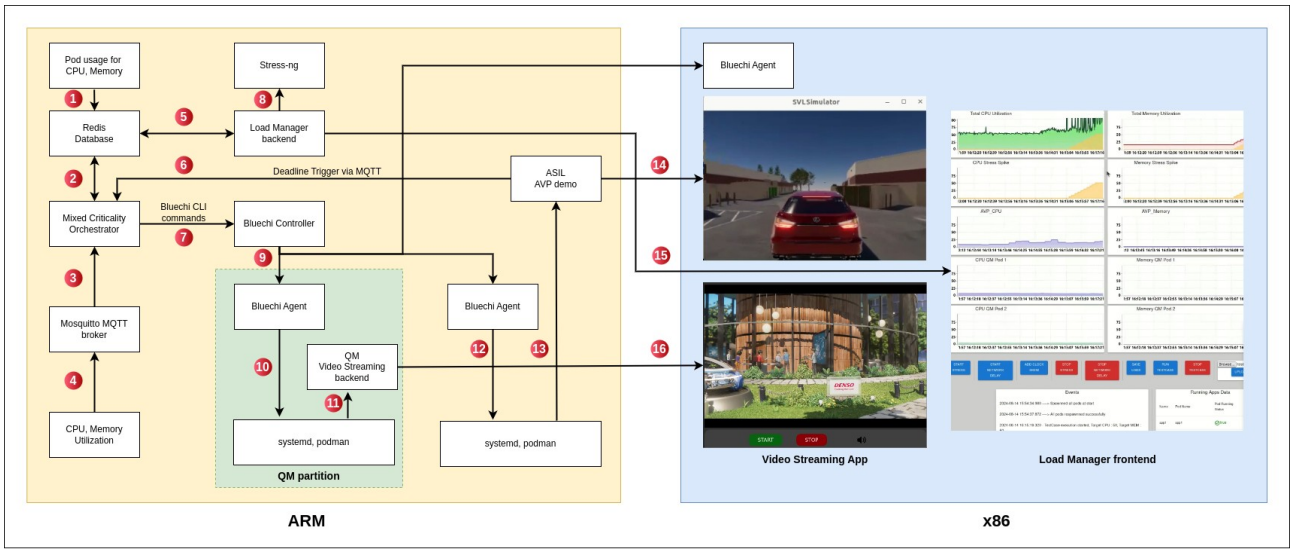
Architecture document



The Wipro software deliverables has 5 parts. They are
1. Video Streaming application (Python)
2. Pod usage application (Python)
3. Load Manager (Python, JavaScript, React)
4. HPC resource utilization module (Rust)
5. Mixed Criticality Orchestrator (C++)

All these modules run in the ARM instance. The x86 instance is responsible for displaying the Load manager dashboard, Video streaming application and AVP simulator.

Following are the responsibilities and operations performed by each of them.

Video Streaming application:
This is a containerized application which streams a video file from backend (ARM machine) to frontend (x86 machine). This application can also play audio along with video. Video streaming application is used as a QM application during our demo.

Pod Usage application:
This application continuously monitors the CPU and Memory utilization of all the pods running in the ARM instance. Once the usage is calculated, it is saved in Redis database which is in ARM.

Load Manager:
This application plots the CPU and Memory utilization of the entire ARM machine and also plots the utilization of AVP pods and QM pods as well. It also plots CPU, memory stress applied on the ARM machine.

Options are available to apply a sudden stress or implement a stress test case on a single core or on entire ARM machine using stress-ng tool. A seperate dashboard shows all the events happening and current status of the running QM pod. All data can be saved into an excel file just by the click of a button.

HPC resource utilization module:
This module is responsible for calculating the overall CPU, Memory utilization of the host machine (ARM) and publish it via Mosquitto MQTT broker to MCO.

Mixed Criticality Orchestrator:
MCO is responsible for doing following operations.
1. Initially spawnning the Video streaming app.
2. Subscribe to the CPU, Memory utilization data of host machine from MQTT broker.
3. Calculating average CPU, Memory and saving into database.
4. Check if host CPU or Memory utilization is below the threshold value.
5 .Continuously check for the “deadline missed trigger” from AVP.
6. If trigger is received from AVP, MCO gives bluechi command to delete Video Streaming pod.

Architecture diagram explanation:

Arrow 1:
Writing pod’s CPU and Memory uasge data into Redis database.

Arrow 2:
Reading/Writing status flags by MCO into Redis database.

Arrow 3, 4:
HPC resource utilization module sending CPU, Memory data to MCO via MQTT broker.

Arrow 5:
Load manager backend retriving necessary data from database when a REST GET request comes from frontend.

Arrow 8:
Load Manager internally used the stress-ng tool to execute the test cases as per users requirement.

Arrow 15:
Once Load Manager is running, its corresponding Browser based UI pops up in the x86 machine.

Arrow 6:
AVP demo giving trigger to MCO via MQTT broker, when deadline miss is detected by Lingua Franca.

Arrow 7, 9, 10, 11:
MCO sending Bluechi command to spawn the QM app. This command triggers the bluechi controller which routes it to the corresponding bluechi agent. The bluechi agent sends this to systemd which uses Podman to spwan the QM Video Streaming container.

Arrow 16:
Once the Video Streaming pod is ready, the QM pod’s frontend starts to display video in the x86 machine.

Arrow 12, 13, 14:
AVP start script is run to start the AVP demo. This AVP demo’s UI pops up in the x86 machine.

Equivalent commands of Kubernetes Implemention in Bluechi

Functions in codebase	Purpose	Kubernetes	Bluechi
		Using .yaml file	***Using .service file***
1.Get Nodes	Lists all the nodes with details like (its name, status etc.) present in the cluster	$ kubectl get nodes	$ bluechictl status
2.Apply Pods	Starts a specified container service in bluechi environment / Deploy a specific application in K3s environment.	$ kubectl apply -f <pod_definition.yml>	$ bluechictl daemon-reload [managed node] $ bluechictl start [managed node] [container.service]

3.Get Pods	Lists all units (with their status) managed by BlueChi / List all the running pods in a K3s cluster	$ kubectl get pods	$ bluechictl list-units
			or
			$ podman ps

4.Pod Usage	Displays the resource usage statistics (CPU, memory, etc.)	$ kubectl top pod	$ podman stats --no-stream (as equivalent in bluechictl command doesn't exist, podman can be used)
5.Delete Pods	Terminates or stop a particular service in bluechi environment / Terminates and remove any resource in K3s	$ kubectl delete -f <pod_definition.yml>	$ bluechictl stop [managed node] [container.service]