Dynamic Templating

Notification Center customization is powered by the Tera templating language. Tera enables you to create highly flexible and dynamic templates that respond to a variety of contexts, formats, and use cases. You can use dynamic templating to personalize your connectors, presets, and routing rules.

Global context available to all templates

All templates have access to the `_context` variable, which contains metadata about the notification source type, including system identifiers and trigger details. This context allows you to dynamically customize your template content based on the notification's origin.

`_context` reference

Variable	Description
`_context`	Contains metadata about the context of the notification source type
`_context.entityType`	The notification source type (e.g., "alerts")
`_context.entitySubType`	The notification source subtype, if any
`_context.entityLabels`	Notification source type labels associated with the context (e.g., "host", "region")
`_context.system`	Information about the system, including its ID and name
`_context.system.id`	The system's unique identifier
`_context.system.name`	The system's name (e.g., "acme-prod")
`_context.trigger`	The trigger that initiated the notification source type
`_context.trigger.automaticTrigger`	Automatic trigger details (if applicable)
`_context.trigger.manualTrigger`	Manual trigger details (e.g., user email)
`_context.trigger.manualTrigger.userEmail`	The email address of the user who manually triggered the notification source type

Variable	Description
`_context.trigger.type`	The type of trigger that initiated the notification source type (e.g., "manual")

Inspecting context with `get_context`

Use the custom `get_context` function to view the entire template context. This is useful for debugging and discovering available variables.

Name	Type	Description	Usage example
`get_context`	Custom	Returns the entire template context as an object, which can be used to troubleshoot and view available variables	`{{ get_context() }}`

Example: Using `get_context` for debugging

View the entire context and all variables that are available for the notification source type.

Input

```
{{ get_context() | json_encode(pretty = true) }}
```

Example output

This output displays all possible variables for this alert notification source in a readable JSON format:

```
{
  "_context": {
    "entityType": "alerts",
    "entitySubType": "metricThresholdMoreThanTriggered",
    "entityLabels": {},
    "trigger": {
      "type": "manual",
      "manualTrigger": {
        "userEmail": "user@email.com"
      },
      "automaticTrigger": null
    },
```

```
    "system": {
      "id": "17266",
      "name": "onlineboutique"
    }
  },
  "alert": {
    "timestamp": 1638403600,
    "id": "497f6eca-6276-4993-bfeb-53cbbbba6f08",
    "status": "Triggered",
    "groups": [
      {
        "status": "Triggered",
        "priority": "P1",
        "keyValues": {
          "host": "server1"
        },
        "details": {
          "$type": "metricThreshold",
          "metricThreshold": {
            "conditionType": "More than threshold",
            "fromTimestamp": 1638400000,
            "toTimestamp": 1638403600,
            "maxValueOverThreshold": 95,
            "minValueOverThreshold": 85,
            "avgValueOverThreshold": 90,
            "pctOverThreshold": 90,
            "isUndetectedValue": false
          }
        }
      }
    ]
  },
  "alertDef": {
    "name": "CPU Utilization Alert",
    "description": "An alert for monitoring CPU utilization.",
    "entityLabels": {
      "environment": "production",
      "region": "us-west"
    }
  }
}
```

This gives you a complete snapshot of the current data available for the alert, which can be useful for debugging and customizing notifications.

Filters

Filters let you transform data, manipulate variables, or format strings, numbers, and dates. These can be used to clean up or format data to fit your notification needs.

Name	Type	Description	Usage Example
`json_escape`	Custom	Converts an object to a JSON string	`{{
`date`	Built-in	Converts a timestamp into a formatted date string	`{{ 1616346000
`default`	Built-in	Returns a default value if the variable is undefined	`{{ value

Example: Using `json_escape` for output

If you need to display a complex object as a string, use the `json_escape` filter:

Input

```
{{ get_context() | json_escape }}
```

Example output

This transforms any value into its JSON string representation:

```
{\"_context\":
{\"entityType\":\"alerts\",\"entitySubType\":\"metricThresholdMoreThanTrigger
... }}
```

Example: Formatting dates with the `date` filter

The `date` filter allows you to format timestamps for easier readability. Here's an example of how to use it:

Input

```
{{ alert.timestamp | date(format="%Y-%m-%d %H:%M") }}
```

```
We detected a change in the values for the following hosts:
{% for i in alert.groups %}
{{ i.priority }} / {{ i.keyValues | json_escape }} / {{
i.details.metricThreshold.fromTimestamp | date(format="%Y-%m-%d %H:%M") }}
{% endfor %}
```

Example output

```
2021-12-02 00:06

We detected a change in the values for the following hosts:
P1 / {"host": "server1"} / 2021-12-01 23:06
```

Example: Using `default` to handle missing values

The `default` filter helps you provide fallback values if a variable is missing or undefined:

Input

```
{{ alertDef.entityLabels.myVal | default(value="no value") }}
```

Example Output

```
no value
```

Variables

Variables refer to the dynamic data or objects that are available within the template's execution context. Coralogix supports the `_context` variable, containing metadata about the context of the notification source type. For alerts, this includes source type, trigger details, and system information.

Example 1: To view the context

Input

```
{{ get_context() | json_encode(pretty = true) }}
```

Example output

Displays the entire alert context in a nicely formatted JSON object.

Example 2: Viewing the context directly

Input

```
{{ _context }}
```

Example output

```
alerts
```

Example 3: Accessing a specific field in the context (`entitySubType`)

Input

```
{{ _context.entitySubType }}
```

Example output

```
logsThresholdMoreThanTriggered
```

Example 4: Viewing the `system` context

Input

```
{{ _context.system | json_encode(pretty = true) }}
```

Example output

```
{
  "id": "17266",
  "name": "onlineboutique"
}
```

The output consists of your Coralogix team ID and team name.

Example 5: Viewing the trigger details

Input

```
{{ _context.trigger | json_encode(pretty = true) }}
```

Example output

This output displays what triggered the notification, showing whether it was manual or automatic:

```
{
  "type": "manual",
  "manualTrigger": {
    "userEmail": "user@email.com"
  },
  "automaticTrigger": null
}
```

In this example, a manual trigger was used by a user.

Tips for troubleshooting and debugging

- **Use `get_context()` to view all available variables**: This is especially useful when you're working with custom variables or trying to figure out which data is available in the template context.
- **Use `json_encode` or `json_escape` to format data**: When working with complex objects, these filters can help you convert them into a readable format, which is helpful for debugging.
- **Check for undefined variables**: Always use the `default` filter to provide fallback values in case a variable is missing from the alert context. This ensures that notifications are always delivered.

Practical examples for alert notification source type

Below are examples of how to use the functions, filters, and variables to customize your notifications effectively. These examples show how to incorporate dynamic values, handle conditional logic, and format the content to make your alerts more informative. The examples include alerts notification source type schema, which can be found here.

Using variables in alert names & descriptions

You can include dynamic variables in an alert's name or description to propagate them into your notification content. For example:

```
Alert has triggered for subsystem
{{alert.groups[0].keyValues['coralogix.metadata.subsystemName']}}
```

Date formatting

Alert timestamps are typically in Epoch format (e.g., `1735655158`). You can use the `date` function to convert these to human-readable format:

- Example: `{{alert.timestamp | date(format="%Y-%m-%d %H:%M")}}`
- This converts an Epoch timestamp (e.g., `1638403600`) into a readable format like `2021-12-02 00:06`.
- Example: `{{alert.timestamp | date(format="%Y-%m-%d %H:%M")}}`
- This converts an Epoch timestamp (e.g., `1638403600`) into a readable format like `2021-12-02 00:06`.

By leveraging these functions, filters, and variables, you can create customized, informative alerts tailored to your specific notification needs.

Example 1: Conditional alert handling based on alert definition name

This example customizes the notification content based on the alert definition name. It checks the alert type and formats the message accordingly:

```
{% if alertDef.name == "RDS Instance Low CPU" %}
affected db: {{alert.groups[0].keyValues.DBInstanceIdentifier}}
credit balance is below: {{i.condition.threshold}}
owning team: {{alert.groups[0].keyValues.Team}}

{% elif alertDef.name == 'OOM killed pod' %}
pod: {{alert.groups[0].keyValues.k8s_pod_name}}
container: {{alert.groups[0].keyValues.k8s_container_name}}

{% else %}
{{alertDef.name}} is triggering

{% endif %}
```

This conditional logic allows the notification to dynamically change depending on the alert's name:

- For "RDS Instance Low CPU", it includes the affected database identifier and the team responsible for the alert.
- For "OOM killed pod", it includes the pod and container names.
- If neither condition is met, it simply outputs the alert's name.
- For "RDS Instance Low CPU", it includes the affected database identifier and the team responsible for the alert.

- For "OOM killed pod", it includes the pod and container names.
- If neither condition is met, it simply outputs the alert's name.

Example 2: Logs threshold alert description

This example formats a notification for a log threshold alert, displaying detailed information about the alert and its associated values. This template includes detailed information about the alert, such as the priority, affected values, thresholds, and the specific rules that caused the alert.

```
{{alert.timestamp | date(format="%Y-%m-%d %H:%M")}}

We've detected that the query result has dropped below the threshold for
the following values of {{alertDef.groupByKeys}}:

Priority / Values / Threshold / Start Time / End Time

{% for i in alert.groups %}
{{i.priority}} / {{i.keyValues | json_escape}} /
{{i.details.logsThreshold.fromTimestamp | date(format = "%Y-%m-%d %H:%M")}}
/ {{i.details.logsThreshold.toTimestamp}}
{% endfor %}

Alert Query:
{{alertDef.typeDefinition.logsThreshold.luceneQuery}}

Alert Condition Rules:
Condition type - {{
alertDef.typeDefinition.logsThreshold.rules[0].condition.conditionType }}
Condition rules -
{% for i in alertDef.typeDefinition.logsThreshold.rules %}
Threshold: {{i.condition.threshold}}, timeframe:
{{i.condition.timeWindow.logsTimeWindowSpecificValue}}.
{% endfor %}

@jane.doe
@here
```

- It uses the `date` filter to format timestamps and `json_escape` to ensure the output is safe for JSON.
- Note also that Jane Doe and all channel users are tagged as part of this Slack notification.

Example 3: Grouping by service

If your alert is grouped by service, you can include dynamic information related to the affected service. Here's an example where the group-by key is `service_name`:

```
{
  "alert_url": "https://teamname-
prod.app.eu2.coralogix.com/#/alerts/{{alert.id}}",
  "service": "{{alert.groups[0].keyValues.service_name}}",
  "priority": "{{alertDef.priority}}",
  "timestamp": "{{alert.timestamp | date(format='%Y-%m-%d %H:%M')}}",
  "status": "{{alert.status}}"
}
```

The service name is extracted from the alert group's key-value pair (`service_name`) and is included in the notification. This allows the notification to dynamically reflect which service the alert pertains to.

Last updated: April 5, 2025