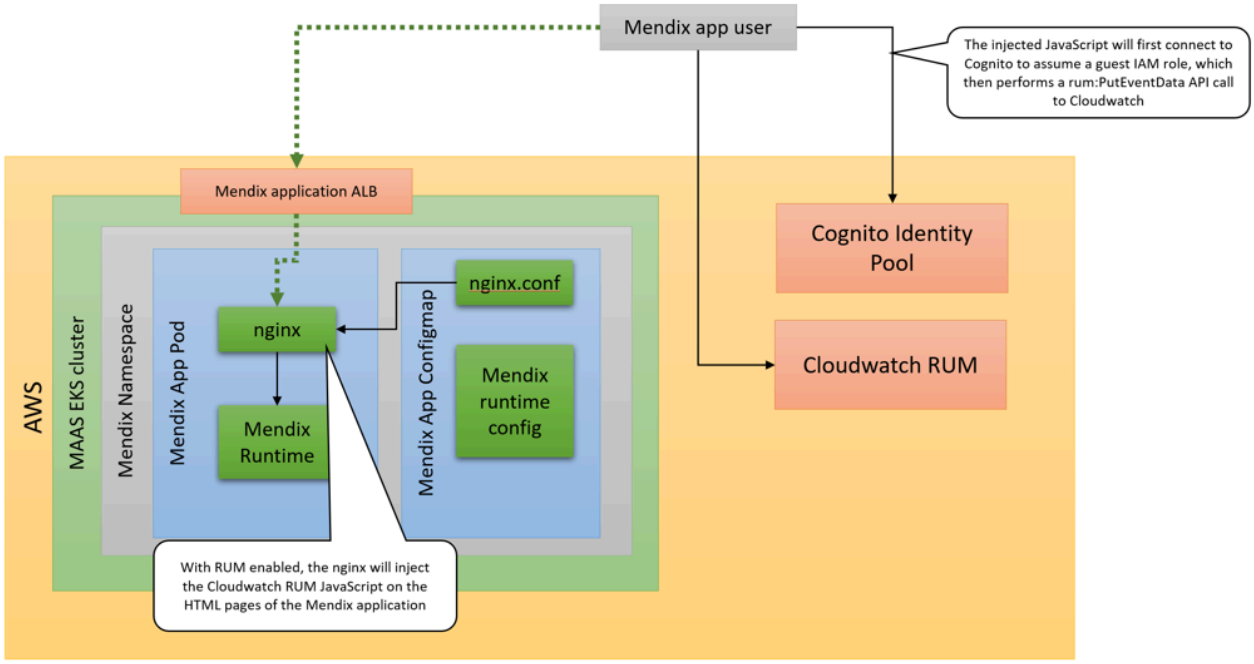
Cloudwatch Real User Monitoring (RUM)

Real User Monitoring captures data from users using a webapplication and captures things like load times, browser performance, errors and more. We use this for the Mendix applications to get a better understanding how users are actually using the app and what errors they get. This information gets passed onto the developers to improve the application.

> Note that RUM is only enabled on production from a cost management perspective. Technically there is nothing preventing it from running on other environments as well.

RUM overview



There are two separate components that make up the Cloudwatch RUM setup; a client instrumentation side and a server side. The server side consists of an AWS Cognito Pool, which is required for the browser to acquire a set of temporary credentials so it can impersonate an IAM role. This IAM role will have privileges to only call the PutEventData API of Cloudwatch RUM. This way the JavaScript can push browser data into Cloudwatch RUM. On the instrumentation side we rely on the nginx container of the Mendix Application pod to modify the HTTP response to inject the Javascript. This is not strictly required as the Mendix

applications themselves could also add this Javascript themselves, but this method removes the need for application side changes to at least get the basics of RUM operational.

Note that we only need a single Cognito Identity Pool with a single IAM role. All requests are unauthenticated which is what we expect.

> While the auto injection done here is sufficient for the basics, application developers can still embed the framework into their codebase to get additional information, should that be required.

Nginx configuration

This snippet highlights the settings for the supermarket app. Each application will have its own unique ID, and the hosted Javascript is always part of the built container. Most of these settings are either default values or are overridden via the SSM parameter store (see next section for detail).

```
# HTTP content modification for injecting RUM Javascript
set $CLOUDWATCH_RUM_ID '37b75f14-7a79-481d-a18d-d1a97c4cdcd5';
set $REGION 'eu-west-1';
set $COGNITO_POOL_ID "eu-west-1:0a6f020d-94b5-4f9b-8d2e-5b14ccd369e2";
set $RUM_ENDPOINT "https://dataplane.rum.${REGION}.amazonaws.com";
set $GUEST_ROLE_ARN 'arn:aws:iam::896172553483:role/cognito-identitypool-cloudwatch-rum-guest-role';
set $ALLOW_COOKIES 'true';
set $ENABLE_XRAY 'false';

set $HTML_SEARCH_STRING '<head>';
set $JS_LOCATION 'https://supermarketone.mx.dsm.app/js/cwr.js';
set $SESSION_SAMPLE_RATE '1'; # percentage of sessions to sample in a float
set $TELEMETRIES_INTERACTION_OPTIONS '{ events: [{ event: \'click\', element: document }]}'; # JSON string format
set $TELEMETRIES_PERFORMANCE_OPTIONS '{ enableMutationObserver: true }'; # JSON string format
set $TELEMETRIES_HTTP_OPTIONS '{ recordAllRequests: false }';

# The actual javascript to inject using the variables from above. Must be single line
set $JAVASCRIPT_STRING "<snipped for brevity, this contains the actual JavaScript to inject with all previously mentioned variables>"

location / {
    if ($request_uri ~ ^/(.*\.(css|js)|forms/.*|img/.*|pages/.*|mxclientsystem/images/.*)\?[0-9]+$) {
            expires 1y;
    }
    if ($request_uri ~ ^/((index[\w-]*|login)\.html)?$) {
        add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval'; connect-src 'self' https://cognito-identity.eu-west-1.amazonaws.com https://sts.eu-west-1.amazonaws.com https://dataplane.rum.eu-west-1.amazonaws.com; font-src 'self' data:; img-src 'self' data:; style-src 'self' 'unsafe-inline'; base-uri 'self'; form-action 'self'; object-src 'none'; frame-ancestors 'self";
```

```
26          add_header Cache-Control "no-cache";
27      }
28      sub_filter ${HTML_SEARCH_STRING} ${HTML_SEARCH_STRING}\n<script>${JAVASCRIPT_STRING}
    </script>;
29      sub_filter_once on;
30      proxy_pass http://mendix;
31  }
```

One important piece is the location of the injection itself; this has to be done **before** any other JavaScript is running; otherwise it doesn't work. Hence line 10 defines the search string on the HTML `<head>` tag to ensure it runs first. You will also see the **Content-Security-Policy** header which must be present at all times; otherwise the browser will reject assuming the role, generating the temporary token via STS or sending data to CloudWatch RUM. The Mendix Build Pipeline contains all the logic for it and will be automatically added, regardless if additional HTTP headers are configured in the SSM parameter store or not. In case of this example, the supermarket app does not supply a CSP in its configuration and therefore the pipeline logic will add the required HTTP response header.

Infrastructure as code

There is an overlap between the *monitoring* repository and the *kubernetes-mendix-deployment* repository in terms of code. The server side is configured in the former repository and uses the CDK to configure Cloudwatch accordingly. The Javascript injection is done on the Mendix application side and it will enable RUM when the `/dsm/mendix/apps/<app_name>/config/rum/enabled` parameter is set in the SSM parameter store. The Helm chart will respond on this and will set the above configuration in the nginx's ConfigMap. In addition, the following SSM parameters are used as part of RUM (and again, they are consumed by the Mendix pipelines):

- `/dsm/mendix/apps/<app_name>/config/rum/cloudwatch_rum_id` - the ID of the Cloudwatch RUM monitor. Created by CDK
- `/dsm/cloudwatch/rum/guest_role_arn` - The ARN of the IAM role used by unauthorized clients to push RUM data. Created by CDK
- `/dsm/cloudwatch/rum/cognito_pool_id` - The ID of the cognito pool. Created by CDK

The CDK code creates the underlying CloudWatch infrastructure and the RUM monitors themselves, as each application as its own RUM monitor. To ease making config changes and

adding / removing RUM monitors this uses a YAML file to configure. This file lives in the `configuration/rum.yaml` file of the *monitoring* repository. This looks as follows:

```
supermarketone:
  samplerate: 1
  cloudwatchLogs: false
  extendedCloudwatchMetrics: true
anh-pricing:
  samplerate: 1
  cloudwatchLogs: false
  extendedCloudwatchMetrics: true
co2trust:
  samplerate: 1
  cloudwatchLogs: false
  extendedCloudwatchMetrics: true
[snipped for brevity]
```

Each application is a map in YAML, with the application name being the same as the name used in the SSM parameter store. To add an application, simply append the 4 lines (e.g. copy line 1-4, paste it at the bottom of the file and change the supermarketone name to the name of your application) and set the parameters to your needs. At this time of writing there are three configurable parameters:

- samplerate: A float indicates a percentage of sessions sending data to RUM. Useful for high traffic websites to not capture everything and reduce costs. A value of 1 means all sessions are captured.
- cloudwatchLogs: By default the data is stored for 30 days and it can optionally store data for longer periods of time in Cloudwatch logs. Disabled by default.
- extendedCloudwatchMetrics: Write additional metrics as Cloudwatch Metrics such as the web vitals. Enabled by default but can be disabled if needed.

⚠ When adding a new RUM monitor, make sure you **first** add it to the YAML file mentioned above and then run the *monitoring* pipeline. Once this is done you can redeploy the application using the Mendix Build / Release pipeline to enable RUM in its entirety.

Helm chart

For reference, these are the values that the Helm template expects in order to enable RUM. These map to the SSM Parameter Store like most things map to the SSM parameter store. These settings are used in the `configmap.yaml` part of the helm template to set the nginx configuration as mentioned above

```
# AWS Cloudwatch Real User Monitoring support. Enabling this enables injection of the RUM
javascript done
```

```
# in the nginx configuration of the Mendix application. The enabled param must be a string.
rum:
  enabled: 'false'
  cloudwatch_rum_id:
  cognito_pool_id:
  guest_role_arn:
```

Dockerfile

The Dockerfile always includes the `cwr.js` of the build, which is irrespective of RUM being enabled or not. Like this we can assume the JavaScript needed for the injection is always present. For reference is a snippet of the Dockerfile used to build the container, and this is in line 5. The file is downloaded at the very late stages of buildtime because we need the `unzip` to be completed as mentioned in line 4.

```
RUN chmod 755 /opt/mendix/bin/startup.py && \
    useradd -d /opt/mendix -s /usr/sbin/nologin mendix && \
    cd /opt/mendix/app && \
    unzip application.mda && \
    wget -qO /opt/mendix/app/web/js/cwr.js https://client.rum.us-east-1.amazonaws.com/1.15.0/cwr.js && \
    mkdir -p data/database data/files data/tmp data/model-upload && \
    chown -R mendix:mendix /opt/mendix
```

Further reading

- SSM Parameter Store & Secrets Manager reference - MAAS Team Space - Confluence (atlassian.net)
- Pipeline workflows - MAAS Team Space - Confluence (atlassian.net)
- Use CloudWatch RUM - Amazon CloudWatch
- aws-cdk-lib.aws_rum module · AWS CDK (amazon.com)