Cyclops Documentation

Release 3.0

CAB Team@SPLab/ICCLab - ZHAW

Jun 27, 2019

Contents:

1 Building Cyclops **3**
1.1 Building from source 3

2 Preparing the host **7**
2.1 Adding a dedicated user 7
2.2 Optional softwares 7
2.3 System folders 7
2.4 Bootstrapping Postgresql 8
2.5 Configuring RabbitMQ 8

3 Install & Configure: UDR **11**
3.1 Preparing the host machine 11
3.2 Preparing the Postgressql / TimescaleDB 12
3.3 Preparing RabbitMQ 12
3.4 Configuring UDR 13
3.5 Fixing permissions 14
3.6 Setup as a service 14

4 Install & Configure: CDR **15**
4.1 Preparing the host machine 15
4.2 Preparing the Postgressql / TimescaleDB 16
4.3 Preparing RabbitMQ 16
4.4 Configuring CDR 17
4.5 Fixing permissions 18
4.6 Setup as a service 18

5 Install & Configure: Billing **19**
5.1 Preparing the host machine 19
5.2 Preparing the Postgressql / TimescaleDB 20
5.3 Preparing RabbitMQ 20
5.4 Configuring Billing 21
5.5 Fixing permissions 22
5.6 Setup as a service 23

6 Install & Configure: Rule Engine **25**
6.1 Setup & configuration: coincdr 25
6.2 Setup & configuration: coinbill 28

7 Managing Cyclops **33**
7.1 Configuring a collector . . . 33
7.2 Rules management . . . 33
7.3 Generation of a bill . . . 36

8 Advanced customizations **37**
8.1 Working with the dashboard . . . 37
8.2 Advanced RabbitMQ setup . . . 37

9 Rule Versioning **39**
9.1 Setup rule versioning . . . 39
9.2 Setting up checkpoints for the rules . . . 39
9.3 Rolling back rules and affected records . . . 39

10 Forecasting and Estimation Engine **41**
10.1 Per account forecast: (Forecast command) . . . 41
10.2 Model based global forecast: (Forecast command with no account specified) . . . 41
10.3 2D pattern based forecast: (GlobalForecast command) . . . 41
10.4 Evaluation rules/Pricing models under evaluation: . . . 42

11 Indices and tables **43**

Cyclops is an open source, community driven project led by Cloud Accounting and Billing (CAB) initiative @ SPLAB, part of InIT - ZHAW, for creating a flexible accounting and billing framework for IT services. Cyclops has been specifically designed keeping requirements of popular cloud native applications, platforms and services in mind. Widely used platforms such as OpenStack, CloudStack, Apache Hadoop, etc. are already supported, meaning - these can be billed out of box through Cyclops framework via appropriate collectors.

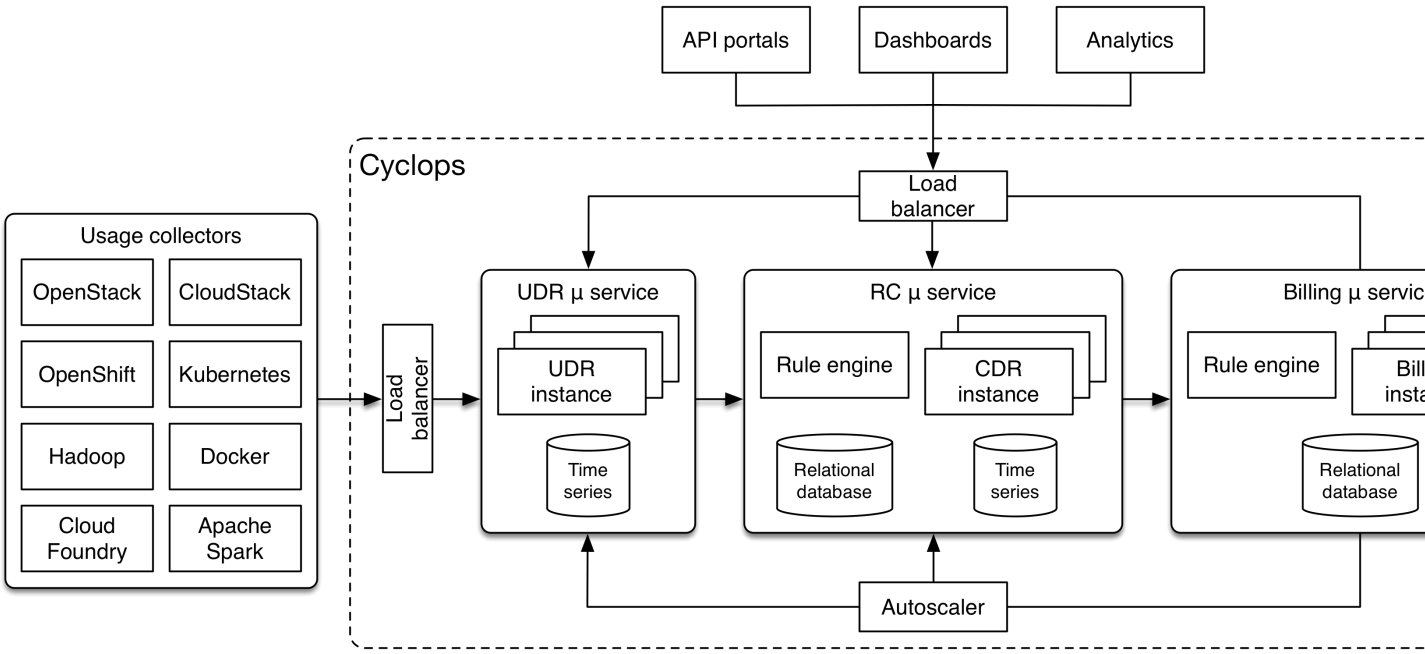

Fig. 1: Figure 1: Cyclops framework architecture (v3.0)

This manual covers only installation and administration of Cyclops installation.

See also:

You may want to read Cyclops's Developer's guide (WiKi) – the first bit, at least – to get an idea of the concepts required for extending the framework.

CHAPTER 1

Building Cyclops

Cyclops framework is made available as a set of docker images, and full source code is available licensed under ASL 2.0.

1.1 Building from source

1.1.1 Requirements

You will need following software packages to be installed before you start the build and setup process from source files:

Dependencies	Supported Version
Java	Oracle Java 8 and higher
Maven	3.0.5 or higher
Git	2.x.x or higher

When deploying, additionally you will need the following services installed and reachable by the Cyclops framework components.

Dependencies	Supported Version
RabbitMQ	3.6
Postgresql	9.6

During configuration and setup phase of all the microservices, it is assumed that the RabbitMQ management plugin is enabled in the service. Post configuration and installation, this plugin can be disabled if not needed elsewhere.

Note: the default package version available under your linux/unix distribution for the above listed dependecies may be different that listed. In that case, please make sure you get the specific versions directly from the respective developer's website.

1.1.2 Download the source

Download the full source code via Git clone

```
git clone https://github.com/icclab/cyclops.git
```

Once source code download finishes, check the folder structure and you should see all the microservices in their separate subfolders. We will now proceed with building every microservice individually.

1.1.3 Building the binaries

Cyclops framework comprises of these micro services:

- usage data record generation microservice (udr)
- rating and charging microservice (cdr)
- billing microservice (billing)
- rule engine (coin)

Each one of the above needs to be built individually. Before proceeding with the build phase of any component, make sure your *JAVA_HOME* environment variable is properly set. On Ubuntu 16.04 machine, this can be normally be done through -

```
source /etc/environment
export JAVA_HOME="/usr/lib/jvm/java-8-oracle"
```

Make sure you change the path appropriately.

Building udr

Change directory to UDR subfolder within *cyclops* folder.

```
mvn dependency:tree
mvn package assembly:single
mv target/cyclops-udr-3.0.0-jar-with-dependencies.jar target/udr.jar
```

The java binary file is located within *cyclops/UDR/target/* as **udr.jar**

Building cdr

Change directory to CDR subfolder within *cyclops* folder.

```
mvn dependency:tree
mvn package assembly:single
mv target/cyclops-cdr-3.0.0-jar-with-dependencies.jar target/cdr.jar
```

The java binary file is located within *cyclops/CDR/target/* as **cdr.jar**

Building billing

Change directory to Billing subfolder within *cyclops* folder.

```
mvn dependency:tree
mvn package assembly:single
mv target/cyclops-billing-3.0.0-jar-with-dependencies.jar target/billing.jar
```

The java binary file is located within *cyclops/Billing/target/* as **billing.jar**

Building rule-engine (coin)

Change directory to Coin subfolder within *cyclops* folder.

```
mvn dependency:tree
mvn package assembly:single
mv target/cyclops-coin-1.1-jar-with-dependencies.jar target/coin.jar
```

The java binary file is located within *cyclops/Coin/target/* as **coin.jar**

CHAPTER 2

Preparing the host

Now that you have successfully compiled and built binaries of each individual Cyclops components, let us understand how to properly install and configure them.

Assumption: An Ubuntu 16.04 OS is installed on nodes where Cyclops framework services will be executed

A few recommended housekeeping steps are receommended before actually starting with the individual service configurations.

2.1 Adding a dedicated user

Since cyclops services will be executed as system processes, it is highly recommended to create a dedicated user to execute these services.

```
sudo useradd -s /sbin/nologin cyclops
```

2.2 Optional softwares

As cyclops framework components generate extensive log messages, it is highly recommended to setup **logrotate** process to ensure log files do not consume the entire usable disk space.

cURL is used to setup RabbitMQ bindings for various Cyclops services later on. It is recommended to install it to avoide setup using the graphical interface.

2.3 System folders

Lets set up appropriate directories to place the binaries, configuration files, and log files.

```
sudo mkdir -p /var/log/cyclops/
sudo mkdir -p /etc/cyclops/
sudo mkdir -p /usr/local/bin/cyclops/
sudo mkdir -p /var/lib/cyclops/
```

We will move the compiled binaries into */usr/local/bin/cyclops/* subtree, configuration files under */etc/cyclops/* subtree, and the log files will be stored within */var/log/cyclops/* directory subtree. */var/lib/cyclops/* is used in case the services require a folder to store additional files.

2.4 Bootstrapping Postgresql

Please use the following statements to allow Cyclops micro-services to setup service specific tables.

```
sudo -i -u postgres psql -c "alter system set idle_in_transaction_session_timeout=
↪'5min';"
sudo -i -u postgres psql -c "DROP USER IF EXISTS cyclops;"
sudo -i -u postgres psql -c "CREATE USER cyclops WITH PASSWORD 'pass1234';"
sudo -i -u postgres psql -c "ALTER USER cyclops CREATEDB;"
sudo -i -u postgres psql -c "CREATE DATABASE cyclops;"
sudo -i -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE cyclops TO cyclops;"
```

Please set a reasonably strong password while creating a Cyclops DB user

2.5 Configuring RabbitMQ

Since Cyclops services uses RabbitMQ for inter-process communication, it is important that the messaging system is already preconfigured to enable communication.

```
sudo rabbitmq-plugins enable rabbitmq_management
sudo rabbitmqctl add_user cyclops pass1234
sudo rabbitmqctl set_user_tags cyclops administrator
sudo rabbitmqctl add_vhost cyclops
sudo rabbitmqctl set_permissions -p cyclops cyclops ".*" ".*" ".*"
```

Please set a reasonably strong rabbitmq user password

For sake of ease, we will continue using *pass1234* in subsequent pages, do replace it with the actual value that was used instead. Figure 1 shows the global exchange and queue bindings maps and relationship between various Cyclops framework services.

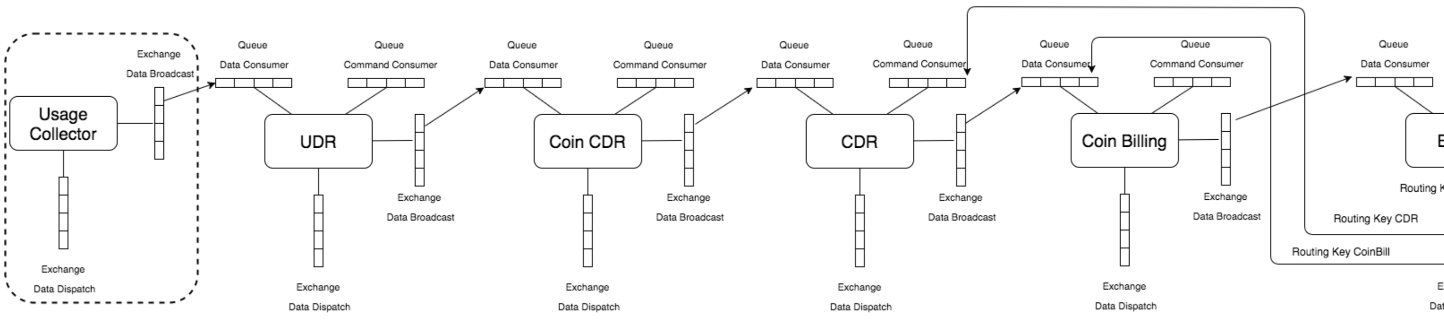

Fig. 1: Figure 1: Global bindings map and relations with Cyclops services

The above bindings will be setup part by part while setting up respective services. Follow through the guide for installing each service individually.

CHAPTER 3

Install & Configure: UDR

Let us setup UDR micro-service to run as a linux system service.

3.1 Preparing the host machine

Start by creating system folders for UDR service.

```
sudo mkdir -p /var/log/cyclops/udr/
sudo mkdir -p /etc/cyclops/udr/
sudo mkdir -p /usr/local/bin/cyclops/udr/
```

For logging to work properly, these files must exist, perform the next commands to ensure the same.

- errors.log
- trace.log
- rest.log
- dispatch.log
- data.log
- commands.log
- timeseries.log

```
sudo touch /var/log/cyclops/udr/errors.log
sudo touch /var/log/cyclops/udr/trace.log
sudo touch /var/log/cyclops/udr/rest.log
sudo touch /var/log/cyclops/udr/dispatch.log
sudo touch /var/log/cyclops/udr/data.log
sudo touch /var/log/cyclops/udr/commands.log
sudo touch /var/log/cyclops/udr/timeseries.log
```

Let's move the binary and the configuration files from the compiled locations to the target system destinations.

```
sudo mv UDR/target/udr.jar /usr/local/bin/cyclops/udr/
sudo mv UDR/config/udr.conf /etc/cyclops/udr/
```

3.2 Preparing the Postgressql / TimescaleDB

Before working with the udr service, it is necessary to setup the appropriate database and table schemas. This can be achieved by executing the following commands on the host where the Postgresql service is running.

```
psql -U postgres -h localhost <<EOF
CREATE DATABASE cyclops_udr WITH OWNER cyclops;
GRANT ALL PRIVILEGES ON DATABASE cyclops_udr TO cyclops;
EOF
```

```
psql -U cyclops -h localhost -d cyclops_udr <<EOF
CREATE TABLE IF NOT EXISTS usage (
  time      TIMESTAMP         NOT NULL,
  metric    TEXT              NOT NULL,
  account   TEXT              NOT NULL,
  usage     DOUBLE PRECISION  NOT NULL,
  data      JSONB,
  unit      TEXT
);
CREATE INDEX IF NOT EXISTS usage_metric ON usage (metric, time DESC);
CREATE INDEX IF NOT EXISTS usage_account ON usage (account, time DESC);
CREATE INDEX IF NOT EXISTS usage_unit ON usage (unit, time DESC);
CREATE INDEX IF NOT EXISTS usage_data ON usage USING HASH (data);
EOF
```

```
psql -U cyclops -h localhost -d cyclops_udr <<EOF
CREATE TABLE IF NOT EXISTS udr (
  time_from TIMESTAMP         NOT NULL,
  time_to   TIMESTAMP         NOT NULL,
  metric    TEXT              NOT NULL,
  account   TEXT              NOT NULL,
  usage     DOUBLE PRECISION  NOT NULL,
  data      JSONB,
  unit      TEXT
);
CREATE INDEX IF NOT EXISTS udr_metric ON udr (metric, time_from DESC);
CREATE INDEX IF NOT EXISTS udr_account ON udr (account, time_from DESC);
CREATE INDEX IF NOT EXISTS udr_unit ON udr (unit, time_from DESC);
CREATE INDEX IF NOT EXISTS udr_data ON udr USING HASH (data);
EOF
```

3.3 Preparing RabbitMQ

Assuming that RabbitMQ is running on the same machine where the following commands are to be executed, running these will setup necessary exchanges, queues and bindings between them for udr process to function properly.

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.udr.consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.udr.commands
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.udr.
↪broadcast
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"direct", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.udr.
↪dispatch
```

In the above commands, do not forget to replace **-u** values *cyclops* and *pass1234* to correct RabbitMQ user/pass values that was setup earlier.

3.4 Configuring UDR

You can configure the service endpoints and dependencies in the configuration file located under */etc/cyclops/udr/*

Default content is shown next:

```
# HTTP and/or HTTPS port to be exposed at
ServerHTTPPort=4567
#ServerHTTPSPort=5567
#ServerHTTPSCertPath=/path/to/cert.p12
#ServerHTTPSPassword=password

# Health check every X seconds
ServerHealthCheck=30
ServerHealthShutdown=false

# Database credentials to TimescaleDB
DatabasePort=5432
DatabaseHost=localhost
DatabaseUsername=cyclops
DatabasePassword=password
DatabaseName=cyclops_udr
DatabasePageLimit=500
DatabaseConnections=4

# Publisher (RabbitMQ) credentials
PublisherHost=localhost
PublisherUsername=cyclops
PublisherPassword=password
PublisherPort=5672
PublisherVirtualHost=cyclops
PublisherDispatchExchange=cyclops.udr.dispatch
PublisherBroadcastExchange=cyclops.udr.broadcast

# Consumer (RabbitMQ) credentials
ConsumerHost=localhost
ConsumerUsername=cyclops
ConsumerPassword=password
ConsumerPort=5672
ConsumerVirtualHost=cyclops
```

(continues on next page)

(continued from previous page)

```
ConsumerDataQueue=cyclops.udr.consume
ConsumerCommandsQueue=cyclops.udr.commands
```

- ServerHTTPPort / ServerHTTPSPort: You can configure the port where the service will be running at. HTTPS is supported if you provide a valid certificate and the associated password.
- TimescaleDB parameters are same as Postgressql parameters
- RabbitMQ block configures how this service communicates with an existing RabbitMQ service endpoint, they are defined for both the consumer as well as publisher process.

3.5 Fixing permissions

Before running any of the Cyclops framework services via *systemctl* command, make sure that the process user *cyclops* which was created earlier to run the process has full read/write access to Cyclops specific system folder and files.

```
sudo chown -R cyclops:cyclops /var/log/cyclops/
sudo chown -R cyclops:cyclops /usr/local/bin/cyclops/
sudo chown -R cyclops:cyclops /etc/cyclops/
sudo chown -R cyclops:cyclops /var/lib/cyclops/
```

3.6 Setup as a service

Create a file called *cyclops-udr.service* in */etc/systemd/system/* directory. Add the following content to this file:

```
[Unit]
Description=Cyclops UDR Service
After=network.target rabbitmq-server.service postgresql-9.6.service

[Service]
ExecStartPre=/bin/sleep 2
Type=simple
User=cyclops
ExecStart=/usr/bin/java -jar /usr/local/bin/cyclops/udr/udr.jar /etc/cyclops/udr/udr.
↪conf
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

This assumes that the rabbitmq and postgres server is running in the same machine where you are setting up udr service. If not then remove them from the dependencies list by changing the **After** line above. *Do make sure that these services are running and reachable before udr service is started.*

You can enable and manage the udr service and start it by using the following systemctl commands.

```
sudo systemctl enable cyclops-udr.service
sudo systemctl start/stop/restart/status cyclops-udr.service
```

CHAPTER 4

Install & Configure: CDR

Let us setup CDR micro-service to run as a linux system service.

4.1 Preparing the host machine

Start by creating system folders for CDR service.

```
sudo mkdir -p /var/log/cyclops/cdr/
sudo mkdir -p /etc/cyclops/cdr/
sudo mkdir -p /usr/local/bin/cyclops/cdr/
```

For logging to work properly, these files must exist, perform the next commands to ensure the same.

- errors.log
- trace.log
- rest.log
- dispatch.log
- data.log
- commands.log
- timeseries.log

```
sudo touch /var/log/cyclops/cdr/errors.log
sudo touch /var/log/cyclops/cdr/trace.log
sudo touch /var/log/cyclops/cdr/rest.log
sudo touch /var/log/cyclops/cdr/dispatch.log
sudo touch /var/log/cyclops/cdr/data.log
sudo touch /var/log/cyclops/cdr/commands.log
sudo touch /var/log/cyclops/cdr/timeseries.log
```

Let's move the binary and the configuration files from the compiled locations to the target system destinations.

```
sudo mv CDR/target/cdr.jar /usr/local/bin/cyclops/cdr/
sudo mv CDR/config/cdr.conf /etc/cyclops/cdr/
```

4.2 Preparing the Postgressql / TimescaleDB

Before working with the cdr service, it is necessary to setup the appropriate database and table schemas. This can be achieved by executing the following commands on the host where the Postgresql service is running.

```
psql -U postgres -h localhost <<EOF
CREATE DATABASE cyclops_cdr WITH OWNER cyclops;
GRANT ALL PRIVILEGES ON DATABASE cyclops_cdr TO cyclops;
EOF
```

```
psql -U cyclops -h localhost -d cyclops_cdr <<EOF
CREATE TABLE IF NOT EXISTS cdr (
  time_from TIMESTAMP         NOT NULL,
  time_to   TIMESTAMP         NOT NULL,
  metric    TEXT              NOT NULL,
  account   TEXT              NOT NULL,
  charge    DOUBLE PRECISION  NOT NULL,
  data      JSONB,
  currency  TEXT
);
CREATE INDEX IF NOT EXISTS cdr_metric ON cdr (metric, time_from DESC);
CREATE INDEX IF NOT EXISTS cdr_account ON cdr (account, time_from DESC);
CREATE INDEX IF NOT EXISTS cdr_currency ON cdr (currency, time_from DESC);
CREATE INDEX IF NOT EXISTS cdr_data ON cdr USING HASH (data);
EOF
```

4.3 Preparing RabbitMQ

Assuming that RabbitMQ is running on the same machine where the following commands are to be executed, running these will setup necessary exchanges, queues and bindings between them for cdr process to function properly.

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.cdr.consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.cdr.commands
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.
↪coincdr.broadcast
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"direct", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.cdr.
↪dispatch
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.cdr.
↪broadcast
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPOST -d '{}' http://
↪localhost:15672/api/bindings/cyclops/e/cyclops.coincdr.broadcast/q/cyclops.cdr.
↪consume
```

In the above commands, do not forget to replace the **-u** values *cyclops* and *pass1234* to correct RabbitMQ user/pass values that was setup earlier.

4.4 Configuring CDR

You can configure the service endpoints and dependencies in the configuration file located under */etc/cyclops/cdr/*

Default content is shown next:

```
# HTTP and/or HTTPS port to be exposed at
ServerHTTPPort=4568
#ServerHTTPSPort=5568
#ServerHTTPSCertPath=/path/to/cert.p12
#ServerHTTPSPassword=password

# Health check every X seconds
ServerHealthCheck=30
ServerHealthShutdown=false

# Database credentials to TimescaleDB
DatabasePort=5432
DatabaseHost=localhost
DatabaseUsername=cyclops
DatabasePassword=password
DatabaseName=cyclops_cdr
DatabasePageLimit=500
DatabaseConnections=2

# Publisher (RabbitMQ) credentials
PublisherHost=localhost
PublisherUsername=cyclops
PublisherPassword=password
PublisherPort=5672
PublisherVirtualHost=cyclops
PublisherDispatchExchange=cyclops.cdr.dispatch
PublisherBroadcastExchange=cyclops.cdr.broadcast

# Consumer (RabbitMQ) credentials
ConsumerHost=localhost
ConsumerUsername=cyclops
ConsumerPassword=password
ConsumerPort=5672
ConsumerVirtualHost=cyclops
ConsumerDataQueue=cyclops.cdr.consume
ConsumerCommandsQueue=cyclops.cdr.commands
```

- ServerHTTPPort / ServerHTTPSPort: You can configure the port where the service will be running at. HTTPS is supported if you provide a valid certificate and the associated password.
- TimescaleDB parameters are same as Postgressql parameters
- RabbitMQ block configures how this service communicates with an existing RabbitMQ service endpoint, they are defined for both the consumer as well as publisher process.

4.5 Fixing permissions

Before running any of the Cyclops framework services via *systemctl* command, make sure that the process user *cyclops* which was created earlier to run the process has full read/write access to Cyclops specific system folder and files.

```
sudo chown -R cyclops:cyclops /var/log/cyclops/
sudo chown -R cyclops:cyclops /usr/local/bin/cyclops/
sudo chown -R cyclops:cyclops /etc/cyclops/
sudo chown -R cyclops:cyclops /var/lib/cyclops/
```

4.6 Setup as a service

Create a file called *cyclops-cdr.service* in */etc/systemd/system/* directory. Add the following content to this file:

```
[Unit]
Description=Cyclops CDR Service
After=network.target rabbitmq-server.service postgresql-9.6.service

[Service]
ExecStartPre=/bin/sleep 2
Type=simple
User=cyclops
ExecStart=/usr/bin/java -jar /usr/local/bin/cyclops/cdr/cdr.jar /etc/cyclops/cdr/cdr.
↪conf
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

This assumes that the rabbitmq and postgres server is running in the same machine where you are setting up cdr service. If not then remove them from the dependencies list by changing the **After** line above. *Do make sure that these services are running and reachable before cdr service is started.*

You can enable and manage the cdr service and start it by using the following systemctl commands.

```
sudo systemctl enable cyclops-cdr.service
sudo systemctl start/stop/restart/status cyclops-cdr.service
```

CHAPTER 5

Install & Configure: Billing

Let us setup Billing micro-service to run as a linux system service.

5.1 Preparing the host machine

Start by creating system folders for Billing service.

```
sudo mkdir -p /var/log/cyclops/billing/
sudo mkdir -p /etc/cyclops/billing/
sudo mkdir -p /usr/local/bin/cyclops/billing/
```

For logging to work properly, these files must exist, perform the next commands to ensure the same.

- errors.log
- trace.log
- rest.log
- dispatch.log
- data.log
- commands.log
- timeseries.log

```
sudo touch /var/log/cyclops/billing/errors.log
sudo touch /var/log/cyclops/billing/trace.log
sudo touch /var/log/cyclops/billing/rest.log
sudo touch /var/log/cyclops/billing/dispatch.log
sudo touch /var/log/cyclops/billing/data.log
sudo touch /var/log/cyclops/billing/commands.log
sudo touch /var/log/cyclops/billing/timeseries.log
```

Let's move the binary and the configuration files from the compiled locations to the target system destinations.

```
sudo mv Billing/target/billing.jar /usr/local/bin/cyclops/billing/
sudo mv Billing/config/billing.conf /etc/cyclops/billing/
```

5.2 Preparing the Postgressql / TimescaleDB

Before working with the billing service, it is necessary to setup the appropriate database and table schemas. This can be achieved by executing the following commands on the host where the Postgresql service is running.

```
psql -U postgres -h localhost <<EOF
CREATE DATABASE cyclops_billing WITH OWNER cyclops;
GRANT ALL PRIVILEGES ON DATABASE cyclops_billing TO cyclops;
EOF
```

```
psql -U cyclops -h localhost -d cyclops_billing <<EOF
CREATE TABLE IF NOT EXISTS billrun (
  id        SERIAL            primary key,
  time      TIMESTAMP         NOT NULL,
  data      JSONB
);
EOF
```

```
psql -U cyclops -h localhost -d cyclops_billing <<EOF
CREATE TABLE IF NOT EXISTS bill (
  id        SERIAL,
  run       INTEGER           REFERENCES billrun,
  time_from TIMESTAMP         NOT NULL,
  time_to   TIMESTAMP         NOT NULL,
  account   TEXT              NOT NULL,
  charge    DOUBLE PRECISION  NOT NULL,
  discount  TEXT,
  data      JSONB,
  currency  TEXT
);
CREATE INDEX IF NOT EXISTS bill_account ON bill (account, time_from DESC);
CREATE INDEX IF NOT EXISTS bill_currency ON bill (currency, time_from DESC);
CREATE INDEX IF NOT EXISTS bill_data ON bill USING HASH (data);
EOF
```

5.3 Preparing RabbitMQ

Assuming that RabbitMQ is running on the same machine where the following commands are to be executed, running these will setup necessary exchanges, queues and bindings between them for billing process to function properly.

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.billing.consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.billing.commands
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.
↪coinbill.broadcast
```

(continues on next page)

(continued from previous page)

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"direct", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.
↪billing.dispatch
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.
↪billing.broadcast
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPOST -d '{}' http://
↪localhost:15672/api/bindings/cyclops/e/cyclops.coinbill.broadcast/q/cyclops.billing.
↪consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.cdr.commands
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.coinbill.consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPOST -d "{\"routing_
↪key\":\"CDR\"}" http://localhost:15672/api/bindings/cyclops/e/cyclops.billing.
↪dispatch/q/cyclops.cdr.commands
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPOST -d "{\"routing_
↪key\":\"CoinBill\"}" http://localhost:15672/api/bindings/cyclops/e/cyclops.billing.
↪dispatch/q/cyclops.coinbill.consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPOST -d "{\"routing_
↪key\":\"SelfPublish\"}" http://localhost:15672/api/bindings/cyclops/e/cyclops.
↪billing.dispatch/q/cyclops.billing.commands
```

In the above commands, do not forget to replace the **-u** values *cyclops* and *pass1234* to correct RabbitMQ user/pass values that was setup earlier.

5.4 Configuring Billing

You can configure the service endpoints and dependencies in the configuration file located under */etc/cyclops/billing/*

Default content is shown next:

```
# HTTP and/or HTTPS port to be exposed at
ServerHTTPPort=4569
#ServerHTTPSPort=5569
#ServerHTTPSCertPath=/path/to/cert.p12
#ServerHTTPSPassword=password

# Health check every X seconds
ServerHealthCheck=30
ServerHealthShutdown=false

# Database credentials to TimescaleDB
```

(continues on next page)

(continued from previous page)

```
DatabasePort=5432
DatabaseHost=localhost
DatabaseUsername=cyclops
DatabasePassword=password
DatabaseName=cyclops_billing
DatabasePageLimit=500
DatabaseConnections=2

# Publisher (RabbitMQ) credentials
PublisherHost=localhost
PublisherUsername=cyclops
PublisherPassword=password
PublisherPort=5672
PublisherVirtualHost=cyclops
PublisherDispatchExchange=cyclops.billing.dispatch
PublisherBroadcastExchange=cyclops.billing.broadcast

# Consumer (RabbitMQ) credentials
ConsumerHost=localhost
ConsumerUsername=cyclops
ConsumerPassword=password
ConsumerPort=5672
ConsumerVirtualHost=cyclops
ConsumerDataQueue=cyclops.billing.consume
ConsumerCommandsQueue=cyclops.billing.commands

# Bill generation workflow
PublishToCDRWithKey=CDR
PublishToCoinBillWithKey=CoinBill
PublishToSelf=SelfPublish

# Connection to customer-database
CustomerDatabaseHost=localhost
CustomerDatabasePort=8888
```

- ServerHTTPPort / ServerHTTPSPort: You can configure the port where the service will be running at. HTTPS is supported if you provide a valid certificate and the associated password.
- TimescaleDB parameters are same as Postgressql parameters
- RabbitMQ block configures how this service communicates with an existing RabbitMQ service endpoint, they are defined for both the consumer as well as publisher process.

5.5 Fixing permissions

Before running any of the Cyclops framework services via *systemctl* command, make sure that the process user *cyclops* which was created earlier to run the process has full read/write access to Cyclops specific system folder and files.

```
sudo chown -R cyclops:cyclops /var/log/cyclops/
sudo chown -R cyclops:cyclops /usr/local/bin/cyclops/
sudo chown -R cyclops:cyclops /etc/cyclops/
sudo chown -R cyclops:cyclops /var/lib/cyclops/
```

5.6 Setup as a service

Create a file called *cyclops-billing.service* in */etc/systemd/system/* directory. Add the following content to this file:

```
[Unit]
Description=Cyclops billing Service
After=network.target rabbitmq-server.service postgresql-9.6.service

[Service]
ExecStartPre=/bin/sleep 2
Type=simple
User=cyclops
ExecStart=/usr/bin/java -jar /usr/local/bin/cyclops/billing/billing.jar /etc/cyclops/
↪billing/billing.conf
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

This assumes that the rabbitmq and postgres server is running in the same machine where you are setting up billing service. If not then remove them from the dependencies list by changing the **After** line above. *Do make sure that these services are running and reachable before billing service is started.*

You can enable and manage the billing service and start it by using the following systemctl commands.

```
sudo systemctl enable cyclops-billing.service
sudo systemctl start/stop/restart/status cyclops-billing.service
```

CHAPTER 6

Install & Configure: Rule Engine

Coin is our rule engine and microservice enabling model insertion and its execution. It supports the cdr and billing processes in model based pricing, charging and discounting workflows.

Two rule engine processes have to be setup for each of the following microservices -

- cdr
- billing

These two rule engine processes in this guide are therefore named as follows -

- coincdr
- coinbill

6.1 Setup & configuration: coincdr

6.1.1 Preparing the host machine

Start by creating system folders for coincdr service.

```
sudo mkdir -p /var/log/cyclops/coincdr/
sudo mkdir -p /etc/cyclops/coincdr/
sudo mkdir -p /usr/local/bin/cyclops/coincdr/
```

For logging to work properly, these files must exist, perform the next commands to ensure the same.

- errors.log
- trace.log
- hibernate.log
- facts.log
- rules.log

- timeline.log
- dispatch.log
- stream.log

```
sudo touch /var/log/cyclops/coincdr/errors.log
sudo touch /var/log/cyclops/coincdr/trace.log
sudo touch /var/log/cyclops/coincdr/hibernate.log
sudo touch /var/log/cyclops/coincdr/facts.log
sudo touch /var/log/cyclops/coincdr/rules.log
sudo touch /var/log/cyclops/coincdr/timeline.log
sudo touch /var/log/cyclops/coincdr/dispatch.log
sudo touch /var/log/cyclops/coincdr/stream.log
```

Let's move the binary and the configuration files from the compiled locations to the target system destinations.

```
sudo cp Coin/target/coin.jar /usr/local/bin/cyclops/coincdr/coincdr.jar
sudo cp CDR/config/coin.conf /etc/cyclops/coincdr/coincdr.conf
```

6.1.2 Preparing RabbitMQ

Assuming that RabbitMQ is running on the same machine where the following commands are to be executed, running these will setup necessary exchanges, queues and bindings between them for coincdr process to function properly.

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.coincdr.consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.udr.
↪broadcast
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPOST -d '{}' http://
↪localhost:15672/api/bindings/cyclops/e/cyclops.udr.broadcast/q/cyclops.coincdr.
↪consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.
↪coincdr.broadcast
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"direct", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.
↪coincdr.dispatch
```

In the above commands, do not forget to replace the **-u** values *cyclops* and *pass1234* to correct RabbitMQ user/pass values that was setup earlier.

6.1.3 Configuring coincdr

You can configure the service endpoints and dependencies in the configuration file located under */etc/cyclops/coincdr/*

Default content is shown next:

```
# HTTP and/or HTTPS port to be exposed at
ServerHTTPPort=4570
#ServerHTTPSPort=5570
#ServerHTTPSCertPath=/path/to/cert.p12
#ServerHTTPSPassword=pass1234

# Health check every X seconds
ServerHealthCheck=30
ServerHealthShutdown=false

# Hibernate connection credentials
HibernateURL=jdbc:postgresql://localhost/cyclops_cdr
HibernateUsername=cyclops
HibernatePassword=pass1234
HibernateDriver=org.postgresql.Driver
HibernateDialect=org.hibernate.dialect.PostgreSQL9Dialect

# Publisher (RabbitMQ) credentials
PublisherHost=localhost
PublisherUsername=cyclops
PublisherPassword=pass1234
PublisherPort=5672
PublisherMngtPort=15672
PublisherVirtualHost=cyclops
PublisherDispatchExchange=cyclops.coincdr.dispatch
PublisherBroadcastExchange=cyclops.coincdr.broadcast

# Consumer (RabbitMQ) credentials
ConsumerHost=localhost
ConsumerUsername=cyclops
ConsumerPassword=pass1234
ConsumerPort=5672
ConsumerMngtPort=15672
ConsumerVirtualHost=cyclops
ConsumeFromQueue=cyclops.coincdr.consume

# Bind Coin CDR with UDR (flushing UDR records)
BindWithUDR=cyclops.udr.broadcast
```

- ServerHTTPPort / ServerHTTPSPort: You can configure the port where the service will be running at. HTTPS is supported if you provide a valid certificate and the associated password.
- Hibernate connections parameters are same as Postgressql parameters
- RabbitMQ block configures how this service communicates with an existing RabbitMQ service endpoint, they are defined for both the consumer as well as publisher process.

6.1.4 Fixing permissions

Before running any of the Cyclops framework services via *systemctl* command, make sure that the process user *cyclops* which was created earlier to run the process has full read/write access to Cyclops specific system folder and files.

```
sudo chown -R cyclops:cyclops /var/log/cyclops/
sudo chown -R cyclops:cyclops /usr/local/bin/cyclops/
sudo chown -R cyclops:cyclops /etc/cyclops/
sudo chown -R cyclops:cyclops /var/lib/cyclops/
```

6.1.5 Setup as a service

Create a file called *cyclops-coincdr.service* in */etc/systemd/system/* directory. Add the following content to this file:

```
[Unit]
Description=Cyclops Coin CDR Service
After=network.target rabbitmq-server.service postgresql-9.6.service

[Service]
ExecStartPre=/bin/sleep 2
Type=simple
User=cyclops
ExecStart=/usr/bin/java -jar /usr/local/bin/cyclops/coincdr/coincdr.jar /etc/cyclops/
↪coincdr/coincdr.conf
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

This assumes that the rabbitmq and postgres server is running in the same machine where you are setting up coincdr service. If not then remove them from the dependencies list by changing the **After** line above. *Do make sure that these services are running and reachable before coincdr service is started.*

You can enable and manage the coincdr service and start it by using the following systemctl commands.

```
sudo systemctl enable cyclops-coincdr.service
sudo systemctl start/stop/restart/status cyclops-coincdr.service
```

6.2 Setup & configuration: coinbill

6.2.1 Preparing the host machine

Start by creating system folders for coinbill service.

```
sudo mkdir -p /var/log/cyclops/coinbill/
sudo mkdir -p /etc/cyclops/coinbill/
sudo mkdir -p /usr/local/bin/cyclops/coinbill/
```

For logging to work properly, these files must exist, perform the next commands to ensure the same.

- errors.log
- trace.log
- hibernate.log
- facts.log
- rules.log
- timeline.log
- dispatch.log
- stream.log

```
sudo touch /var/log/cyclops/coinbill/errors.log
sudo touch /var/log/cyclops/coinbill/trace.log
sudo touch /var/log/cyclops/coinbill/hibernate.log
sudo touch /var/log/cyclops/coinbill/facts.log
sudo touch /var/log/cyclops/coinbill/rules.log
sudo touch /var/log/cyclops/coinbill/timeline.log
sudo touch /var/log/cyclops/coinbill/dispatch.log
sudo touch /var/log/cyclops/coinbill/stream.log
```

Let's move the binary and the configuration files from the compiled locations to the target system destinations.

```
sudo mv Coin/target/coin.jar /usr/local/bin/cyclops/coinbill/coinbill.jar
sudo mv CDR/config/coin.conf /etc/cyclops/coinbill/coinbill.conf
```

6.2.2 Preparing RabbitMQ

Assuming that RabbitMQ is running on the same machine where the following commands are to be executed, running these will setup necessary exchanges, queues and bindings between them for coinbill process to function properly.

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"durable
↪":true}' http://localhost:15672/api/queues/cyclops/cyclops.coinbill.consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.cdr.
↪broadcast
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPOST -d '{}' http://
↪localhost:15672/api/bindings/cyclops/e/cyclops.cdr.broadcast/q/cyclops.coinbill.
↪consume
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"fanout", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.
↪coinbill.broadcast
```

```
curl -u "cyclops:pass1234" -H "content-type:application/json" -XPUT -d '{"type":
↪"direct", "durable":true}' http://localhost:15672/api/exchanges/cyclops/cyclops.
↪coinbill.dispatch
```

In the above commands, do not forget to replace the **-u** values *cyclops* and *pass1234* to correct RabbitMQ user/pass values that was setup earlier.

6.2.3 Configuring coinbill

You can configure the service endpoints and dependencies in the configuration file located under */etc/cyclops/coinbill/*

Default content is shown next:

```
# HTTP and/or HTTPS port to be exposed at
ServerHTTPPort=4571
#ServerHTTPSPort=5571
#ServerHTTPSCertPath=/path/to/cert.p12
#ServerHTTPSPassword=pass1234

```

(continues on next page)

(continued from previous page)

```
# Health check every X seconds
ServerHealthCheck=30
ServerHealthShutdown=false

# Hibernate connection credentials
HibernateURL=jdbc:postgresql://localhost/cyclops_billing
HibernateUsername=cyclops
HibernatePassword=pass1234
HibernateDriver=org.postgresql.Driver
HibernateDialect=org.hibernate.dialect.PostgreSQL9Dialect

# Publisher (RabbitMQ) credentials
PublisherHost=localhost
PublisherUsername=cyclops
PublisherPassword=pass1234
PublisherPort=5672
PublisherMngtPort=15672
PublisherVirtualHost=cyclops
PublisherDispatchExchange=cyclops.coinbill.dispatch
PublisherBroadcastExchange=cyclops.coinbill.broadcast

# Consumer (RabbitMQ) credentials
ConsumerHost=localhost
ConsumerUsername=cyclops
ConsumerPassword=pass1234
ConsumerPort=5672
ConsumerMngtPort=15672
ConsumerVirtualHost=cyclops
ConsumeFromQueue=cyclops.coinbill.consume

# Bind Coin Bill with CDR (flushing CDR records)
BindWithCDR=cyclops.cdr.broadcast
```

- ServerHTTPPort / ServerHTTPSPort: You can configure the port where the service will be running at. HTTPS is supported if you provide a valid certificate and the associated password.
- Hibernate connections parameters are same as Postgressql parameters
- RabbitMQ block configures how this service communicates with an existing RabbitMQ service endpoint, they are defined for both the consumer as well as publisher process.

6.2.4 Fixing permissions

Before running any of the Cyclops framework services via *systemctl* command, make sure that the process user *cyclops* which was created earlier to run the process has full read/write access to Cyclops specific system folder and files.

```
sudo chown -R cyclops:cyclops /var/log/cyclops/
sudo chown -R cyclops:cyclops /usr/local/bin/cyclops/
sudo chown -R cyclops:cyclops /etc/cyclops/
sudo chown -R cyclops:cyclops /var/lib/cyclops/
```

6.2.5 Setup as a service

Create a file called *cyclops-coinbill.service* in */etc/systemd/system/* directory. Add the following content to this file:

```
[Unit]
Description=Cyclops Coin Bill Service
After=network.target rabbitmq-server.service postgresql-9.6.service

[Service]
ExecStartPre=/bin/sleep 2
Type=simple
User=cyclops
ExecStart=/usr/bin/java -jar /usr/local/bin/cyclops/coinbill/coinbill.jar /etc/
↪cyclops/coinbill/coinbill.conf
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

This assumes that the rabbitmq and postgres server is running in the same machine where you are setting up coinbill service. If not then remove them from the dependencies list by changing the **After** line above. *Do make sure that these services are running and reachable before coincdr service is started.*

You can enable and manage the coinbill service and start it by using the following systemctl commands.

```
sudo systemctl enable cyclops-coinbill.service
sudo systemctl start/stop/restart/status cyclops-coinbill.service
```

CHAPTER 7

Managing Cyclops

Now that we have the framework configured properly, lets look at how to manage a live Cyclops service.

7.1 Configuring a collector

7.2 Rules management

It is assumed that you already know how to read/understand/write Drools rule. If not please read further here.

7.2.1 Managing rules in coincdr

Cyclops data transformation workflow is heavily guide by pricing and billing models injected within the rule engines attached to the microservices *cdr* and *billing*. These are called **coincdr** and **coinbill**.

Assuming that the usage data being sent to cyclops has the following form -

```
{
  "metric":"somemeter",
  "account":"customer-account",
  "usage":2,
  "unit":"GB",
  "time":1507593601000,
  "data":{
    "serviceId":"user1@cust-x.ch",
    "billingModel":"Smart"
  }
}
```

You can inject rules within the **coincdr** rule engine to manipulate any fields you see in the JSON above. Fields inside the *data* block is accessible via the corresponding Map object.

A sample rule is shown below -

```
import ch.icclab.cyclops.facts.Usage;
import ch.icclab.cyclops.facts.Charge;

rule "Rate somemeter usage value"
salience 50
when
  $usage: Usage(metric == "somemeter" && data != null && data contains "billingModel"
↪&& data["billingModel"]=="Smart")
then
  Charge charge = new Charge($usage);
  charge.setCharge($usage.getUsage() * 0.4);

  insert(charge);
  retract($usage);
end
```

Analyzing the rule above, if the usage record being processed contains a data block and an element *billingModel*, then generates the charge by multiplying the **usage** value with **0.4**.

This example simply shows how with ease, Cyclops rule engines can be programmed.

You can have multiple rules which can be potentially apply in a given situation, but which one is triggered can be controlled by the weight assigned to a rule. The weight is controlled via the **salience** parameter.

Lets assume one wishes to have a catch all rule for processing usage. This can be written as shown below -

```
import ch.icclab.cyclops.facts.Usage;
import ch.icclab.cyclops.facts.Charge;

rule "Remaining services for free"
salience 40
when
  $usage: Usage()
then
  Charge charge = new Charge($usage);
  charge.setCharge(0);

  insert(charge);
  retract($usage);
end
```

Since the *salience* of the rule is lesser than the first rule, it will be applied only when the first rule mentioned in this page is inapplicable.

You can even control data transmission behavior via rules. Say we want to push all generated charge records over to a channel, specially within the Cyclops framework we must push the cdr records to a specific RabbitMQ exchange, it can be achieved via the following rule within *coincdr*.

```
import ch.icclab.cyclops.facts.Charge;
import java.util.List;
global ch.icclab.cyclops.publish.Messenger messenger;

rule "Broadcast CDRs"
salience 20
when
  $charge: List( size > 0 ) from collect ( Charge() )
then
  messenger.broadcast($charge);
```

(continues on next page)

(continued from previous page)

```
  $charge.forEach(c->retract(c));
end
```

7.2.2 Managing rules in coinbill

Just like the rules for *coincdr* that governs the transformation of udr records to cdr records, one needs to manage the rules in coinbill to govern the generation of bill from cdr records.

Lets look at a sample *coinbill* rule that upon receipt of the bill generation command and the list of cdr records, creates the bill for the requested set of accounts -

```
import ch.icclab.cyclops.facts.BillRequest;
import ch.icclab.cyclops.facts.Charge;
import ch.icclab.cyclops.facts.Bill;
import java.util.List;

rule "Collect CDRs for the Bill Request"
salience 50
when
  $request: BillRequest($accounts: accounts)
  $CDRs: List(size > 0) from collect (Charge(account memberOf $accounts))
then
  // bills for each currency of account\'s CDRs
  List<Bill> bills = $request.process($CDRs);

  // add bills to the working memory
  bills.forEach(bill->insert(bill));

  // remove processed CDRs and the bill request
  $CDRs.forEach(c->retract(c));
  retract($request);
end
```

The statements of the rule above should be self explanatory. Similar to *coincdr* where one had to prepare a rule for sending the generated records to next stop in the data path, here too in Cyclops framework, the generated bill records should be moved to the next stage in the messaging setup -

```
import ch.icclab.cyclops.facts.DatonusBill;
import java.util.List;

global ch.icclab.cyclops.publish.Messenger messenger;

rule "Broadcast generated Datonus bills"
salience 30
when
  $bills: List(size > 0) from collect (DatonusBill())
then
  // broadcast and remove processed bills
  messenger.broadcast($bills);
  $bills.forEach(bill->retract(bill));
end
```

As you can notice, usually all Java language constructs and objects are available to you while formulating a rule.

7.2.3 Rule management endpoints

The above shown example rules and any other that one may create must be uploaded to the corresponding rule engines. This is achieved by sending a HTTP POST request to the rule engine endpoint

- coin-cdr-url-or-ip:port/rule
- coin-bill-url-or-ip:port/rule

7.3 Generation of a bill

CHAPTER 8

Advanced customizations

8.1 Working with the dashboard

8.2 Advanced RabbitMQ setup

CHAPTER 9

Rule Versioning

It is possible to use git integration and rollback coin rules to previous versions. This will also roll back all affected CDRs and bills

9.1 Setup rule versioning

To enable this functionality, a git repository needs to be created for the rules. The credentials for this repository need to be specified in the *conf* file of the *coin* service.

```
# Git credentials:
GitRepo=
GitUsername=
GitPassword=
GitProjectPath=
```

9.2 Setting up checkpoints for the rules

To be able to roll back rules, they need to be uploaded to the git repository after they are applied to *coin*. Different versions of the rules can be **tagged** for easier reference.

9.3 Rolling back rules and affected records

To rollback rules to a previous version, a command request needs to be made to *coin* with the following format:

```
{
  "commits": [
      {
          "added": [
```

(continues on next page)

(continued from previous page)

```
                <list of rules added by the commit>
            ],
            "modified": [
                <list of rules modified by the commit>
            ]
        }
    ],
    "project_id": <id of the project>,
    "ref": <tag or branch to roll back to>,
    "time_from": <time of the commit to be undone>
}
```

A list of 'bad' commits can be provided, with lists of files added or modified in those commits. The **time_from** parameter is important, as it will set the checkpoint for the rolling back of CDRs and bills. This request is made to:

```
<coinurl>/newrule?execute=true
```

The execute parameter forces all rules to be fired when a rule is rolled back.

CHAPTER 10

Forecasting and Estimation Engine

Description of the implementation of the forecasting and estimation engine and how it can be used to create cost forecasts and evaluate different pricing models.

10.1 Per account forecast: (Forecast command)

- All historical records for account are retrieved from DB
- They are grouped by usage type
- A set of forecast records are generated for each usage type using the ARIMA model, UDRs, CDRs and Bills are generated using ‘evaluation rules’

10.2 Model based global forecast: (Forecast command with no account specified)

- Same as per account forecast, but ignores account and aggregates all records
- Depends on the ARIMA model itself to determine usage and account activity patterns and customize the forecast

10.3 2D pattern based forecast: (GlobalForecast command)

- Creates usage patterns by grouping the historical records by account and by type
- Creates an activity pattern by counting how many users were active for each day in the history
- Uses the ARIMA model to create a future activity forecast (how many accounts are expected to be active each day in the forecast period)
- For each active account for each day in the forecast period, assigns one of the generated usage patterns and uses ARIMA model to forecast the usage for each usage type

- As before, UDRs, CDRs and Bills are generated using ‘evaluation rules’

10.4 Evaluation rules/Pricing models under evaluation:

- Rules that are fired only when the records have a specific tag
- Groups of rules can have the same tag target so that they can be grouped into a separate pricing model to be evaluated
- The pricing model is marked by its tag, and the target model is specified by its tag in the forecast command payload

Request example (can be used as a payload either to the command HTTP endpoint or to send a command through AMPQ:

```
{

   "command": "GlobalForecast",

   "target": "test1",

   "forecastSize": 15

}
```

Rule example:

```
rule "Test 1 rule for ram (12:3:58.4 11/Jun/2019)"

salience 60

when

   $usage: Usage(metric == "memory" && account.contains("test1"))

then

...
```

The important things to note about the rules are:

- The salience must be higher than the ‘real’ rules, so that it gets checked first, or it will never be triggered. This will not affect ‘real’ records, as they will not be tagged.
- The tag for this rule is ‘test1’. The forecast generator tags the records it creates in the account and data fields, so the rule can look for the tag in either place. In this example, for readability, the rule checks for its tag in the account field. In a real case, it is safer to check the data field for the “target”:”test1” pair. That excludes the possibility that real records from a user named ‘test1’ will fire the test rules.
- Rules with the same tag (as long as they comply with the above two points) can be targeted to evaluate a whole new pricing model (a set of charging and pricing rules in this context)

CHAPTER 11

Indices and tables

- genindex
- modindex
- search