and stopping of the server as root

- EL6: `service ingest-server start|stop`
- EL7: `systemctl start ingest-server`

Installation Notes

Installed files for RHEL6

```
/etc/init.d/ingest-server
/usr/local/chronopolis/ingest
/usr/local/chronopolis/ingest/application.yml
/usr/local/chronopolis/ingest/ingest-server.jar
```

Installed Files for RHEL7

```
/usr/lib/systemd/system/ingest-server.service
/usr/local/chronopolis/ingest
/usr/local/chronopolis/ingest/application.yml
/usr/local/chronopolis/ingest/ingest-prepare
/usr/local/chronopolis/ingest/ingest-server.jar
```

When running, the startup scripts will check for the following directories and create/apply permissions if they do not match:

- Logging: /var/log/chronopolis/

User Creation

A `chronopolis` user is also needed which can write to `/var/log/chronopolis` and perform various read and write tasks as needed from the Token Staging Area. This is no longer installed as part of the rpm, but should be managed separately and configured in the ingest-server startup script.

Configuration

The ingest server reads from the /usr/local/chronopolis/ingest/application.yml configuration file:

application.yml

```
# Ingest Configuration Properties

# Ingest Cron job properties
# tokens: the rate at which to check for bags which have all tokens and can have a Token Store written
# request: the rate at which to check for bags which need their initial replications created
# tokenize: the rate at which to check for local bags which need tokens created - DEPRECATED in 3.0
ingest.cron:
  tokens: 0 0/10 * * * *
  request: 0 0/10 * * * *
  tokenize: 0 0 * * * * *

# Ingest AJP Settings
# enabled: flag to enable an AJP connector
# port: the port for the connector to listen on
ingest.ajp:
  enabled: false
  port: 8009

# The staging area for writing Token Stores. Nonposix support not yet implemented.
## id: The id of the StagingStorage in the Ingest server
## path: The path to the filesystem on disk
chron.stage.tokens.posix.id: -1
chron.stage.tokens.posix.path: /dev/null

# If Local Tokenization is desired include properties for the Ingest API user, staging information for Bags,
and ACE IMS connection information
# username: The name of the user who created the bags ingest will be tokenizing
ingest.api.username: bag-creator

## id: The id of the StagingStorage which the Ingest Server will read from  - DEPRECATED in 3.0
## path: The path to the filesystem on disk  - DEPRECATED in 3.0
chron.stage.bags.posix.id: -1
chron.stage.bags.posix.path: /dev/null

## port: the port to connect to the ims with
## waitTime: the time to wait between token requests
## endpoint: the fqdn of the ims
## queueLength: the maximum number of requests to send at once
ace.ims:
  port: 80
  waitTime: 5000
  endpoint: ims.umiacs.umd.edu
  queueLength: 1000

# Database connection
# Initialize should be kept false so that the server does not attempt to run a drop/create on the tables
spring.datasource:
  url: jdbc:postgresql://localhost/ingest
  username: postgres
  password: dbpass
  initialize: false

# Specify the active profile for configuring services as a comma separated list
# production - remove stdout/stderr from printing and run without accepting input
# disable-tokenizer - disable local tokenization services from running
spring.profiles.active: production
spring.pid.file: /var/run/ingest-server.pid

# debug: true
server.port: 8080

# Logging properties
logging.file: ingest.log
logging.level:
  org.springframework: ERROR
  org.hibernate: ERROR
  org.chronopolis: DEBUG
```

Notes on Configuratiom

- An AJP connector can now be configured with the server, meaning SSL can be served through apache httpd instead of a java keystore
- The pid file should probably not be updated unless you update the init files with any corresponding changes

Administration

Database Setup

Database initialization can be done through the flyway-maven-plugin provided that you have a PostgresQL database which you can connect to and create /drop tables for. The flyway plugin will create load the first version of the schema and baseline the database at 1.0 so that all migrations can be applied. This requires the chronopolis-core repository to be cloned so that the flyway plugin can be run.

To run the flyway plugin, `cd` into the `ingest-rest` directory and use `flyway:baseline` and provide configuration for the `flyway.user`, `flyway.password`, and `flyway.url` configuration parameters. Once the database is baselined, the ingest server can be started and will apply all migrations.

Example PostgresSQL Setup

```
[chronopolis-core] $ cd ingest-rest
[chronopolis-core/ingest-rest] $ ../mvnw -Dflyway.user=postgres -Dflyway.password=mysecretpassword -Dflyway.
url=jdbc:postgresql://172.17.0.2/ingest flyway:baseline
```

Preparing the DB for Schema Migrations

As of version 1.1.0, the database has a schema_version table for handling schema migrations. This is managed automatically through flyway, so that the server can be upgraded without needing to worry about manually applying patches. Flyway provides a jar file which we can use to prepare the database for migrations, something which can be applied to previous versions as well.

1. Download and untar/unzip the Flyway Command Line Tool
    a. The Ingest Server currently uses Flyway 5.2.4; if possible the binary for that version should be used
2. Edit the conf/flyway.conf
    a. some properties follow the same pattern as our application properties (connecting to the database)
    b. specify the version which you are creating the baseline (using the MAJOR.MINOR number of the ingest server version)

    Flyway Configuration Example

    ```
    #
    # Copyright 2010-2015 Axel Fontaine
    #
    ...

    # Jdbc url to use to connect to the database
    flyway.url=jdbc:postgresql://localhost/ingest

    # Fully qualified classname of the jdbc driver (autodetected by default based on flyway.url)
    # flyway.driver=

    # User to use to connect to the database (default: <<null>>)
    flyway.user=chron

    # Password to use to connect to the database (default: <<null>>)
    flyway.password=my-postgresql-password

    ...
    flyway.baselineVersion=3.0
    ```

3. Use the flyway bash script to update the database

    Flyway Baseline Migration

    ```
    $ ./flyway baseline
    ```

Local Tokenization SINCE 2.3.0

When doing local ingestion of bags through the Ingest Server, it is possible to have the Ingest Server create ACE Tokens for the files in a Bag. This can be enabled through the application.yml configuration file. Note that because staging areas can be shared, the user creating the Bags in the Ingest Server should be unique to the `local` processes.

Once enabled, rudamentary information of tokenization can be viewed on the webui at `/status/supervisor`.

Tokenization Configuration

```
# Ingest Tokenizer Settings
# cron: the cron timer for running local-tokenization
# enabled: flag to enable Local tokenization of bags
# username: the 'creator' to check for when depositing bags (defaults to to 'admin')
# staging.id: the ID of the StorageRegion to write tokens into
# staging.path: the path to the filesystem on disk
ingest.tokenizer:
  cron: 0 0 * * * *
  enabled: true
  username: admin
  staging.id: -1
  staging.path: /dev/null
```

Local File Scanning SINCE 3.1.0

Bags on a filesystem local to the Ingest Server can have their files and fixities registered. This is done through a task which periodically fires and queries the database for Bags awaiting scanning. Note that because staging areas can be shared, the user creating the Bags in the Ingest Server should be unique to the `local` processes.

Resetting Passwords SINCE 1.4.0

As of version 1.4.0, passwords for users are now encoded using bcrypt. In the event a user forgets their password, we will need to reset it for them. As we do not have email notifications or anything of the like setup, for the moment everything will need to be done manually. We will first need to run the password through an encoder, which can be found online. If you aren't sure how many rounds to use, check the database as the information is kept as part of the encoding, i.e. $2a$08 uses 8 rounds; $2a$10 uses 10 rounds.

Then we connect to the database and issue a simple update:

User update

```
 UPDATE users SET password = '$2a$10$hEYYHV/Fri00RRHjWPIAWuH3NxYpPPjbMU5OsJfH1SAenajQqKjhK' WHERE username =
'user_resetting_password';
```

Storage Regions SINCE 2.0.0

With the release of version 2.0.0, StorageRegions have been introduced in order to facilitate distribution of content from many nodes in Chronopolis. A StorageRegion contains information about what type of data is held and the total capacity of the StorageRegion. Currently the capacity only serves as a reference, and can be exceeded if the Ingest Server does not know data has been removed. A `note` can also be provided to display additional information about a StorageRegion (e.g. 500TB XFS JBOD)

StorageRegions also provide configuration information for creating replications:

ReplicationConfiguration

- Replication Server: The server which will be connected to for transferring data
- Replication Path: The path which should be used to point to the files, e.g. `/export/bags`, or with a chroot env `/bags`
- Replication Username: The username clients should connect as, or null if they should use their node username