Configuring data collection for Node.js applications (draft)

Contents

Configuring Node.js application monitoring

Before you can use the Winterfell Node.js data collector to monitor your Node.js applications in IBM Cloud Private, you must configure data collection.

Configuring the Node.js data collector

Complete the following steps to configure Node.js data collector for your applications:

1. (Conditional) If the service account that is used to configure the data collector does not have access to Kubernetes resources through Kubernetes API, you must first the service account with appropriate access.

   For more information, see "(Conditional) Authorizing the data collector to access Kubernetes resources."
2. For the data collector to connect to the Winterfell server, configure server connection for the data collector.

   For more information, see "Configuring server connection for the data collector" on page 2.
3. Configure data collection for your Node.js applications in IBM Cloud Private.
   - If you deploy the Node.js-based microservices by using IBM Microservice Builder, the Winterfell Node.js data collector is automatically installed in the microservices so you can start monitoring quickly. For more information, see "Deploying a Node.js-based microservice by using Microservice Builder" on page 4.
   - If you have Node.js applications that are NOT created and deployed with Microservice Builder, manually update your current deployment to configure data collection for the applications. For more information, see "Manually configuring data collection for Node.js applications" on page 4.

Disabling the Node.js data collector

If you do not need data collection for your application, disable the Winterfell Node.js data collector. For more information, see "Disabling the Node.js data collector" on page 5.

Upgrading the Node.js data collector

When there is a new version of data collector available, you can upgrade the data collector for Node.js application monitoring to get data collector enhancements. For more information, see "Upgrading the Node.js data collector" on page 6.

(Conditional) Authorizing the data collector to access Kubernetes resources

The service account that you use to configure the data collector must have access to Kubernetes resources through Kubernetes API. Otherwise, you must authorize the service account with appropriate access before you configure the data collector.

About this task

Run the following command on the master node to decide whether the data collector can access Kubernetes resources with the service account, where *kube_namespace* is the target namespace and *service_account* is the account that you use to configure the data collector.

```
kubectl auth can-i list nodes --as system:serviceaccount:kube_namespace:service_account
kubectl auth can-i get pods --as system:serviceaccount:kube_namespace:service_account
kubectl auth can-i list services --as system:serviceaccount:kube_namespace:service_account
```

The following procedure authorizes the service account using Role-Based Access Control (RBAC) authorization. For other authorization methods, refer to Kubernetes documentation.

Procedure

1. Create a `rolebinding.yaml` to bind the service account to a Role that has access to query Kubernetes resources in the RBAC mode.

   The following example binds the `system:serviceaccount:ops-am:default` account to the `admin` ClusterRole.

   ```
   kind: RoleBinding
   apiVersion: rbac.authorization.k8s.io/v1
   metadata:
     name: get-pods
     namespace: ops-am
   subjects:
   - kind: User
     name: system:serviceaccount:ops-am:default
     apiGroup: rbac.authorization.k8s.io
   roleRef:
     kind: ClusterRole
     name: admin
     apiGroup: rbac.authorization.k8s.io
   ```

2. Run the following command:

   ```
   kubectl create -f rolebinding.yaml
   ```

3. Create a `clusterrolebinding.yaml` file to bind the service account to a ClusterRole that has access to query Kubernetes resources in the RBAC mode.

   The following example binds the `system:serviceaccount:ops-am:default` account to the `cluster-admin` ClusterRole.

   ```
   kind: ClusterRoleBinding
   apiVersion: rbac.authorization.k8s.io/v1
   metadata:
     name: list-cluster
   subjects:
   - kind: User
     name: system:serviceaccount:ops-am:default
     apiGroup: rbac.authorization.k8s.io
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: cluster-admin
   ```

4. Run the following command:

   ```
   kubectl create -f clusterrolebinding.yaml
   ```

Configuring server connection for the data collector

For the data collector to connect to the Winterfell server, you must configure server connection for the data collector.

About this task

To configure server connection for the data collector, you must create the server secret in the same namespace where the to-be-monitored microservices or applications are running. The URL for the Winterfell server must be specified in the server secret. If you do not know the URL, contact the server administrator.

Tip: The Winterfell server URL is in the format of `http://`*`ingress_service_name`*`.kube_namespace.svc.cluster.local/1.0/data`, where *ingress_server_name* is the ingress service name in the **ibm-apm-server-dev** Helm release and *kube_namespace* is the namespace of the Winterfell server. You can run the `kubectl get svc –namespace=`*`kube_namespace`* command to look for the ingress service name.

Procedure

Complete the following steps in the namespace where the microservices or applications that are to be monitored are running:

1. Create a `yaml` file named `apm-server-config-secret.yaml` on your local system.
2. Edit the `apm-server-config-secret.yaml` file to add the following information:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: apm-server-config
   data:
     ibm_apm_ingress_url: winterfell_server_url
   ```

   where, *winterfell_server_url* is the base64 encoded URL of the Winterfell server.

   For example, run the following command on a Linux system to get the encoded URL for the Winterfell server:

   ```
   echo -n http://am-server01-ingress.ops-am.svc.cluster.local/1.0/data | base64
   ```

   The returned encoded URL is as follows:

   ```
   aHR0cDovL2FtLXNlcnZlcjAxLWluZ3Jlc3Mub3BzLWFtLnN2Yy5jbHVzdGVyLmxvY2FsLzEuMC9kYXRh
   ```

3. On the Kubernetes master node and in the same namespace of your application, create the server secret by running the following command:

   ```
   kubectl create -f apm-server-config-secret.yaml
   ```

What to do next

Configure data collection for your Node.js applications in IBM Cloud Private.

- If you deploy the Node.js-based microservices by using IBM Microservice Builder, the Winterfell Node.js data collector is automatically installed in the microservices so you can start monitoring quickly. For more information, see “Deploying a Node.js-based microservice by using Microservice Builder” on page 4.
- If you have Node.js-based microservices or applications that are NOT created and deployed with Microservice Builder, manually update your current deployment to configure data collection for the microservices or applications. For more information, see “Manually configuring data collection for Node.js applications” on page 4.

Deploying a Node.js-based microservice by using Microservice Builder

If you used IBM Microservice Builder to create your Node.js microservices, the Winterfell Node.js data collector is automatically installed in the microservice so you can start monitoring quickly.

Before you begin

If the Microservice Builder pipeline, which is required to deploy microservices from GitHub or GitHub Enterprise to IBM Cloud Private was not previously set up, set it up now. For more information, see Setting up the Microservice Builder pipeline.

Procedure

1. Create a Node.js microservice by running the following command. For more information, see Create and deploy a simple microservice.

   ```
   bx dev create
   ```

   Remember: After you run the **bx dev create** command and follow the prompts to create a microservice, select **Node** as the language.
   When this step completes, a directory is created within the current directory with the project name that you specify.
2. Modify the project files to customize your application.
3. Deploy your application with the Microservice Builder pipeline by pushing the project to a new repository in the GitHub organization that the Microservice Builder pipeline is monitoring.

What to do next

From your browser, log in to the Winterfell console to review the health status of your Java services in the dashboards. .

You can also configure thresholds for the monitored microservices to be alerted of real and potential issues.

Manually configuring data collection for Node.js applications

If your Node.js applications are not deployed using Microservice Builder, manually update the deployment to configure data collection for the your applications.

Before you begin

If your environment does not have internet connection, run the **npm install ibmapm** command from a system that has internet connection. Then, copy the entire `node_modules` folder in the output directory to the target system.

Procedure

1. Update the Dockerfile for your Node.js application to add the following line that start with `RUN`:

   ```
   RUN npm install ibmapm --save
   ```

   No internet connection: If your current system cannot access your NPM package repository, go to a system that can connect to the repository, run the

npm install ibmapm command. In the application directory, copy the `node_modules` directory to your current system.

2. Add the following line to the beginning of the main file of your Node.js application:

   ```
   require('ibmapm');
   ```

   Tip: If you start your application by running the **node app.js** command, `app.js` is the main file of your application.

3. Remove the current Docker image by running the **docker rmi** command.

   The following example removes the Docker image by using the image tag:

   ```
   docker rmi -f mycluster.icp:8500/admin/trader:node.js
   ```

4. Build and tag the Docker image again with the updated Dockerfile by running the following command:

   ```
   docker build -t new_image_tag .
   ```

   where *new_image_tag* is the Docker image tag, which must contain the IBM Cloud Private registry. Example:

   ```
   docker build -t mycluster.icp:8500/admin/trader:newnode.js .
   ```

   where `mycluster.icp:8500` is the IBM Cloud Private registry.

5. Push the new Docker image to your registry by running the following command:

   ```
   docker push new_image_tag
   ```

   where *new_image_tag* is the Docker image tag that you specified in the previous step. Example:

   ```
   docker push mycluster.icp:8500/admin/trader:newnode.js
   ```

6. Edit the Node.js application deployment `yaml` file by adding the following variables to the container specification. Specify your application name in the APPLICATION_NAME value.

   ```
   - name: APPLICATION_NAME
     value: your_app_name
   - name: IBM_APM_INGRESS_URL
     valueFrom:
       secretKeyRef:
         name: apm-server-config
         key: ibm_apm_ingress_url
         optional: true
   ```

7. Update the Node.js application deployment by running the **kubectl replace** command.

   ```
   kubectl replace -f updated_nodejs_app_deployment.yaml
   ```

What to do next

From your browser, log in to the Winterfell console to review the health status of your Java services in the dashboards. .

You can also configure thresholds for the monitored microservices to be alerted of real and potential issues.

Disabling the Node.js data collector

To disable the Node.js data collector, roll back the changes that you have made to your application and then update the application deployment.

Procedure

1. Edit the main file of your Node.js application to remove the following line:
   ```
   require('ibmapm');
   ```
2. If the Node.js application is deployed by using Microservice Builder, push your project to a new repository that the Microservice Builder pipeline is monitoring.
3. If the Node.js application is not deployed by using Micorservice Builder, complete the following steps to update the application deployment.
   a. Remove the current Docker image by running the **docker rmi** command.

      The following example removes the Docker image by using the image tag:

      ```
      docker rmi -f mycluster.icp:8500/admin/trader:node.js
      ```
   b. Build and tag the Docker image again with the updated Dockerfile by running the following command:

      ```
      docker build -t docker_image_name:image_tag .
      ```

      where *image_tag* is the Docker image tag, which must contain the IBM Cloud Private registry. Example:

      ```
      docker build -t mycluster.icp:8500/admin/trader:node.js .
      ```

      where `mycluster.icp:8500` is the IBM Cloud Private registry.
   c. Push the new Docker image to your registry by running the following command:

      ```
      docker push image_tag
      ```

      where *image_tag* is the Docker image tag that you specified in the previous step. Example:

      ```
      docker push mycluster.icp:8500/admin/trader:node.js
      ```
   d. Edit the application deployment `yaml` file to use the new Docker image.
   e. Update the Node.js application deployment by running the **kubectl replace** command.

      ```
      kubectl replace -f updated_nodejs_app_deployment.yaml
      ```

Upgrading the Node.js data collector

When there is a new version of Node.js data collector available, to upgrade the data collector, repeat the steps that you completed to configure data collection for the first time.

Procedure

- If you deploy the Node.js-based microservices by using Microservice Builder, see "Deploying a Node.js-based microservice by using Microservice Builder" on page 4.
- If your Node.js application is not deployed by using Microserver Builder, see "Manually configuring data collection for Node.js applications" on page 4.