Koku Documentation

Release 0.0.1

Red Hat, Inc.

Jul 13, 2020

Contents:

Koku, accessed through the our web user interface or application program interface (API), is a cost management and reporting tool. It is designed to identify and report cost data associated with sources (e.g. Amazon Web Services accounts), such as the total cost which can be broken down by services, accounts, etc. and instance counts and storage usage month over month. It is designed to provide insights on cost across your enterprise.

CHAPTER 1

User Management

This section covers the customer owner, adding and removing users.

1.1 Definitions

Project Koku uses terminology in certain ways to describe key concepts. These are definitions of the term based on how they work within the context of the Koku application.

Customer: An organization or entity that uses Project Koku for cost management analysis.

Source: A cloud resource provider or cloud data provider. An entity that produces cost and resource usage data. This could be a public or private cloud.

User - A user of the Project Koku application. Users map to an individual person or login with access to *customer* data.

1.2 Development

Authentication for Koku is expected to be managed by an external service. Authentication information is expected to be provided to Koku through an HTTP header - `HTTP_X_RH_IDENTITY`.

For development purposes, if the environment variable `DEVELOPMENT=True` is set, Koku will authenticate using its dev_middleware, which bypasses authentication and authorizes any request as valid.

This is an example for making authenticated HTTP requests to the Koku API when `DEVELOPMENT=True`.

```
#!/bin/bash
HOST='localhost'
IDENTITY=$(echo '{"identity":{"account_number":"10001","user":{"username":"test_
↪customer","email":"koku-dev@example.com"}}}' | base64 | tr -d '\n')
curl -g -H "HTTP_X_RH_IDENTITY: ${IDENTITY}" 'http://'${HOST}'/api/v1/reports/
↪inventory/aws/instance-type/'
```

CHAPTER 2

Source Management

This section covers managing sources. Currently the supported source types are Amazon Web Services *(AWS)* account and OpenShift Container Platform *(OCP)*. Each provider requires special configuration in order to be created. The specifics for adding each provider type are described below along with information to remove a provider.

2.1 Removing a Source

A source can currently be removed by any user.

2.2 Adding an AWS Account

This section describes how to configure your AWS account to allow Koku access. Configuring your account involves configuring three AWS services. Setting up the AWS account for cost and usage reporting to an S3 bucket to make the cost and usage data available. Creating an Identity Access Management (IAM) Policy and Role to be utilized by Koku to process the cost and usage data. AWS organization setup in order to capture member account names when using consolidated billing (optional).

2.2.1 Configuring an Amazon Account for Cost & Usage Reporting

Follow the instructions described in the Amazon article, Setting Up an Amazon S3 Bucket for AWS Cost and Usage Reports. When creating a report choose the following options:

- **Report name:** koku
- **Time unit:** Hourly
- **Include:** Resource IDs
- **Enable support for...:** Redshift, QuickSight
- **Report path prefix:** *(leave blank)*

Make a note of the name of the S3 bucket as you will need it when creating an IAM policy and creating the provider.

2.2.2 Creating an IAM Policy and Role for Cost & Usage Consumption

In order for Koku to provide data within the web interface and API it must be able to consume the cost and usage reports produced by AWS. In order to obtain this data with the minimal amount of access follow the steps below to create an IAM Policy and Role for Koku to utilize.

Sign in to the AWS Management Console as an administrator of the account you wish to add, and open the IAM console at https://console.aws.amazon.com/iam/.

Creating an IAM Policy

1. Browse to the *Policies* tab.
2. Choose *Create Policy*.
3. Select the *JSON* tab and paste the following JSON policy into the editor replacing *bucket_name* with the S3 bucket that was configured in the *Configuring an Amazon Account for Cost & Usage Reporting* section.
4. Including Action *iam:ListAccountAliases* will allow Koku to display the AWS account alias rather than the account id.
5. Including Actions *organization:List** and *organizations:Describe** will allow Koku to obtain the display names of AWS member accounts if you are using consolidated billing rather than the account id.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::bucket_name",
                "arn:aws:s3:::bucket_name/*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "organizations:List*",
                "organizations:Describe*",
                "s3:ListAllMyBuckets",
                "iam:ListAccountAliases",
                "s3:HeadBucket",
                "cur:DescribeReportDefinitions"
            ],
            "Resource": "*"
        }
    ]
}
```

1. Choose the *Review Policy* button and complete the creation of the policy by naming the policy `CostUsage`.

Creating an IAM Role

1. Browse to the *Roles* tab.
2. Choose *Create role*.
3. Select the type of trusted entity as *Another AWS Account*.
4. Provide the Account ID `589173575009`, no options need to be selected.
5. Move to the *Permissions* selection and search for the policy just created `CostUsage`.
6. Choose the *Review* button and complete the creation fo the role by naming the role `CostManagement`.
7. Select the newly created role `CostManagement` to view the summary.
8. Capture the *Role ARN* as it will be used in the provider creation.

2.2.3 Create an AWS Account Source

Using the information from the steps above which included the *S3 bucket name* and the *Role ARN* you can create an AWS account provider. Once created the cost and usage data will be processed and made viewable within the system.

2.3 Adding an OCP Source

This section describes how to configure your Openshift Container Platform (OCP) cluster to provide Koku operator metering usage data. Configuring your OCP cluster involves configuring four setup steps. Obtaining a login token for your reporting-operator service account. Downloading our Ansible playbook and performing the setup step to configure connectivity to the OCP cluster and generate report resources. Create a cron job that will collect the operator metering usage data on an interval and send this data to our upload service. Create an OCP source in Koku.

2.3.1 Dependencies

We require the following dependencies in order to obtain the OCP operator metering usage data.

- You must be running OCP version 3.11 or newer.
- You must install Operator Metering on your OCP cluster.
- The Operator Metering installation must expose a route for the reporting operator.
- You must install and configure the Red Hat Insights Client on a system with network access to you OCP cluster.
- You must install Ansible and the EPEL repository on the system where the Red Hat Insight Client is installed.
- You must install the OCP commandline, oc, on the system where the Red Hat Insighs Client is installed.

2.3.2 Obtaining login credentials

During the installation process for Operator Metering a service account, *reporting-operator*, is created in the Operator Metering namespace. Login to your OCP cluster with a user that has access to the Operator Metering namespace (e.g. *metering*), like a sysadmin.

```
# oc login

# oc project metering
```

After logging into the cluster you can obtain the login token for the *reporting-operator* service account with the following command which will place the token into a file, *ocp_usage_token*:

```
# oc serviceaccounts get-token reporting-operator > ocp_usage_token
```

The token for the *reporting-operator* service account does not expire, as such the token file should be placed on file system with limited access to maintain security This service account token will be used to obtain metering data on a scheduled basis using a cron job.

2.3.3 Download and Configure OCP Usage Collector (Korekuta)

The OCP Usage Collector is an Ansible playbook wrapped in a minimal script to help provide simpler usage. The OCP Usage Collector has two phases, setup and collect. During the *setup* phase configuration is stored for connectivity to the OCP cluster and usage reports resources are created to enable the on-going collection of usage data. During the *collect* phase usage data is retrieved from the Operator Metering endpoint and compressed into a package that is uploaded for processing by Koku via the Red Hat Insights Client.

You can download this tool with your browser or via the command line with the following command:

```
# curl -LOk https://github.com/project-koku/korekuta/archive/master.zip
```

The OCP Usage Collector should be downloaded on the same system where the Red Hat Insights Client was installed.

Once download you can unzip the tool and open the created directory:

```
# unzip master.zip

# cd korekuta-master
```

In the directory you will find the *ocp_usage.sh* script, this script will be used to run both phases of the OCP Usage Collector. In order to configure the tool you need to the following information:

- OCP API endpoint (e.g. https://api.openshift-prod.mycompany.com)
- OCP *reporting-operator* token file path
- OCP Operator Metering namespace (e.g. metering)
- The URL of the route exposed for the reporting operator in the Operator Metering namespace (e.g. https://metering.metering.api.ocp.com)
- Sudo password for installing dependencies

Now you can trigger the setup of the tool providing the above data as seen in the example below:

```
# ./ocp_usage.sh --setup -e OCP_API="https://api.openshift-prod.mycompany.com"  -e␣
↪OCP_METERING_NAMESPACE="metering" -e OCP_TOKEN_PATH="/path/to/ocp_usage_token" -e␣
↪METERING_API="https://metering.metering.api.ocp.com"
```

You will be prompted for your sudo password and the Ansible playbook will execute to capture the configuration information and create the usage reports on your OCP cluster. When complete you will see the following message:

```
TASK [setup : Display New Cluster Identifier] *********************************
ok: [localhost] => {
    "msg": "Use the following value, <YOUR_OCP_IDENTIFIER>, for the cluster␣
↪identifier when configuring an OCP source in Cost Management."
}
```

Record the cluster identifier noted in this step. This value is also stored in a configuration file, *config.json*, located in your *~/.config/ocp_usage/* directory.

Note: The OCP Usage Collector defaults the OCP command line, oc, to exist at */usr/bin/oc*. If the oc command line is installed in a different location then you can supply the *-e OCP_CLI=</path/to/oc>* when executing the *ocp_usage.sh* command.

2.3.4 Uploading data with OCP Usage Collector (Korekuta)

As mentioned earlier during the *collect* phase of the OCP Usage Collector usage data is retrieved from the Operator Metering endpoint and compressed into a package that is uploaded for processing by Koku via the Red Hat Insights Client. Collection of data is performed via the *ocp_usage.sh* script as follows:

```
./ocp_usage.sh --collect --e OCP_CLUSTER_ID=<YOUR_OCP_IDENTIFIER>
```

The command above would perform a one time extraction of usage data based on the defined report created during the setup step above. In order to get on-going usage data this command must be run on a regular interval. Uploading the data on a regular basis will be achieved utilizing a cron job. Use the following command to edit the crontab for the user that will execute this scheduled upload:

```
# crontab -u <username> -e
```

Note: The crontab user must have access to the file with *reporting-operator* token.

Create an entry to run the OCP Usage collector every 45 minutes:

```
 */45 * * * * /path/to/ocp_usage.sh
--collect --e OCP_CLUSTER_ID=<YOUR_OCP_IDENTIFIER>
```

Note: The cron user will also need sudo authority to interact with the Red Hat Insights Client. Below is an example of the addition need to the */etc/sudoers* file to provide password-less sudo for an example user *ocpcollector*:

```
ocpcollector    ALL=(ALL)    NOPASSWD: ALL
```

Note: The OCP Usage Collector defaults the OCP command line, oc, to exist at */usr/bin/oc*. If the oc command line is installed in a different location then you can supply the *-e OCP_CLI=</path/to/oc>* when executing the *ocp_usage.sh* command.

2.3.5 Create an OCP Source

Using the information from the steps above which included the *cluster identifier* you can create an OCP source. Once created the cost and usage data will be processed and made viewable within the system.

2.4 Adding an Azure Account

This section describes how to configure your Azure account to allow Koku access. Configuring your account involves setting up a Service principal account, creating a storage account, and scheduling cost exports.

2.4.1 Creating a Service principal

Follow the instructions described in the Azure article, How to: Use the portal to create an Azure AD application and service principal that can access resources .

The following are details needed when following the linked Azure guide to create a service principal account.

Sections:

Create an Azure Active Directory application

Assign the application to a role:

- Select **Subscriptions** from the left hand navigation list and take note of the desired Subscription ID.
- Select **Access control (IAM)** in secondary panel.
- Select **Add role assignment** in the right panel.
- Set role to **Storage Blob Data Reader**
- Leave default access.
- Type display name created in previous steps.

Get values for signing in

- Select **Azure Active Directory** in the left hand panel.
- Select **App registrations** from secondary panel.
- Select newly created registration.
- Make note of the App Client ID, dir, and tenant id.

Certificates and secrets

- Select **Certificates & secrets** in the secondary panel.
- Add description and click **Add**.
- Make note of your secret.

Make a note of the name of the client id, dir, tenant id, and client secret. They will be needed when creating a provider.

2.4.2 Creating a storage account

Follow the instructions in the Azure article, Create a storage account .

Make a note of the resource group and storage account name. They will be needed when creating a provider.

2.4.3 Setup Azure cost export schedule

Follow the instructions in the azure article, Tutorial: Create and manage exported data .

Ensure that the export type is **Daily export of billing-period-to-date costs**

2.4.4 Create an Azure Account Source

Using the information from the steps above which included the *Client ID, Directory, Tenant ID, Client Secret, Resource Group name, and storage account name* you can create an Azure account provider. Once created the cost export data will be processed and made viewable within the system.

2.5 Adding a Local Source

This section describes the local sources that are used for development and testing purposes. Local sources give Koku the ability to access test data without requiring a dependency on an external service, such as AWS S3 or a pre-existing OpenShift cluster (OKD).

2.5.1 AWS Local Source

These steps will allow you to configure a local source for a Koku instance deployed into an OpenShift environment.

1. Deploy an OKD cluster: `make oc-up`
2. Deploy Koku into the cluster: `make oc-create-koku`
3. **Add a persistent volume to the Koku pod** Mount point: `/tmp/koku`
4. Wait for Koku to redeploy to verify the persistent volume is available.
5. Use Nise to generate test data.
6. Upload the test data to the Koku pod: `oc rsync <data_on_host> <koku_pod>:/tmp/masu`
7. **Create the local source in the Koku app.** Source Resource Name: `arn:aws:iam::111111111111:role/LocalAWSSource` Source Type: `AWS` Bucket: `/tmp/koku/local/<report_name>`

Once configured, you should be able to use Masu's download endpoint to queue tasks to download and process the local source's data.

2.6 Running Openshift on AWS

If sources have been successfully configured for AWS and OpenShift and the OpenShift cluster is running on AWS, then the AWS cost and usage related to running the OpenShift cluster can be automatically tracked. By default, AWS compute usage and costs are tracked by associating the Amazon EC2 instance id with the OpenShift node running on that instance. Further association can be made by associating AWS tags with OpenShift labels.

2.7 Matching AWS Tags with OpenShift Labels

Resource tagging can also be used to identify AWS resources associated with an OpenShift cluster running on AWS. The matched AWS resources need not be running explicitly within the OpenShift cluster, but can be any AWS resource that allows tagging. For example an AWS RDS instance would not run inside OpenShift, but the cost of running RDS may be associated with OpenShift projects in the cluster. If an AWS user-defined tag matches an OpenShift label, that RDS instance usage and cost can be assocaited with the OpenShift cluster.

AWS documents the process of creating user-defined tags and activating them for cost and usage reporting here: https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/custom-tags.html.

Information about creating OpenShift labels can be found here: https://docs.okd.io/latest/architecture/core_concepts/pods_and_services.h

In order for an AWS tag to match an OpenShift label it must meet one of the following criteria.

1. The lowercase version of the AWS tag key must match the lowercase version of the OpenShift label key. The lowercase version of the AWS tag value must match the lowercase version of the OpenShift label value.

   The following examples would meet this requirment. (AWS key:value) == (OpenShift key:value) environment:production == environment:production Environment:production == environment:production Environment:Production == environment:PRODUCTION

2. The lowercase version of the AWS tag key is *openshift_cluster* and the lowercase value of the tag matches the lowercase OpenShift cluster name.

   The following example would meet this requirment. (AWS key:value) (OpenShift cluster name)
   Openshift_cluster:Openshift_1 == Openshift_1

3. The lowercase version of the AWS tag key is *openshift_project* and the lowercase value of the tag matches a lowercase OpenShift project name in the cluster.

   The following example would meet this requirment. (AWS key:value) (OpenShift project name)
   Openshift_Project:Cost_Management == Cost_Management

4. The lowercase version of the AWS tag key is *openshift_node* and the lowercase value of the tag matches a lowercase OpenShift node name in the cluster.

   The following example would meet this requirment. (AWS key:value) (OpenShift node name)
   Openshift_node:compute_1 == COMPUTE_1

If an AWS resource tag matches with multiple OpenShift projects, the cost and usage of that resource are split evenly between the matched projects. Note that this is not the case with AWS compute resources that are matched via the instance id - node relationship. In that case cost and usage are broken down using information about a project's resource consumption within the OpenShift environment.

2.8 Controling Visibility of Resources

In certain cases you may not wish users to have access to all cost data, but instead only data specific to their projects or organization. Using role based access control you can limit the visibility of resources involved in cost management reports. Role based access control utilizes groups of which users are members. These groups can be associated with one or more policies. A policy associates a group with a set of roles. A role defines a permission and a set of resource definitions. By default a user who is not an account administrator will not have access to data, but instead must be granted access to resources. Account administrators can view all data without any configuration within the role based access control.

Reports involving Amazon Web Services can be scoped by usage account or its account alias. For example if an administrator has configured eight Amazon accounts (sources) specific to different organizations within the company, but only wished a user to be able to see the data associated with their organization/account; role based access control can be utilized to filter the data set to only that specific account. This can be accomplished by creating a group that encompasses the users for that organization. A role can be defined that allows "read" permission on the specified Amazon account. Lastly a policy can be created that associates the group and role. Now any user in that group will have visibility limited to just that Amazon account.

Reports involving OpenShift can be scoped by the cluster, node, and project. In order to see report data users must have visibility to a cluster. Visibility can be further limited by specifying nodes or projects. For example an administrator might have two clusters configured (one for pre-production and one for production) and they wish to expose the cost of projects deployed on these clusters to the associated development teams. Groups would be created for each development team. Roles would be created to give "read" access to each of the clusters but only the specific project. Lastly a policy would be created to associate a specific group with the role that provides them visibility to only their project.

Reports involving OpenShift running on Amazon Web Services require visibility to be setup on both the Amazon account and the OpenShift resources as described above.

CHAPTER 3

Development Information

3.1 Koku README

3.1.1 About

Koku's goal is to provide an open source solution for cost management of cloud and hybrid cloud environments. This solution is offered via a web interface that exposes resource consumption and cost data in easily digestible and filterable views. The project also aims to provide insight into this data and ultimately provide suggested optimizations for reducing cost and eliminating unnecessary resource usage.

Full documentation is available through readthedocs.

To submit an issue please visit https://issues.redhat.com/projects/COST/

3.1.2 Getting Started

This project is developed using Python 3.6. Make sure you have at least this version installed.

Prerequisites

- Docker
- PostgreSQL

For Mac OSX

Install Docker for Mac

Install brew

Install PostgreSQL:

```
brew install postgresql
```

3.1.3 Development

To get started developing against Koku you first need to clone a local copy of the git repositories.

```
git clone https://github.com/project-koku/koku
git clone https://github.com/project-koku/nise
```

This project is developed using the Django web framework. Many configuration settings can be read in from a `.env` file. To configure, do the following:

1. Copy `.env.example` into a `.env`
2. Obtain AWS values and update the following in your `.env`:

```
AWS_ACCESS_KEY_ID=YOUR_AWS_ACCESS_KEY
AWS_SECRET_ACCESS_KEY=YOUR_AWS_SECRET_KEY
AWS_RESOURCE_NAME=YOUR_COST_MANAGEMENT_AWS_ARN
```

3. (Mac Only) If you are on Mac, do the following note that psycopg2 is a dependency of Django and installing the psycopg2 wheel will likely fail. The following steps should be taken to allow installation to succeed:

```
brew install openssl
brew unlink openssl && brew link openssl --force
```

4. (Mac Only) Also add the following to your `.env`:

```
LDFLAGS="-L/usr/local/opt/openssl/lib"
CPPFLAGS="-I/usr/local/opt/openssl/include"
```

5. Developing inside a virtual environment is recommended. A Pipfile is provided. Pipenv is recommended for combining virtual environment (virtualenv) and dependency management (pip). To install pipenv, use pip

```
pip3 install pipenv==2018.11.26
```

6. Then project dependencies and a virtual environment can be created using

```
pipenv install --dev
```

7. If dependency installation still fails, try using

```
pipenv install --dev --sequential
```

8. To activate the virtual environment run

```
pipenv shell
```

9. Install the pre-commit hooks for the repository

```
pre-commit install
```

Developing with Docker Compose

This will explain how to start the server and its dependencies using Docker, create AWS/OCP sources, and view reports. This will not cover all API or scenarios but should give you an end to end flow.

Starting Koku using Docker Compose

1. Start the docker containers:

```
make docker-up
```

2. Display log output from the docker containers. It is recommended that logs be kept in a second terminal

```
docker-compose logs -f koku-server koku-worker
```

3. Install koku-nise:

```
pip install koku-nise
```

Run AWS Scenario

1. Create AWS Source:

```
make aws-source aws_name=AWS-SOURCE-001 bucket=cost-usage-bucket
```

2. Verify source exists by visiting http://127.0.0.1:8000/api/cost-management/v1/sources/?name=AWS-SOURCE-001
3. Trigger MASU processing by visiting http://127.0.0.1:5000/api/cost-management/v1/download/
4. Wait for processing to complete
5. Verify data existing using AWS API endpoints
    - http://127.0.0.1:8000/api/cost-management/v1/reports/aws/instance-types/
    - http://127.0.0.1:8000/api/cost-management/v1/reports/aws/costs/
    - http://127.0.0.1:8000/api/cost-management/v1/reports/aws/storage/

Run OCP Scenario

1. Create OCP Source:

```
make ocp-source-from-yaml cluster_id=my_test_cluster srf_yaml=../nise/example_ocp_
↪static_data.yml ocp_name=my_ocp_name
```

2. Verify provider exists by visiting http://127.0.0.1:8000/api/cost-management/v1/sources/?name=my_ocp_name
3. Trigger MASU processing by visiting http://127.0.0.1:5000/api/cost-management/v1/download/
4. Wait for processing to complete
5. Verify data exists using API endpoints
    - http://127.0.0.1:8000/api/cost-management/v1/reports/openshift/volumes/

- http://127.0.0.1:8000/api/cost-management/v1/reports/openshift/memory/
- http://127.0.0.1:8000/api/cost-management/v1/reports/openshift/compute/

Stopping Koku using Docker Compose

To bring down all the docker containers, run the following command:

```
make docker-down
```

Database

PostgreSQL is used as the database backend for Koku. A docker-compose file is provided for creating a local database container. Assuming the default .env file values are used, to access the database directly using psql run

```
PGPASSWORD=postgres psql postgres -U postgres -h localhost -p 15432
```

Note: There is a known limitation with docker-compose and Linux environments with SELinux enabled. You may see the following error during the postgres container deployment:

```
"mkdir: cannot create directory '/var/lib/pgsql/data/userdata': Permission denied"␣
↪can be resolved by granting ./pg_data ownership permissions to uid:26 (postgres␣
↪user in centos/postgresql-96-centos7)
```

If you see this error, run the following command (assuming you are at the project top level directory):

```
setfacl -m u:26:-wx ./pg_data
```

See https://access.redhat.com/containers/?tab=overview#/registry.access.redhat.com/rhel8/postgresql-12

Database Query Monitoring

A basic level of query monitoring has been included leveraging a local grafana container which will be built with the *docker-up* make target.

To use the monitor, open a new web browser tab or window and enter the following URL:

http://localhost:3001

You will be presented with the grafana login page. For this monitor, use the following credentials:

```
User: admin
Password: admin12
```

Once you have logged into the server, you will be taken straight to the main dashboard. It will have 5 panels.

Query statistics	
Connect States	Active Queries
Lock Types	Lock Detail

- Query Statistics - The max execution time, the mean execution time, number of calls and the query text
- Connect States - Shows the connection states (active, idle, idle in transaction, etc)
- Active Queries - Shows the approximate run time (based on the probe time) and the query text of queries detected

- Lock Types - Shows the discrete lock types detected during the probe
- Lock Detail - Shows any detail informaiton for the lock and the affected query.

The Query Statistics panel is cumulative. The remaining panels are ephemeral.

Information about PostgreSQL statistics can be found here: https://www.postgresql.org/docs/10/monitoring-stats.html

Information about Grafana dashboards can be found here: https://grafana.com/docs/grafana/latest/features/dashboard/dashboards/

Developing with OpenShift

Our production deployment runs on OpenShift. At times you may need to run on OpenShift if you are working on deployment templates or would like to test in a production like environment. This is a more advanced scenario that many new developers will not need. To learn how to run OpenShift refer to Working with Openshift.

Testing

Koku uses tox to standardize the environment used when running tests. Essentially, tox manages its own virtual environment and a copy of required dependencies to run tests. To ensure a clean tox environment run

```
tox -r
```

This will rebuild the tox virtual env and then run all tests.

To run unit tests specifically:

```
tox -e py36
```

To run a specific subset of unit tests, you can pass a particular module path to tox. To do this, use positional args using the – separator. For example:

```
tox -e py36 -- masu.test.external.downloader.azure.test_azure_services.
↪AzureServiceTest
```

To run IQE Smoke, Vortex or API tests, while on the Red Hat network and koku deployed via docker-compose run:

```
make docker-iqe-smokes-tests
make docker-iqe-vortex-tests
make docker-iqe-api-tests
```

Individual IQE tests can be ran with run_test.sh:

```
<koku_topdir>/testing/run_test.sh iqe tests plugin hccm -k test_api_cost_model_markup_
↪calculation_ocp
```

Linting

This repository uses pre-commit to check and enforce code style. It uses Black to reformat the Python code and Flake8 to check it afterwards. Other formats and text files are linted as well.

To run pre-commit checks:

```
pre-commit run --all-files
```

pgAdmin

If you want to interact with the Postgres database from a GUI:

1. Copy the *pgadmin_servers.json.example* into a *pgadmin_servers.json* file and if necessary, change any variables to match your database.
2. *docker-compose up* causes pgAdmin to run on http://localhost:8432
3. In the login screen, the default login email is *postgres*

Side note: The *pgadmin_servers.json* file uses [pgadmin servers.json syntax](https://www.pgadmin.org/docs/pgadmin4/development/import_export_servers.html#json-format)

3.1.4 Contributing

Please refer to Contributing.

3.2 Contributing to Koku

Thank you for your interest in contributing to this project!

The following are a set of guidelines for contributing to Koku. These are guidelines, not rules. Use your best judgement. Feel free to suggest changes to this document in a pull-request.

This document uses RFC 2119 keywords to indicate requirement levels.

3.2.1 Reporting Bugs & Requesting Features

We use Github Issues to track bug reports and feature requests.

When submitting a bug report, please be as detailed as possible. Include as much of these items as you have:

1. steps to reproduce the bug
2. error messages with stacktraces
3. logs
4. any relevant configuration settings
5. environment details

When submitting a feature request, please submit them in the form of a user story with acceptance criteria:

> As a [user], I want [a thing], So that [some goal].
>
> When complete, I will be able to:
>
> 1. [do this]
> 2. [do that]
> 3. [do another]

3.2.2 Contributing Code (Pull Requests)

All code contributions MUST come in the form of a pull-request. Pull-requests will be reviewed for a variety of criteria. This section attempts to capture as much of that criteria as possible.

Certificate of Origin

By contributing to this project you agree to the Developer Certificate of Origin (DCO). This document was created by the Linux Kernel community and is a simple statement that you, as a contributor, have the legal right to make the contribution. See the DCO file for details.

3.2.3 Readability and Style considerations

In general, we believe that code is read just as much as it is executed. So, writing readable code is just as important as writing functional code.

Pull-requests MUST follow Python style conventions (e.g. PEP 8 and PEP 20 and conform to generally recognized best practices. Pull-requests MAY also choose to conform to additional style guidelines, e.g. Google's Python Style Guide.

We do use automation whenever possible to ensure a basic level of acceptability. Pull-requests MUST pass a linter (flake8) without errors.

We do recognize that sometimes linters can get things wrong. They are useful tools, but they are not perfect tools. Pull-requests SHOULD pass a linter without warnings.

Pull-requests MAY include disabling a specific linter check. If your pull-request disables linting it MUST include a comment block detailing why that particular check was disabled and it MUST be scoped as narrowly as possible. i.e. Don't disable linting on an entire class or method when disabling the check for a single statement will do.

This repository uses pre-commit to check and enforce code style. It uses Black to reformat the Python code and Flake8 to check it afterwards. Other formats and text files are linted as well.

Install pre-commit hooks to your local repository by running:

```
`bash $ pre-commit install `
```

After that, all your commited files will be linted. If the checks don't succeed, the commit will be rejected. Please make sure all checks pass before submitting a pull request.

3.2.4 Code testing considerations

We believe that well-tested code is a critical component to every successful project. For this reason, all pull-requests MUST include unit test cases and those unit tests MUST pass when run.

The unit tests SHOULD cover all of the code in the pull-request. Our goal is to maintain at least 90% test coverage.

In general, the test cases SHOULD cover both success and failure conditions.

An attempt SHOULD be made to cover all code branches. You SHOULD also attempt to include tests for all class and method parameters. e.g. If a method accepts a boolean, there should be tests for when that boolean is True, False, and None.

3.3 Developer Tools

This section describes tooling and features included in Koku to assist developers contributing to Koku.

3.3.1 Environment Variables

Koku makes use of environment variables to configure features and application capabilities.

In the repository, there is an `.env.example` file with sample environment settings. The `.env` file is used by Django's tools as well as Koku's scripting. It is the recommended way to configure your local development environment settings.

This section documents environment variables that may be of interest to developers.

`DEVELOPMENT=(True|False)` - Enables development features. Not for production use.
`DEVELOPMENT_IDENTITY=(JSON)` - A JSON Object representing a User

3.3.2 Authentication

When `DEVELOPMENT` is not set, Koku expects to use an external service to handle authentication and access control for most API endpoints.

When `DEVELOPMENT` is set, the development middleware is enabled. This allows passing in custom identity information into Koku for development and testing purposes using the `DEVELOPMENT_IDENTITY` variable.

Example DEVELOPMENT_IDENTITY object:

```
{
    "identity": {
        "account_number": "10001",
        "type": "User",
        "user": {
            "username": "user_dev",
            "email": "user_dev@foo.com",
            "is_org_admin": False
            "access": {
                "aws.account": {
                    "read": ["1234567890AB", "234567890AB1"]
                }
                "azure.subscription_guid": {
                    "read": ["*"]
                }
                "openshift.cluster": {
                    "read": ["*"]
                }
                "openshift.project": {
                    "read": ["*"]
                }
                "openshift.node": {
                    "read": ["*"]
                }
            }
        },
    },
    "entitlements": {"cost_management": {"is_entitled": True}},
}
```

Note: This example is pretty-printed for readability. When setting the enviroment variable, it should be collapsed to one single line.

3.4 Developing using OpenShift

The recommended development workflow is to develop using the same tools the application uses as its target environment. In this case, Koku is intended for use inside an OpenShift deployment. Therefore, it is recommended that Koku development also use an OpenShift deployment. Developing using OpenShift will align not only the software architecture, but also ensures the developer builds familiarity with the toolchain.

3.4.1 Prerequistes

Developing Koku using OpenShift requires prerequisite knowledge and workstation configuration. Please ensure that you are familiar with the following software and have configured your workstation accordingly.

- OpenShift
- Kubernetes
- Docker

When ready, your workstation should be able to run containers and deploy OpenShift, either using minishift or an alternative installer.

3.4.2 Local Development

Minishift (OKD 3.11)

The recommended way to deploy a local OpenShift 3.x installation on Linux for Koku development is to use minishift. This runs an OpenShift cluster inside of a VM.

Installing and configuring minishift is outside the scope of this document. Please refer to the minishift documentation for details.

In order to access RHEL images for building Koku, you must configure Red Hat Registry Authentication:

For username/password, you can use the minishift's `redhat-registry-login` addon:

:: minishift addons enable redhat-registry-login minishift addons apply redhat-registry-login –addon-env REGISTRY_USERNAME=${USERNAME} –addon-env REGISTRY_PASSWORD=${PASSWORD}

For token-based authentication, you will need to configure the secret manually in your project:

:: # this extracts the nested object from the file distributed by https://access.redhat.com/terms-based-registry cat /path/to/registry-pull-secret.yaml |

python -c 'import yaml, sys; print(yaml.safe_load(sys.stdin).get("data").get(".dockerconfigjson"))' | base64 -d | oc create secret generic registry-redhat-io-secret

–from-file=.dockerconfigjson=/dev/stdin -n myproject –type=kubernetes.io/dockerconfigjson

oc secrets link default registry-redhat-io-secret -n myproject –for=pull oc secrets link builder registry-redhat-io-secret -n myproject

CodeReady Containers (OKD 4.x)

The recommended way to deploy a local OpenShift 4.x installation on Linux for Koku development is to use crc. This runs an OpenShift cluster inside of a VM.

Installing and configuring crc is outside the scope of this document. Please refer to the crc documentation for details.

In order to access RHEL images for building Koku, you must configure Red Hat Registry Authentication.

Running locally in OpenShift using e2e-deploy

The script `scripts/e2e-deploy.sh` handles setup and configuration of crc or OKD 3.11, including Red Hat Registry Authentication. To use the script, complete the following steps.

1. First, make sure that you have cloned and followed the setup instructions of the following repos:

```
- https://github.com/project-koku/koku
- https://github.com/RedHatInsights/e2e-deploy
- https://gitlab.cee.redhat.com/insights-qe/iqe-tests.git
```

2. Next, make sure that you have the following tools installed:

```
- oc
- ocdeployer
- iqe
- python
- base64
```

3. Finally, either export the following environment variables or add them to a .env file and enter the IQE virtual environment:

```
OPENSHIFT_API_URL=YOUR_OPENSHIFT_API_URL
REGISTRY_REDHAT_IO_SECRETS=PATH_TO_RH_REGISTRY_YAML
E2E_REPO=PATH_TO_LOCAL_E2E_REPO
```

If you do not have access to some of the repositories or resources discussed in this document, please contact a member of the Koku development team. Some resources mentioned are internal to Red Hat Associates. If you are unable to access those resources, we will work with you to identify suitable alternatives.

4. Once the IQE virtual environment is activated, change directories to the scripts folder inside of the koku repo (`koku/scripts`). Make sure that you have a running OpenShift cluster before proceeding. The setup script handles all configuration once the cluster is running. With your OpenShift cluster running, execute the e2e-deploy script. This will set up three projects (`secrets`, `buildfactory`, & `hccm`); it will pull the required imagestreams and build the container images for koku; once the container pods are built, the script will deploy the project.

Note: If you are getting intermittent deployment failures that don't have an obvious root cause in the pod's runtime logs or in the OpenShift cluster's Event logs, the most common reason is that there are insufficient resources to deploy all pods. Try increasing the memory and CPUs allocated to your openshift cluster or to the individual pods.

5. To delete all of the objects created by running the e2e-deploy script, run `make oc-delete-e2e`.

If you need to test a specific Koku branch using e2e-deploy, edit the `buildfactory/hccm/koku.yaml` template and change the `SOURCE_REPOSITORY_REF` in the parameters to have a value that is set to the name of the branch of Koku that you are testing. For example, if you are testing the branch `issues/123`, the parameter for the `SOURCE_REPOSITORY_REF` should look like the following:

```
- description: Set this to a branch name, tag or other ref of your repository if you
    are not using the default branch.
  displayName: Git Reference
  name: SOURCE_REPOSITORY_REF
  required: false
  value: issues/264
```

Deploying Services

Koku is implemented as a collection of services. During development, it is not required to deploy all services. It is possible to deploy subsets of services based on the focus of the development effort.

The `Makefile` in the Koku git repository provides targets intended to assist with development by enabling deployment and management of Koku's services within a local OpenShift installation. See `make help` for more information about the available targets.

Service Dependencies

- PostgreSQL: the database is required for most Koku services.
- RabbitMQ: the message bus is required for report polling and processing.
- Redis: the key-value store is required for caching credentials from an external authentication service.

OpenShift Templates

OpenShift templates are provided for all service resources. Each template includes parameters to enable customization to the target environment.

The `Makefile` targets include scripting to dynamically pass parameter values into the OpenShift templates. A developer may define parameter values by placing a parameter file into the `koku.git/openshift/parameters` directory.

Examples of parameter files are provided in the `koku.git/openshift/parameters/examples` directory.

The `Makefile` scripting applies parameter values only to matching templates based on matching the filenames of each file. For example, parameters defined in `koku-api.env` are applied *only* to the `koku-api.yaml` template. As a result, common parameters like `NAMESPACE` must be defined consistently within *each* parameter file.

3.4.3 General Platform information

When developing using OpenShift, there are different setup requirements for Linux and Mac OS. Linux instructions are provided for Fedora/RHEL/CentOS.

CLI Tab Completion

The Openshift client (`oc`) does offer shell/tab completion. It can be generated for either bash/zsh and is available by running *oc completion bash|zsh* The following example generates a shell script for completion and sources the file.

```
oc completion zsh > $HOME/.oc/oc_completion.sh
source $HOME/.oc/oc_completion.sh
```

Mac OS

There is a known issue with Docker for Mac ignoring *NO_PROXY* settings which are required for OpenShift. (https://github.com/openshift/origin/issues/18596) The current solution is to use a version of Docker prior to 17.12.0-ce, the most recent of which can be found at docker-community-edition-17091-ce-mac42-2017-12-11

Docker needs to be configured for OpenShift. A local registry and proxy are used by OpenShift and Docker needs to be made aware.

Add *172.30.0.0/16* to the Docker insecure registries which can be accomplished from Docker -> Preferences -> Daemon. This article details information about insecure registries Test an insecure registry | Docker Documentation

Add *172.30.1.1* to the list of proxies to bypass. This can be found at Docker -> Preferences -> Proxies

Troubleshooting

- When running a cluster locally for developement, it is recommended that your workstation can allocate at least 4 GB of memory available for use.
- Accessing the database when it is running inside an OpenShift deployment will require either a remote shell or port forwarding. The `Makefile` provides targets for managing port forwarding.

3.5 Testing Koku

3.5.1 Unit testing

Unit testing for Koku is handled through tox.

Running the test suite requires a minimum of an accessible database and Koku API server.

Unit testing using docker-compose

Example:

```
$ make docker-up
$ tox
$ make docker-down
```

Unit testing using OpenShift

The `Makefile` provides convenience commands for enabling port-forwarding. This is required for accessing a database pod hosted within an OpenShift environment.

Example:

```
$ make oc-create-all
$ make oc-forward-ports
$ tox
$ make stop-forwarding-ports
```

Unit testing log messages

The logger is disabled by default during unit tests. If you are building a unit test that asserts log messages, you must re-enable the logger. For example:

```
import logging
with self.assertLogs(logger='masu.external.downloader.aws.aws_report_downloader',
level='WARN') as cm:
            logging.disable(logging.NOTSET)
            self.aws_report_downloader._remove_manifest_file("None")
            self.assertEqual(['WARN: Could not delete manifest file at'], cm.output)
```

If you observe the following error when running the unit tests, it may be a false error. This can be due to some tests expecting the DEBUG setting to be TRUE:

```
FAIL: test_delete_single_provider_skips_delete_archived_data_if_customer_is_none
(api.provider.test.tests_models.ProviderModelTest)
Assert the delete_archived_data task is not called if Customer is None.
----------------------------------------------------------------------
Traceback (most recent call last):
File "/usr/local/Cellar/python/3.7.4_1/Frameworks/Python.framework/Versions/3.7/lib/
↪python3.7/unittest/mock.py", line 1209, in patched
return func(*args, **keywargs)
File "/Users/nbonilla/Documents/Koku/koku/koku/api/provider/test/tests_models.py",␣
↪line 43, in test_delete_single_provider_skips_delete_archived_data_if_customer_is_
↪none
self.aws_provider.delete()
AssertionError: no logs of level WARNING or higher triggered on api.provider.models
```

Testing using OpenShift

If your test environment is hosted on an OpenShift cluster, you will need to ensure that routes are created to expose the Koku and Masu API endpoints.

You can use the `oc expose` command to accomplish this.

Examples

settings.local.yaml:

```
local:
  hccm:
    ocp_dir: /var/tmp/masu/insights_local
    aws_dir: /tmp/local_bucket
    azure_dir: /tmp/local_container
    masu:
      path: api/cost-management/v1
      port: 5000
      hostname: masu-hccm.apps-crc.testing
      scheme: http
  main:
    hostname: koku-hccm.apps-crc.testing
    scheme: http
    default_user: default
    api_path: api/cost-management/v1
  http:
    default_auth_type: basic
  users:
    default:
      username: user_dev@foo.com
      password: password
```

Smoke test:

```
$ cd koku/testing
$ ./run_smoke_tests.sh
```

Run specific unit tests

To run a specific subset of unit tests, you can pass a particular module path to tox. To do this, use positional args using the -- separator. For example:

```
tox -e py36 -- masu.test.external.downloader.azure.test_azure_services.
↪AzureServiceTest
```

The previous command will selectively run only the AzureServiceTest module The following are examples of valid module paths:

- masu.test.external
- masu.test.external.downloader.azure.test_azure_services.AzureServiceTest.specific_test

3.5.2 Functional Testing

It is often necessary to load test data into a local development environment to test functionality of a new feature.

Using default test data

Within the koku repo there are scripts to automate the creation of a test customer with a default set of sources. This will create one AWS, one Azure, and three OpenShift sources to simulate the following scenarios.

- OpenShift running on AWS
- OpenShift running on Azure
- OpenShift on premises

There is a complementary script to subsequently load test data for each source. The nise project is used to generate this test data. Nise is a dev dependency of koku, so a working copy of the nise command line tool shoudl be availble after running *pipenv install –dev*. A local clone of the nise repository is also required as it contains static YAML files used by this script. Before using this script, make sure the following variables are defined in your environment. See .env.example for example values.

- API_PATH_PREFIX
- NISE_REPO_PATH
- KOKU_API_HOSTNAME
- MASU_API_HOSTNAME
- KOKU_PORT (if using docker-compose)
- MASU_PORT (if using docker-compose)

Examples

Example 1. Using docker-compose to wipe and rebuild the local database with test data:

```
make docker-reinitdb-with-sources
make load-test-customer-data
```

Example 2. Assuming the database is already clean and does not need to be rebuilt:

```
make create-test-customer
make load-test-customer-data
```

Example 3. Using custom date ranges:

```
make load-test-customer-data start=2020-01-01 end=2020-02-29
```

Manually loading test data into the database

Before running any functional testing, you will need to populate your database with test data.

Using nise for generating test data is recommended.

3.5.3 Testing with Ingress

It may be necessary to test changes related to Kafka using Ingress in a local development environment to test the functionality of a new feature.

Setting up Ingress

First, you need to obtain the source for the ingress project:

```
git clone https://github.com/RedHatInsights/insights-ingress-go
```

Next, you should add/modify the following variables in the existing `.env` file for the ingress environment:

```
STORAGE_DRIVER=localdisk
ASYNC_TEST_TIMEOUT=10
MINIO_DATA_DIR=/tmp/hccm/mnt/data
MINIO_CONFIG_DIR=/tmp/hccm/mnt/config
INGRESS_VALID_TOPICS=testareno,advisor,hccm
```

Since both Ingress and Koku are ran locally via docker-compose files, we must ensure that all of the services are on the same network. We can do that by creating a network in the Ingress `docker-compose.yml` file and adding it to each of the services in both Ingress and Koku. To create a network, add the following to the Ingress `docker-compose.yml`:

```
networks:
  myNetwork:
    driver: bridge
```

Add `myNetwork` to each of the Ingress services using the following example:

```
kafka:
  image: confluentinc/cp-kafka
  ports:
    - 29092:29092
  depends_on:
    - zookeeper
  environment:
    - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka:29092
    - KAFKA_BROKER_ID=1
    - KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1
```

(continues on next page)

(continued from previous page)

```
    - KAFKA_ZOOKEEPER_CONNECT=zookeeper:32181
    - LISTENER_EXTERNAL=PLAINTEXT://localhost:29092
  networks:
    - myNetwork
```

Now, we must define the external network that we just created in Ingress in the `docker-compose.yml` for koku. Modify the file to include the external network and add it to each service using the kafka example above, noting that the name of the external network is `insightsingressgo_myNetwork`:

```
networks:
  insightsingressgo_myNetwork:
    external: true
```

Next, install the development requirements, enter the pip environment and bring up the ingress service:

```
pipenv install --dev
pipenv shell
docker-compose up --build
```

If necessary, you can bring up a consumer to see the contents of messages that are uploaded to the `hccm` topic using the following command within the ingress environment:

```
docker-compose exec kafka kafka-console-consumer --topic=platform.upload.hccm --
↪bootstrap-server=localhost:29092
```

Finally, you can bring up Koku project via docker-compose and check the koku-listener logs to ensure the listener has successfully connected and is listening for messages.

3.5.4 Debugging Options

PDB in koku container

While koku-server is running in a docker container:

1. Ensure all migrations are run.
2. Stop the server *docker-compose stop koku-server*
3. Run the server with service-ports: `docker-compose run --service-ports koku-server`
4. set a breakpoint using `import pdb; pdb.set_trace()`

PDB in IQE container

Set the environment variable ENV_FOR_DYNACONF=local While IQE (the integration test suite) is running a docker container:

Start a shell session in the docker container that runs IQE:

```
koku/testing/run_test.sh bash
```

The following command runs all QE api tests. The optional `--pdb` flag will cause any failed test to automatically start a pdb session:

```
iqe tests plugin hccm -k test_api --pdb
```

To run a subset of the above tests, for example only smoke tests:

```
iqe tests plugin hccm -k test_api -m hccm_smoke --pdb
```

To run the vortex the tests:

```
iqe tests plugin hccm -k test_api -m qa --pdb
```

To run a specific subset of the integration test suite, you can specify a single test using the `-k` flag. The single test names can be found in the IQE repo. Here is an example of running a single test named `test_api_aws_storage_filtered_top`:

```
iqe tests plugin hccm -k test_api_aws_storage_filtered_top --pdb
```

The single test name above was found in the hccm plugin repo itself at https://gitlab.cee.redhat.com/insights-qe/hccm-plugin/blob/master/iqe_hccm/tests/rest_api/v1/test_aws_storage_reports.py#L245 Any function definition name in this file can be passed in as the parameter for `-k` to run specifically that test. To find more specific test names, search that repo for test function names.

3.5.5 Smoke testing with IQE

Prerequisites:

- koku is running and accessible via the network
- you are connected to the Red Hat internal network

For a quick start on smoke testing, continue to the section Running IQE in Docker below. Otherwise, for more in-depth information on IQE, see https://gitlab.cee.redhat.com/insights-qe/hccm-plugin/tree/master

Running IQE in Docker

To run IQE Smoke, Vortex or API tests, run one of the following commands, respectively:

```
make docker-iqe-smokes-tests
make docker-iqe-vortex-tests
make docker-iqe-api-tests
```

3.5.6 Testing Performance with cProfile

Tools such as cProfile and pstats can be used to identify potential performance problems within the code base.

Creating the Profiler Code

The cProfile code written in this step will heavily depend on what part of the project's code you are trying to profile. For example, the code below is profiling the `DateHelper` function inside of `koku.api.utils`:

```
import cProfile
profile_text = """
# The section of code you want to profile:
from api.utils import DateHelper
dh = DateHelper
```

(continues on next page)

(continued from previous page)

```
dh.now
dh.now_utc
"""
cProfile.run(profile_text, filename="ouput_filename.file")
```

Runnin the Profiler Code

The profiler code must be executed inside of a django environment in order to import koku's python modules. The profiler code can be directly ran through a shell through the following method:

```
DJANGO_READ_DOT_ENV_FILE=True python koku/manage.py shell
>>> import cProfile
>>> profile_text = """
... from api.utils import DateHelper
... dh = DateHelper
... dh.now
... dh.now_utc
... """
>>> cProfile.run(profile_text, filename="ouput_filename.file")
```

However, if the profiler code is rather large it can be saved into a python script and executed inside of the django environment:

```
DJANGO_READ_DOT_ENV_FILE=True python koku/manage.py shell < profile_code.py
```

Analyzing the Results

After running the profiler code, statistics regarding the code is stored in the filename specified here: `cProfile.run(profile_text, filename="ouput_filename.file")`. Pstats can be used to organize the information inside of this file, for example:

```
import pstats

ps = pstats.Stats("ouput_filename.file")

print('###### All Results #######')
ps.strip_dirs().sort_stats().print_stats()

print('#### SORTED BY TOP 25 CUMULATIVE TIME ####')
ps.strip_dirs().sort_stats('cumtime').print_stats(25)

print('#### SORTED BY TOP 25 TOTAL TIME ####')
ps.strip_dirs().sort_stats('tottime').print_stats(25)

print('#### SORTED BY TOP 25 NUMBER OF CALLS ####')
ps.strip_dirs().sort_stats('ncalls').print_stats(25)
```

CHAPTER 4

Installation

Project Koku is divided into logical components. To achieve a fully functional application, it requires the deployment of multiple micro-applications.

The primary applications for Project Koku are:

- Koku API (Reporting and Query API service)
- Koku UI (Front-end Web UI)
- Masu (Data ingestion service)

For development and testing, Project Koku also includes:

- Nise (Test data generator)

This guide will focus on deploying Project Koku into an existing OpenShift cluster.

4.1 Deploying the Koku API

The Koku application contains two components - a web service and database.

OpenShift

A basic deployment configuration is contained within the application's openshift template file. This template should be acceptable for most use cases. It provides parameterized values for most configuration options.

To deploy the Koku API application using the provided templates, you can use the provided `Makefile`: (the openshift CLI is required to be installed to run the following command)

```
make oc-create-all
```

To deploy individual components, there are also `make` commands provided for your convenience:

Deploy the API web application: `make oc-create-koku` Deploy the PostgreSQL database: `make oc-create-db`

Docker Compose

The Koku API can also be deployed with Docker Compose with the following steps:

- Create a Docker bridge network named `koku-network`: `docker network create koku-network`
- Set AWS credential environment variables: `AWS_SECRET_ACCESS_KEY` and `AWS_ACCESS_KEY_ID`
- Start koku server and database: `make docker-up`

This command will run database migraitons and start the API server. Once complete the API server will be running on port 8000 on your localhost.

4.2 Deploying the Koku UI

The Koku-UI application is the web-based frontend for Project Koku. It is built using Patternfly and ReactJS.

Instructions for deploying the Koku UI can be found here: https://github.com/project-koku/koku-ui#getting-started

4.3 Deploying Masu

The Masu application contains several components - a web service, message bus, and workers. Masu also uses the Koku database. Configuration and management of the database are controlled from the Koku API application.

Instructions for deploying Masu can be found here: https://github.com/project-koku/masu#getting-started

Masu can alternatively be deployed with Docker Compose: Follow the steps above to start the Koku API with Docker Compose

Start masu services: `masu_branch> make docker-up` To trigger Masu to process data, send a GET request to http://<baseUrl>:5000/api/cost-management/v1/download/ You may want to see Masu logs using *docker logs koku_worker -f*

CHAPTER 5

Indices and tables

- genindex
- modindex
- search