5G/6G Cloud Network - IP Report

Advisor: Dr. Rinku Shah
Kartik Prasad (2022240), Swara Parekh (2022524), Varun Kumar(2022563)

EdgeRIC: Empowering Realtime Intelligent Optimization and Control in NextG Cellular Networks

This paper introduces EdgeRIC, a real-time Radio Access Network (RAN) Intelligent Controller (RIC) designed to enhance the performance of NextG cellular networks. It addresses the limitations of existing near-RT and non-RT RICs by enabling sub-millisecond control synchronized with the RAN's Transmission Time Interval (TTI) timescale.
EdgeRIC is composed of two major disaggregated components:

1. **EdgeRIC Execution Module:** A disaggregated microservice co-located with the O-DU (Open Distributed Unit) in the O-RAN architecture, running on edge computing. It operates independently from the time-critical RAN stack to ensure robustness.
   It consisted of
   - Real-time E2 Protocol: Facilitates low-latency communication between μApps and the RAN.
   - Open AI Gym Wrapper: Allows for integration of reinforcement learning environments.
   - In-memory Redis Database: Enables rapid data exchange between μApps and the RAN stack.

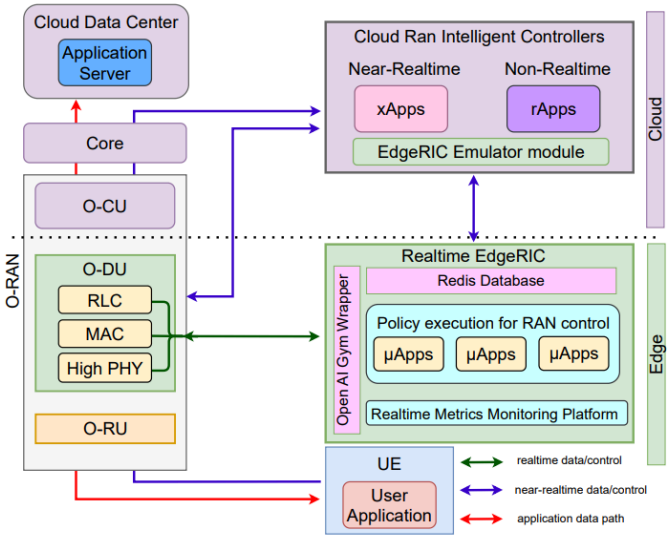
   It executes μApps for real-time tasks like resource scheduling, using cross-layer data (e.g., channel quality, buffer status, application states). These μApps are driven by cross-layer data inputs and it ensures TTI-level synchronization without violating RAN timing constraints.

2. **EdgeRIC Emulator Module:** It supports training and validation of intelligent control strategies, EdgeRIC includes an emulator built on the srsRAN codebase, incorporating:
   - Virtual radios and simulated channels
   - ZeroMQ-based communication stack
   - Realistic UE behavior emulation

   This supports the training of RL (Reinforcement Learning) algorithms, including Proximal Policy Optimization (PPO), enabling effective “sim-to-real” transfer. Trained policies are exported to the execution module as μApps for deployment in live systems.

3. **Integration with O-RAN:** EdgeRIC conforms to the open and disaggregated O-RAN architecture, interfacing directly with the PHY and MAC layers of the O-DU.
   It maintains vendor-agnostic operability and ultra-low latency (as low as 100 μs). This

makes it well-suited for real-time edge deployments—in contrast to cloud-hosted near-RT RICs (10–30 ms) or non-RT RICs (>1 s latency).



Results of the paper:

- EdgeRIC's real-time control (100 µs latency) outperforms near-RT RIC (15–30 ms latency) by achieving up to 50% higher system throughput in emulation and 14–33% in real-world tests with 4 Turntable UEs.
- RL-based scheduling µApps which were trained using PPO, outperform traditional algorithms (Max CQI, Proportional Fair, Max Weight) by 5–25% in throughput across diverse scenarios.
- In a video streaming case study with 4 UEs (2 streaming, 2 background traffic), EdgeRIC's RL-based µApp reduced video stalls by ~66% compared to standard algorithms, leveraging application state (media buffer length).
- EdgeRIC maintains sub-millisecond feedback loops for up to 100 simulated UEs, with inference times under 400 µs.

On reading this paper thoroughly, we understood the disaggregated architecture, comprising the execution and emulator modules of EdgeRIC. It supports robust, vendor-agnostic integration with O-RAN.

NS-3 ORAN:

Introduction:

This section outlines the changes made to the oran-lte-2-lte-rsrp-handover-lm-example.cc code from the ns3-oran repository to fix issues encountered during simulation. The original code simulates an LTE network with

2 eNBs and 1 UE, where the ORAN Near-RT RIC manages handovers based on RSRP measurements. The initial modified version had a segmentation fault, failed to update base stations, kept the UE stuck on one base station, and did not log RSRP values in the lteuersrprsrq table. This report details the fixes applied, along with an overview of the ns3-oran module, its repository structure, and ORAN's role in ns-3.

What is ns3-oran?

The ns3-oran module extends ns-3 to model O-RAN architectures, focusing on the Near-RT RIC and E2 Terminators for communication between the RIC and network nodes. It provides a framework for researchers to test O-RAN solutions like intelligent handovers without modifying core ns-3 models, using a SQLite database for logging.

What Does ORAN in ns-3 Do?

ORAN in ns-3 enables simulation of intelligent RAN management. In the oran-lte-2-lte-rsrp-handover-lm-example.cc, the Near-RT RIC collects RSRP data from a UE moving between eNBs and uses a Logic Module (OranLmLte2LteRsrpHandover) to trigger handovers, supporting features like conflict mitigation and periodic reporting.

ns3-oran Repository Structure

The ns3-oran repository follows the standard ns-3 module structure:

- src/oran/: Core source code.
    - model/: Classes like oran-near-rt-ric.cc, oran-reporter-lte-ue-rsrp-rsrq.cc, and oran-lm-lte-2-lte-rsrp-handover.cc.
    - helper/: Helper classes like oran-helper.cc.
- examples/: Example simulations.
    - oran-lte-2-lte-rsrp-handover-lm-example.cc: Simulates LTE handover using ORAN.
    - Other examples: May include scenarios like load balancing or ML-based handovers.
    - Subdirectories: Likely includes data/ for outputs (e.g., oran-repository.db) and scripts/ for analysis.
- test/: Unit tests for ORAN components.
- wscript: Build script for integration.

Modifications and Fixes to the Code

Increased Number of eNBs and Adjusted Simulation Parameters

1. The distance was increased to 80m to space out the eNBs over a wider area (total 320m between the first and last eNB), and the speed was doubled to 4 m/s to ensure the UE would cover this distance within the simulation time. The simulation time was extended

to 60 seconds to allow sufficient time for the UE to move across all eNBs and perform multiple handovers.

Original Configuration: Defined 2 eNBs, 50m distance, 2 m/s UE speed, 30-second simulation time, and 15-second reversal interval.

```
uint16_t numberOfEnbs = 2;
double distance = 50;
double speed = 2;
Time simTime = Seconds(30);
Time interval = Seconds(15);
```

Modified Configuration: Changed to 5 eNBs, 80m distance, 4 m/s speed, 60-second simulation time, and 80-second reversal interval.

```
uint16_t numEnbs = 5;
double distance = 80.0;
double speed = 4.0;
Time simTime = Seconds(60);
Time hoInterval = Seconds((distance * (numEnbs - 1)) / speed);
```

2. To simulate handovers across more base stations. This caused a segmentation fault, fixed by adjusting ReverseVelocity to use iterators.

Original loop in reverse velocity

```
for (uint32_t idx = 0; idx < nodes.GetN(); idx++)
{
    Ptr<ConstantVelocityMobilityModel> mobility = nodes.Get(idx)-
>GetObject<ConstantVelocityMobilityModel>();
    mobility->SetVelocity(Vector(mobility->GetVelocity().x * -1, 0, 0));
}
```

Modified loop with iterators

```
for (NodeContainer::Iterator it = nodes.Begin(); it != nodes.End(); ++it)
{
    Ptr<ConstantVelocityMobilityModel> cv = (*it)-
>GetObject<ConstantVelocityMobilityModel>();
    Vector v = cv->GetVelocity();
    cv->SetVelocity(Vector(-v.x, 0, 0));
}
```

Changed UE Initial Position and Attachment Method

3. To allow the UE to traverse the network and dynamically connect to the nearest eNB. Initially, the UE was stuck on one base station.

Original Configuration: UE started at 10m and attached to the first eNB.

```
positionAlloc->Add(Vector((distance / 2) - (speed * (interval.GetSeconds() / 2)),
0, 1.5));
lteHelper->Attach(ueLteDevs.Get(i), enbLteDevs.Get(0));
```

Modified Configuration: UE starts at -10m and uses AttachToClosestEnb

```
uePos->Add(Vector(-10.0, 0, 1.5));
lte->AttachToClosestEnb(ueDevs.Get(0), enbDevs);
```

Fixed UE Mobility and Base Station Updates

The UE remained stuck on one base station despite 5 eNBs being registered and visible in logs. Fixed by ensuring the OranLmLte2LteRsrpHandover module received RSRP data for handover decisions (see below).
The UE now hands over between base stations, verified via handover.tr in the examples folder.

Added Missing Trace Connection for RSRP/RSRQ Logging

RSRP values were not logged in the lteuersrprsrq table due to a missing trace connection. Added the trace connection in the UE terminator setup.

Original trace connection:

```
uePhy->TraceConnectWithoutContext(
    "ReportUeMeasurements",
    MakeCallback(&ns3::OranReporterLteUeRsrpRsrq::ReportRsrpRsrq,
rsrpRsrqReporter));
```

Modified Trace connection:

```
for (uint32_t netDevIdx = 0; netDevIdx < (*it)->GetNDevices(); netDevIdx++)
{
    Ptr<LteUeNetDevice> lteUeDevice = (*it)->GetDevice(netDevIdx)-
>GetObject<LteUeNetDevice>();
    if (lteUeDevice)
    {
        Ptr<LteUePhy> uePhy = lteUeDevice->GetPhy();
        uePhy->TraceConnectWithoutContext("ReportUeMeasurements",
MakeCallback(&ns3::OranReporterLteUeRsrpRsrq::ReportRsrpRsrq, rsrp));
    }
}
```

RSRP/RSRQ measurements are logged in oran-repository.db in the examples folder, enabling RIC handover decisions.

Restored Tracing for Signal Quality and Positions

Original Configuration: Included RSRP/RSRQ/SINR and position tracing.

```
Ptr<OutputStreamWrapper> rsrpRsrqSinrTraceStream =
    Create<OutputStreamWrapper>("rsrp-rsrq-sinr.tr", std::ios::out);
uePhy->TraceConnectWithoutContext(
    "ReportCurrentCellRsrpSinr",
    MakeBoundCallback(&TraceRsrpRsrqSinr, rsrpRsrqSinrTraceStream));
Ptr<OutputStreamWrapper> positionTraceStream =
    Create<OutputStreamWrapper>("positions.tr", std::ios::out);
Simulator::Schedule(Seconds(1), &PositionTrace, positionTraceStream, ueNodes);
```

Modified configuration: to help our analysis we also saved a trace file

```
Ptr<OutputStreamWrapper> rsrpRsrqSinrTraceStream =
    Create<OutputStreamWrapper>("rsrp-rsrq-sinr.tr", std::ios::out);
for (NetDeviceContainer::Iterator it = ueDevs.Begin(); it != ueDevs.End(); ++it)
{
    Ptr<NetDevice> device = *it;
    Ptr<LteUeNetDevice> lteUeDevice = device->GetObject<LteUeNetDevice>();
    if (lteUeDevice)
    {
        Ptr<LteUePhy> uePhy = lteUeDevice->GetPhy();
        uePhy->TraceConnectWithoutContext(
            "ReportCurrentCellRsrpSinr",
            MakeBoundCallback(&TraceRsrpRsrqSinr, rsrpRsrqSinrTraceStream));
    }
}
Ptr<OutputStreamWrapper> positionTraceStream =
    Create<OutputStreamWrapper>("positions.tr", std::ios::out);
Simulator::Schedule(Seconds(1), &PositionTrace, positionTraceStream, ues);
```

Purpose: To verify signal quality and UE movement, with outputs in the examples folder.

Removed these to just focus on the ORAN functionality and for ease of debugging if some issues arrive, and decrease output size, to share easily among teammates.

```
oranHelper->SetAttribute("E2NodeInactivityThreshold", TimeValue(Seconds(2)));
oranHelper->SetAttribute("E2NodeInactivityIntervalRv",
StringValue("ns3::ConstantRandomVariable[Constant=2]"));
lteHelper->EnablePhyTraces();
lteHelper->EnableMacTraces();
lteHelper->EnableRlcTraces();
lteHelper->EnablePdcpTraces();
```

The initial modifications scaled the simulation to 5 eNBs but introduced a segmentation fault (fixed by adjusting ReverseVelocity), caused the UE to remain stuck on one base station (resolved by ensuring ORAN logic and RSRP data), and failed to log RSRP values (corrected by adding the trace connection). Restored tracing for signal quality and positions, simplified ORAN settings, and enabled SQL query logging by default. These changes ensured the simulation worked, with the UE handing over between base stations and RSRP values logged.

Oran-repository.db - Database file

```
ubuntuip@ubuntuip-VirtualBox:~/ns-allinone-3.41/ns-3.41$ sqlite3 oran-repository
.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> .tables
cmmaction          lteenb             lteuersrprsrq      nodelocation
lmaction           lteue              node               noderegistration
lmcommand          lteuecell          nodeapploss        terminatorcommand
```

```
sqlite> .schema lteuersrprsrq
CREATE TABLE lteuersrprsrq (entryid       INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, nodeid         INTEGER
               NOT NULL, simulationtime INTEGER                           NOT NULL, rnti           INTEGER
             NOT NULL, cellid         INTEGER                          NOT NULL, rsrp           REAL
           NOT NULL, rsrq           REAL                             NOT NULL, serving        BOOLEAN
         NOT NULL, ccid           BOOLEAN                          NOT NULL, FOREIGN KEY(cellid) REFERENCES lteenb(cel
lid), FOREIGN KEY(nodeid) REFERENCES lteue(nodeid));
```

lteuersrprsrq: Keeps track of signal strength and quality (RSRP/RSRQ) for LTE devices.
nodes: Stores info about network devices (like IDs and types, e.g., UE or eNB).
commands: Logs the RIC's instructions, like handover commands, with timestamps.
reports: Gathers updates sent to the RIC, such as device measurements, for making decisions.
logs: Records what's happening in the simulation and RIC for troubleshooting.

Decision Tree Classifier Model:

We developed a **Decision Tree Classifier** that predicts the most suitable gNB (cell) for each UE to connect to, based on network conditions and mobility features.

We opted for a Decision Tree model because:

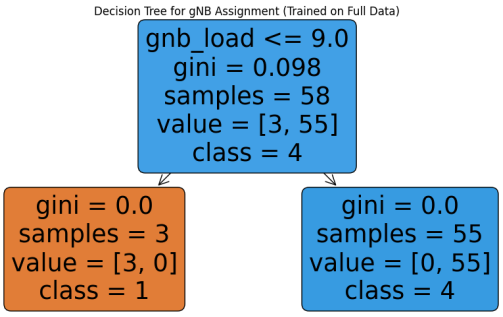
- It is fast to train and lightweight for inference, which is better for real-time or near-real-time applications.
- Decision trees handle both numerical and categorical features effectively without requiring feature scaling.
- Unlike neural network models, it's easier to integrate it within the ns-3 or xApp testing pipelines.

We used the **lteuecell table** from the **oran-repository.db** database, which logs:

rnti (UE ID),
cellid (connected gNB),
simulationtime (timestamp),
nodeid (node reference).

From this raw data, we formed the following variables:

gnb_load: The number of UEs connected to each gNB (cell load).
ue_mod: A proxy for UE mobility, estimated using the RNTI % 10.
time_bin: The simulation time to capture load evolution over time.



The model classifies into class=1 and class=4 only because, they are the least the loaded gNBs

For supervised learning, we generated labels by selecting the **least loaded gNB (excluding the current one)** as the best candidate for handover. Once the decision tree model was trained, we saved the model in a pkl file. This can be used during the inference time after incorporating it into the x-apps.

SD-RAN-in-a-Box and xApp Integration

1. Objectives

- Deploy SD-RAN-in-a-Box
- Configure ONOS-based near-Real-Time RAN Intelligent Controller
- Install and test xApps: Physical Cell Identity (PCI), Mobility Load Balancing (MLB), Mobility Handover Optimization (MHO), Fraction-Based Automated Handover (FBAH), RIM Detection Optimization (RIMDEO)
- Simulate Radio Access Network using RANSim
- Analyze xApp decisions via logs and monitoring metrics

2. SD-RAN-in-a-Box Components

- **ONOS-based near-RT RIC**: Hosts and manages xApps
- **RAN Simulator (RANSim)**: Simulates evolved NodeBs (eNBs), next-generation NodeBs (gNBs), and User Equipments (UEs)
- **E2 Termination Service**: Manages E2 Application Protocol (E2AP) messages
- **xApps**: Modular applications implementing control logic
- **Helm Charts**: For deployment of ONOS-RIC, RANSim, and xApps
- **Prometheus + Grafana**: Used for metrics collection and visualization

3. xApps Installed and Tested

3.1 Physical Cell Identity (PCI) xApp

- **Function**: Automatic assignment of PCI values
- **Objective**: Eliminate PCI collision and confusion in network cells
- **Input**: Cell topology and identifiers from RANSim
- **Output**: PCI reconfiguration commands via E2 interface
- **Validation**: Verified PCI updates through ONOS logs

3.2 Mobility Load Balancing (MLB) xApp

- **Function**: Distributes traffic across cells to balance network load
- **Objective**: Avoid overload on individual base stations
- **Input**: Load metrics from simulated cells
- **Output**: Adjustment of handover thresholds
- **Validation**: Verified balanced traffic distribution behavior

3.3 Mobility Handover Optimization (MHO) xApp

- **Function**: Enhances handover decisions to minimize failures and unnecessary handovers
- **Objective**: Improve handover performance and quality of service
- **Input**: Signal measurements and User Equipment mobility data
- **Output**: Optimized handover commands
- **Validation**: Confirmed fewer failed or ping-pong handovers

3.4 Fraction-Based Automated Handover (FBAH) xApp

- **Function**: Makes handover decisions using fractional load and signal ratio thresholds
- **Objective**: Enable intelligent and dynamic handover control
- **Input**: Signal quality, load statistics, and user positioning
- **Output**: Precise handover execution commands
- **Validation**: Verified appropriate handovers using FBAH logic

3.5 RIM Detection Optimization – Time Series (RIMDEO_TS) xApp

- **Function**: Detects inter-cell interference using time-series analysis of measurement data
- **Objective**: Identify and mitigate Radio Interference Management (RIM) issues
- **Input**: Measurement Reports (MRs) from User Equipments
- **Output**: Reconfiguration of interfering cell parameters
- **Validation**: Verified interference detection and mitigation logic via logs and metrics

4. Installation Steps for Each xApp

1. Clone SD-RAN-in-a-Box repository
2. Modify `values.yaml` with specific xApp parameters
3. Deploy xApp via Helm:
   ```
   helm install <xapp-name> ./charts/<xapp>
   ```
4. Launch RANSim with appropriate simulation config
5. Monitor execution using ONOS logs and Grafana dashboards

5. Tools Used

- **Docker**: Containerization of all SD-RAN components
- **Kubernetes**: Service orchestration
- **Helm**: Deployment management
- **Prometheus**: Collection of time-series metrics
- **Grafana**: Visualization of network statistics
- **ONOS CLI + Logs**: xApp behavior monitoring
- **RANSim**: Realistic emulation of User Equipments and base stations

6. Configuration Files Used

- `values.yaml`: xApp configuration parameters
- `onos-config.json`: Contains E2 service model definitions
- `rnasim-config.yaml`: Defines User Equipment behavior and cell layout
- `charts/`: Contains Helm deployment files for each component

7. Issues and Debugging

- E2 connection failure due to mismatched service model versions (resolved via alignment)
- Conflicting commands from multiple xApps (resolved using selective xApp deployment)
- Inconsistent metric timestamps (resolved by configuring Prometheus scrape intervals)

ONOS API and Custom xApp Development

The `onos-api` repository provides the core interface definitions (Protocol Buffers `.proto` files) used by xApps to communicate with μONOS components such as the near-Real-Time RAN Intelligent Controller (near-RT RIC), E2 Termination service, topology service, and configuration management.

Repository: https://github.com/onosproject/onos-api

1. Purpose

- Provides standardized gRPC APIs for xApp interaction with ONOS-based RIC.
- Enables subscription to RAN metrics, access to RAN topology, and configuration of network elements.
- Supports auto-generation of API client libraries for xApp development in languages like Go and Python.

2. Key API Modules

E2 Subscription API: Enables xApps to subscribe to RAN metrics such as Reference Signal Received Power (RSRP), Radio Resource Control (RRC) events, and handover triggers.

Topology API: Allows xApps to discover and monitor network topology including evolved NodeBs (eNBs), gNodeBs (gNBs), cells, and User Equipments.

Configuration API: Provides the ability to apply control commands to the RAN, such as Physical Cell Identity (PCI) reassignment and handover threshold updates.

RAN Simulation API: Supports integration with RANSim, enabling xApps to emulate and monitor network behavior during testing.

Atomix API: Supplies distributed state storage mechanisms required for coordination across ONOS microservices.

Kubernetes Metadata API: Contains configuration structures necessary for container-based deployment and orchestration.

3. xApp Development Workflow using ONOS API

First, the xApp uses the Topology API to retrieve the layout and attributes of all base stations and cells in the network.

Next, the xApp uses the E2 Subscription API to subscribe to relevant telemetry data streams such as signal strength or handover events from User Equipments.

Based on the received metrics and control logic, the xApp applies configuration changes using the Configuration API to modify RAN parameters dynamically.

In simulation environments, the RAN Simulation API is used to interact with and control User Equipment and base station behavior to validate the effectiveness of the xApp.

4. Code Generation and Usage

All Protocol Buffer files are compiled using `buf` and `protoc`. The repository provides a build script `build/bin/compile-protos.sh` to generate client bindings for multiple programming languages. xApps utilize these bindings to invoke gRPC services and interact with ONOS components.

5. Integration with SD-RAN-in-a-Box

The PCI, MLB, MHO, FBAH, and RIMDEO_TS xApps rely on the ONOS APIs for executing their control logic. These xApps use the Topology API to understand the network layout, the E2 Subscription API to receive RAN metrics, and the Configuration API to push control decisions such as PCI changes or handover threshold updates. This modular API architecture supports scalable and maintainable xApp development within the SD-RAN-in-a-Box framework.

Partial completion of CUDA course offered by Oak Ridge National Laboratory

Introduction
I decided to take the CUDA course offered by Oak Ridge National Laboratory and here is the link https://www.olcf.ornl.gov/cuda-training-series/ because I wanted to get better at GPU programming, which is so important for high-performance computing tasks. It was a great introduction to using NVIDIA GPUs, and I made it through up to the "Advanced Features and Memory Management" part.

Foundations of CUDA Programming
This part taught me the essentials of CUDA, like how to write GPU kernels and handle data transfers between the CPU and GPU. I also learned about shared memory, which really helps speed up data access and is key for writing efficient GPU programs.

Optimization Techniques for High Performance
I got to explore ways to make my code run faster, like using memory coalescing and reducing thread divergence. The course introduced me to NVIDIA Nsight, which was super helpful for finding and fixing performance issues in my programs.

Advanced Features and Memory Management
In this section, I learned about advanced stuff like managed memory, which makes handling data between the CPU and GPU much easier. I also picked up concurrency techniques that let me overlap computation and data transfers, boosting overall efficiency.

Personal Reflections
One of my favorite moments was when I optimized a kernel and saw a real performance improvement—it felt so rewarding! The hands-on exercises really helped me wrap my head around some of the trickier GPU programming concepts.

Conclusion
Even though I only got up to "Advanced Features and Memory Management," I feel like I've built

a strong foundation in GPU programming. I'm already finding these skills useful and can't wait to put them to work on computational projects, with plans to finish the rest of the course later.

References:
https://www.youtube.com/watch?v=3c-iBn73dDE
http://acl.digimat.in/nptel/courses/video/128108025/L05.html

Docker and Kubernetes:

Overview
Docker is a tool that makes it easy to package and run apps in containers. We learnt it by thinking of containers as lightweight, portable boxes that hold your app and everything it needs to work, like libraries and settings. They're less bulky than virtual machines since they share the host's operating system, and they ensure that the app runs the same everywhere.

Basics of Using Docker:

Docker Engine: part which runs everything.
Docker Images: blueprints for your containers,
Docker Containers: running instances of those images.
Dockerfile: text file with instructions to build an image.

How To Use It:

- Write a Dockerfile to set up your app's environment.
- Build an image with docker build -t my-app .
- Run a container with docker run my-app.
- Manage things with commands like docker ps to see what's running, docker stop to pause a container, or docker rm to delete one.

Basic Commands:

- docker pull nginx: Grabs an image (like the Nginx web server) from Docker Hub.
- docker images: what images you've got.
- docker push my-app: Shares your image with others.
- docker logs my-container: Checks what's going on inside a container.

Advantages:

- App works the same everywhere.
- Avoids conflicts.
- We can use Kubernetes for managing tons of containers.