REVISION HISTORY

Version	Date	Change description	Created by	Updated by	Reviewed by
V 1.0	November, 2022	Release 6.0	Hemanth/Yash	Raksha	Chiranjib

Deployment Monitoring

Overview

After the transition of CAN solution to a micro-service architecture in CAN 5.5 release, there are several improvements like reduction in deployment time, ability to deploy anywhere because of containerization, flexibility in hybrid and multi-cloud environments, horizontal pod auto-scaling, increased data security and many more. This transition towards micro-service based application indicates that monitoring the complete Kubernetes cluster (Pod/Service monitoring) along with the monitoring of resource utilization of underlying server (Node monitoring) is a crucial aspect.

From the past deployment experiences, there are certain concerns related to increased operational complexity, Inter-Service communication breakdown, debugging issues and many more. Below is the list of few challenges posted from a day-to-day basis:

- Is the application up and operating all the time?
- Are messages being routed to and from the designated application correctly?
- Does each component of the application have sufficient resources (memory, disk, and network) to operate successfully?
- Are service nodes failing to start? Are they failing after a period? Are they failing when idle? Do they seem to fail randomly?

These challenges are addressed effectively in the 6.0 release with the help of monitoring tools like **Prometheus** (Time series database), **Grafana** (Visualization software) and **Kiali** (Service mesh dashboard) by taking appropriate measures like:

- Continuously monitoring CPU / Memory / Network utilisation of each pod.
- Creating alerts and sending notifications in the event of specified condition or a failure.
- Ability to view the logs of all the pods without SSH access of the worker nodes.
- Ability to create alerts based on any type of business metrics, resource metrics or any custom defined metrics.

Architecture

CAN 6.0 monitoring solution uses layered architecture that helps the user to operate without having to know the details of underlying APIs. The architecture mainly includes independent entities such as Keycloak, Prometheus, Grafana and Kiali.

1. **Keycloak:** Responsible for Authentication, Authorization, SSO, Session management and routing the requests for the designated underlying pods.
2. **Prometheus:** Responsible for collecting and storing Business metrics, Resource metrics and Custom metrics from all the available pods in all the namespaces.
3. **Grafana:** Responsible for displaying metrics and log data in the form of visualisations and reporting dashboards. It also provides alert creation and notification functionality.

4. **Kiali:** Responsible for providing a web-based graphical user interface to view service mesh related graphs and Istio configuration objects. It also provides the functionality of log monitoring without an SSH access.

Following is the high-level architecture view of the CAN 6.0 monitoring solution:

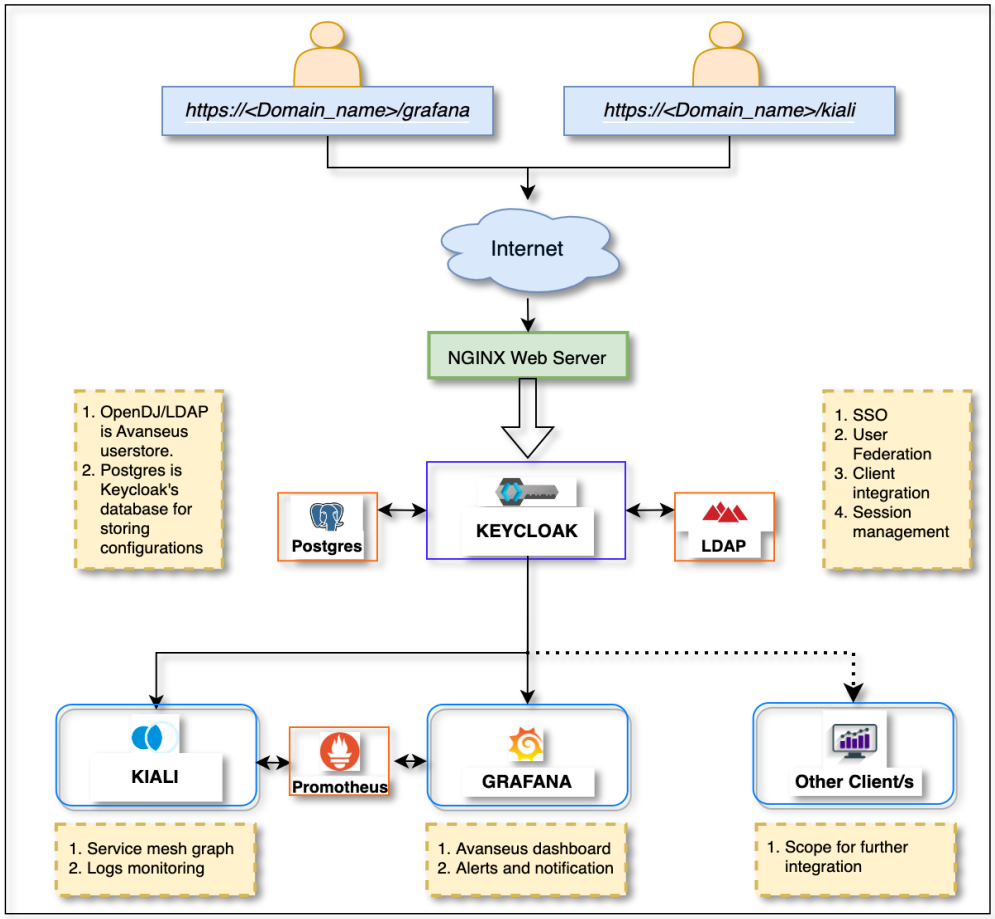

Figure 1. High-level architecture

Components

This section gives a high-level understanding of the functionalities offered by each of the software components used in the architecture such as Keycloak, Prometheus, Grafana and Kiali. For detailed explanation and for hands-on section, refer the below mentioned documents:

1. Keycloak and its Integration with Grafana and Kiali.
2. Prometheus role in CAN monitoring.
3. Monitoring deployments using Grafana.
4. Recommended Alerts using Grafana.
5. Customizing Alerts using Grafana.

Keycloak

Keycloak is an authentication layer to support Single-Sign On for multiple clients with minimum effort. In this context, Grafana and Kiali are the clients. In future, if there is a requirement to add any other software(s), it will be added as a client in keycloak to leverage the benefits such as SSO, Authorization, Session management etc.,

The below diagram shows the authentication flow between user, keycloak and applications (Grafana and Kiali).

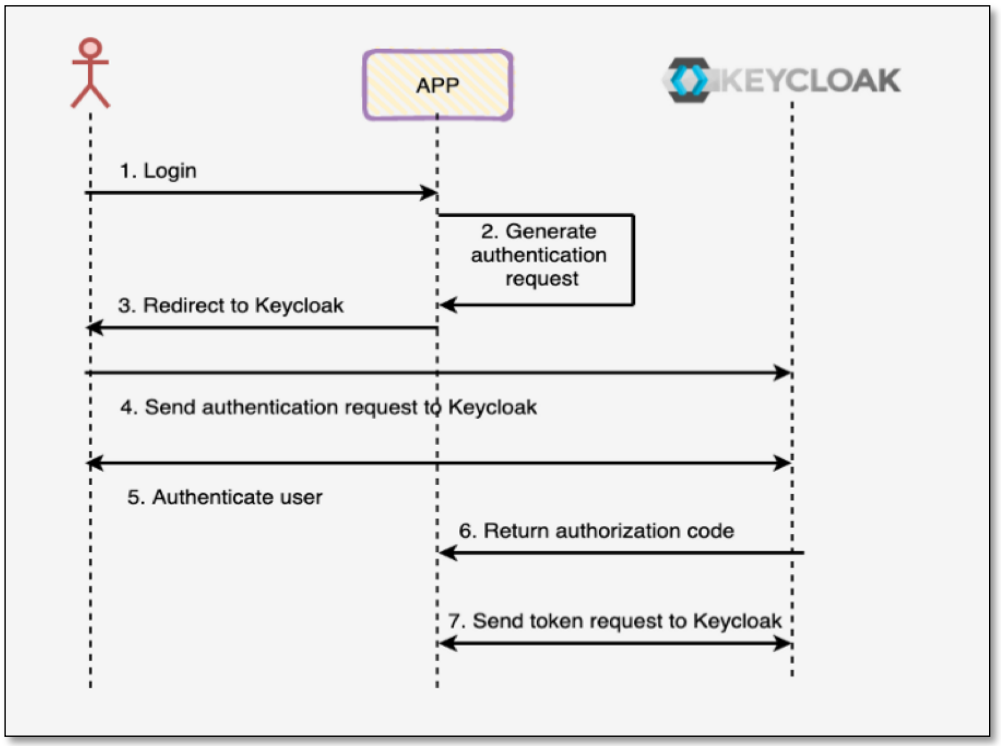

Figure 2. Authentication using keycloak

For ease of use in Keycloak, the Admin console is pre-configured with a custom realm named **"Avanseus-keycloak"** in which the following pre-requisite steps are performed:

1. Addition of existing LDAP server as a provider in the user federation section.
2. Configuration of Grafana and Kiali as clients, so that grafana / kiali supports SSO.

Along with this, user can manage permissions and sessions in the admin console. For more details about configuration, refer to ***Keycloak and its Integration with Grafana and Kiali*** document.

Prometheus

Prometheus is an open-source monitoring solution for collecting and aggregating metrics as time series data, i.e., metrics information is stored with the timestamp at which it was recorded.

Prometheus fundamentally relies on scraping, or pulling, metrics from defined endpoints. Prometheus stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data. In our case, we have used Grafana and Kiali to visualize and to generate alerts from the collected data.

For more details about Prometheus like installation, Architecture, metric types, custom metrics and many more please refer ***Prometheus role in CAN monitoring*** document.

Grafana

Grafana is a multi-platform open source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources.

Prometheus is the data source. Avanseus query Prometheus data and display it on customizable charts for further analysis. Grafana possesses a variety of visualisation options to view and understand data like CPU usage, Memory usage etc.

For ease of use in Grafana, a pre-configured dashboard is created by name **"Avanseus-dashboard"** with some custom panels. For more details, refer to ***Monitoring deployments using Grafana*** document.

Figure 3. Avanseus custom dashboard

It is also possible to configure alert and send notifications from grafana. This notification is helpful to take reactive actions to prevent or mitigate the possible downtime of pods/nodes.

To create an alert, create and configure an alert rule first. This rule serves as a trigger for the alert, such that whenever said rule is broken, an alert notification is sent via the alert notification channel. For more details about how to configure alerts, refer to ***Recommended Alerts using Grafana*** document.

Kiali

Kiali is a management console for Istio service mesh. Kiali is installed as an Istio add-on to leverage its features. Avanseushas integrated with Kiali to understand the structure and health of service mesh, to monitoring the traffic flow, to infer the topology, to view the logs and to report errors.

Avanseus solutions uses Kiali for the below features:

1. **Overview Tab**

   The overview tab provides detailed information like health status, and a detailed mini-graph of the current traffic of all namespaces. The full set of tabs, as well as the detailed information, varies based on the namespace.

2. **Graph**

   The graph provides a powerful way to visualize the topology of the service mesh. It shows which services communicate with each other and the traffic rates and latencies between them, which helps to visually identify problem areas and pinpoint issues quickly.

3. **Details**

   Kiali provides filtered list views of all the service mesh definitions. Each view provides health, details, YAML definitions and links to visualize mesh. There are list and detail views for,

   - Applications
   - Workloads
   - Services

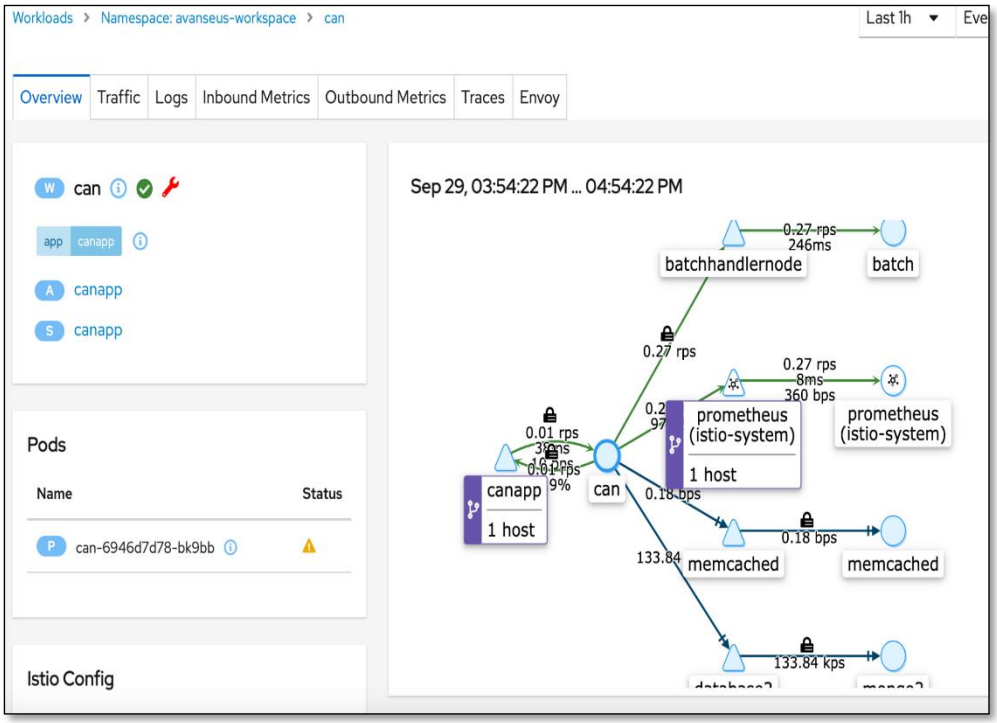

Figure 4. Kiali dashboard (Describing CAN pod)

Selecting an object from the list will navigate to its detail page that includes the below tabs:

1. **Overview:** The overview tab provides detailed information; spark line graph and health overview for a specific Application, Workload or Service.
2. **Traffic:** The traffic tab collects all connections from and to a specific Application, Workload and Service. It groups them in Inbound and Outbound tables providing traffic details and links to specific detailed metrics.
3. **Logs:** The Workload detail offers a special logs tab. Kiali allows side-by-side viewing of both the application and the proxy logs of a selected pod.
4. **Detailed Metrics:** Application and workload detail views show request and response metrics such as volume, duration, size, or TCP traffic. The service detail view shows request and response metrics for inbound traffic.
5. **Detailed Traces:** Navigate to the traces tab to browse filtered traces for a given service in the time interval or to show details for a single trace. After selecting a trace, Kiali shows the information related to that trace like number of spans, spans grouped by operation name, duration and date.