ORACLE

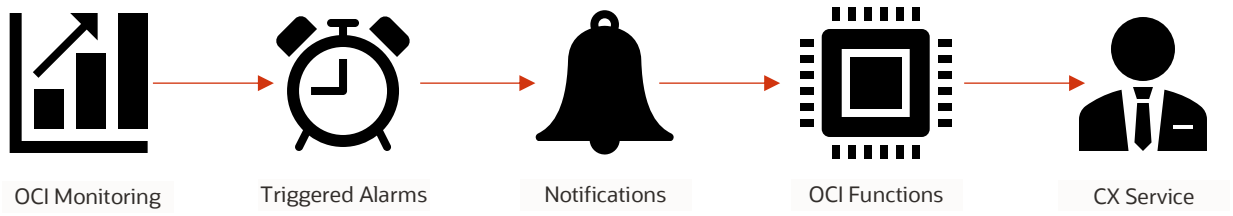
# Use Oracle CX Service to Manage OCI Alarm Events

Learn how to automate service operations for your OCI events using Oracle CX Service

August, 2020 | Version 1.01

## Introduction

Oracle customers may use Oracle Cloud Infrastructure (OCI) Alarms to provide automatic IT support based on the performance of Cloud Infrastructure. Oracle B2C Service and Oracle B2B Service may both be used to manage incidents or service requests that require service engagement, such as notifying staff of broader service issues, proactive customer interaction, or other services managed through the contact center. The process below demonstrates the flow of information from Oracle Cloud Infrastructure to CX Service in order to trigger a service process.




To illustrate the ease of configuration we will use an example alarm scenario that will trigger an incident in B2C Service. An alarm configured for vault secret creation will trigger a notification that generates the B2C Service incident. The alarm is an arbitrary choice to easily demonstrate how this process works; the process is the same for any OCI alert.

## Design Overview

The solution is comprised of the following components from OCI:

- OCI Monitoring service allows customers to create alarms to monitor resources and system behavior significant to their operation. Setting these alarms allows administrators to focus their time and energy on higher value tasks rather than regularly monitoring the resources via metrics and charts. Once alarms are set, OCI will notify of any configured events as defined in alarm definitions. Managing alarms is described here in OCI documentation and some best practices related to alarms are here.
- Alarms use OCI Notifications service to notify system administrators about these events. Notifications service uses topics and subscriptions to send out notifications. Messages are published to topics and then sent out to all the subscription channels defined for that topic. OCI Notifications service supports a rich set of subscriptions – Email, PagerDuty, Slack, HTTPS URLs and Oracle Functions. More information about topics and subscriptions is here.
- Oracle Functions allows development, deployment, and execution of applications that implement business logic without worrying about the infrastructure where it will execute. This enables very quick release cycles as you are only concerned with the development of your business logic as the infrastructure is managed automatically.

## Implementation Details

The remainder of this post will outline function implementation in detail. References to product documentation will be provided for context while this post focuses on the design elements of the solution.

**Documentation Links:**

- OCI Monitoring Service Alarms
- OCI Notifications Service

An alarm message belongs one of these four types: OK_TO_FIRING, FIRING_TO_OK, REPEAT and RESET. These message types are described here. Alarm message data depends on the message types. Message type data formats are described here. The documentation has an example of an alarm message.

Following are some salient points about the alarm messages:

- Each alarm message has a “dedupeKey” that can be used to distinguish between messages from different alarms. This example uses a custom attribute on the incident to maintain context of this event from alert-to-alert.
- An alarm goes from OK to FIRING state when the metric threshold as defined in alarm definition is violated for any of the resources included by metric description. The corresponding message type is OK_TO_FIRING. This message type will include dimension data for all the resources for which the alarm is firing. Please note that the dimension data may be different depending on the resource type and metric available. If you bring your own custom metrics to OCI monitoring service, then you could create alarms on those metrics as well. The following picture shows an example of this message type.


```
{
    "dedupeKey": "9a38cd1c-f808-4871-8c2a-abc4a72a9868",
    "title": "Test Alert",
    "body": "An alert is firing that needs attention in CX Service",
    "type": "OK_TO_FIRING",
    "severity": "INFO",
    "timestampEpochMillis": 1597101180000,
    "alarmMetaData": [
        {
            "id": "ocid1.alarm.oc1.iad.<id>",
            "status": "FIRING",
            "severity": "INFO",
            "query": "PutRequests[1m]{resourceID = \"ocid1.bucket.oc1.iad.<id> \"}.sum()
> 0",
            "totalMetricsFiring": 1,
            "dimensions": [
                {
                    "resourceID": "ocid1.bucket.oc1.iad.<id> ",
                    "resourceDisplayName": "bucket-20200810-1902"
                }
            ]
        }
    ],
    "version": 1
```

```
}
```

- The alarm body is mapped to a new incident thread.
- The alarm goes back to OK state when the metric is under the defined threshold for all the resources. The corresponding message type is FIRING_TO_OK and doesn't have any dimension data. The following picture shows an example of this message type (relevant fields highlighted):

```
{
    "dedupeKey": "9a38cd1c-f808-4871-8c2a-abc4a72a9868",
    "title": "Test Alert",
    "body": "An alert is firing that needs attention in CX Service",
    "type": "FIRING_TO_OK",
    "severity": "INFO",
    "timestampEpochMillis": 1597101180000,
    "alarmMetaData": [
        {
            "id": "ocid1.alarm.oc1.iad.<id>",
            "status": "OK",
            "severity": "INFO",
            "query": "PutRequests[1m]{resourceID = \"ocid1.bucket.oc1.iad.<id> \"}.sum()
> 0",
            "totalMetricsFiring": 1,
            "dimensions": []
        }
    ],
    "version": 1
}
```

Please note that the "dedupeKey" is identical for both these notifications. This is used to ensure that the same message does not regenerate incidents in the destination system.

Repeat messages are like OK_TO_FIRING but have a message type of REPEAT. It is possible that Repeat messages may have dimension data for different resources than the original OK_TO_FIRING message because the state of the system might have changed. For example, let's say that we have a cluster of three servers – "server A", "server B" and "server C" -- for a web application and we have defined an alarm for monitoring memory utilization on these servers. When things are running smoothly, the alarm is in the OK state. Then server A's memory utilization breaks the threshold limit. The alarm switches from OK to FIRING state and an OK_TO_FIRING notification is triggered which will include server A's details in the dimension data. Things calm down on server A, however before the alarm goes to OK, server B's memory utilization breaks the threshold limit. If configured, repeat notifications will trigger – however, instead of server A, the dimension data will have details of server B because the system state has changed now. Please note that there can be multiple resources

in the dimension data. In our example, each new thread will contain the specific dimension data, but it is not parsed into unique incidents beyond the dedupeKey.

## Function Implementation

If you are new to Oracle Functions, here is good quick-start tutorial to get you up to speed. The service documentation is here and more details on how they work are here

Oracle Functions is based on open source F(n) project. You can use different programming languages, libraries, and runtimes to develop your function code. You have complete control of how you want to build the integration. You could also invoke OCI APIs from inside your function code for various-use-cases. For example, instead of storing sensitive information like passwords, tokens and other secrets in plaintext in either Function configuration or environment variables, you could manage secrets in OCI Vault service and read them using APIs, which is a more secure way of storing sensitive data.

**Note:** *Python is the language used for this example. However, the concepts remain the same in other languages as well. Please see the full script example at the end of this document for reference.*

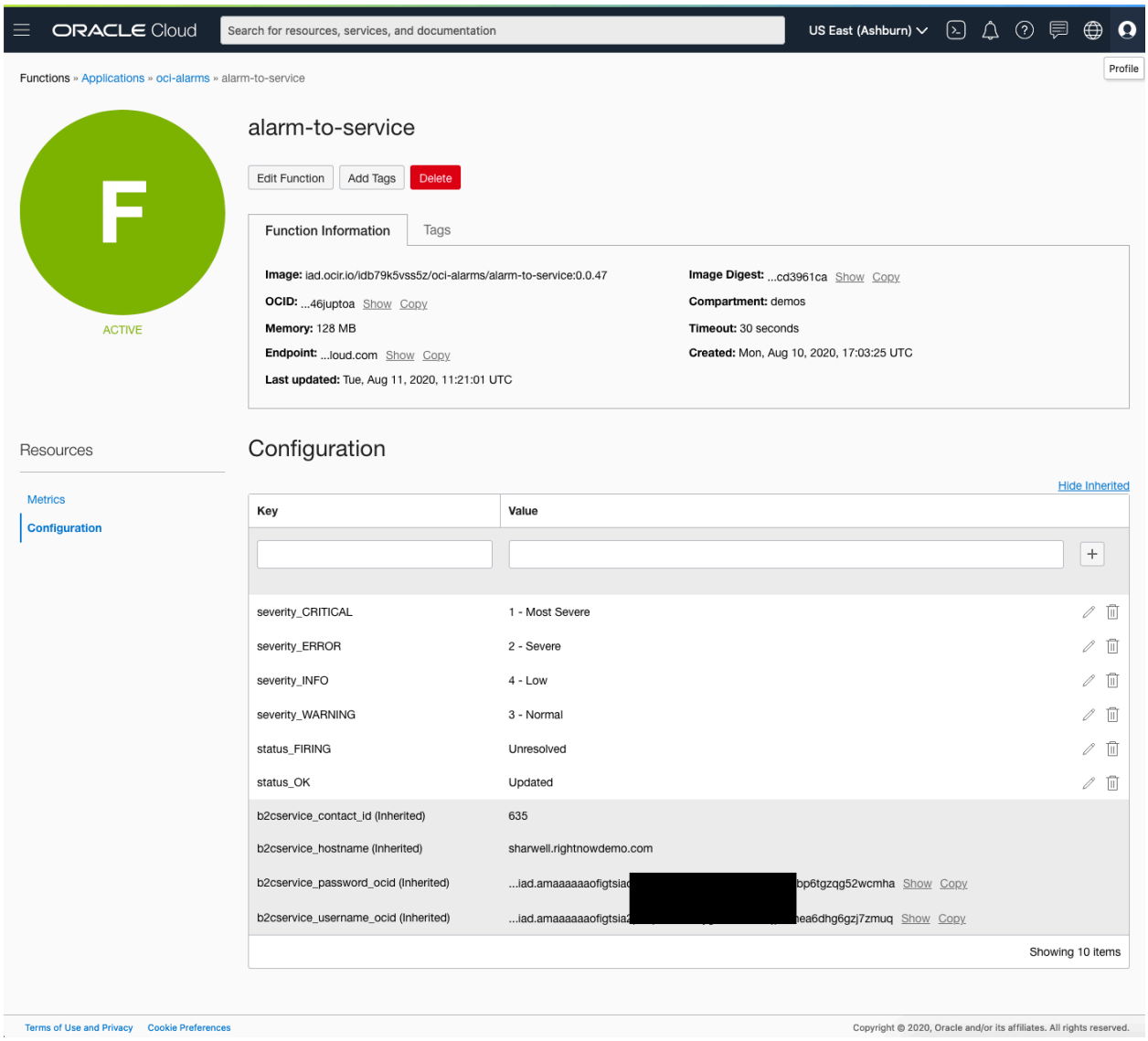
As mentioned above, in case of alarm notifications, input data is a serialized JSON object that can be converted to JSON using something like:

```
alarm_body = json.loads(data.getvalue())
```

Once you have deserialized the alarm string into an object, you then use Python's standard object and dictionary access and mutation techniques to extract and analyze the contained information and take appropriate action.

Functions also accept configuration parameters that are passed in the "ctx" parameter. For example, we can pass the B2C Service hostname, user id, and password through the function's configuration parameters to keep from hard coding any run-time requirements.

In our example, we store the user id and password in a vault as secrets, and then pass the OCIDs of these secrets into the Function via its configuration. The Function can then use OCI APIs to get the user id and password secrets from the vault. You will need to grant the function privileges for reading secrets from a particular vault using OCI IAM Policies. Here is a screen capture of the function's configuration where the B2C Service configuration parameters are managed:




This data is available inside the function in a dictionary like object:

```
# Get environment vars from F(n)
hostname = os.getenv("b2cservice_hostname")
username_ocid = os.getenv("b2cservice_username_ocid")
password_ocid = os.getenv("b2cservice_password_ocid")
```

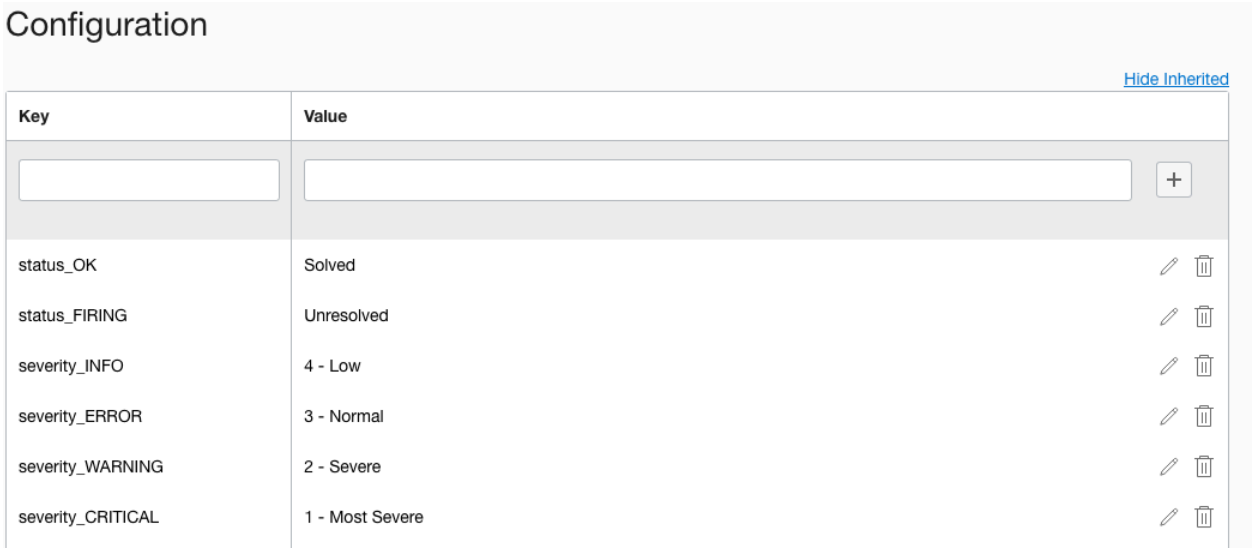
Please note that “b2cservice_username_ocid” and “b2cservice_password_ocid” are OCIDs of the corresponding secrets stored in an OCI vault. OCI provides APIs to read secrets from vault provided proper authorizations are in place. The Python SDK APIs are here.

## B2C Service Integration

For integration with B2C Service, the information from alarm notification data is used to create a B2C Service incident; the subject, severity, and status are set based on parameters in the alarm payload. The alarm message is added as a thread to the created incident. You have the full freedom to create and enrich the B2C service

incident as per your use-cases by using OCI APIs and the power that the programming environment provides. Here are a couple of examples of such mapping:

- The severity levels for OCI alarms are – CRITICAL, WARNING, ERROR and INFO whereas B2C Service expects Custom Menu variables per your environment. This example assumes that your B2C Service severity custom menu options match this OCI values.
- Similarly, “status” from dimension data could be used for “Status” attribute of the B2C Service incident.



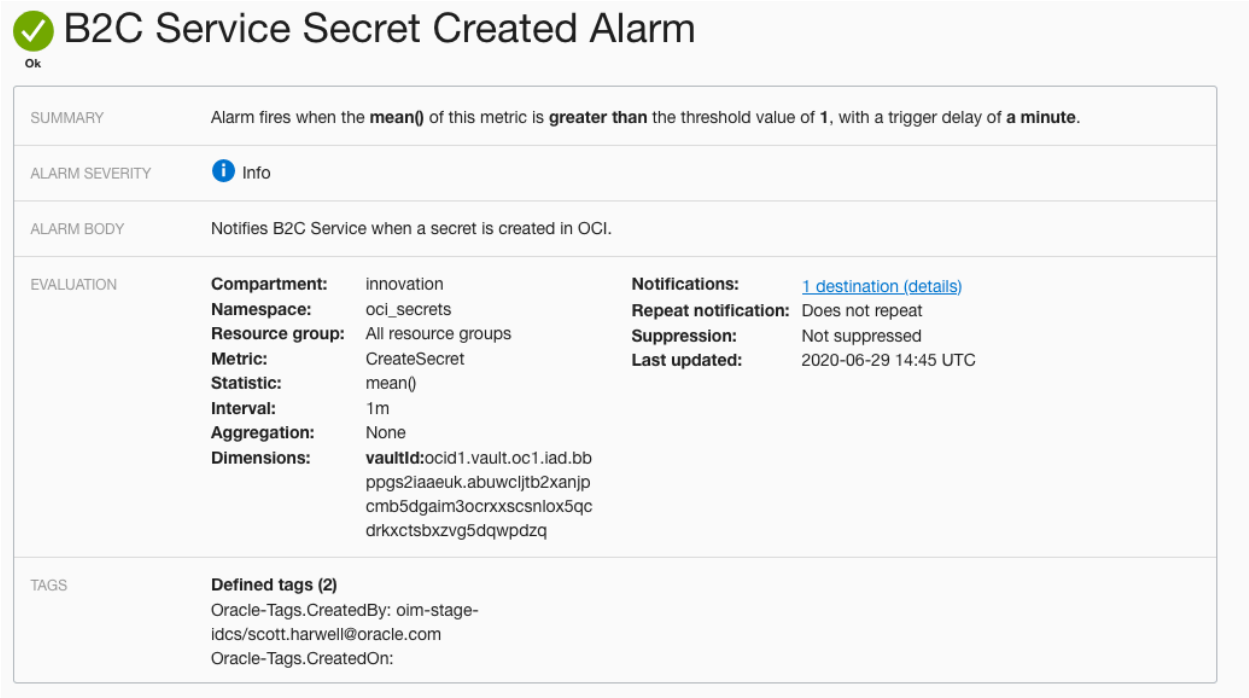

Once you have constructed the incident data, it is sent via a HTTP POST call to the “https://your-site.custhelp.com/services/rest/connect/latest/incidents” endpoint exposed by B2C Service.

For advanced integration scenarios, you could also keep track of (in a cache such as OCI NoSQL or object storage) what individual resources the alarm has fired for so that when the alarm switches back to OK, appropriate CLEAR events could be created in B2C Service.

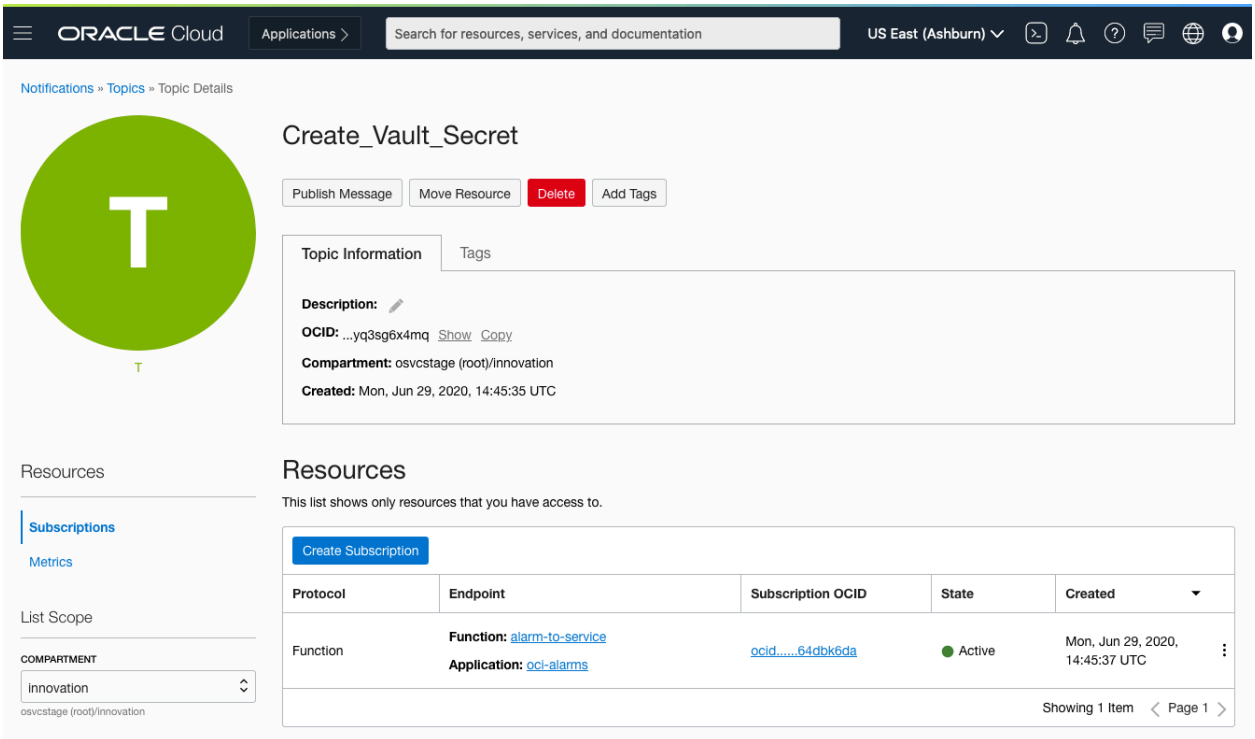
## Solution Configuration

Below demonstrates the end-to-end example configuration for the solution:

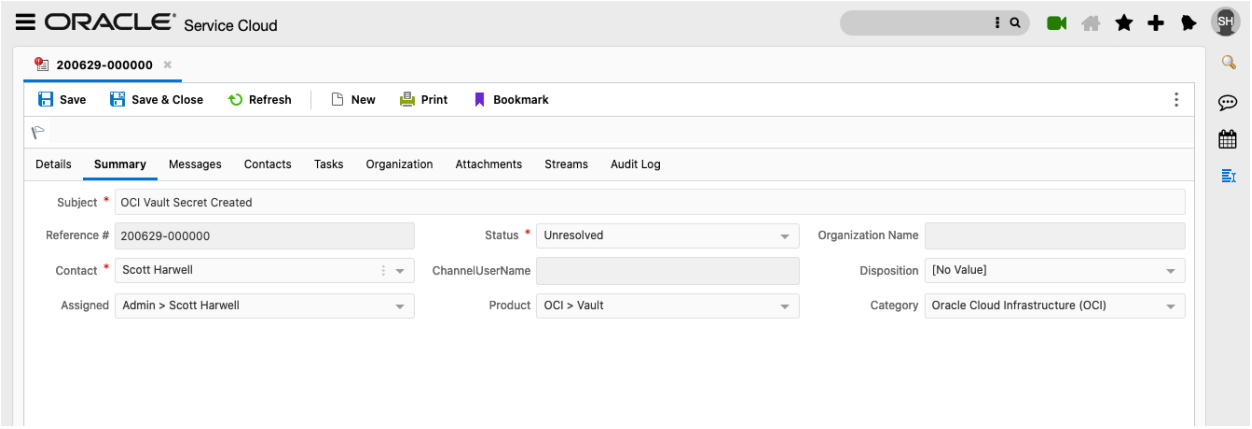
- Alarm Definition: Alarm Definition has Notification Topic configured as destination (pointed by red arrow).




- Notification Topic Configuration: A Notification Topic has one or more subscriptions. Please make note of the Function subscription (highlighted by a red oval)




When the alarm fires, the notification is invoked which in turn triggers the Function, and if the Function is set-up properly then it will in turn create an incident in B2C Service. The following screenshot demonstrates an incident created from the OCI alarm:




## Conclusion

Once the incident exists in B2C Service, then the business processes defined for that event can be initiated for rapid response, such as employee notification, customer outreach, process automation, and other services.

With this example, you should be able to create and configure OCI alarms that initial service-driven business processes when those events occur.

## CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com facebook.com/oracle twitter.com/oracle