SERVERLESS IOT PLATFORM

INTRODUCTION

A server-less IoT solution development and management platform offers core functionality for managing the entire lifecycle of IoT solution development. A configuration based environment allows users to quickly design, develop, deploy and manage complex deployments. Easily build and manage complex business logic, integrations and devices. The solution allows extensibility and customization.

BACKGROUND WORK

If you are building an IoT solution in the cloud, chances are your focus is on the devices and what you can accomplish with them. You might want to process data coming from a network of devices in real time, analyze the data to gain insights, get alerted for special conditions, manage the devices themselves, and so on. What is less interesting to you is setting up and managing the infrastructure in the cloud, which will enable you to do the above. This is where serverless comes in.

Serverless technologies, like Azure Functions, take away the burden of managing infrastructure and enable you to focus on your IoT-powered business logic. IoT projects usually have variable traffic, which means accumulating infrastructure to account for peak loads, isn't the best strategy. Adopting serverless allows your solutions to scale dynamically while keeping costs low.

RELATED WORK

Related Works include-

# Microsoft Azure IOT	# Google Cloud IOT	# IOT with OpenWhisk
# Kubeless	# Foundry	# Kappa
# Amazon AWS	# Protego	

In this paper, we try to propose the loopholes in our implementation of such a platform when compared to existing already built ones. Specifically, we try to compare a few modules of an IOT Platform.

STUDY & COMPARISON

Action & Notification Service-

(**NOTE**:- Used Apache Kafka as the Technology to build the Action & Notification Module)

✔Comparison with Azure IOT's Notification Hubs

- We did not provide support for message broadcast, Notification Hubs used in Azure IOT, however support fast broadcast push to millions of devices/ customers with a single API call.
- Kafka(used in our service) lacks any kind of security mechanism as of today. Azure Event Hub, on the other hand, is very secure.

- Kafka persists the messages even after consumed until a certain retention period, which is of no use in the notification service as a customer has to be notified about an action only once after which the message is of no use. But in Notification Hubs (AMQP) messages are stored only until a receiving application connects and receives a message off the queue. The client can either ack (acknowledge) the message when it receives it or when the client has completely processed the message. In either situation, once the message is acked, it's removed from the queue.
- Delivering messages with AMQP gives reliability and being asynchronous allows not to worry about the delivery at all. No such guarantee is met in Kafka.
- Notification Hubs supports something called priority queues, meaning that a queue can be set to have a range of priorities. The priority of each message can be set when it is published and so urgent notifications can be sent first. However,a message cannot be sent with a priority level, nor be delivered in priority order, in Kafka. All messages in Kafka are stored and delivered in the order in which they are received regardless of how busy the consumer side is.

Reference- https://azure.microsoft.com/en-in/services/notification-hubs/

✔Comparison with OpenWhisk's HTTP REST API

- As per our implementation we notify the developer/ customer when a certain event condition is met, say scheduling the application is done or deploying is completed. However, say we want 'on-demand' notifications i.e. as and when the user demands of a notification, our platform is incapable of doing the same. OpenWhisk HTTP API is capable of doing so.
- Kafka stores data in the topic until it is retained so an intruder can seek back and forth to get user credentials until it is managed to delete the data from the Kafka topic manually. However, in HTTP REST once the response is over, it is over.
- We failed to make our notification service compatible with audio messages(it can be done though), however OpenWhisk has a support for audio notifications as well.

Reference- https://openwhisk.apache.org/documentation.html

✔Comparison with AWS's SNS/SQS

- Amazon SNS/SQS(as used by AWS IOT platform) has notifications for email, SMS, SQS, HTTP built-in. With Kafka, we had to code it using some python libraries.
- SQS is serverless, while there is a Kafka Server.
- Messages can be sent, received or deleted in batches of up to 10 messages or 256KB in Amazon AWS. Though it can be done by configuring some settings in Kafka too, we have not handled that to be able to notify our customer with a batch of notifications.

Reference-
https://aws.amazon.com/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc

Health Checkup & Monitoring Services-

(**NOTE**:- Used a combination of HTTP REST API and Apache Kafka to implement Health Checkup and Monitoring Service)

✔Comparison with AWS's Cloudwatch

- Auto Recovery can be implemented with the help of CloudWatch. If an instance fails a system status check, CloudWatch can automatically reboot or recover it. The Health Checkup Module as per our implementation requires manual work to do so.

- Cloudwatch provides features of troubleshooting and recommendation for the future and how to avoid the existing errors, it can monitor the resultant performance and cost of the application. Our Monitoring Service, on the other end, provides no such facilities.
- This technology is more user friendly as compared to our monitoring system as it provides statistics by creating charts describing the workload level of cloud services.
- Cloudwatch offers control access to the AWS Personal Health Dashboard and AWS Health using IAM to create identities (users, groups, or roles), and then give those identities permissions to access the Personal Health Dashboard and AWS Health API. There is no such feature in our implemented Health Check-up Module.

Reference- https://aws.amazon.com/cloudwatch/
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

✔Comparison with Kubeless Prometheus & Grafana

- Kubeless provides a barometer to check the health of apps and self-heal apps with auto-placement, auto-restart, auto-replication, and autoscaling. There is no such automatic recovery in our implemented Health Service, it only keeps an eye as to which server is healthy, if any of it goes down, the load balancer comes into picture to start a new similar service/ server but there is no healing of the ill service/ server.
- We could have used Grafana as Prometheus does which works as a sandbox for graphic visualization, which enables creating themed dashboards with several graphics by collecting data from different data sources and metrics.
- Prometheus monitors the actual amount of time between target scrapes configured using the *prometheus_target_interval_length_seconds* metric which helps to identify latency issues whenever a sudden rise in latency is something that needs to be explored. Our implementation makes no latency checks as such.

Reference- https://kubeless.io/docs/monitoring/

✔Comparison with Google Cloud IOT's Cloud Monitoring

- Cloud Monitoring provides alerts to send notifications when certain metric thresholds are exceeded. No thresholds check is maintained here, only load distribution is done as per the metrics of the different servers.
- Application Performance Management (APM) includes tools to help you reduce latency and cost, so you can run more efficient applications. With Cloud Trace, Cloud Debugger, and Cloud Profiler, you gain insight into how your code and services are functioning and troubleshoot if needed.
- More or less all the above listed advantages of Monitoring Service in other Platforms are also manifested by Google Cloud Monitoring.

Reference- https://cloud.google.com/run/docs/monitoring
https://cloud.google.com/products/operations

CONCLUSION

Serverless IoT architecture can help to provide IoT businesses with incredible savings, scalability and performance. By leveraging the power of the serverless IoT Rules Engine, one can take advantage of creating a flexible and scalable IoT backend that can be used to manage your growing Internet of Things fleet.