Documentation of Someip

Index

1.vSomeIP Overview:

vSomeIP (Virtualized Scalable Service-Oriented MiddlewarE over IP) is an implementation of the SOME/IP protocol, which is a communication protocol used in automotive and other embedded systems for service-oriented communication over IP networks. vSomeIP enables inter-process communication between software components in a distributed system. It provides features like message serialization, service discovery, and end-to-end (E2E) protection.

2.Prerequisites:

Make sure you have installed docker and Kubernetes on the local system.

2.1 Docker Installation:

Docker provides installation guides for various operating systems on their official website. You can find the appropriate installation guide for your OS here

You can find the Docker Desktop installation instructions for Windows here.

Follow the instructions provided in the Docker installation guide for your specific operating system, and it will walk you through the installation process.

2.2 Kubernetes Installation:

Kubernetes provides installation guides for various operating systems on their official website. You can find the appropriate installation guide for your OS here

You can find the Kubernetes installation instructions for Windows here.

Follow the instructions provided in the Kubernetes installation guide for your specific operating system, and it will walk you through the installation process.

3.Installation and Upgrade:

Step 1: Pull the image from the Azure Container Registry (ACR) To pull the Docker image from the ACR, you use the **docker pull** command followed by the image name and tag/version:

```
docker pull scutiregistry.azurecr.io/someip_example:version
```

Replace **someip_example** with the name of the image and **version** with the specific version or tag you want to pull from the registry.

Step 2: Add the image in the YAML file

In this step, you are adding the pulled image to a Kubernetes YAML file, where you specify the image in the **spec** section of a container definition. Here's an example of how you can do that:

```
apiVersion: v1
kind: Pod
metadata:
  name: vsomeip-notify
spec:
  containers:
  - name: someip-example-container
    image: scutiregistry.azurecr.io/someip-example:v1.0.1190-scuti
    command: ["sh", "/start-notify-udp.sh"]
```

In this example, we define a Kubernetes Pod that runs a single container with the image **scutiregistry.azurecr.io/someip-example:v1.0.1190-scuti**. We also specify the command to run inside the container using the **command** field.

Step 3: Run the YAML file

To deploy the YAML file and create the specified Pod with the container, you use the **kubectl apply** command:

```
kubectl apply -f file-name.yaml
```

Replace **file-name.yaml** with the actual name of your YAML file.

After running this command, Kubernetes will create the necessary resources based on the YAML file you provided, including the Pod with the specified container.

```
pod/vsomeip-notify created
pod/vsomeip-subscriber created
```

4.Configuration:

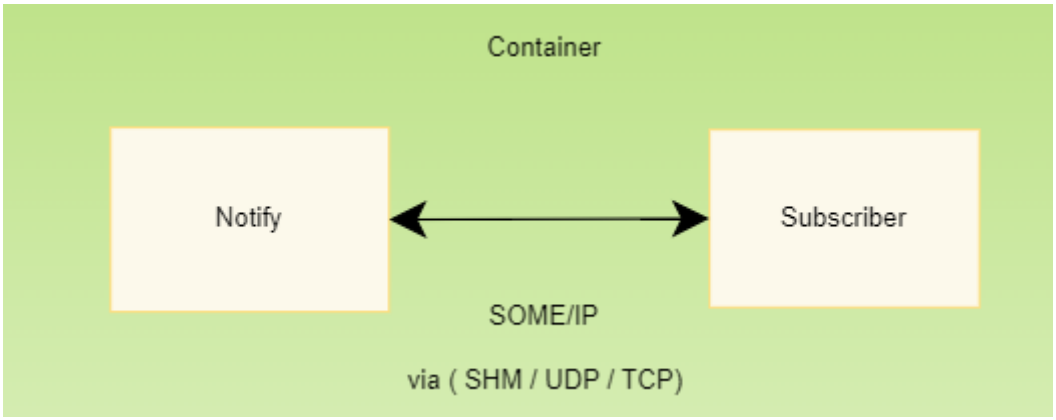
4.1 TCP

4.2 UDP

4.3 Shared Memory

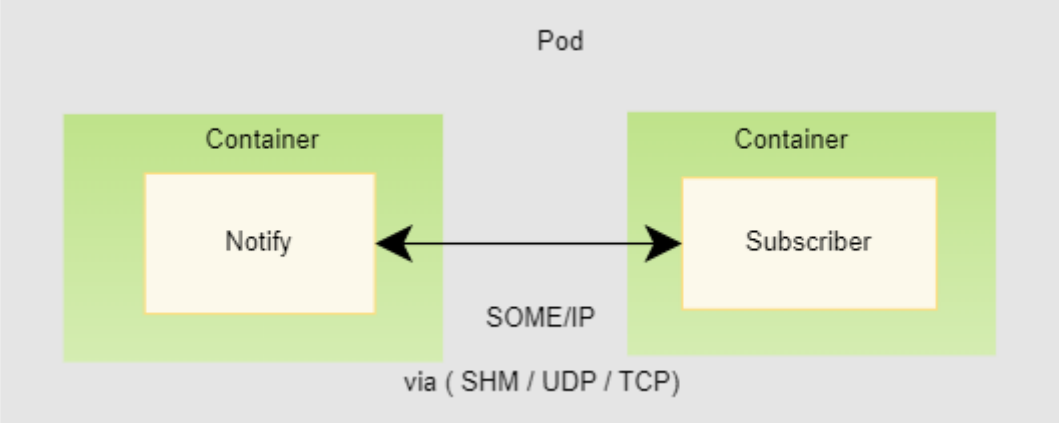
5.Communication styles:

5.1 Applications running in same containers :



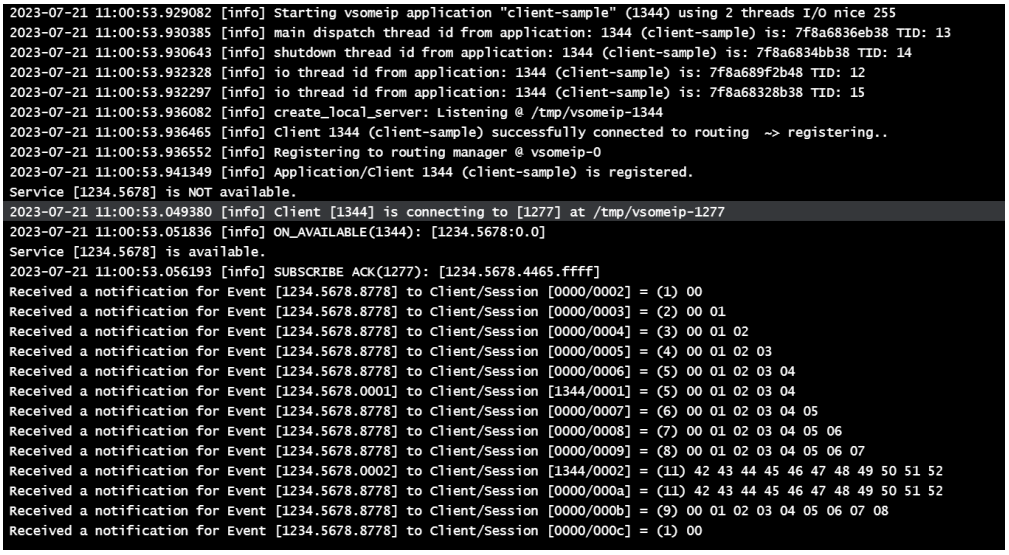
- **Container**: Represents a single container that includes both the "Notifier" and "Subscriber" applications running together.
- **Applications**: The container contains two applications: the **"Notifier"** application and the **"Subscriber"** application.
- **Notifier**: Represents the "Notifier" application that generates events or notifications.
- **Subscriber**: Represents the "Subscriber" application that listens for and processes the events or notifications sent by the "Notifier."
- In this scenario, both the "Notifier" and "Subscriber" applications communicate smoothly and exchange information through the SOME/IP protocol inside the container, without needing any external networks or devices

5.2 Applications running in different containers, in a same Pod:



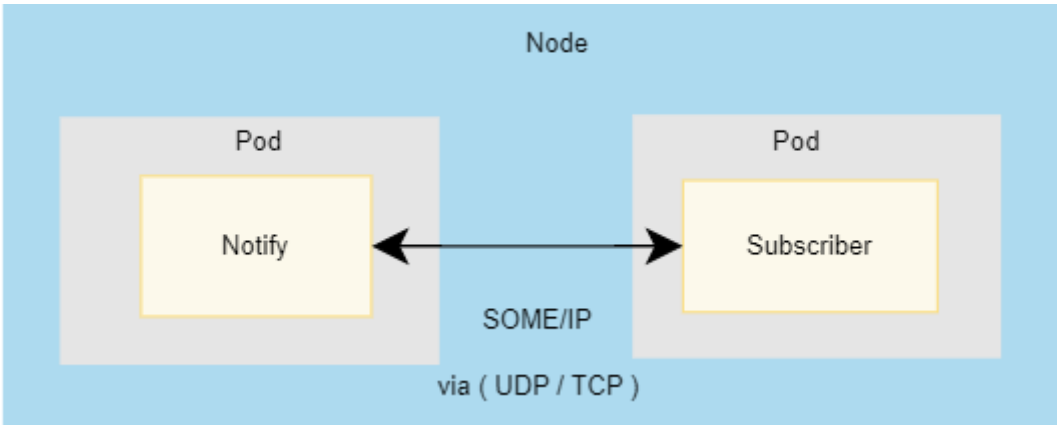

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

No resources found in default namespace.
PS C:\Users\ANET1KOR> kubectl apply -f .\vsomeip-sharedmemory.yaml
pod/vsomeip-multipod created
PS C:\Users\ANET1KOR> kubectl get pods
NAME               READY   STATUS    RESTARTS   AGE
vsomeip-multipod   2/2     Running   0          26s
PS C:\Users\ANET1KOR>
```

```
2023-07-21 11:00:53.670940 [info] Application(service-sample, 1277) is initialized (11, 100).
2023-07-21 11:00:53.671077 [info] offer_event: Event [1234.5678.8778] uses configured cycle time 0ms
2023-07-21 11:00:53.671312 [info] REGISTER EVENT(1277): [1234.5678.8778:is_provider=true]
2023-07-21 11:00:53.671868 [info] Starting vsomeip application "service-sample" (1277) using 2 threads I/O nice 255
2023-07-21 11:00:53.671990 [info] OFFER(1277): [1234.5678:0.0] (true)
2023-07-21 11:00:53.674198 [info] create_local_server: Listening @ /tmp/vsomeip-1277
2023-07-21 11:00:53.673916 [info] shutdown thread id from application: 1277 (service-sample) is: 7f87f505ab38 TID: 17
2023-07-21 11:00:53.673945 [info] Client [1277] routes unicast:10.0.2.15, netmask:255.255.255.0
2023-07-21 11:00:53.674372 [info] main dispatch thread id from application: 1277 (service-sample) is: 7f87f507db38 TID: 16
Setting event (Length=1).
2023-07-21 11:00:53.678187 [info] Watchdog is disabled!
Application service-sample is registered.
2023-07-21 11:00:53.679140 [info] io thread id from application: 1277 (service-sample) is: 7f87f500fb38 TID: 19
2023-07-21 11:00:53.679216 [info] io thread id from application: 1277 (service-sample) is: 7f87f57d2b48 TID: 12
2023-07-21 11:00:53.679436 [info] vSomeIP 3.3.8 | (default)
2023-07-21 11:00:53.937364 [info] Application/Client 1344 is registering.
2023-07-21 11:00:53.938291 [info] Client [1277] is connecting to [1344] at /tmp/vsomeip-1344
2023-07-21 11:00:53.942428 [info] REGISTERED_ACK(1344)
2023-07-21 11:00:53.943362 [info] REGISTER EVENT(1344): [1234.5678.8778:eventtype=2:is_provided=false:reliable=255]
2023-07-21 11:00:53.047425 [info] REQUEST(1344): [1234.5678:255.4294967295]
2023-07-21 11:00:53.054267 [info] SUBSCRIBE(1344): [1234.5678.4465:ffff:0]
Setting event (Length=2).
Setting event (Length=3).
Setting event (Length=4).
Setting event (Length=5).
Setting event (Length=6).
Setting event (Length=7).
```



- In this scenario, The pod contains two containers: "Container 1" (Notify) and "Container 2" (Subscriber).
- The **"Notifier"** container (Container 1) is responsible for publishing events or notifications using the SOME/IP protocol. It encodes the event data into SOME/IP messages and sends them out.
- On the other hand, the **"Subscriber"** container (Container 2) is responsible for subscribing to these events through the SAME/IP protocol. It receives the SOME/IP messages sent by the "Notifier," decodes them to extract the event data, and processes the events accordingly.
- Both the **"Notifier"** and **"Subscriber"** containers are co-located within the same pod, which means they share the same network namespace. This allows them to effectively communicate with each other using the "localhost" network interface. The shared network namespace enables seamless and efficient communication between the containers, facilitating the exchange of SOME/IP messages for the publish-subscribe pattern to work effectively within the Kubernetes pod.

5.3 Applications running in different Pod, in a same Node :




```
PS C:\Users\ANET1KOR> kubectl get pods
No resources found in default namespace.
PS C:\Users\ANET1KOR> kubectl apply -f .\vsomeip-udp.yaml
pod/vsomeip-notify created
pod/vsomeip-subscriber created
PS C:\Users\ANET1KOR> kubectl get pods
NAME                 READY   STATUS    RESTARTS   AGE
vsomeip-notify       1/1     Running   0          3s
vsomeip-subscriber   1/1     Running   0          3s
PS C:\Users\ANET1KOR>
```

```
2023-07-21 11:20:13.358971 [info] Application(service-sample, 1277) is initialized (11, 100).
2023-07-21 11:20:13.359076 [info] offer_event: Event [1234.5678.8778] uses configured cycle time 0ms
2023-07-21 11:20:13.359211 [info] REGISTER EVENT(1277): [1234.5678.8778:is_provider=true]
2023-07-21 11:20:13.359363 [info] Starting vsomeip application "service-sample" (1277) using 2 threads I/O nice 255
2023-07-21 11:20:13.360081 [info] main dispatch thread id from application: 1277 (service-sample) is: 7f259eee7b38 TID: 17
2023-07-21 11:20:13.360251 [info] Client [1277] routes unicast:10.1.1.101, netmask:255.255.255.0
2023-07-21 11:20:13.360893 [info] shutdown thread id from application: 1277 (service-sample) is: 7f259eec4b38 TID: 18
2023-07-21 11:20:13.360685 [info] OFFER(1277): [1234.5678:0.0] (true)
2023-07-21 11:20:13.362652 [info] Watchdog is disabled!
Application service-sample is registered.
2023-07-21 11:20:13.363694 [info] io thread id from application: 1277 (service-sample) is: 7f259ee7eb38 TID: 20
2023-07-21 11:20:13.363732 [info] io thread id from application: 1277 (service-sample) is: 7f259f63cb48 TID: 13
2023-07-21 11:20:13.364517 [info] vSomeIP 3.3.8 | (default)
2023-07-21 11:20:13.364800 [info] Network interface "eth0" state changed: up
2023-07-21 11:20:13.365802 [info] Route "default route (0.0.0.0/0) if: eth0 gw: 10.1.0.1" state changed: up
2023-07-21 11:20:13.366682 [info] udp_server_endpoint_impl: SO_RCVBUFFORCE successful.
2023-07-21 11:20:13.366731 [info] udp_server_endpoint_impl: SO_RCVBUF is: 1703936 (1703936) local port:30490
2023-07-21 11:20:13.366818 [debug] Joining to multicast group 224.244.224.245 from 10.1.1.101
2023-07-21 11:20:13.367358 [info] udp_server_endpoint_impl: SO_RCVBUFFORCE successful.
2023-07-21 11:20:13.367899 [info] udp_server_endpoint_impl: SO_RCVBUF is: 1703936 (1703936) local port:30509
2023-07-21 11:20:13.368097 [info] SOME/IP routing ready.
2023-07-21 11:20:13.368260 [warning] Route "224.244.224.245/32 if: eth0 gw: n/a" state changed: up
2023-07-21 11:20:13.367377 [info] udp_server_endpoint_impl<multicast>: SO_RCVBUFFORCE: successful.
2023-07-21 11:20:13.369013 [info] udp_server_endpoint_impl<multicast>: SO_RCVBUF is: 1703936 (1703936) local port:30490
2023-07-21 11:20:13.370488 [info] create_local_server: Listening @ /tmp/vsomeip-1277
Setting event (Length=1).
2023-07-21 11:20:13.885074 [info] REMOTE SUBSCRIBE(0000): [1234.5678.4465] from 10.1.1.102:40000 unreliable was accepted
```

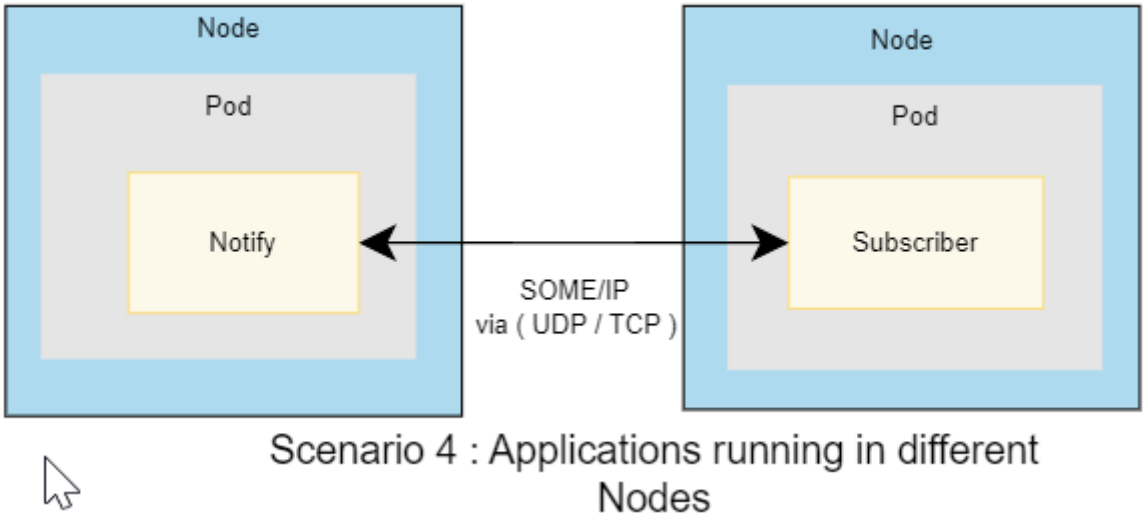
```
2023-07-21 11:20:13.022608 [info] vSomeIP 3.3.8 | (default)
2023-07-21 11:20:13.024902 [info] create_local_server: Listening @ /tmp/vsomeip-1343
2023-07-21 11:20:13.023589 [info] Network interface "eth0" state changed: up
2023-07-21 11:20:13.027033 [info] Route "default route (0.0.0.0/0) if: eth0 gw: 10.1.0.1" state changed: up
2023-07-21 11:20:13.027832 [info] udp_server_endpoint_impl: SO_RCVBUFFORCE successful.
2023-07-21 11:20:13.028198 [info] udp_server_endpoint_impl: SO_RCVBUF is: 1703936 (1703936) local port:30490
2023-07-21 11:20:13.028817 [debug] Joining to multicast group 224.244.224.245 from 10.1.1.102
2023-07-21 11:20:13.029551 [info] SOME/IP routing ready.
2023-07-21 11:20:13.030106 [info] udp_server_endpoint_impl<multicast>: SO_RCVBUFFORCE: successful.
2023-07-21 11:20:13.030575 [info] udp_server_endpoint_impl<multicast>: SO_RCVBUF is: 1703936 (1703936) local port:30490
2023-07-21 11:20:13.030303 [warning] Route "224.244.224.245/32 if: eth0 gw: n/a" state changed: up
2023-07-21 11:20:13.845696 [info] configuration_impl:find_specific_port #1: service: 1234 instance: 5678 reliable: 0 return specific port: 40000
2023-07-21 11:20:13.846735 [info] endpoint_manager_impl::create_remote_client: 10.1.1.101:30509 reliable: 0 using local port: 40000
2023-07-21 11:20:13.847165 [info] udp_client_endpoint_impl::connect: SO_RCVBUFFORCE successful!
2023-07-21 11:20:13.847720 [info] udp_client_endpoint_impl::connect: SO_RCVBUF is: 1703936 (1703936) local port:40000 remote:10.1.1.101:30509
2023-07-21 11:20:13.848352 [warning] sd::get_eventgroup_reliability: couldn't determine eventgroup reliability type for [1234.5678.4465] using reliability type:  0002
Service [1234.5678] is available.
Received a notification for Event [1234.5678.8778] to Client/Session [0000/0002] = (1) 00
Received a notification for Event [1234.5678.8778] to Client/Session [0000/0003] = (2) 00 01
Received a notification for Event [1234.5678.8778] to Client/Session [0000/0004] = (3) 00 01 02
Received a notification for Event [1234.5678.8778] to Client/Session [0000/0005] = (4) 00 01 02 03
Received a notification for Event [1234.5678.8778] to Client/Session [0000/0006] = (5) 00 01 02 03 04
Received a notification for Event [1234.5678.0001] to Client/Session [1343/0001] = (5) 00 01 02 03 04
Received a notification for Event [1234.5678.8778] to Client/Session [0000/0007] = (6) 00 01 02 03 04 05
2023-07-21 11:20:18.811196 [warning] Didn't receive a multicast SD message for 2200ms.
2023-07-21 11:20:18.812388 [debug] Leaving the multicast group 224.244.224.245 from 10.1.1.102
2023-07-21 11:20:18.813153 [debug] Joining to multicast group 224.244.224.245 from 10.1.1.102
```

- In this scenario, We have two pods running on the same node: one pod acts as the "Notifier" and the other as the "Subscriber," both communicating via the SOME/IP protocol Kubernetes Node: Represents a physical or virtual machine in the Kubernetes cluster where the two pods are scheduled to run.
- **Pod 1 (Notify)**: The first pod contains the "Notifier" container responsible for publishing events

or notifications.

- **Notifier Container**: The "Notifier" container runs within Pod 1 and communicates using the SOME/IP protocol. It generates events and formats them into SOME/IP messages for transmission.
- **Pod 2 (Subscribe)**: The second pod contains the "Subscriber" container responsible for subscribing to and processing events.
- **Subscriber Container**: The "Subscriber" container runs within Pod 2 and communicates using the SOME/IP protocol. It listens for incoming SOME/IP messages from the "Notifier" container.
- **Communication via SOME/IP**: The "Notifier" container in Pod 1 sends SOME/IP messages containing events to the "Subscriber" container in Pod 2 over the node's internal network.
- **Node**: Since both pods are scheduled on the same node, they can communicate directly using the node's internal network without involving external network hops. This allows for low-latency communication between the "Notifier" and "Subscriber" containers.
- The communication flow between the "Notifier" and "Subscriber" containers running in different pods on the same Kubernetes node. The SOME/IP protocol facilitates efficient communication between the two containers for event notification and processing.

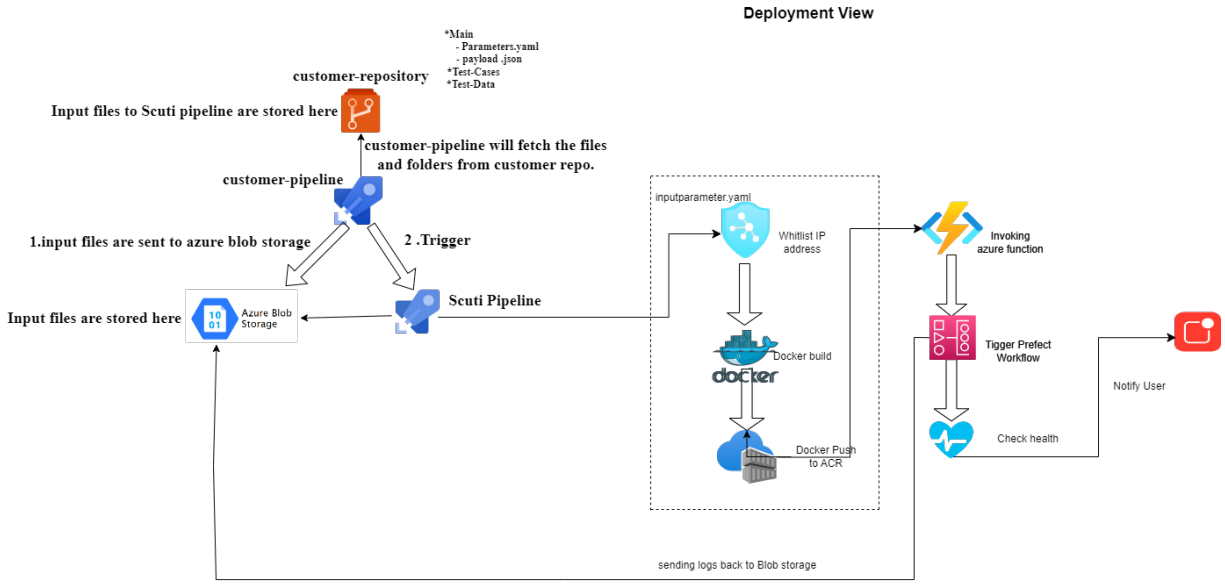
5.4 Applications running in different Nodes:



Scenario 4 : Applications running in different Nodes

- In this scenario, the "Notifier" and "Subscriber" applications are running in different pods, and these pods are scheduled on different nodes within a Kubernetes cluster. They communicate with each other via the SOME/IP protocol.
- **Kubernetes Cluster**: Represents the entire Kubernetes environment, consisting of multiple nodes that can be physical or virtual machines.
- **Node 1 (Pod 1)**: Represents one of the nodes within the Kubernetes cluster. This node hosts "Pod 1."
- **Pod 1**: Contains the "Notifier" application, which generates events and communicates via the SOME/IP protocol.
- **Notifier**: The "Notifier" application runs inside "Pod 1" on "Node 1." It generates events and sends SOME/IP messages.
- **Node 2 (Pod 2)**: Represents another node within the Kubernetes cluster. This node hosts "Pod 2."
- **Pod 2**: Contains the "Subscriber" application, which listens for events and communicates via the SOME/IP protocol.
- **Subscriber**: The "Subscriber" application runs inside "Pod 2" on "Node 2." It receives SOME/IP messages and processes events accordingly.
- **Communication via SOME/IP**: The "Notifier" application sends SOME/IP messages with events to the "Subscriber" application. These messages travel over the network connecting "Node 1" and "Node 2" within the Kubernetes cluster.
- **Different Nodes and Pods**: Since "Notifier" and "Subscriber" are running in different pods on different nodes, they communicate through the Kubernetes network and the underlying network infrastructure that connects the nodes in the cluster. The SOME/IP protocol enables them to exchange data effectively despite being deployed on separate nodes.

6.CI/CD process



- **Customer Pipeline Steps:**
- Read parameters.yaml: The customer pipeline reads the parameters.yaml file from the main folder in the repository. This file contains essential information provided by the customer, such as the image name, repository name, and deployment tag.
- Upload Folders to Azure Blob Storage: Based on the customer's choice specified in parameters.yaml, the customer pipeline uploads the selected folders (main, test-cases, and test-data) to their respective containers (main, test-cases, and test-data) in Azure Blob Storage.
- **Triggering the Scuti-Pipeline:**
- Scuti Pipeline Execution: After being triggered by the customer pipeline, the Scuti-Pipeline starts its execution.
- **Scuti Pipeline Steps:**
- Read parameters.yaml from Azure Blob Storage: The Scuti pipeline accesses Azure Blob Storage and retrieves the parameters.yaml file from the appropriate container (main).
- Extract Image Name, Deployment Tag, and Repository Name: The Scuti pipeline parses the content of parameters.yaml to extract the necessary information, such as the image name, deployment tag, and repository name. This extracted data will be used for the next steps.
- Build the Docker Image: Using the extracted image name, deployment tag, and repository name, the Scuti pipeline initiates the building of the Docker image. It references the payload.json file and any other required dependencies to create a custom Docker image.
- Push the Image to Azure Container Registry (ACR): Once the Docker image is built, the Scuti pipeline pushes the image to the specified Azure Container Registry. This step makes the custom Docker image available for deployment and use in various environments.
- Pipeline Template Repository: The pipeline_template repository contains the configuration and scripts required for the Scuti pipeline. It acts as a blueprint for setting up and running the pipeline effectively.