# IT Admin Guide: Review Document

# Contents

# About the IT Administrator

An individual responsible for maintaining the PHEMI Central system should have skills and expertise in the following:

- Linux system administration and network security
- Database administration
- Apache Hadoop ecosystem, including Ambari, YARN, and Accumulo
- MongoDB
- Docker containers
- Ansible playbooks

# Your Deployment

Your PHEMI Central system is installed by PHEMI Professional Services. The PHEMI Professional Services team will:

- Set up your base system of cluster nodes
- Perform initial configuration of the system components and initialize them

The system consists includes a number of components:

- The Hadoop ecosystem, including YARN and Accumulo
- Ambari, a web-based console for provisioning, managing, and monitoring the cluster
- MongoDB, a high availability, scalable document database
- Tornado, the Python-based framework on which the PHEMI Central web application runs
- Nginx, used as a web proxy server by PHEMI Central

A number of PHEMI Central components are implemented as Docker containers. PHEMI Professional Services will install Docker as part of system installation.

The PHEMI Central platforms can support multiple data contexts, called "tenants." PHEMI Professional Services will have installed some number of tenants for your deployment. These tenants need to be included in system administration.

# Starting and Stopping the System

From a cold state, the components of PHEMI Central must be started in order. From a running system, PHEMI Central components are stopped in essentially the reverse order to that in which they were started.

## Start the System

First, start the Hadoop cluster.

1. Start the service for the Ambari server.

   Log on to the node running the Ambari server as a user with sudo privileges. At the command prompt, enter the following command:

   ```
   sudo service ambari-server start
   ```

2. When the command line returns a success message, use a web browser to access the Ambari console, which runs on port 8080.
3. In the Ambari console, select your cluster.

   The cluster services show yellow question mark icons, which means that the Ambari server has not discovered the services yet. Wait for the Ambari server to establish communication with the Ambari service on the cluster.

   When the server and the cluster have discovered one another, the Ambari dashboard will learn that the services are stopped. The yellow question marks change to red "stopped" icons. This can take a few minutes.
4. Start Accumulo and the Hadoop cluster.

   Log on to the Accumulo master node as a user with sudo privileges. From the command line, become the Accumulo user.

   ```
   sudo su accumulo
   ```

   Start the Hadoop cluster.

   ```
   /usr/lib/accumulo/bin/start-cluster.sh
   ```

5. Verify that the cluster has started correctly.

   When the server returns a success response to the command starting the cluster, return to the Ambari console. The Ambari console should show green checkmark icons next to each service to indicate that the service is running.

   If any service still shows as stopped, you can individually start the service from within Ambari. Select the service, click the **Service-Actions** button, and select **Start**.

After the Hadoop cluster is started, start the Docker containers that make up the PHEMI Central application.

6. Start the Docker service.

   Log on to the server running PHEMI Central as a user with sudo privileges. Start the Docker service.

   ```
   sudo service docker start
   ```

7. Start the MongoDB container.

   ```
   sudo docker start phemi_mongo
   ```

8. Start the PHEMI Central container.

```
sudo docker start phemi_central
```

9. Start the Nginx container.

```
sudo docker start phemi_nginx
```

10. Check that the containers are all running.

```
sudo docker ps
```

The system displays a list of running Docker containers. The list should show all the containers you started.

Once the containers are started, start the web proxy server and the PHEMI Central tenants.

11. Start the Nginx web proxy server.

From the PHEMI Central server command line, attach to the Nginx container.

```
sudo docker attach phemi_nginx
```

You may have to press <Enter> for the container command prompt to appear. At the container command line, become the Ubuntu user, so that you have sudo privileges. Start the Nginx server.

```
su ubuntu
sudo /etc/init.d/nginx start
```

12. Detach from the Nginx container but be careful not to kill it.

To detach from the container without killing it, press the key sequence <Ctrl>+p, <Ctrl>+q.

13. Start the PHEMI Central tenants.

From the PHEMI Central server command line, attach to the PHEMI Central container.

```
sudo docker attach phemi_central
```

You may have to press <Enter> for the container command prompt to appear. At the container command line, become the Ubuntu user.

```
su ubuntu
```

Navigate to the directory that contains your tenants.

```
cd /home/ubuntu/agile/bin/
```

Start each tenant in your deployment.

```
./<tenant_01>/start-phemi.sh
./<tenant_02>/start-phemi.sh
./<tenant_03>/start-phemi.sh
```

14. Detach from the PHEMI Central container but be careful not to kill it.

To detach from the container without killing it, press the key sequence <Ctrl>+p, <Ctrl>+q.

## Stop the System

First, stop the PHEMI Central Docker containers.

1. Log on to the server running PHEMI Central as a user with sudo privileges.

2. Stop the Nginx container.

```
sudo docker stop phemi_nginx
```

3. Stop the PHEMI Central container.

```
sudo docker stop phemi_central
```

4. Stop the MongoDB container.

```
sudo docker stop phemi_mongo
```

5. Check that the containers are all stopped.

```
sudo docker ps
```

   The system displays a list of running Docker containers. The list should be empty.

6. Stop the Docker service.

```
sudo service docker stop
```

After stopping the PHEMI Central Docker containers, stop the Hadoop cluster and the Ambari server.

7. Stop the cluster.

   Log on to the Accumulo master node as a user with sudo privileges. From the command line, become the Accumulo user.

```
sudo su accumulo
```

   Stop all the cluster and accumulo services.

```

/usr/lib/accumulo/bin/stop-cluster.sh
```

8. Verify that all the services are stopped.

   Use a web browser to access the Ambari console, which runs on port 8080

   In the Ambari console, select your cluster.

   The cluster services show yellow question mark icons, which means that the Ambari server has not discovered the services yet. Wait for the Ambari server to establish communication with the Ambari service on the cluster.

   When the server and the cluster have discovered one another, the Ambari dashboard will learn that the services are running. The yellow question marks change to red "stopped" icons. This can take a few minutes.

9. Stop the Ambari server.

   Log on to the node running the Ambari server as a user with sudo privileges. Stop the server.

```
sudo service ambari-server stop
```

# Backup and Restore

Backup files, regardless of component, are stored on the PHEMI Central server. Backups are stored in the directory /backup.

The most recent backups are stored in /backup/latestOlder backups reside in directories named /backup/*timestamp*, where *timestamp* is a timestamp in Epoch time format.

## Back Up the Database

You back up the PHEMI Central database using an Ansible playbook. The playbook backs up data each tenant deployed on your system, including the Accumulo database, MongoDB, and all deployed Data Processing Functions (DPFs), as well as the Ansible configuration.

You access the playbook and run it from the command line of the PHEMI Central Docker container.

1. Log on to the server running PHEMI Central as a user with sudo privileges.
2. From the PHEMI Central server command line, attach to the PHEMI Central container and access its command line.

   ```
   sudo docker attach phemi_central
   ```

   After attaching to the process, you may have to press <Enter> for the container command prompt to appear.
3. Navigate to the directory where the backup script is located.

   ```
   cd ~/agile/ansible/
   ```
4. Run the script.

   ```
   ansible-playbook -u ubuntu -i inventory/template backup-prod.yml
   ```

## Restore Data from a Backup

Restore each of Accumulo, MongoDB, Data Processing Functions (DPFs), and Ansible configuration separately.

Before beginning any restore procedure, check the running processes to ensure that no instance of the agile.py process is running.

### Restore Accumulo

Accumulo backups preserve the data, Accumulo splits, Accumulo configuration, and the logical time information for the Accumulo table.

You restore Accumulo information using a Python script.

Accumulo information for each PHEMI Central tenant is restored separately.

1. Log on to the server running PHEMI Central as a user with sudo privileges.
2. From the PHEMI Central server command line, attach to the PHEMI Central container and access its command line.

   ```
   sudo docker attach phemi_central
   ```

   After attaching to the process, you may have to press <Enter> for the container command prompt to appear.

3. At the container command line, become the Ubuntu user.

```
su ubuntu
```

4. Navigate to the PHEMI Central web application folder.

```
cd /home/ubuntu/agile/agile-web-ui/
```

5. For each tenant in your deployment, execute the restore script.

```
python -m utils/backup.accumulo_backup --config
{{ tenant_id }}.central --backup-dir /backup/timestamp/accumulo
--restore
```

where *tenant_id* is the identifier of the tenant (for example, tenant_01) and *timestamp* is either latest (for the most recent backup) or a timestamp in Epoch time format.

## Restore MongoDB

The backup script uses the mongodump utility to back up MongoDB data. The mongodump utility creates a binary backup.

You use the mongorestore utility to restore from the binary backup.

MongoDB information for each PHEMI Central tenant is restored separately.

1. Log on to the server running PHEMI Central as a user with sudo privileges.
2. From the PHEMI Central server command line, attach to the PHEMI Central container and access its command line.

```
sudo docker attach phemi_central
```

After attaching to the process, you may have to press <Enter> for the container command prompt to appear.

3. At the container command line, become the Ubuntu user, so that you have sudo privileges.

```
su ubuntu
```

4. Navigate to the PHEMI Central web application folder.

```
cd /home/ubuntu/agile/agile-web-ui/
```

5. For each tenant in your deployment, execute the restore script.

```
mongorestore --host tenant_host --port tenant_port
-u tenant_username -p tenant_password
--authenticationDatabase tenant_database /backup/timestamp/mongodb
```

where:

- *tenant_host* is the name or IP address of the host running the tenant's MongoDB server
- *tenant_port* is the port being used for MongoDB on the tenant's host
- *tenant_username* is the username for logging on to the tenant's MongoDB server
- *tenant_password* is the password for logging on to the tenant's MongoDB server
- *tenant_database* is the name of the tenant's MongoDB database.
- *timestamp* is either latest (for the most recent backup) or a timestamp in Epoch time format.

## Restore DPFs

A Data Processing Function, or DPF, is an executable piece of code that supplies the instructions for processing raw data to extract meaningful, context-specific information (such as a temperature reading or blood glucose measurement) that can be queried or exported for analysis.

The set of code that makes up a DPF is called a DPF archive. A DPF archive is delivered as a ZIP file archive. It consists of two parts: a manifest file and a code library. To associate a DPF with a data collection, the DPF archive is ``registered`` with the data collection by uploading the DPF archive. Registering the DPF is part of data collection configuration.

DPF archives are backed up into the directory /backup/*timestamp*/dpf, where *timestamp* is either latest (for the most recent backup) or a timestamp in Epoch time format.

Once retrieved, the DPF archive may need to be re-registered with the appropriate data collection in PHEMI Central. Your PHEMI Administrator can do this.

## Retrieve an Ansible Playbook

The PHEMI Central platform uses Ansible to set up and configure various components during installation.

The PHEMI Central Ansible Playbooks are backed up into the directory /backup/*timestamp*, where *timestamp* is either latest (for the most recent backup) or a timestamp in Epoch time format.

# Glossary of Terms and Concepts

**access policy**

An access policy is a set of rules that specifies how users can consume data stored in PHEMI Central. The access policy lists what user authorizations are required to interact with data tagged with specified visibility. Access policies can be applied to data collections and datasets.

**Accumulo**

Apache Accumulo is a high-performance distributed key/value data storage and retrieval system.

**Ambari**

Apache Ambari is an open-source interface for provisioning, managing, and monitoring Hadoop clusters.

**authorizations**

User authorizations are configurable attributes you can assign to PHEMI Central users. Authorizations are defined in PHEMI Central by the PHEMI Administrator, who sets them in accordance with the organization's governance policies.

**field**

A field is the smallest unit of data storage in PHEMI Central. A field is a single data item, which can range from a single byte up to gigabytes, plus the metadata associated with the data item. Any piece of raw data, regardless of size, is stored in a single field. Elements of derived data (transformed from the raw data) are also each stored individually in fields. Any field can be protected by applying data visibilities. For derived data, each derived item can be individually assigned a visibility (which may be different than that configured for the data collection) by the DPF performing the processing.

**code library**

A code library is a package of executable code that is included in a DPF archive. Whether the code is source or compiled depends on the coding language. Code libraries must be portable and self-contained; that is, all dependencies required for the DPF to function must be bundled inside the library, in the appropriate way, for whatever language is being used.

**data category**

Data categories are a way to classify data into broader groupings. Examples of data categories are "Research Reports," "X-Rays," and "Prescriptions."

**data collection**

In PHEMI Central, a data collection is the set of management and governance rules and policies, as well as the DPF processing, that will be applied to some set of data. A data collection configuration should be defined for each set of data that is to be stored and managed according to the same retention, legal, and governance rules.

**Data Processing Function, DPF**

A Data Processing Function, or DPF, is an executable piece of code that supplies the instructions for processing raw data to extract meaningful, context-specific information (such as a temperature reading or blood glucose measurement) that can be queried or exported for analysis.

The code is executed by the PHEMI Central DPF Engine, which uses it to direct curation of the data. The input to a DPF is the raw binary data ingested into the system. The output of a DPF is a set of structured elements, each of which includes a type property (for example, INT or STRING) and can selectively specify data visibilities (for example, SECRET or IDENTIFIABLE) on a per-field basis. The data elements output by a DPF are called derived data. The collection of derived data produced by a DPF is automatically indexed in PHEMI Central.

**data visibilities**

See visibilities.

**dataset**

A dataset is a view, or map, of an underlying set of data. Data items in a dataset can be selected from across multiple data collections. The dataset is a view, or map, to the underlying data. The actual content of the dataset (that is, the dataset's data) is generated when the dataset is executed or when it is queried against.

**derived data**

Derived data is data that has been parsed, extracted, or otherwise enriched or processed by running a DPF on stored raw data. The set of derived data items can be searched, queried, further processed, or exported from the system.

**digital asset**

A digital asset is any piece of data stored with metadata in the system. This may be raw data that has had metadata applied during ingestion, or it may be derived data that has been parsed, indexed, catalogued, and/or enriched with additional metadata.

**DPF archive**

The set of code that makes up a DPF is called a DPF archive. A DPF archive is delivered as a ZIP file archive. It consists of two parts: a manifest file and a code library. To associate a DPF with a data collection, the DPF archive is ``registered`` with the data collection by uploading the DPF archive. Registering the DPF is part of data collection configuration.

**ETL**

Extract, transform, and load. In databases, a set of tools or processes that extracts data from sources, transforms the format or structure for storage, query, and analysis, and loads it into the receiving or consuming system.

**HDFS**

The Hadoop Distributed File System, a distributed file system designed for scalability on commodity hardware.

**Hadoop**

Hadoop is an open-source platform for distributed computing.

**ingestion**

Ingestion is the process by which data is brought into in PHEMI Central. Often, the sending system submits the data to PHEMI Central, which listens for the data, often using a web service. Data can also be ingested manually, by using the PHEMI Central Management and Governance Console.

**JSON**

JSON stands for JavaScript Object Notation. JSON is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON is used in the body of several REST requests in the PHEMI RESTful API. PHEMI Central also includes a system DPF that can create derived data from JSON objects, providing the objects conform to PHEMI’s JSON specification.

**key-value pairs**

A key-value pair is a set of two linked data items: a key which uniquely identifies some item of data, and the value, which is the data itself. PHEMI Central uses key-value store to efficiently store, process, and retrieve data.

**M2M**

M2M is a way of referring to machine-to-machine interfaces, used in machine-to-machine communication.

**manifest file**

A manifest file is a JSON file that specifies the output of a DPF. With the code library, the manifest file makes up the DPF archive that is uploaded to register the DPF with a data collection. The manifest file should include the properties of the DPF along with the details of each derived data item to be generated.

**metadata**

Metadata is information about a piece of data. In PHEMI Central, metadata is information about how a given piece of data is to be managed. When a piece of raw data is ingested into PHEMI Central, information from the connection (for example, the timestamp) together with policy information configured for the data collection (for example, the data visibility) and some derived information (for example, a "time to live," as derived from the timestamp and the data retention policy) is used to create metadata properties that are stored with the data. Further, PHEMI Central also automatically indexes and catalogues all stored data, whether raw or derived; the indexes and catalogues can also be considered a kind of metadata.

**MongoDB**

MongoDB is a high-performance, high availability, scalable document database.

**Nginx**

(Pronounced "engine-X.") Nginx is an open-source reverse proxy server for the HTTP, HTTPS, SMTP, POP3, and IMAP protocols. It can also act as a load balancer, HTTP cache, and a web origin server .

**PII**

Personally Identifiable Information, or PII, is a legal concept used in US privacy law and information security to mean information that can be used on its own or with other information to identify, contact, or locate a single person or to identify an individual in context. When thinking about PII, it is important to distinguish legal requirements to remove attributes uniquely identify an individual from a general technical ability to identify individuals. Because of the versatility and power of modern re-identification algorithms, together with the amount of information freely available from all sources, the absence of PII data does not guarantee that de-identified data cannot be used, perhaps in combination with other data, to identify individuals.

**raw data**

In PHEMI Central, raw data items are files, objects, records, images, and so on that are submitted for ingestion into the system. Raw data is stored exactly as received, along with the metadata generated for it on ingestion.

**REST, RESTful API**

Representational Statement Transfer (REST) is an architectural style that uses HTTP requests and associated methods (POST, PUT, GET, and DELETE) to create, update, read, and delete data. A RESTful API is an application programming interface (API) based on REST.

**tenant**

A tenant is a virtual context within the PHEMI Central data store. Within the tenancy, data is private from other tenants. Using tenancies, more than one organization can use PHEMI Central without necessarily sharing data.

**Tornado**

Tornado is a Python-based web server and web application framework.

**VCF**

A Variant Call Format (VCF) file is a text file containing tab-separated marker and genotype data. VCF data is used in bioinformatics to store gene sequence variations. As such, VCF files are very large, and a a VCF file can document hundreds, thousands, and even millions of gene sites in a single file.

**visibilities**

All raw data and derived data stored in PHEMI Central can be tagged with labels that provide information about the data's sensitivity. This sensitivity is described in terms of the visibility the data should have to different

system users. The visibility tags you define for your data should reflect the visibility the data is intended to have according to your organization's governance policy.

Two words. (Not "whitepaper.")