Collaboration Curry

Christian Drefke

Sep 01, 2021

CONTENTS:

1 Getting Started 3

2 Introduction to Rules 5

3 The Ruleset 7

4 Environment 15

5 Modules 17

6 About 19

7 Indices and tables 21

Project Homepage: https://gitlab.com/MrCollaborator/collabcurry

CHAPTER

ONE

GETTING STARTED

1.1 Requirements

The repository is primarily built around Docker. Though it is possible to run the Flask application on it's own after installing all necessary requirements, runnin the environment in Docker is recommended.

Basic Requirements:

- Docker
- Docker-Compose (automatically installed if you are using Docker for Windows)
- Git
- Internet Access for the Setup

1.2 Setup

1.2.1 Cloning the repository

Clone the repository from Gitlab.

via SSH

```
git clone git@gitlab.com:MrCollaborator/collabcurry.git
```

via HTTPS

```
git clone https://gitlab.com/MrCollaborator/collabcurry.git
```

Switch to directory

```
cd collabcurry
```

1.2.2 Example Config

Copy the example config from /setup/_examples/default to /setup.

Linux / MacOSX

```
cp setup/_examples/. setup/
```

Windows

```
cp setup\_examples\* setup\
```

Edit the files as necessary.

See chapter *Introduction to Rules* for further documentation on rules and actions.

See chapter *Environment* for information about environment variables.

1.2.3 Build and start Docker Containers

Use Docker Compose to build and start the containers. "–parallels" is optional but faster on multi-core hosts.

```
docker-compose -f production.yml build --parallels
```

```
docker-compose -f production.yml up -d
```

The container running the Flask application is mapping the /rules directory to read the rules files. This allows for subsequent changes to the rules files during runtime.

Attach to the main container for logging purposes.

```
docker logs -f collabcurry_curri_prod_1
```

1.2.4 Configure Cisco UCM ECC Profile

The URL for Cisco UCMs ECC Profile configuration is:

> http://{[}collabcurry ip]:80/ecc/request

Note: The port ':80' is mandatory. Without, CUCM will not parse the url correctly.

CHAPTER

TWO

INTRODUCTION TO RULES

The rules configuration (which should actually be called rules and actions) is based on YAML files that reside in the '/rules' directory. This directory is mapped as volume during runtime of the container. Therefore it is possible to edit and manipulate the individual rules during runtime.

At the moment there is only the filename 'rules.yml' supported for a single ruleset. There are plans to enhance this to multiple rule files in the future but we are not there yet.

Although CollabCurry is checking for errors after loading and sending out appropriate validation results, it makes absolutely sense to do it right the first time. The file needs to be written in valid YAML. The base structure including all currently possible options can be taken from the examples file 'rules_all.yml' in the '/examples/rules' directory.

The following section tries to explain the seperate parts of the file, it's structure and options.

CHAPTER

THREE

THE RULESET

A ruleset is the collection of all individual rules in a rules file. A ruleset can contain an basically unlimited number of rules though there might be a performance impact with hundreds of rules.

Warning: No performance testing was done yet to validate a maximum number of rules before noticing impact. But a handful shouldn't be harmful.

Example from the examples directory (shortened for better readability):

```
---
ruleset:
  - rule:
      description: 'Continue'
      *snip*
            continue:
                modify:
                    callingnumber: '+49123456789'
                    callednumber: '+49123456789'
                    callingname: 'Hans Wurst'
                    calledname: 'Zwerg Nase'
                greetingid: 'ConferenceNowEnterPIN'
  - rule:
      description: 'Divert'
      *snip*
            divert:
                destination: '+49123456789'
                modify:
                    callingnumber: '+49123456789'
                    callednumber: '+49123456789'
                    callingname: 'Hans Wurst'
                    calledname: 'Zwerg Nase'
                resetcallhistory: 'resetAllHops'
  - rule:
      description: 'Reject'
      *snip*
            reject:
                announcementid: 'ConferenceNowEnterPIN'
                reason: 'blocked'
```

This ruleset contains of three rules: 'Continue', 'Divert' and 'Reject'.

After accepting a request from the UCMs ECC interface, all rules from a ruleset will be subsequently processed. The first rule where the defined conditions match the request will be selecet.

Note: There is **no** further processing after the first rule matched it's condition. Subsequent rules and it's actions are simply skipped.

When a rule matches, it's actions will be executed and the request answered with a CURRI result based on default settings or the matching rules actions.

3.1 The Curry Rule

A rule consists at least of the following elements:

- A 'name' attribute
- 'Conditions' to match an incoming request
- 'Actions' to be performed on a successful match

Example of a rule:

```
- rule:
  name: 'Continue'
  conditions:
    AND:
      callingnumber:
        - '^\+(?!49)[\d]+$'
      callednumber: '7100'
  actions:
    curri:
      decision: 'Permit'
      directives:
        continue:
            modify:
                callingname: 'International Caller'
            greetingid: 'WelcomeEN'
```

3.1.1 Conditions

Conditions define if a rule is matched and therefore it's actions are performed. The first level of the conditions tree is always an operator. Currently only the 'AND' and 'OR' operators are supported.

At the moment all valid parameters for a condition derive from Cisco UCMs ECC requests. The request contains these attributes:

- callingnumber
- callednumber
- transformedcgpn
- transformedcdpn
- triggerpointtype

In most cases only 'callingnumber' and/or 'callednumber' will be used to match a condition.

Parameters can be defined as a single value or a YAML list of values. A list of values is always validated with the OR operator. All values have to be strings.

Example for a single value:

```
callednumber: '7100'
```

Example for a list of values:

```
callingnumber:
  -  '^\+(?!49)[\d]+$'
  -  '^\+49123456789$'
```

This example means that no phone pattern with the German +E.164 prefix (+49) will be matched with the exception of the exact pattern '+49123456789'.

3.2 Actions

Actions are performed when the conditions of a rule match. Currently supported actions:

- CURRI Action
- LDAP Action
- EMail Action
- UCCX Action

Each type of Action has it's own section under the "actions" tree:

```
- rule:
  name: ...
  conditions:
    ...
  actions:
    curri:
      ...
    ldap:
      ...
    email:
      ...
    uccx:
      ...
```

3.2.1 Action Execution order

CURRI actions are currently executed in a fixed order.

1. uccx
2. ldap
3. email
4. curri

The order in which the different methods are called is important since we want to use the modified calling number of the UCCX action first. Then, if the LDAP action might overwrite again. At the end, the result of all the Transformations is sent out via e-mail.

Warning: This needs to be kept in mind when overwriting the calling number especially with the LDAP action.

3.2.2 CURRI Action

CURRI actions are all possible reply actions that are supported by Cisco UCMs ECC interface.

Note: See Cisco's documentation on developer.cisco.com for further information on the ECC interface.

An ECC has basically three major decisions:

- Permit
- Divert
- Reject

Permit

A permit decision gives is first of all signalling that the call shall be normally processed by the UCM based on it's call routing configuration.

Directive information can be added to this decision to malform calling or called party information. The base tag for permit directives is called 'continue'. Probably the most used feature here is to malform the calling party name. This is usually done for phone number to name resolutions again a LDAP, SQL, CSV, whatever phone directory. The advantage of this name substitution through ECC is, that the UCM is setting updating the alterting name. This name is then signalled through all it's call protocol interfaces where calling name support is given, including analog devices (SCCP only), SIP trunks or CTI.

Example or a permit decision:

```
curri:
  decision: 'Permit'
  directives:
    continue:
      modify:
          callingnumber: '+49123456789'
          callednumber: '+49123456789'
          callingname: 'Hans Wurst'
          calledname: 'Zwerg Nase'
      greetingid: 'ConferenceNowEnterPIN'
```

Divert

The divert is actually also configured as a permit decision in terms of the CURRI syntax but enhanced with an addition. When configuring a divert, the directives parameter 'destination' is used for setting the diversion target.

There are again additional directives. The base tag here is 'divert'.

Example of a divert decision:

```
curri:
  decision: 'Permit'
  directives:
```

(continues on next page)

(continued from previous page)

```
    divert:
      destination: '+49123456789'
      modify:
          callingnumber: '+49123456789'
          callednumber: '+49123456789'
          callingname: 'Hans Wurst'
          calledname: 'Zwerg Nase'
      resetcallhistory: 'resetAllHops'
```

Reject

Use this decision to reject a call. The directives base tag is called 'reject'.

Example of a reject decision:

```
curri:
  decision: permit
  directives:
    reject:
        announcementid: 'ConferenceNowEnterPIN'
        reason: 'blocked'
```

3.2.3 E-Mail Action

E-Mail actions are used to send out an e-mail .

Note: Configure the SMTP server information through the environment variables.

Example of the mail action section:

```
mail:
  active: "true"
  subject: "mail/curri_subject.txt"
  body: "mail/curri_body.txt"
  mail_from: "abc.def@ghj.com"
  mail_to:
    - "abc.def@ghj.com"
```

Parameters in E-Mail action

active If False, the e-mail action won't be executed.

subject Path to a text file for the e-mail subject. Jinja2 is used as template engine.

body Path to a text file for the e-mail body. Jinja2 is used as template engine. In addition, a .html file but the same name can be used to use html formatted mail body. A .txt file is still mandatory though.

mail_from E-Mail address to be used as the sender address.

mail_to A list of e-mail addresses to send the e-mail to.

E-Mail templates

The templates have to be located under the /templates directory.

E-Mail template variables

A dictionary with variables is forwarded to the Jinja2 template engine and can be used as information in the e-mail. The structure of this dictionary is:

```
{
  ecc:
    {
      'callingnumber': ...,
      'callednumber': ...,
      ... ecc request information
    },
  curri:
    {
      'decision': ...,
      'directives': ...,
      ... content of the matched rule
    }
```

3.2.4 LDAP Action

The LDAP action can be used to lookup a phone number for name resolution.

Example of the LDAP action section:

```
ldap:
  match_against:
    - telephoneNumber
    - mobile
  return: '{{ sn }} {{ company }}'
```

Parameters in LDAP action

match_against A list of LDAP attributes the calling number is matched against.

return A string that can contain LDAP attributes. The result is used as the new calling name in the ECC response. Jinja2 is used as template engine.

LDAP Synch and Database Backend

All LDAP lookups will be done against a database backend. No ECC requests will be queried directly against LDAP do limit the load on the service.

Instead MongoDB is used to store the imported data. As NoSQL database which allows for dynamic content per object, MongoDB can adapt to number and name of LDAP attributes to be imported.

LDAP Service Configuration

Edit the environment file to define LDAP connection and import settings.

3.2.5 UCCX Action

The standard setup of Cisco Contact Center Express together with a Cisco Unified Communications Manager has a certain flaw when it comes to the Caller Id signalling. This is a result of the CCXs behaviour when transferring a call from a CCX queue to an agent. When the agents phone rings, the phone display will only show an internal directory number of the UCM CTI Port that is used to forward the call. The agent can see the Caller Id in CCX Finesse but not* on the phone.

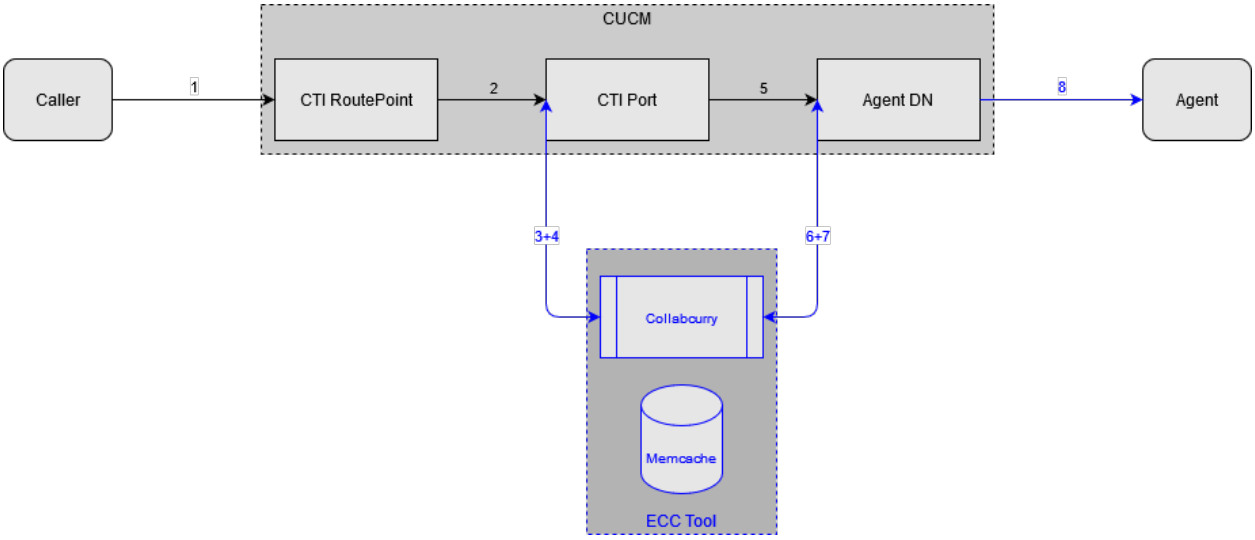
If the agent has some kind of local CTI application running that shall be used for caller recognition then this recognition will also not work for CCX forwarded calls.

To overcome this limitation you can use the UCCX Action. It will basically store and forward the Caller Id for CCX routed calls.

To be fair: There is a way of pushing the Caller Id to the phone but it's not pretty and won't work for CTI apps.

UCCX-ECC Flow

These are the basics of the UCM-UCCX-ECC call flow required by Collaburry to perform a UCCX action.



1. The call comes in to a CTI Routepoint (Trigger in UCCX)
2. UCCX forwards the call to a free CTI Port (Port Group in UCCX)
3. The ECC profile in UCM is configured on the CTI ports. UCM sends an ECC request to CollabCurry.
4. Collabcurry checks the UCCX rule. If the conditions fit it will store the Caller Id to the cache.
5. When an agent is available, UCCX makes a consult transfer to the agents Directory Number.
6. The ECC profile needs to be configured on the agents DN. UCM will send an ECC request.
7. Collabcurry will check if there is a cached Caller Id for the transferring CTI Port. ECC reply to UCM.
8. UCM rings the agents phone. If there was a cached number, the agent will see this instead of the CTI Port DN.

Example of the mail action section:

```
uccx:
  ctiports:
    - '^9\d\d\d'
  agents:
    - '^1\d\d\d'
```

Parameters

ctiports Regex single entry or list of all CTI Port directory numbers that shall be monitored.

agents Regex single entry or list of all Agent directory numbers that shall receive the cached Caller Id.

Example Rule

This is an example of how a single rule can be configured for the inbound call to CCX and the outbound call to the agent. It is also possible to split these up into multiple rules.

Example Rule:

```
- rule:
      name: 'UCCX'
      conditions:
        AND:
          callednumber:
            - '^9\d\d\d'
            - '^1\d\d\d'
      actions:
        uccx:
          ctiports:
            - '^9\d\d\d'
          agents:
            - '^1\d\d\d'
        curri:
          decision: 'Permit'
```

- callednumber: Both, the CTI Port DNs and Agent DNs are configured as list (OR Pattern)
- ctiports: All CTI Port DNs that shall be monitored are defined here.
- agents: All Agent DNs that shall receive the substituted Caller Id.
- decision: Set the curri permit decision to give UCM permission to forward the call.

CHAPTER

FOUR

ENVIRONMENT

CHAPTER

FIVE

MODULES

CHAPTER

SIX

ABOUT

6.1 Collab Curry

6.1.1 Working the the Cisco Unified Routing XML Interface

This small python application running with Flask is accepting requests through Cisco UCMs[1] External Call Control (ECC) interface. Also known as CURRI. Incoming requests are then parsed and checked against a set of rules, provided through YAML files from the /rules directory. Depending on the rules, various actions can be started. The following actions are currently implemented:

Planned but not yet implemented actions

- Caller Recognition against SQL, ...
- Out events through Syslog, SNMP, REST, ...

Other Todos

- Performance testing for design recommendations
- REST API for features like importing rules...

[1] Cisco Unified Communications Manager

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search