RL for Adaptive Cloud Resource Allocation - Project Resources Guide

Problem Definition (Why this is important)

Cloud providers (AWS, GCP, Azure, Kubernetes clusters, HPC centers) struggle with allocating resources (CPU, GPU, memory, network bandwidth) fairly and efficiently.

Current Issues:

- Static allocation → over-provisioning (wastes money & energy) or under-provisioning (apps crash/slow)
- Reinforcement Learning (RL) can learn policies that dynamically adapt to workloads → auto-tuning

Research Angle: Most current solutions either focus only on throughput or only on fairness. You can propose a multi-objective RL scheduler (fairness + efficiency + energy-awareness).

Feasibility Analysis

1. Data / Workloads (✅ Available)

You don't need access to AWS/GCP internals. Open workload traces exist:

- **Google Borg traces** (ClusterData 2011, 2019) → real jobs run on Google datacenters
- **Alibaba cluster traces** (2018) → CPU/memory usage patterns
- **HPC logs** from Parallel Workloads Archive (MIT, Hebrew University)

These traces can be replayed in a simulated Kubernetes/HPC cluster environment for testing your RL agent.

2. Simulation Environment (✅ Buildable)

Options:

- Kubernetes + Minikube or SimGrid / CloudSim / OpenDC for cluster simulation
- Simpler approach: simulate workloads in Python → queue of jobs with resource demands, cluster with N nodes, RL agent allocates
- Eventually, you can deploy a prototype Kubernetes custom scheduler (written in Go or Python)

3. ML / RL Frameworks (✅ Manageable)

Libraries: Stable Baselines3, Ray RLlib, PyTorch RL

Algorithms: Start with DQN, PPO, A3C → extend to multi-agent RL (MARL) where each agent controls a cluster node

Research Novelty:

- Reward = weighted function of throughput + latency + fairness + energy cost
- Try hierarchical RL (global agent + node agents)

4. Division of Work (4 Members Team)

- **Person A:** Systems setup → Kubernetes/SimGrid + workload traces preprocessing
- **Person B:** RL model design + training pipeline (Stable Baselines3, PyTorch)
- **Person C:** Experimentation → benchmarking against baselines (First-Fit, Random, Heuristic schedulers)
- **Person D:** Results + Visualization + Research writing

This way, no one is idle and it scales well across a year.

5. Complexity Level

- **Engineering challenge:** Setting up simulation of workloads
- **Research challenge:** Designing a reward function that balances multiple objectives (throughput, fairness, energy)
- **Feasibility:** Definitely doable in 1 year, and you can push it towards publishable novelty

Resources Needed

1. Hardware Resources

You don't need industrial-scale datacenters. A modest setup is enough for simulation-based research.

Laptops/Desktops (each team member's system can be used):

- **Minimum:** 8 GB RAM, i5/Ryzen processor
- **Recommended** (for training RL models): 1 system with GPU (NVIDIA GTX 1660 or better) → if not available, you can use Google Colab / Kaggle / cloud credits

Optional: Cloud Credits Many student programs give free credits:

- Google Cloud Student Credits ($300 free)
- AWS Educate
- Microsoft Azure for Students ($100 free)

You can deploy Kubernetes clusters on cloud VMs if needed.

> **Feasibility:** Even if you don't get GPUs, you can simulate smaller workloads on CPU — RL training might be slower but still manageable.

2. Software / Tools

(a) Cluster / Workload Simulation

Option 1 (Realistic):

- Kubernetes + Minikube (run on local system for cluster simulation)
- Workload submission with KubeFlow or custom scripts

Option 2 (Lightweight simulation):

- CloudSim Plus (Java) or SimGrid (C++/Python)
- Or write a Python simulator: cluster = set of nodes, jobs = queue with CPU/memory demands

> **Suggestion:** Start with Python simulator → once RL works, deploy it as a Kubernetes custom scheduler for real-world feel.

(b) Datasets (Workload Traces)

Open-source workload traces to test your scheduler:

- **Google Cluster Data** (2011, 2019) → job CPU/memory demands
- **Alibaba Cluster Trace** (2018) → real production workloads
- **Parallel Workloads Archive (PWA)** → HPC traces (jobs, runtimes, nodes)

You can replay these traces in your simulator/Kubernetes environment.

(c) ML / RL Frameworks

- Python (3.9+)
- PyTorch or TensorFlow (prefer PyTorch for flexibility)

- Stable Baselines3 (great for standard RL algorithms like PPO, DQN, A2C)
- Ray RLlib (scales well if you want to run multi-agent RL)

(d) Visualization & Metrics

- **Grafana + Prometheus** (if you use Kubernetes, for live monitoring)
- **Matplotlib, Seaborn** → plotting RL results

Metrics you'll need:

- Resource Utilization (CPU/mem)
- Job Completion Time / Makespan
- Fairness Index (Jain's index)
- Energy Consumption (can simulate via CPU utilization × power model)

3. Human Resources (Team Roles)

- **Team Member 1 (Systems Engineer):** Sets up Kubernetes/SimGrid, manages workload traces
- **Team Member 2 (RL Developer):** Implements RL agents in PyTorch/Stable Baselines3
- **Team Member 3 (Benchmarking & Analysis):** Runs experiments, compares with heuristics (First-Fit, Round-Robin, Default K8s scheduler)
- **Team Member 4 (Research & Writing):** Analyzes results, writes paper, handles visualization/metrics

4. Time Resources (Effort Breakdown)

- **Learning curve** (2–3 months) → Kubernetes basics, RL basics
- **System Setup** (2 months) → simulator + baseline schedulers
- **RL Agent Implementation** (3–4 months)
- **Experiments & Tuning** (2 months)
- **Paper Writing & Finalization** (1–2 months)

5. Reference Material (to speed things up)

Papers to start with:

- "Deep Reinforcement Learning for Resource Scheduling in Cloud" (IEEE, 2020)
- "A Survey on Deep Reinforcement Learning for Resource Management in Cloud Computing" (ACM, 2021)

Books / Courses:

- Sutton & Barto – Reinforcement Learning: An Introduction
- Kubernetes official docs (for scheduling)
- Stable Baselines3 tutorials

Expected Deliverables

- **Simulator + RL Scheduler** (working code)
- **Comparison against baseline schedulers** (e.g., Kubernetes default scheduler)
- **Graphs & results:**
  - Resource utilization vs. job completion time
  - Fairness index (e.g., Jain's index)
  - Energy savings (optional)
- **Research paper** (~8–10 pages IEEE/ACM style)

Publication Targets (realistic for BTech level)

- IEEE ICAC (International Conference on Autonomic Computing)
- ACM Middleware / IEEE CLOUD (student track)
- Springer LNCS / Elsevier FGCS (if aiming journal level)
- Or at least IEEE Xplore student conferences in India (ICCCNT, INDICON, TENCON, etc.)

Novelty Angles You Can Explore

- Multi-Agent RL for fairness across nodes
- Carbon-aware scheduling → RL agent learns to allocate based on energy costs
- Adaptive reward shaping → RL balances short-term vs long-term gains
- Explainability in RL decisions (why a job got fewer resources)

Final Verdict on Feasibility ✅

With 4 motivated members + 1 year, this is very doable.

You'll have both:

- A working system demo (cluster + scheduler)
- Research novelty (multi-objective RL policy)

Strong enough for a paper publication + demo at project expo.

Bottom Line on Resources

- **Hardware:** A few decent laptops + maybe cloud GPU credits
- **Software:** Kubernetes/SimGrid + Python (PyTorch + RL libraries)
- **Data:** Open workload traces (Google, Alibaba, PWA)
- **People:** Divide roles smartly, each member contributes to both coding + paper