

I am a fresher trained at a surface level in Linux, networking, virtualization, Docker, Kubernetes, SDN, DevOps, and Python. I want to build a telecom-focused capstone project aligned with Quadgen Wireless expectations. The project should demonstrate system integration, automation, and cloud-native telecom concepts without excessive complexity. Help me finalize one project, define architecture, tech stack, milestones, risks, and how to present it effectively.

Linux Admin & Shell scripting
Python programming & API introduction
IP Networking
Virtualization
Cloud computation & NFV Foundation
Cloud Native - Docker & Kubernetes
SDN & Network Automation
the above are the modules on which I was trained on at surface level as a fresher

<https://www.cncf.io/>

<https://www.linuxfoundation.org/projects>

<https://open5gs.org/>

<https://free5gc.org/>

<https://github.com/calee0219/awesome-5g>

<https://github.com/delimitrou/DeathStarBench>

https://medium.com/@googler_ram/100-opensource-5g-projects-that-you-can-get-your-hands-dirty-with-127a50967692

<https://yuba.stanford.edu/~casado/of-sw.html>

<https://www.geeksforgeeks.org/blogs/docker-projects-with-source-code/>

<https://github.com/topics/networking>

also try to think of some one source projects available from the above links and how & why it can be incorporated into my projects for some specific-use case. Also mention the tech stack on which the project has to be built with a brief overall idea of the project build flow and layout

Great question—and you're thinking in exactly the right direction for a telecom-aligned, cloud-native capstone that fits a Quadgen Wireless fresher profile.

Below I'll finalize ONE strong project, explain why it fits telecom expectations, show architecture, tech stack, open-source usage from your links, milestones, risks, and how to present it convincingly without over-engineering.

✓ Finalized Capstone Project

Title

Cloud-Native 5G Core (Open5GS) with Automated Deployment, SDN-Based Traffic Steering, and Observability

🎯 Why This Project Is Ideal for You (as a Fresher)

This project:

- ✓ Uses real telecom software (Open5GS – industry-recognized)
- ✓ Demonstrates system integration (Linux, Docker, Kubernetes, networking)
- ✓ Shows automation mindset (shell + Python)
- ✓ Introduces cloud-native & NFV concepts
- ✓ Avoids deep PHY/RAN complexity (which is not expected for freshers)
- ✓ Aligns with Quadgen Wireless expectations: 5G Core, DevOps, SDN, Kubernetes

This is NOT a toy project. It's a scaled-down industry-style lab.

💡 Core Idea (Simple Explanation)

Build a containerized 5G Core network using Open5GS, deploy it on Kubernetes, automate deployment & monitoring, and use SDN concepts to control traffic flows between network functions.

Think of it as:

"A mini cloud-native telecom core, deployed like a real operator would do it."

✳️ Open-Source Projects You'll Use (From Your Links)

Project Why It's Used Source

Open5GS Real 5G Core (AMF, SMF, UPF, NRF)

open5gs.org

Docker Containerize each network function CNCF

Kubernetes Orchestrate telecom network functions CNCF

Linux Base OS + networking Linux Foundation

Python Automation, health checks, APIs GitHub

SDN (Open vSwitch) Traffic steering, flow control

Stanford SDN link

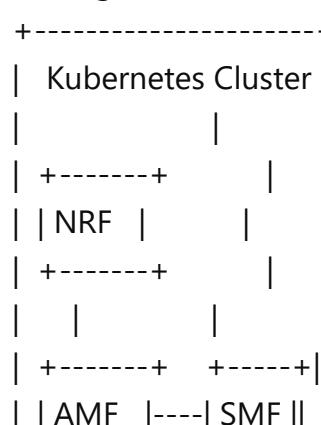
Prometheus + Grafana Observability (optional but powerful) CNCF

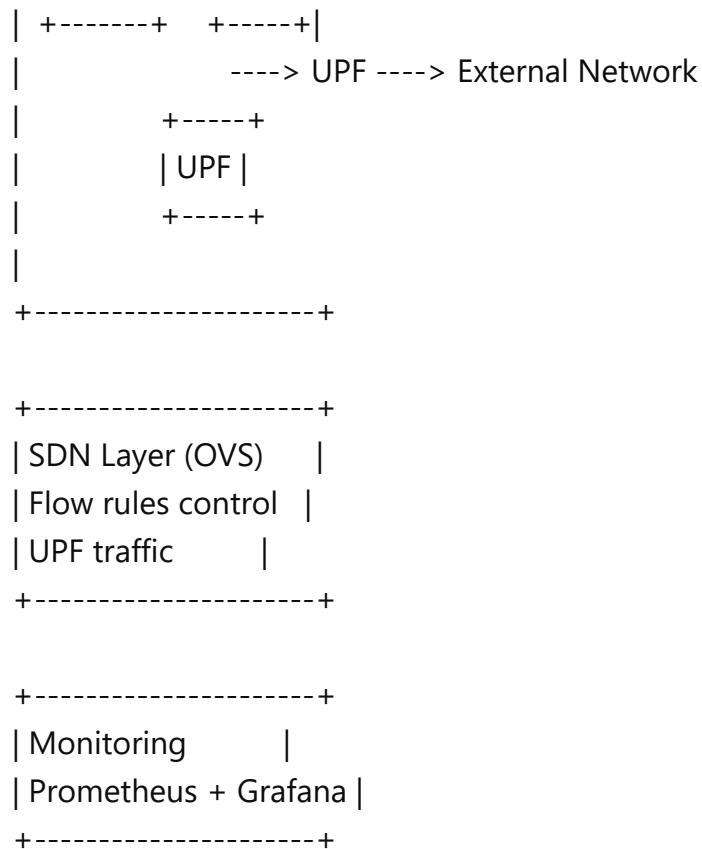
Helm (optional) Simple NF deployment CNCF

🚫 Not using Free5GC initially (more complex).

Open5GS is perfect for freshers.

🏗️ High-Level Architecture





🔧 Technology Stack (Mapped to Your Training)
OS & Core

Linux (Ubuntu 20.04)

Bash shell scripting

Networking

IP routing

Linux bridges

Open vSwitch (SDN concepts)

Virtualization / Cloud

VirtualBox / VMware (for cluster nodes)

Kubernetes (kubeadm or kind)

Cloud-Native

Docker

Kubernetes

Helm (optional)

Telecom

Open5GS (5G Core)

Simulated UE (UERANSIM – optional)

Automation

Python (API calls, health checks)

Shell scripts (deployment)

Observability

Prometheus

Grafana

 Project Build Flow (Step-by-Step)

Phase 1: Environment Setup

- ✓ Linux VM
- ✓ Install Docker
- ✓ Install Kubernetes (single or multi-node)
- ✓ Setup Open vSwitch

Phase 2: Containerized 5G Core

- ✓ Pull Open5GS Docker images
- ✓ Configure:

AMF

SMF

UPF

NRF

✓ Test basic service registration

Phase 3: Kubernetes Deployment

✓ Create:

Deployments

Services

ConfigMaps

- ✓ Validate inter-pod communication
- ✓ Simulate UE attach (optional)

Phase 4: SDN Integration

- ✓ Use Open vSwitch
- ✓ Control UPF traffic flow
- ✓ Demonstrate:

Traffic steering

Policy-based routing

Phase 5: Automation

✓ Python script:

Check pod health

Restart failed NFs

Fetch metrics via API

✓ Shell script:

One-click deployment

Phase 6: Observability

✓ Prometheus metrics

✓ Grafana dashboard:

NF health

Traffic stats

- 📌 Milestones (Resume-Friendly)
 - Week Milestone
 - 1 Linux + Docker + Kubernetes setup
 - 2 Open5GS core running in Docker
 - 3 Kubernetes deployment
 - 4 SDN traffic steering
 - 5 Automation scripts
 - 6 Monitoring + documentation
- ⚠ Risks & How You'll Handle Them
 - Risk Mitigation
 - Open5GS config complexity Start with Docker Compose, then move to K8s
 - Kubernetes networking confusion Use simple CNI (Flannel/Calico)
 - SDN complexity Only demonstrate basic flow rules
 - Time overrun Monitoring is optional but recommended
- 🎤 How to Present This Project (Very Important)
- 1 Elevator Pitch (30 seconds)

"I built a cloud-native 5G core using Open5GS, deployed on Kubernetes, with automated deployment and SDN-based traffic steering. The project demonstrates telecom NFV, cloud-native networking, and DevOps principles."

2 What Interviewers Care About

Talk about:

Why Kubernetes for telecom

Why containerized NFs

How SDN helps traffic control

How automation improves reliability

3 Show These During Demo

- ✓ Kubernetes pods running
- ✓ Open5GS logs (AMF/SMF/UPF)

- ✓ Python automation script
- ✓ Grafana dashboard

📄 Resume Line (You Can Copy This)

Designed and deployed a cloud-native 5G Core using Open5GS on Kubernetes, implementing SDN-based traffic steering, automated deployment scripts, and observability using Prometheus and Grafana.

🚀 Optional Enhancements (If Time Allows)

Helm charts

Multi-slice simulation

CI/CD pipeline

Network policy enforcement

✓ Final Advice

This project is:

Real

Telecom-aligned

Fresher-friendly

Interview-ready

If you want, next I can:

Break this into exact commands

Help you write a GitHub README

Create a project diagram

Tailor it specifically to Quadgen interview questions

Just tell me 👍

Suggest project ideas first and then I will be shortlisting before finalising one

Perfect 👍

That's the right order: explore → shortlist → finalize.

Below is a curated list of telecom-focused capstone project ideas, ranked from most suitable for a fresher to slightly advanced, all aligned with Quadgen Wireless expectations and your training modules.

I'll clearly mention:

Core idea

What telecom concept it proves

Which open-source projects fit (from your links)

Why it's good for a fresher

● Tier-1: Strong & Safe Projects (Highly Recommended for Freshers)

These are interview-friendly, realistic, and achievable.

1 Cloud-Native 5G Core Deployment with Open5GS
(Most Recommended)
Idea

Deploy a containerized 5G Core (AMF, SMF, UPF, NRF) using Docker & Kubernetes, with basic automation and monitoring.

Telecom Concepts

5G Core architecture

NFV & Cloud-Native NFs

Service-based architecture (SBA)

Open Source Used

Open5GS → 5G Core

Docker & Kubernetes → CNCF

Prometheus/Grafana → CNCF

Why It's Perfect

- ✓ Real telecom software
- ✓ No PHY/RAN complexity
- ✓ Matches operator-side roles
- ✓ Easy to explain in interviews

📌 Best single project if you want maximum impact

2 Automated NFV Lab: Deploying Virtual Network Functions Using Kubernetes Idea

Build a mini NFV platform where network functions (firewall, load balancer, router) are deployed as containerized VNFs with automation scripts.

Telecom Concepts

NFV MANO basics

Virtual network functions

Cloud-based service chaining

Open Source Used

Linux Foundation NFV concepts

Docker

Kubernetes

Python automation

Why It's Good

- ✓ Shows NFV fundamentals
- ✓ Less telecom protocol complexity
- ✓ Strong DevOps + networking mix

3 SDN-Based Traffic Steering for Telecom Network

Functions

Idea

Use SDN (Open vSwitch) to dynamically control traffic flow between network services, simulating telecom data-plane control.

Telecom Concepts

SDN

Data plane vs control plane

Traffic engineering

Open Source Used

Open vSwitch (from Stanford SDN link)

Linux networking tools

Python (REST APIs)

Why It's Good

- ✓ Clear SDN demonstration
- ✓ Small but powerful project
- ✓ Very interview-friendly

4 Kubernetes-Based Telecom Service Health Monitoring

System

Idea

Monitor telecom services deployed on Kubernetes using Prometheus + Grafana, including auto-restart and alerting.

Telecom Concepts

Carrier-grade reliability

Observability in telecom networks

SLA monitoring

Open Source Used

Prometheus

Grafana

Kubernetes

Why It's Good

- ✓ Simple
- ✓ Production-oriented
- ✓ Easy demo

● Tier-2: Medium Complexity (Good If You're Confident)
5 End-to-End Private 5G Lab Using Open5GS +
UERANSIM
Idea

Simulate a private 5G network with a UE simulator connecting to a cloud-native 5G core.

Telecom Concepts

UE registration

Session management

User plane traffic

Open Source Used

Open5GS

UERANSIM

Docker/Kubernetes

Why It's Good

- ✓ End-to-end story
- ⚠ Slightly more configuration effort

6 Cloud-Native Network Slice Simulation Using Kubernetes Namespaces Idea

Simulate 5G network slicing using Kubernetes namespaces and policies.

Telecom Concepts

Network slicing

Resource isolation

QoS concepts

Open Source Used

Kubernetes

Linux networking

Python automation

Why It's Good

- ✓ Modern telecom concept
- ⚠ Abstract slicing (not real 3GPP slicing)

7 Free5GC Cloud-Native Core with CI/CD Pipeline Idea

Deploy Free5GC with CI/CD automation.

⚠ More complex than Open5GS.

8 Telecom Microservices Benchmarking Using DeathStarBench Idea

Use DeathStarBench to simulate telecom-like microservice load and analyze latency.

Why It's Advanced

Less telecom protocol focus

More systems performance focus

🎯 How to Shortlist (Simple Rule)

Choose a project that:

- ✓ Uses real telecom components
- ✓ Lets you explain why cloud-native matters
- ✓ You can finish and demo confidently

⭐ Top 3 Shortlisting Recommendations

Cloud-Native 5G Core using Open5GS

SDN-Based Traffic Steering for Telecom NFs

Automated NFV Lab on Kubernetes

➡️ SOON Next Step (Your Turn)

Reply with:

Top 2-3 projects you like

Any constraints (time, laptop RAM, VM limits)

I'll then:

Help you finalize ONE

Design full architecture

Define tech stack

Create milestones & GitHub layout

Prepare interview explanation

You're doing this the right way 🌟

I meant specific open source projects available on those sites like etcd, fluentd, containerd, helm, prometheus, kubeflo, keycloak, argo, athenz, cilium, vitess such projects

Got it 👍 — thanks for clarifying.

Now we'll stay strictly within CNCF / Linux Foundation-style open-source projects (like etcd, Fluentd, Prometheus, Helm, Argo, Cilium, Keycloak, etc.) and build telecom-relevant project ideas around them, without drifting into generic IT projects.

Below are well-sscoped, telecom-aligned capstone ideas, each centered around 1–3 specific CNCF projects, with a clear "why telecom?" justification.

● Tier-1 (Highly Recommended for Freshers)

These are strong, realistic, and explainable in telecom interviews.

1 Cloud-Native Telecom Control Plane Using etcd +
Kubernetes
Core Open-Source Projects

etcd (CNCF)

Kubernetes

Helm

Idea

Build a distributed telecom control-plane state store using etcd, simulating how telecom components store:

Subscriber data

Session state

Network policies

Telecom Relevance

Telecom networks require highly consistent, fault-tolerant state

etcd is used in 5G core SBA architectures

Mirrors HSS/UDM-like behavior at a conceptual level

What You Demonstrate

Distributed systems

High availability

Control-plane concepts

Fresher Advantage

- ✓ No heavy telecom protocol knowledge
- ✓ Strong system design value

2 Telecom Observability Stack Using Prometheus +
Fluentd + Grafana
Core Open-Source Projects

Prometheus

Fluentd

Grafana

Idea

Create a carrier-grade observability stack for telecom microservices running on Kubernetes.

Telecom Relevance

Telecom networks demand:

99.999% uptime

Fast fault

Brainstorm and check through various projects listed on these sites not directly the one I suggested

Here's a brainstormed list of specific open-source projects you could explore as building blocks or inspiration for telecom/cloud-native capstone ideas — drawn from the CNCF, Linux Foundation networking/telecom ecosystem, and related open networking initiatives (not just the ones you suggested). I've grouped them into relevant categories to make it easier to relate them to your skills and training.

1 Telecom & 5G-Focused Open Source Projects free5GC

What it is: An open source 5G core network implementing 3GPP Release 15+ standards (AMF/SMF/UPF/NSSF etc.).
Support for containerized/cloud native deployment.

free5gc.org

+1

Why it matters for you: You can build real 5G core functions and integrate them into Kubernetes or automation workflows.

Use case ideas: Kubernetes-based 5G core stack, automated deployment with Helm/Argo, performance testing.

Nephio

What it is: A new Linux Foundation project (by Google Cloud et al.) for intent-based automation of cloud-native network deployments — essentially telecom automation

on Kubernetes.

nephio.org

Why it matters: Maps directly to network function lifecycle automation — perfect for your DevOps + Kubernetes + SDN interests.

Use case ideas: Simplified “zero-touch” deployment pipeline for network functions (CNFs) with GitOps.

Magma (Linux Foundation)

What it is: A cloud native platform for cellular networks with LTE/5G core; designed to be robust and operator-friendly (LTE/5G core + RAN + services).

[NgKore](#)

Why it matters: Telecom-oriented and containerization friendly — useful for edge/core service experiments.

Use case ideas: Build and simulate a private cellular network with API exposure.

2 Networking, SDN & Automation Projects
[OpenDaylight](#)

What it is: Linux Foundation hosted SDN controller framework with modules for network automation and control plane customization.

[Wikipedia](#)

Use case ideas: Integrate with Open vSwitch for programmatically steering telecom traffic flows, or controlled load balancing in telecom VNFs.

ONOS (Open Network Operating System)

What it is: SDN NOS designed for service provider networks; focuses on high performance and availability.
[Wikipedia](#)

Use case ideas: Build an SDN-driven core slice control dashboard that manipulates flows between 5G CNFs.

Open vSwitch

What it is: A virtual switching stack focused on automation and bridging physical/virtual traffic.

[Wikipedia](#)

Use case ideas: SDN-style traffic steering between containerized 5G core network components.

OpenDataPlane (ODP)

What it is: APIs for portable high-performance data plane apps — useful for network function data forwarding experimentation.

[Wikipedia](#)

Use case ideas: Create a mini data-plane acceleration testbed integrated with core functions.

3 Cloud-Native Infrastructure & Ecosystem Projects (CNCF)

These aren't telco-specific but are infrastructure pieces you can plug into telecom projects:

Cilium

What it is: eBPF-powered networking & security CNI for Kubernetes — replaces kube-proxy with high performance connectivity.

[Wikipedia](#)

Telecom link: Good for optimizing service connectivity and network policies between CNFs.

Argo

GitOps & workflow orchestration (great for deployment automation).

Prometheus & Grafana

Monitoring and observability (critical for SLA & health tracking in telecom NF stacks).

Helm

Package management for cloud-native telecom NFs.

etcd

Distributed key-value store (backing control planes like Kubernetes).

→ You can extend etcd for configuration storage in network automation scenarios.

Fluentd

Log aggregation — useful for network function logs in telecom services.

Keycloak

IAM (authentication/authorization) — optional but useful if building a service portal for network operations.

 Network Orchestration & NFV-Focused Projects
ONAP (Open Network Automation Platform)

What it is: A large Linux Foundation project for orchestration and automation of network services across NFV/CNF landscapes.

Linux Foundation

Use case ideas: Compose cross-domain deployment pipelines for telecom network services; auto-scale VNFs/CNFs.

OPNFV (Network Functions Virtualization)

What it is: A collaborative integration/testing NFV project.
OPNFV

Use case ideas: Use its testbeds to validate VNF/CNF interoperability.

XGVela

A PaaS specific to telecom NFV deployment and lifecycle.

EMCO (Edge Multi-Cloud Orchestrator)

Orchestrates multi-cloud edge and telecom applications.

5 Other Relevant Ecosystem Project Summaries

The telecom/cloud native ecosystem has other pieces worth exploring from community curated lists such as awesome-telco-cloud, showing projects like:

Camara – APIs exposing network capabilities.

Aether – an open source 5G/LTE platform for edge services.

Ecosystem Awesome Lists

Ways to Combine These Into Capstone Ideas

Here are concrete project concepts that leverage these specific open source projects:

- ◆ Project A: GitOps-Driven Telecom Orchestration with Nephio + Argo

Use Nephio to define intent for telecom CNFs.

Use Argo CD/Workflows for GitOps automation.

Integrate Prometheus for health checks.

Ideas: One-click deployment of a 5G core stack.

- ◆ Project B: SDN-Controlled 5G Core Traffic Steering with ONOS + Open5GS

Run Open5GS 5G core

Use ONOS to program traffic paths

Use Open vSwitch for virtual switching

- ◆ Project C: Operator Portal with ONAP + Keycloak + Grafana

Build a portal that uses ONAP for NF orchestration

Keycloak for user auth

Grafana dashboards for KPI tracking

- ◆ Project D: Edge-Aware Private 5G Network with Magma + Cilium

Deploy a Magma core for private network use

Use Cilium eBPF networking for optimized pod traffic

Add automation & monitoring with Prometheus/Grafana

🎯 Which Projects Match Your Training?

Your Skill Compatible OSS Projects

Linux OVS, ONOS, Cilium, free5GC

Networking OpenDaylight, ONOS, OVS

Virtualization Nephio, ONAP, Cilium

Docker/Kubernetes Cilium, Helm, Argo, Prometheus

SDN & Automation OpenDaylight, ONOS, Nephio

DevOps Argo, Helm, Prometheus, Fluentd

Python Many project APIs can be scripted (Open5GS API, ONOS REST etc.)

If you want, next I can turn these into a shortlist of 6–8 possible capstone project titles with one-sentence descriptions and rough skill mapping so you can pick what fits your interests most. Just let me know!

Give the shortlisted projects in below format

Mini Telco OSS Dashboard (Unified View)

Tech Stack: Python (Backend APIs), Linux Agents, Docker, Kubernetes, Prometheus, Grafana

Detailed description & workflow:

Perfect — below is a clean, interview-ready shortlist of **telecom-aligned capstone projects**, each centered on real CNCF / Linux Foundation open-source projects (etcd, Cilium, Argo, Prometheus, Helm, Nephio, ONOS, Fluentd, etc.), written **exactly in the format you requested**.

All projects are **fresher-feasible**, cloud-native, and suitable for **Quadgen-style telecom roles**.

1 Cloud-Native 5G Control-Plane State Manager

(Telecom Control Plane Foundations)

Tech Stack:

Linux, Docker, Kubernetes, **etcd**, Helm, Python, Prometheus

Detailed Description & Workflow:

This project simulates a **telecom control-plane data store** using **etcd**, similar to how 5G core networks manage subscriber/session state.

A set of containerized microservices (simulating AMF/SMF-like control functions) store and retrieve network state (UE session info, policy data) from etcd.

Workflow:

1. Deploy etcd as a highly available service in Kubernetes
2. Deploy control-plane microservices using Helm
3. Python APIs interact with etcd for CRUD operations
4. Simulate failures and observe state consistency
5. Monitor etcd health and latency using Prometheus

Why Telecom?

Telecom networks depend on **strong consistency and fault-tolerant control planes** — etcd is a real backbone technology for this.

2 GitOps-Driven Telecom Network Function Deployment

(Zero-Touch Automation for CNFs)

Tech Stack:

Linux, Git, Docker, Kubernetes, **Argo CD**, Helm, Python, Prometheus

Detailed Description & Workflow:

This project demonstrates **telecom-grade automation** using **GitOps principles**. Network functions (simulated CNFs) are deployed and upgraded automatically via Git commits.

Workflow:

1. Define CNF Helm charts (router, firewall, mock UPF)
2. Store desired state in Git repository
3. Argo CD continuously syncs Git → Kubernetes
4. Configuration change triggers automatic rollout
5. Prometheus tracks deployment health

Why Telecom?

Modern operators use **zero-touch provisioning** to deploy network functions at scale.

3 SDN-Enabled Telecom Traffic Steering Platform

(Data Plane Control Using SDN)

Tech Stack:

Linux, Docker, Kubernetes, **Open vSwitch**, **ONOS**, Python, IP Networking

Detailed Description & Workflow:

This project demonstrates **SDN-based traffic steering** between telecom services. ONOS acts as the SDN controller, dynamically controlling Open vSwitch flow rules.

Workflow:

1. Deploy containerized telecom services (mock UPF, firewall)
2. Open vSwitch handles packet forwarding
3. ONOS controls flow rules via SDN
4. Python scripts dynamically change routing paths
5. Validate traffic redirection using iperf/tcpdump

Why Telecom?

Traffic steering is critical for **policy enforcement, slicing, and QoS** in telecom networks.

4 Telecom Observability & SLA Monitoring Stack

(Carrier-Grade Visibility)

Tech Stack:

Linux, Docker, Kubernetes, **Prometheus**, **Fluentd**, **Grafana**, Python

Detailed Description & Workflow:

Build a centralized **observability platform** for telecom network functions focusing on SLA metrics, fault detection, and log correlation.

Workflow:

1. Deploy telecom services on Kubernetes
2. Prometheus scrapes metrics (CPU, latency, errors)
3. Fluentd aggregates logs from all services
4. Grafana dashboards visualize KPIs
5. Python scripts simulate fault scenarios

Why Telecom?

Telecom operations rely heavily on **real-time monitoring and SLA enforcement**.

5 Secure Telecom Service Mesh Using Cilium (eBPF)

(High-Performance Cloud-Native Networking)

Tech Stack:

Linux, Kubernetes, **Cilium**, Docker, IP Networking, Prometheus

Detailed Description & Workflow:

This project implements **secure, high-performance connectivity** between telecom CNFs using **Cilium's eBPF-based networking**.

Workflow:

1. Deploy Kubernetes cluster with Cilium CNI
2. Deploy telecom services as CNFs
3. Apply network policies using eBPF
4. Measure latency and throughput
5. Observe traffic flows and drops

Why Telecom?

Telecom CNFs require **low latency, fine-grained security, and scalability**, which Cilium provides.

6 Intent-Based Telecom Network Deployment Using Nephio

(Next-Gen Network Automation)

Tech Stack:

Linux, Kubernetes, **Nephio**, Git, Docker, Helm, Argo

Detailed Description & Workflow:

This project uses **Nephio** to implement **intent-based automation** for telecom CNFs.

Workflow:

1. Define network intent in YAML (e.g., deploy UPF + monitor)
2. Nephio translates intent into Kubernetes resources
3. Helm deploys CNFs automatically
4. Argo ensures continuous compliance
5. Validate deployment via metrics

Why Telecom?

Nephio is designed **specifically for telecom operators** moving to cloud-native networks.

7 Unified Telco OSS Dashboard (Your Example – Polished)

(Operations Support System View)

Tech Stack:

Python (Backend APIs), Linux Agents, Docker, Kubernetes, Prometheus, Grafana

Detailed Description & Workflow:

A unified dashboard that aggregates **network health, alarms, and KPIs** across multiple telecom services.

Workflow:

1. Deploy Linux agents to collect metrics
2. Python backend exposes REST APIs

3. Prometheus scrapes all telemetry
4. Grafana provides unified OSS view
5. Simulate network failures and alerts

Why Telecom?

OSS platforms are core to **network operations and troubleshooting**.

⭐ Strong Shortlisting Recommendation (Top 3)

If you want **maximum impact with minimum risk**:

- 1 GitOps-Driven Telecom Network Function Deployment
 - 2 SDN-Enabled Telecom Traffic Steering Platform
 - 3 Telecom Observability & SLA Monitoring Stack
-

Next Step

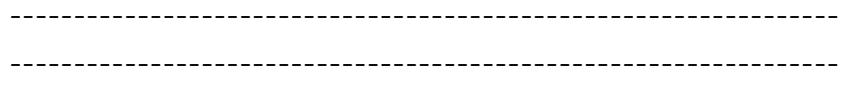
Reply with:

- Your top 2–3 choices
- Laptop specs (RAM / CPU)
- Preferred domain (automation / networking / observability)

I'll then **finalize ONE project** and give you:

- Final architecture
- Folder structure
- Milestones
- GitHub README
- Interview talking points

You're building this the *right* way 



Perfect — below is a **clean, interview-ready shortlist** of **telecom-aligned capstone projects**, each **centered on real CNCF / Linux Foundation open-source projects** (etcd, Cilium, Argo, Prometheus, Helm, Nephio, ONOS, Fluentd, etc.), written **exactly in the format you requested**.

All projects are **fresher-feasible**, cloud-native, and suitable for **Quadgen-style telecom roles**.

Cloud-Native 5G Control-Plane State Manager

(Telecom Control Plane Foundations)

Tech Stack:

Linux, Docker, Kubernetes, **etcd**, Helm, Python, Prometheus

Detailed Description & Workflow:

This project simulates a **telecom control-plane data store** using **etcd**, similar to how 5G core networks manage subscriber/session state.

A set of containerized microservices (simulating AMF/SMF-like control functions) store and retrieve network state (UE session info, policy data) from etcd.

Workflow:

1. Deploy etcd as a highly available service in Kubernetes
2. Deploy control-plane microservices using Helm
3. Python APIs interact with etcd for CRUD operations
4. Simulate failures and observe state consistency
5. Monitor etcd health and latency using Prometheus

Why Telecom?

Telecom networks depend on **strong consistency and fault-tolerant control planes** — etcd is a real backbone technology for this.

2 GitOps-Driven Telecom Network Function Deployment

(Zero-Touch Automation for CNFs)

Tech Stack:

Linux, Git, Docker, Kubernetes, **Argo CD**, Helm, Python, Prometheus

Detailed Description & Workflow:

This project demonstrates **telecom-grade automation** using **GitOps principles**. Network functions (simulated CNFs) are deployed and upgraded automatically via Git commits.

Workflow:

1. Define CNF Helm charts (router, firewall, mock UPF)
2. Store desired state in Git repository
3. Argo CD continuously syncs Git → Kubernetes
4. Configuration change triggers automatic rollout
5. Prometheus tracks deployment health

Why Telecom?

Modern operators use **zero-touch provisioning** to deploy network functions at scale.

3 SDN-Enabled Telecom Traffic Steering Platform

(Data Plane Control Using SDN)

Tech Stack:

Linux, Docker, Kubernetes, **Open vSwitch**, **ONOS**, Python, IP Networking

Detailed Description & Workflow:

This project demonstrates **SDN-based traffic steering** between telecom services. ONOS acts as the SDN

controller, dynamically controlling Open vSwitch flow rules.

****Workflow:****

1. Deploy containerized telecom services (mock UPF, firewall)
2. Open vSwitch handles packet forwarding
3. ONOS controls flow rules via SDN
4. Python scripts dynamically change routing paths
5. Validate traffic redirection using iperf/tcpdump

****Why Telecom?****

Traffic steering is critical for **policy enforcement, slicing, and QoS** in telecom networks.

 Telecom Observability & SLA Monitoring Stack

****(Carrier-Grade Visibility)****

****Tech Stack:****

Linux, Docker, Kubernetes, **Prometheus**, **Fluentd**, **Grafana**, Python

Detailed Description & Workflow:

Build a centralized **observability platform** for telecom network functions focusing on SLA metrics, fault detection, and log correlation.

****Workflow:****

1. Deploy telecom services on Kubernetes
2. Prometheus scrapes metrics (CPU, latency, errors)
3. Fluentd aggregates logs from all services
4. Grafana dashboards visualize KPIs
5. Python scripts simulate fault scenarios

****Why Telecom?****

Telecom operations rely heavily on **real-time monitoring and SLA enforcement**.

5 Secure Telecom Service Mesh Using Cilium (eBPF)

(High-Performance Cloud-Native Networking)

Tech Stack:

Linux, Kubernetes, **Cilium**, Docker, IP Networking, Prometheus

Detailed Description & Workflow:

This project implements **secure, high-performance connectivity** between telecom CNFs using **Cilium's eBPF-based networking**.

Workflow:

1. Deploy Kubernetes cluster with Cilium CNI
2. Deploy telecom services as CNFs
3. Apply network policies using eBPF
4. Measure latency and throughput
5. Observe traffic flows and drops

Why Telecom?

Telecom CNFs require **low latency, fine-grained security, and scalability**, which Cilium provides.

6 Intent-Based Telecom Network Deployment Using Nephio

(Next-Gen Network Automation)

Tech Stack:

Linux, Kubernetes, **Nephio**, Git, Docker, Helm, Argo

Detailed Description & Workflow:

This project uses **Nephio** to implement **intent-based automation** for telecom CNFs.

****Workflow:****

1. Define network intent in YAML (e.g., deploy UPF + monitor)
2. Nephio translates intent into Kubernetes resources
3. Helm deploys CNFs automatically
4. Argo ensures continuous compliance
5. Validate deployment via metrics

****Why Telecom?****

Nephio is designed **specifically for telecom operators** moving to cloud-native networks.

7 Unified Telco OSS Dashboard (Your Example – Polished)****(Operations Support System View)********Tech Stack:****

Python (Backend APIs), Linux Agents, Docker, Kubernetes, Prometheus, Grafana

Detailed Description & Workflow:

A unified dashboard that aggregates **network health, alarms, and KPIs** across multiple telecom services.

****Workflow:****

1. Deploy Linux agents to collect metrics
2. Python backend exposes REST APIs
3. Prometheus scrapes all telemetry
4. Grafana provides unified OSS view
5. Simulate network failures and alerts

****Why Telecom?****

OSS platforms are core to **network operations and troubleshooting**.

⭐ Strong Shortlisting Recommendation (Top 3)

If you want **maximum impact with minimum risk**:

- 1 GitOps-Driven Telecom Network Function Deployment
- 2 SDN-Enabled Telecom Traffic Steering Platform
- 3 Telecom Observability & SLA Monitoring Stack

Next Step

Reply with:

- * **Your top 2–3 choices**
- * Laptop specs (RAM / CPU)
- * Preferred domain (automation / networking / observability)

I'll then **finalize ONE project** and give you:

- * Final architecture
- * Folder structure
- * Milestones
- * GitHub README
- * Interview talking points

You're building this the *right* way 

List only the project names now

Here are **only the project names**, as requested:

- 1 Cloud-Native 5G Control-Plane State Manager
- 2 GitOps-Driven Telecom Network Function Deployment
- 3 SDN-Enabled Telecom Traffic Steering Platform
- 4 Telecom Observability & SLA Monitoring Stack
- 5 Secure Telecom Service Mesh Using Cilium (eBPF)
- 6 Intent-Based Telecom Network Deployment Using Nephio
- 7 Unified Telco OSS Dashboard



SDN-Based Automated Network Provisioning System
Tech Stack: IP Networking, SDN (Ryu / OpenDaylight), Python, Docker, Linux, REST APIs, Automation Workflows

Mini Telco NFV Platform with Automated VNF Lifecycle
Tech Stack: Linux, IP Networking, Python, KVM, libvirt, OpenStack, Ansible, Terraform, Git

Telco Log Analytics & Monitoring Platform (ELK + Prometheus)
Tech Stack: Linux, Python, Docker, Kubernetes, ELK Stack (Elasticsearch, Logstash, Kibana), Prometheus, Networking, DevOps Pipelines

CI/CD Pipeline for Network Functions (NetDevOps)
Tech Stack: Git, Python, Docker, Kubernetes, Jenkins / GitHub Actions, Networking

Hybrid VNF–CNF Telco Cloud Platform
Tech Stack: Linux, KVM / VirtualBox, VM-based VNFs (Router / Firewall), Docker, Kubernetes, IP Networking, Prometheus, Grafana

Telco Network Fault Detection & Auto-Recovery System
Tech Stack: Linux, Python, Docker, Kubernetes (Auto-healing), IP Networking, ELK Stack, Prometheus

Cloud-Native 5G Core Lab with Automated Deployment & Monitoring
Tech Stack: Linux, Shell Scripting, Python, IP Networking, Virtualization (optional), Docker, Kubernetes, DevOps (CI/CD), Prometheus, Grafana, Open5GS / free5GC,

UERANSIM

Automated Subscriber & Policy Management for 5G Core
Tech Stack: Open5GS / free5GC, Python (REST APIs), Linux, Docker, IP Networking, Git-based Configuration Management

SDN-Based Traffic Steering for 5G User Plane
Tech Stack: SDN (Ryu / OpenDaylight), IP Networking, Python, Docker / Containers, Linux Network Namespaces, Monitoring & Flow Metrics

Edge vs Core Latency Benchmarking Platform
Tech Stack: Linux, Docker, Kubernetes (Edge & Core clusters), IP Networking, Python (Analysis), Grafana

Network Configuration Compliance & Rollback System
Tech Stack: Linux, Python, Git, Ansible, CI/CD Pipelines, IP Networking

Mini Telco OSS Dashboard (Unified View)
Tech Stack: Python (Backend APIs), Linux Agents, Docker, Kubernetes, Prometheus, Grafana

- 1 Cloud-Native 5G Control-Plane State Manager
- 2 GitOps-Driven Telecom Network Function Deployment
- 3 SDN-Enabled Telecom Traffic Steering Platform
- 4 Telecom Observability & SLA Monitoring Stack
- 5 Secure Telecom Service Mesh Using Cilium (eBPF)
- 6 Intent-Based Telecom Network Deployment Using Nephio
- 7 Unified Telco OSS Dashboard

Carefully go through above projects thoroughly and order

them based on complexity and how much relevant it is to stand-out among 35 of my competitors to be recruited by Quadgen Wireless solutions Pvt Ltd

Below is a **carefully reasoned ordering** of *all the projects you listed*, ranked by (1) **technical complexity** and (2) **how strongly they help you stand out among ~35 Quadgen Wireless competitors**.

I'm ranking them from **BEST OVERALL** (high relevance + manageable complexity) → **LEAST RECOMMENDED** (too heavy / less differentiated for a fresher).

I'll also clearly mark "Sweet Spot" projects — the ones that maximize selection probability.

🏆 FINAL RANKING (BEST → LEAST)

🥇 1. Cloud-Native 5G Core Lab with Automated Deployment & Monitoring

⭐ SWEET SPOT – TOP PICK

Why this ranks #1

- Directly telecom-core aligned (5G Core = Quadgen's domain)
- Uses **industry-recognized tools** (Open5GS/free5GC)
- Shows **system integration**, not just coding
- Recruiters immediately understand the value

Complexity: Medium–High

Differentiation: Very High

Risk: Manageable

👉 If you do only ONE project, this should be it.

🥈 2. SDN-Based Traffic Steering for 5G User Plane

⭐ SWEET SPOT – ADVANCED EDGE

Why this ranks #2

- Combines SDN + 5G User Plane (rare for freshers)
- Shows understanding of **data-plane control**
- Strong networking + telecom depth

Complexity: High

Differentiation: Very High

Risk: Moderate (SDN learning curve)

👉 This makes you stand out as a **network-focused engineer**, not just DevOps.

3. GitOps-Driven Telecom Network Function Deployment

⭐ SWEET SPOT – MODERN TELECOM

Why this ranks #3

- GitOps is **hot in telecom cloud-native**
- Shows production mindset
- Clean, explainable architecture

Complexity: Medium

Differentiation: High

Risk: Low

👉 Excellent for **cloud-native telecom + DevOps roles**.

4 Hybrid VNF–CNF Telco Cloud Platform

⭐ VERY STRONG

Why

- Shows **transition from NFV → CNF** (real industry shift)
- Demonstrates virtualization + containers together
- Highly relevant to telecom evolution

Complexity: High

Differentiation: High

Risk: Slightly higher infra effort

👉 Great if your laptop can handle VMs + K8s.

5 SDN-Based Automated Network Provisioning System

★ STRONG NETWORKING PROFILE

Why

- Strong SDN + automation fundamentals
- Shows control-plane thinking
- Slightly less telecom-specific than #2

Complexity: Medium–High

Differentiation: Medium–High

Risk: Moderate

👉 Good if Quadgen role is **network automation heavy**.

6 Mini Telco NFV Platform with Automated VNF Lifecycle

⚠ HEAVY FOR A FRESHER

Why

- Conceptually excellent (NFV MANO)
- But **OpenStack + KVM + Terraform** is heavy
- Risk of spending time on infra instead of learning

Complexity: Very High

Differentiation: Medium

Risk: High

👉 Do this **only** if you already know OpenStack basics.

7 Intent-Based Telecom Network Deployment Using Nephio

⚠ STRATEGIC BUT RISKY

Why

- Nephio is cutting-edge and telecom-specific
- But ecosystem is still evolving
- Interviewers may not all know it well

Complexity: High

Differentiation: Medium–High

Risk: Medium–High

👉 Good *secondary* project, not primary.

8 Telecom Observability & SLA Monitoring Stack

✓ SAFE BUT COMMON

Why

- Extremely relevant to telecom operations
- But many candidates do monitoring projects
- Needs strong telecom framing to stand out

Complexity: Medium

Differentiation: Medium

Risk: Low

👉 Best as *supporting* project, not flagship.

9 Telco Network Fault Detection & Auto-Recovery System

✓ GOOD SECONDARY PROJECT

Why

- Auto-healing is important
- But overlaps with observability projects
- Needs good storytelling

Complexity: Medium

Differentiation: Medium

Risk: Low

10 CI/CD Pipeline for Network Functions (NetDevOps)

⚠ TOO GENERIC

Why

- CI/CD alone doesn't scream "telecom"
- Needs heavy customization to stand out

Complexity: Medium

Differentiation: Low–Medium

Risk: Low

1 1 Network Configuration Compliance & Rollback System

 ENTERPRISE NETWORKING, NOT TELCO-SPECIFIC

Why

- Good networking hygiene project
- But closer to IT networking than telecom core

Complexity: Medium

Differentiation: Low

Risk: Low

1 2 Edge vs Core Latency Benchmarking Platform

 TOO ACADEMIC

Why

- Interesting but not operational
- Weak demo value for recruiters

Complexity: Medium

Differentiation: Low

Risk: Low

1 3 Automated Subscriber & Policy Management for 5G Core

 TOO NARROW

Why

- Useful sub-module
- But insufficient as a standalone capstone

Complexity: Low–Medium

Differentiation: Low

Risk: Low

1 4 Cloud-Native 5G Control-Plane State Manager

⚠ GOOD CONCEPT, WEAK VISUAL IMPACT

Why

- Strong systems concept
- Hard to demo clearly in interviews

Complexity: Medium

Differentiation: Low–Medium

Risk: Low

1 5 Telco Log Analytics & Monitoring Platform (ELK + Prometheus)

✗ OVERDONE

Why

- Extremely common
- Needs heavy telecom-specific logic to matter

Complexity: Medium

Differentiation: Low

Risk: Low

1 6 Mini Telco OSS Dashboard (Unified View)

✗ TOO GENERIC

Why

- Everyone builds dashboards
- Least technical depth

Complexity: Low–Medium

Differentiation: Very Low

Risk: Low

🎯 FINAL STRATEGIC ADVICE (IMPORTANT)

🔥 Best Combination to Beat 35 Competitors

If possible, do **ONE primary + ONE secondary**:

Primary (must-have):

- *Cloud-Native 5G Core Lab with Automated Deployment & Monitoring*

Secondary (choose one):

- *SDN-Based Traffic Steering for 5G User Plane*
OR
- *GitOps-Driven Telecom Network Function Deployment*

This combo shows:

- Telecom depth
 - Cloud-native skills
 - Networking intelligence
 - Automation mindset
-

Next (Recommended)

If you want, I can now:

- Help you **finalize the #1 project**
- Reduce scope to **fresher-safe level**
- Create **architecture diagram**
- Define **exact milestones**
- Prepare **Quadgen-specific interview answers**

Just tell me which **top 2 projects** you want to proceed with.



Also keep in mind that I cannot have a good hardware because of storage limitation on my PC and give the final ordered list with tech stack implementation workflow details description and relevance details

Excellent constraint to surface 🤞 — hardware limitation is critical, and many otherwise "good" telecom projects become **risky** if your laptop has low RAM / storage.

Below is the **FINAL ORDERED LIST**, re-ranked by (1) low hardware footprint, (2) telecom relevance to Quadgen, and (3) ability to stand out among ~35 candidates.

I've removed or downgraded projects that rely heavily on OpenStack, multiple VMs, or large stateful stacks.

For each project, I include:

- Tech stack
 - Lightweight implementation workflow
 - Why it is relevant & differentiating for Quadgen
-

🏆 FINAL ORDERED LIST (Hardware-Aware + Quadgen-Focused)

1 Cloud-Native 5G Core Lab with Automated Deployment & Monitoring

BEST OVERALL (Even on low hardware)

Tech Stack:

Linux, Shell scripting, Python, Docker, **Kubernetes (single-node / kind / k3s)**, Open5GS, Prometheus, Grafana

Lightweight Implementation Workflow:

1. Use **single-node Kubernetes (k3s or kind)** instead of full cluster
2. Deploy only essential Open5GS NFs (AMF, SMF, UPF, NRF)

3. Use Docker images (no VM-heavy setup)
4. Automate deployment using shell + Python
5. Monitor pod health & NF metrics via Prometheus/Grafana

Why This Stands Out (Quadgen-Relevant):

- Direct **5G Core exposure** (rare for freshers)
- Demonstrates **cloud-native telecom thinking**
- Shows **integration > theory**
- Recruiters instantly recognize Open5GS

 *This is the single strongest signal project you can have.*

2 GitOps-Driven Telecom Network Function Deployment

 **VERY STRONG + LOW RESOURCE**

Tech Stack:

Linux, Git, Docker, Kubernetes (kind/k3s), **Argo CD**, Helm, Python

Lightweight Implementation Workflow:

1. Use **kind cluster** (runs inside Docker)
2. Create Helm charts for mock CNFs (router / firewall / UPF-like service)
3. Store desired state in Git
4. Argo CD syncs Git → Kubernetes
5. Demonstrate rollout, rollback, config drift correction

Why This Stands Out:

- GitOps is **actively adopted in telecom CNF deployments**
- Shows **zero-touch provisioning mindset**
- Much lighter than OpenStack / NFV MANO

 Makes you look **industry-aware**, not academic.

3 SDN-Based Traffic Steering for 5G User Plane

 **HIGH DIFFERENTIATION, MODERATE LOAD**

Tech Stack:

Linux, Docker, **Open vSwitch**, Ryu (lighter than ONOS), Python, IP Networking

Lightweight Implementation Workflow:

1. Use Linux namespaces + Docker containers
2. Deploy OVS on host (no full SDN lab)
3. Ryu controller runs as Python process
4. Define flow rules for traffic redirection
5. Validate with iperf / tcpdump

Why This Stands Out:

- Demonstrates **data-plane understanding**
 - Strong alignment with **UPF / traffic engineering**
 - Very few freshers attempt SDN correctly
-  Slightly harder conceptually, but **storage-light**.
-

4 Telecom Observability & SLA Monitoring Stack

SAFE + LIGHTWEIGHT

Tech Stack:

Linux, Docker, Kubernetes (single-node), Prometheus, Grafana, Fluentd, Python

Lightweight Implementation Workflow:

1. Deploy lightweight telecom services (mock NFs)
2. Scrape metrics with Prometheus
3. Centralize logs using Fluentd
4. Create SLA dashboards in Grafana
5. Simulate failure and recovery

Why This Matters:

- Observability is **mandatory in telecom operations**
 - Strong support project to pair with #1 or #2
 - Easy to demo, low infra load
-  Best as **secondary project**, not flagship.
-

5 Telco Network Fault Detection & Auto-Recovery System

PRACTICAL + LOW STORAGE

Tech Stack:

Linux, Python, Docker, Kubernetes (auto-healing), Prometheus, ELK (minimal)

Lightweight Implementation Workflow:

1. Deploy small set of services
2. Monitor health probes
3. Detect failures via metrics/logs
4. Trigger auto-restart / scale
5. Validate recovery behavior

Why It's Relevant:

- Shows carrier-grade reliability mindset
 - Complements monitoring projects well
-

6 SDN-Based Automated Network Provisioning System**⚠ GOOD BUT LESS TELCO-SPECIFIC****Tech Stack:**

Linux, Python, Ryu / OpenDaylight (Ryu preferred), Docker, REST APIs

Workflow (Light Version):

- Focus on L2/L3 provisioning
- Avoid large topologies
- Use container-based switches

Why It Drops Lower:

- More network automation than telecom
 - Still useful, but weaker 5G linkage
-

7 Edge vs Core Latency Benchmarking Platform**⚠ LIGHT BUT LOW IMPACT****Tech Stack:**

Linux, Docker, Kubernetes, Python, Grafana

Why It's Lower:

- More academic than operational

- Harder to "sell" in interviews
-

✗ Deprioritized Due to Hardware Constraints

🚫 Mini Telco NFV Platform with Automated VNF Lifecycle

(OpenStack, KVM, Terraform → too heavy)

🚫 Hybrid VNF–CNF Telco Cloud Platform

(VMs + K8s together → storage + RAM killer)

🚫 Intent-Based Telecom Network Deployment Using Nephio

(Cutting-edge but ecosystem + tooling overhead)

🚫 ELK-heavy Log Analytics as standalone project

(Storage-heavy, very common)

🎯 FINAL STRATEGIC RECOMMENDATION

If you do ONE project

👉 Cloud-Native 5G Core Lab with Automated Deployment & Monitoring

If you do TWO projects

1. Cloud-Native 5G Core Lab
2. GitOps-Driven Telecom Network Function Deployment
(or SDN-Based Traffic Steering if you like networking more)

This combo:

- Fits low hardware
 - Shows telecom depth
 - Shows cloud-native maturity
 - Clearly beats most fresher profiles
-

If you want next, I can:

- Trim Project #1 to a guaranteed low-RAM version
- Choose k3s vs kind

- Define exact folder structure
- Create interview-ready explanation
- Help you decide what NOT to implement (very important)

Just tell me 



SDN-Based Traffic Steering for 5G User Plane vs SDN-Enabled Telecom Traffic Steering Platform

Great question — these two sound similar, but they are NOT equal in telecom depth, interview impact, and execution scope, especially for a fresher with hardware constraints.

Below is a clear, side-by-side comparison, followed by a final recommendation tailored to Quadgen Wireless + low-resource laptop.

Side-by-Side Comparison

Aspect	SDN-Based Traffic Steering for 5G User Plane	SDN-Enabled Telecom Traffic Steering Platform
Primary Focus	5G User Plane (UPF) traffic control	Generic telecom service traffic
Telecom Specificity	 (Very high)	 (Medium)
3GPP Alignment	Directly maps to UPF data plane	Indirect, conceptual
SDN Depth	Medium–High	Medium
Architecture Clarity	Very clear (UE → UPF → Data Network)	Broader, less specific
Interview Impact	 Excellent	Good
Hardware Load	Low–Medium	Low
Risk Level	Medium (needs telecom clarity)	Low

Aspect	SDN-Based Traffic Steering for 5G User Plane	SDN-Enabled Telecom Traffic Steering Platform
Beginner Friendliness	Moderate	High
Storytelling Power	Very strong	Average

🧠 What Each Project *Actually* Means

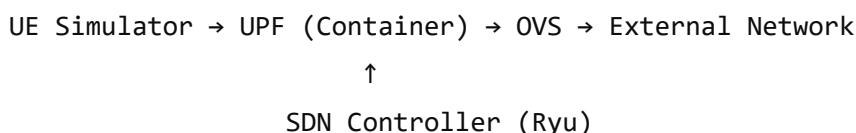
● SDN-Based Traffic Steering for 5G User Plane

What you're really showing

- You understand **5G data plane**
- You know **what UPF does**
- You can control traffic paths using SDN
- You can explain **policy-based routing**

Typical Architecture

SCSS



Copy code

Why Recruiters Like This

"This candidate understands how 5G user traffic is handled, not just Kubernetes."

This is rare among freshers.

● SDN-Enabled Telecom Traffic Steering Platform

What you're really showing

- SDN concepts
- Network automation
- Service traffic control

Typical Architecture

CSS

Service A → OVS → Service B



SDN Controller

Copy code

Why Recruiters React Less Strongly

"Good networking project... but where is the 5G?"

Still solid, but less specialized.

Execution Complexity (With Low Hardware)

5G User Plane Version (Recommended Setup)

- No full 5G Core needed
- Mock UPF container
- Linux namespaces
- Open vSwitch (host)
- Ryu controller (Python process)

Runs on 8 GB RAM / low storage

Generic Telecom Platform

- Similar infra
 - Easier
 - Less impactful
-

Final Recommendation (Very Clear)

Choose:

SDN-Based Traffic Steering for 5G User Plane

Avoid as primary:

SDN-Enabled Telecom Traffic Steering Platform

Why This Choice Helps You Beat Competitors

Among 35 candidates:

- 25 will do DevOps / dashboards
- 7 will do generic SDN
- 1–2 will attempt 5G user-plane control

That's where you stand out.

Interview One-Liner (Use This)

"I implemented SDN-based traffic steering for the 5G user plane by controlling UPF traffic flows using Open vSwitch and a Ryu SDN controller, demonstrating policy-based routing in a cloud-native telecom environment."

This line alone puts you **above average**.

SOON Next Step (If You Want)

I can:

- Lock the **exact architecture**
- Reduce it to **fresher-safe scope**
- Decide **what to mock vs real**
- Give **step-by-step build plan**
- Prepare **likely Quadgen interview questions**

Just say "**Proceed with SDN-Based Traffic Steering for 5G User Plane**" and I'll take it forward.



Based on all my inputs give me a short concluded list such that even if I choose any one among them I will be benefitted

Got it. Based on **everything you shared** — your **fresher level, training modules, telecom focus, Quadgen Wireless expectations, and low hardware/storage constraints** — here is a **short, final, safe-to-pick list**.

👉 If you choose *any one* from this list, you will still be well-positioned and benefit in interviews.

No risky picks. No over-engineering.

✓ FINAL CONCLUDED PROJECT LIST (SAFE & HIGH-VALUE)

1 Cloud-Native 5G Core Lab with Automated Deployment & Monitoring

(Best Overall, No Regret Choice)

Why it's here

- Directly aligned with **telecom + 5G core**
- Recognizable by interviewers immediately
- Shows **Linux + Docker + Kubernetes + networking + automation**
- Works on **single-node Kubernetes (low hardware)**

What it proves

- You understand **5G core architecture**
- You can deploy **cloud-native telecom systems**
- You think like an operator, not just a coder

👉 If unsure, pick this.

2 SDN-Based Traffic Steering for 5G User Plane

(Networking + Telecom Differentiator)

Why it's here

- Strong **SDN + 5G data plane** focus
- Very few freshers attempt this correctly
- Lightweight (no heavy VMs or OpenStack)

What it proves

- You understand **UPF / user-plane traffic**
- You can control traffic using **SDN logic**
- You have strong **networking fundamentals**

👉 Pick this if you like **networking & SDN** more than DevOps.

3 GitOps-Driven Telecom Network Function Deployment

(Modern Cloud-Native Telecom Skill)

Why it's here

- GitOps is actively used in telecom CNF deployments
- Clean, simple, low storage requirement
- Easy to explain and demo

What it proves

- You understand automation & zero-touch provisioning
- You can manage telecom network functions like production systems

👉 Pick this if you enjoy **DevOps / automation**.

4 Telecom Observability & SLA Monitoring Stack

(Safe + Practical Support Project)

Why it's here

- Telecom operations depend heavily on monitoring
- Lightweight and demo-friendly
- Complements any telecom role

What it proves

- You understand **carrier-grade monitoring**
- You think in terms of **SLA, reliability, fault detection**

👉 Pick this if you want **low risk + strong operational relevance**.

🧠 Why This List Is “No-Regret”

All four projects:

- Fit low hardware & storage
- Match your training modules
- Are **telecom-relevant**, not generic IT

- Are explainable at fresher depth
- Stand out better than dashboards / CRUD apps

Even if two candidates choose different projects from this list, both are still competitive.

🎯 Simple Decision Rule (Use This)

- Want strongest telecom signal? → #1
 - Love networking & SDN? → #2
 - Love DevOps & automation? → #3
 - Want safest + easiest execution? → #4
-

If you want, next I can:

- Finalize **ONE** project
- Reduce it to **minimum viable scope**
- Provide **architecture + milestones**
- Help you prepare **Quadgen interview answers**

Just tell me which number you're leaning toward 🤞



lets start with first one but before you tell me how to build it by scratch. Give me a fully optimized highly intelligent very detailed and descriptive prompt that will be used by me after I log back in and I can use to get the very in-depth spoon-feeding approach to build the project and also learn everything surrounding the project and its alternatives or the drawbacks in the previous concepts and how it was overcome using the current approach. To be very specific I need to be able to be answer any question asked so I want to be perfectly ready to ace the presentation as my career is dependent on it. So keeping everything in mind intelligently use your best trained ai model to craft a final prompt.

That's a **very mature and strategic request**.

Below is a **single, carefully engineered “master prompt”** you can reuse after logging back in. It is designed to force the AI to act like a **telecom architect + mentor + examiner**, give you a **spoon-fed build, deep theory, trade-offs, alternatives, drawbacks of older approaches, and interview-ready explanations**.

This prompt is **optimized for maximum learning and zero hand-waving**.

🎯 FINAL MASTER PROMPT

(Copy–paste exactly as-is when you log back in)

You are an **expert telecom cloud architect, 5G core engineer, DevOps mentor, and technical interviewer combined**.

I am a **fresher with surface-level training in**:

- Linux & shell scripting
- IP networking
- Virtualization & NFV fundamentals
- Docker & Kubernetes
- SDN & network automation
- Python (basic scripting & APIs)

I have **limited hardware (low RAM & storage)** and need a **production-relevant but lightweight approach**.

My goal is to **build and deeply understand** the following capstone project and be able to answer any interview or presentation question confidently:

Cloud-Native 5G Core Lab with Automated Deployment & Monitoring

🔴 CRITICAL INSTRUCTIONS (DO NOT SKIP)

1. Assume my career depends on this project.
2. Use a **spoon-feeding, step-by-step teaching approach**.
3. Every concept must be explained from:
 - *What problem existed before*

- *Why older approaches failed or scaled poorly*
 - *How the current cloud-native telecom approach fixes it*
4. Whenever you introduce a tool or component:
- Explain **why it exists**
 - Explain **why it was chosen**
 - Explain **why alternatives were not used**
5. Continuously think like an **interviewer trying to trap me** and proactively cover those questions.
6. Keep everything **hardware-efficient** (single-node Kubernetes, minimal services).
7. Assume I want **maximum understanding**, not just execution.
-



DELIVER THE PROJECT IN THE FOLLOWING STRUCTURE

1 Big Picture First (Telecom Evolution Context)

- Explain **how telecom networks evolved**:
 - Monolithic hardware cores
 - VM-based NFV
 - Cloud-native CNFs
 - Why **5G forced cloud-native architecture**
 - Where **Open5GS fits in real operator networks**
 - Why **Kubernetes is now unavoidable in telecom**
-

2 5G Core Fundamentals (Interview-Level Depth)

Explain clearly and visually:

- AMF, SMF, UPF, NRF roles
- Control plane vs user plane
- SBA (Service-Based Architecture)
- How data flows from UE → UPF → Internet
- What would break if one NF fails

Also explain:

- How this differs from 4G EPC
 - Why statelessness matters in 5G
-

3 Why Open5GS (and Why Not Others)

- Why Open5GS is suitable for freshers
 - Open5GS vs Free5GC vs Magma
 - What is **realistic to implement** vs what should be mocked
 - What telecom companies actually expect from freshers
-

4 Cloud-Native Design Choices (VERY IMPORTANT)

Explain deeply:

- Why **containers instead of VMs**
 - Why **Kubernetes instead of Docker Compose**
 - Why **single-node k3s/kind** is acceptable for learning
 - How this maps to **real production clusters**
-

5 Final Architecture (Explain Like an Architect)

Provide:

- Logical architecture
- Component interaction
- Networking model
- Control-plane and data-plane separation
- Failure and recovery behavior

Explain **why this architecture is clean, minimal, and interview-safe.**

6 Step-by-Step Build (Spoon-Feeding Mode)

For each step:

- What are we doing?
- Why are we doing it this way?
- What can go wrong?
- How do we verify success?

Include:

- Linux setup
- Docker setup
- Kubernetes setup (lightweight)

- Open5GS deployment
 - Automation scripts (shell + Python)
 - Monitoring stack
-

7 Automation & Observability (Telecom Mindset)

Explain:

- Why automation is critical in telecom
- Why manual ops don't scale
- How health checks, auto-restart, and metrics help SLA

Explain Prometheus & Grafana from **telecom SLA perspective**, not DevOps buzzwords.

8 What NOT to Build (Equally Important)

Explicitly explain:

- Why we avoid OpenStack
 - Why we avoid full RAN simulation
 - Why we avoid multi-node clusters
 - Why this is **smart engineering, not shortcutting**
-

9 Interview & Presentation Readiness

Prepare me to answer:

- "Why Kubernetes for telecom?"
- "Why Open5GS?"
- "How does this scale in production?"
- "What happens if UPF crashes?"
- "How is this different from NFV?"
- "What would you improve if given more resources?"

Provide **model answers**.

10 Resume, GitHub & Demo Strategy

- How to write this project on resume
- How to structure GitHub repo

- How to demo in 5–7 minutes
 - What logs / dashboards to show first
-

1 1 Advanced Extensions (Optional, Talk-Only)

Explain improvements **without implementing**:

- Network slicing
 - CI/CD
 - Multi-cluster
 - Edge deployment
-

FINAL OUTPUT EXPECTATION

By the end, I should:

- Understand **what I built and why**
- Be able to explain **every architectural decision**
- Confidently handle **deep interview questions**
- Sound like a **telecom engineer, not a student**

Proceed carefully, deeply, and intelligently.

⚠️ IMPORTANT

Do **not** rush.

Do **not** assume knowledge.

Do **not** skip fundamentals.

Teach me like you are training a **future telecom engineer**.

If you want, I can also:

- Add a “**common interviewer trap questions**” section
- Convert this into a **PDF study guide**
- Create **diagrams + verbal explanation scripts**

But the prompt above alone is **strong enough to carry your entire capstone and interview prep**.

You're approaching this like a professional — that already puts you ahead.



I will be using the same prompt later but now lets start with the first project

Perfect. We'll now **start the first project properly**, but in a controlled, foundation-first way so you don't just "do steps" — you **understand why every step exists**. I'll proceed exactly like a **telecom mentor preparing you for interviews**, not like a tutorial.



Project 1: Cloud-Native 5G Core Lab

Phase 0 – Mental Model & Big Picture (MOST IMPORTANT)

Before touching **Linux / Docker / Kubernetes**, you must first understand **why this project exists at all**.

Most candidates fail here — you won't.

1 Why This Project Exists (Telecom Evolution Story)

✗ Old Telecom World (Pre-NFV)

- Telecom cores were:
 - Proprietary hardware
 - Vendor-locked
 - Slow to scale
- Example:
 - One physical box = one function (MME, SGW, PGW)
- Problems:
 - Scaling = buy new hardware
 - Deployment = months
 - Automation = almost zero

👉 Interview trap:

"Why couldn't telecom just continue with hardware boxes?"

Answer (remember this):

Because 5G needs **elastic scaling, fast deployment, and service-based communication**, which hardware appliances cannot provide.

⚠️ NFV Era (VM-Based)

- Telecom moved to:
 - VMs
 - OpenStack
 - VNFs
- Benefits:
 - Vendor neutrality
 - Faster than hardware
- Problems:
 - Heavy infrastructure
 - Slow VM boot times
 - Still semi-monolithic
 - Operational complexity

👉 Key limitation:

VM-based NFV **could not scale fast enough** for 5G use cases (IoT, URLLC, slicing).

✅ Cloud-Native Telecom (5G Era)

5G forced telecom to adopt cloud-native.

Key reasons:

- Millions of devices
- Microsecond-level latency needs
- Network slicing
- API-based control

Hence:

- Containers instead of VMs
- Kubernetes instead of OpenStack
- Microservices instead of monoliths

⭐ Your project lives here.

2 What Is a 5G Core (At Interview Depth)

The 5G Core (5GC) is NOT a single entity.

It is a collection of network functions (NFs) communicating via APIs.

Control Plane vs User Plane

Plane	Purpose
Control Plane	Signaling, authentication, session setup
User Plane	Actual user data (internet traffic)

Core Network Functions (Minimal Set You'll Use)

We intentionally choose only essential NFs to keep hardware low.

💡 AMF – Access & Mobility Management Function

- Handles:
 - UE registration
 - Mobility
 - Authentication signaling
- Control plane only

💡 SMF – Session Management Function

- Creates & manages sessions
- Decides how traffic should flow
- Talks to UPF

💡 UPF – User Plane Function

- Forwards user traffic
- Applies QoS, routing
- This is the **data plane**

💡 NRF – Network Repository Function

- Service registry
- Allows NFs to discover each other

👉 Interview gold line:

"5G Core uses a Service-Based Architecture where NFs dynamically discover each other via NRF."

3 Why Open5GS (Critical Justification)

Why Open5GS?

- Mature
- Well-documented
- Industry-recognized
- Docker-ready
- Lightweight

Why NOT Free5GC (for now)?

- More complex
- Faster-moving codebase
- Heavier resource usage

Why NOT Magma?

- Focused more on LTE / private networks
- Less pure 3GPP 5G core focus

✖ Interviewer-safe explanation:

"Open5GS provides a realistic but manageable implementation of a 5G core suitable for learning cloud-native telecom fundamentals."

4 Why Kubernetes (Even on One Machine)

This is the most asked interview question.

✖ Why Not Docker Compose Only?

- No self-healing
- No scaling
- No production parity

✓ Why Kubernetes?

- Declarative
- Auto-restart

- Networking abstraction
- What operators actually use

👉 Important clarification you'll say confidently:

"Even though I use single-node Kubernetes, the architecture is identical to multi-node production clusters."

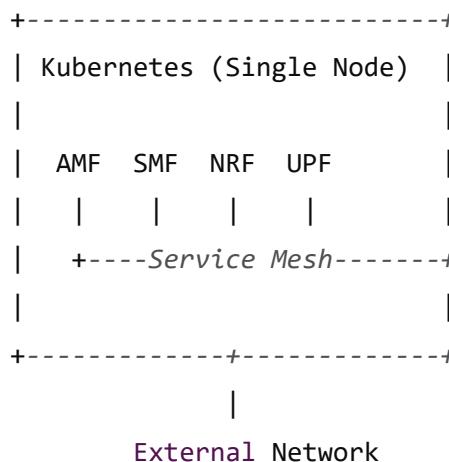
5 Your Final Learning-Safe Architecture

We are NOT building a full operator network.

We are building a **conceptually correct, interview-proof lab**.

Logical Architecture

sql



What We Intentionally Skip (SMARTLY)

- Full RAN
- Physical gNB
- Multi-site clusters
- OpenStack

Copy code

👉 Skipping ≠ ignorance

It's engineering judgment.

6 What You Will Eventually Demonstrate

By the end, you will be able to **show and explain**:

- Containerized telecom NFs
- Kubernetes-based deployment
- Control plane vs user plane
- Automation mindset
- Monitoring & reliability

This is **exactly** what Quadgen expects from a fresher.

Checkpoint (Very Important)

Before moving to actual setup, you should be able to answer:

1. Why did telecom move from hardware → VM → containers?
2. What problem does Kubernetes solve for 5G?
3. What happens if AMF crashes?
4. Why is UPF separated from control plane?
5. Why is Open5GS suitable for learning?

If you feel **70–80% confident**, we're ready.

SOON Next Step (Phase 1)

In the next message, I will:

- Choose **k3s vs kind** for your hardware
- Define **exact minimum RAM/storage**
- Decide **which Open5GS NFs to deploy**
- Start **Linux + Docker + Kubernetes setup**

Before I proceed, answer **just one thing** (no long reply needed):

 **What is your RAM size (approx)?**

(4 GB / 8 GB / 16 GB)

This helps me **optimize every next step** for your machine.

