🎯 PHASED DEVELOPMENT PLAN

Distributed AI Code Review System

Goal: Build a working, demo-ready system in 2-3 weeks **Approach:** Start simple, add features incrementally, test each phase

---

📅 PHASE 1: Foundation (Weekend 1 - Days 1-2)

Goal: Get the basic infrastructure running locally **Time:** 4-6 hours

What You'll Build:

✅ Redis + PostgreSQL running in Docker
✅ Basic API Gateway that receives webhooks
✅ One worker that can process simple jobs
✅ No AI yet - just prove the infrastructure works

Step-by-Step Tasks:

Task 1.1: Setup Project Structure (30 min)

```bash
# Create folders
mkdir -p ai-code-review-system/{api-gateway,worker,shared}
cd ai-code-review-system

# Copy files from the full project:
# - docker-compose.yml
# - .env.example
# - .gitignore
```

Files needed:

- `docker-compose.yml` (simplified - just Redis + Postgres)
- `.env.example`
- `.gitignore`

Task 1.2: Setup Redis + PostgreSQL (1 hour)

```bash
```

```
# Start just infrastructure
docker-compose up redis postgres

# Verify Redis
docker exec -it code-review-redis redis-cli ping
# Should return: PONG

# Verify PostgreSQL
docker exec -it code-review-postgres psql -U postgres -c "SELECT 1"
# Should return: 1
```

Success Criteria:

- ✅ Redis responds to PING
- ✅ PostgreSQL accepts connections
- ✅ No errors in docker-compose logs

Task 1.3: Create Shared Utilities (1.5 hours)

Build only what you need:

- `shared/config.py` - Load environment variables
- `shared/redis_client.py` - Simple push/pop to queue
- `shared/database.py` - One simple table for PR analysis

Test it:

```python
# Test Redis
from shared.redis_client import redis_client
redis_client.push_job({"test": "data"})
job = redis_client.pop_job()
print(job)  # Should print: {"test": "data"}
```

Task 1.4: Build Minimal API Gateway (1.5 hours)

Create `api-gateway/app/main.py`:

- One route: `POST /webhook`
- Receives JSON payload
- Pushes to Redis queue

- Returns 202 Accepted

Test it:

```bash
# Start API Gateway
cd api-gateway
uvicorn app.main:app --reload

# Send test request
curl -X POST http://localhost:8000/webhook \
  -H "Content-Type: application/json" \
  -d '{"pr_number": 123}'

# Should return: {"message": "Queued", "pr_number": 123}
```

Task 1.5: Build Minimal Worker (1.5 hours)

Create `worker/app/worker.py`:

- Polls Redis queue
- Prints job details
- Marks job as complete in database
- No AI analysis yet!

Test it:

```bash
# Start worker
python worker/app/worker.py

# In another terminal, send webhook
curl -X POST http://localhost:8000/webhook \
  -H "Content-Type: application/json" \
  -d '{"pr_number": 123}'

# Worker should print:
# "Processing job: PR #123"
# "Job completed!"
```

✅ Phase 1 Complete When:

- [ ] Redis + PostgreSQL running in Docker
- [ ] API Gateway receives webhooks and queues jobs
- [ ] Worker picks up jobs and logs them
- [ ] No crashes, clean logs
- [ ] You can send 3 webhooks and worker processes all 3

Deliverable: Basic message queue system working end-to-end

📅 PHASE 2: Add Intelligence (Weekend 2 - Days 3-5)

Goal: Integrate Ollama for actual code analysis **Time:** 6-8 hours

What You'll Build:

✅ Ollama integration working
✅ Basic code analysis (find TODOs, console.logs)
✅ LLM generates actual review comments
✅ Results stored in database

Step-by-Step Tasks:

Task 2.1: Setup Ollama (30 min)

```bash
# Install Ollama
curl -fsSL https://ollama.com/install.sh | sh

# Download model
ollama pull codellama

# Test it works
ollama run codellama "Explain this code: def add(a,b): return a+b"
# Should return explanation
```

Task 2.2: Create Simple Code Analyzer (2 hours)

Build `worker/app/code_analyzer.py`:

- Look for console.log statements
- Find TODO/FIXME comments

- Detect long lines (>120 chars)
- Find hardcoded passwords (simple regex)

Test it:

```python
from worker.app.code_analyzer import CodeAnalyzer

analyzer = CodeAnalyzer()
test_code = """
def login():
    password = "hardcoded123"  # TODO: move to env
    console.log("Logging in")
"""

issues = analyzer.analyze_code(test_code)
print(f"Found {len(issues)} issues")
# Should find: hardcoded password, TODO, console.log
```

Task 2.3: Create LLM Analyzer (2 hours)

Build `worker/app/llm_analyzer.py`:

- Connect to Ollama
- Send code to model
- Parse response
- Return structured analysis

Test it:

```python
from worker.app.llm_analyzer import LLMAnalyzer

analyzer = LLMAnalyzer()
result = analyzer.analyze_pr(
    pr_title="Add login feature",
    changed_files=[{"filename": "auth.py", "changes": 10}]
)
print(result["summary"])
# Should print LLM's analysis
```

Task 2.4: Integrate Everything (2 hours)

Update `worker/app/worker.py`:

- Run code analyzer on files
- Send results to LLM analyzer
- Combine both analyses
- Store in PostgreSQL
- Add timing metrics

Test it:

```bash
# Send real-looking PR webhook
curl -X POST http://localhost:8000/webhook \
  -H "Content-Type: application/json" \
  -d '{
    "action": "opened",
    "number": 456,
    "pull_request": {
      "title": "Fix auth bug",
      "files": [
        {
          "filename": "auth.py",
          "patch": "+    password = \"test123\"\n+    # TODO fix"
        }
      ]
    }
  }'

# Worker should:
# 1. Find hardcoded password
# 2. Find TODO
# 3. Ask LLM for review
# 4. Store complete analysis
```

Task 2.5: Add Caching (1.5 hours)

Update Redis client:

- Cache LLM responses (expensive!)
- Key: `{repo}:{pr_number}`

- TTL: 24 hours

Test it:

```bash
# Send same PR twice
curl -X POST http://localhost:8000/webhook -d @test_pr.json
# First: Should take 2-3 seconds

sleep 1

curl -X POST http://localhost:8000/webhook -d @test_pr.json
# Second: Should take <100ms (cached!)
```

✅ Phase 2 Complete When:

- [ ] Ollama responds to requests
- [ ] Code analyzer finds issues in patches
- [ ] LLM provides actual review comments
- [ ] Complete analysis stored in database
- [ ] Caching works (second request is instant)
- [ ] You can query database and see results

Deliverable: Actual AI-powered code reviews working!

📅 PHASE 3: Scale & Polish (Week 2 - Days 6-10)

Goal: Make it production-ready and demo-worthy **Time:** 8-10 hours

What You'll Build:

✅ 3 parallel workers
✅ Proper error handling
✅ Health checks & monitoring
✅ Professional logging
✅ Complete documentation

Step-by-Step Tasks:

Task 3.1: Scale to Multiple Workers (1 hour)

Update `docker-compose.yml`:

```yaml
worker:
  # ... existing config
  deploy:
    replicas: 3  # THIS LINE!
```

Test it:

```bash
docker-compose up --scale worker=3

# Send 5 PRs at once
for i in {1..5}; do
  curl -X POST http://localhost:8000/webhook -d @test_pr.json &
done

# Should see 3 workers processing in parallel!
```

Task 3.2: Add Health Checks (1.5 hours)

Update API Gateway:

- `GET /health` - Redis + DB status
- `GET /metrics` - Queue length, processed count
- `GET /` - API info

Test it:

```bash
curl http://localhost:8000/health
# Returns: {"status": "healthy", "redis": true, "db": true}

curl http://localhost:8000/metrics
# Returns: {"queue_length": 0, "processed_today": 15}
```

Task 3.3: Improve Error Handling (2 hours)

Add to all services:

- Try/catch around all operations
- Store errors in database

- Retry logic for transient failures
- Dead letter queue for permanent failures

Test it:

```bash
# Stop Ollama
ollama stop

# Send PR
curl -X POST http://localhost:8000/webhook -d @test_pr.json

# Worker should:
# - Log error gracefully
# - Mark job as failed in DB
# - Not crash!

# Start Ollama
ollama serve

# Worker should resume processing
```

Task 3.4: Add Professional Logging (1.5 hours)

Update all services:

- Structured JSON logging
- Log levels (INFO, ERROR, DEBUG)
- Request IDs for tracing
- Performance metrics

Example log:

```json
```

```
{
  "timestamp": "2025-02-18T10:30:00Z",
  "level": "INFO",
  "service": "worker",
  "request_id": "uuid-123",
  "message": "Job completed",
  "duration_ms": 2847,
  "issues_found": 3
}
```

Task 3.5: Write Documentation (2 hours)

Create:

- `README.md` - Project overview, features, architecture
- `QUICKSTART.md` - Setup instructions
- `ARCHITECTURE.md` - System design, data flow
- Inline code comments
- API documentation (FastAPI auto-generates this!)

Task 3.6: Create Test Suite (2 hours)

Build `tests/`:

- Unit tests for analyzers
- Integration tests for API
- Load tests for workers

```bash
# Run tests
pytest tests/

# Load test
python tests/load_test.py --prs 50
# Should handle 50 PRs smoothly
```

✅ Phase 3 Complete When:

☐ 3 workers processing in parallel

☐ Health checks return accurate status

- [ ] Errors logged but don't crash system
- [ ] Professional README on GitHub
- [ ] Tests passing
- [ ] You can explain every component

Deliverable: Production-ready system!

🗓️ PHASE 4: Demo Prep (Week 3 - Days 11-14)

Goal: Make it impressive for interviews **Time:** 4-6 hours

What You'll Build:

✅ Architecture diagram
✅ Demo video/GIF
✅ Performance benchmarks
✅ GitHub polish
✅ Interview talking points

Step-by-Step Tasks:

Task 4.1: Create Architecture Diagram (1 hour)

Use draw.io or mermaid:

```mermaid
graph LR
    A[GitHub Webhook] --> B[API Gateway]
    B --> C[Redis Queue]
    C --> D[Worker 1]
    C --> E[Worker 2]
    C --> F[Worker 3]
    D --> G[Ollama LLM]
    E --> G
    F --> G
    D --> H[PostgreSQL]
    E --> H
    F --> H
    D --> I[Redis Cache]
    E --> I
    F --> I
```

Task 4.2: Record Demo (1.5 hours)

Create a screencast showing:

1. Starting the system: `docker-compose up`
2. Sending a PR webhook
3. Workers processing in parallel
4. Viewing results in database
5. Showing cached response (instant!)

Tools: OBS Studio (free) or QuickTime

Task 4.3: Run Benchmarks (1 hour)

Test and document:

```bash
# Single PR analysis time
time curl -X POST http://localhost:8000/webhook -d @test_pr.json
# Record: 2.8s

# Cached analysis time
# Record: 0.1s

# Concurrent handling
# Send 20 PRs simultaneously
# Record: All processed in 8s (2.4 PRs/sec)

# Queue capacity
# Queue 100 PRs, measure completion time
# Record: 95s (1.05 PRs/sec sustained)
```

Task 4.4: Polish GitHub Repo (1.5 hours)

- Add badges (Docker, Python, FastAPI)
- Create demo GIF
- Add architecture diagram
- Write detailed setup instructions
- Add "Tech Stack" section
- Create sample webhook payloads in `examples/`
- Add MIT license

- Write contributing guide

Task 4.5: Prepare Interview Answers (1 hour)

Write answers to:

- "Walk me through the architecture"
- "How does the system scale?"
- "What happens if a worker crashes?"
- "How did you optimize for performance?"
- "What would you do differently?"
- "How would you handle 1000x load?"

✅ Phase 4 Complete When:

- [ ] GitHub repo looks professional
- [ ] Demo GIF shows system working
- [ ] Benchmarks documented in README
- [ ] You can explain design decisions confidently
- [ ] Repo has 10+ stars (share with friends!)

Deliverable: Interview-ready portfolio project!

📊 OVERALL TIMELINE

Week 1: Foundation + Intelligence

- **Days 1-2:** Phase 1 (4-6 hours)
- **Days 3-5:** Phase 2 (6-8 hours)
- **Weekend:** Buffer/testing

Week 2: Production Ready

- **Days 6-10:** Phase 3 (8-10 hours)
- **Weekend:** Testing & polish

Week 3: Demo & Deploy

- **Days 11-14:** Phase 4 (4-6 hours)

- **Day 15:** Deploy to GitHub, update resume

Total Time: 22-30 hours over 2-3 weeks

🎯 MINIMUM VIABLE PROJECT

If you're short on time, you can stop after Phase 2!

That gives you: ✅ Working distributed system
✅ Real AI analysis
✅ Database persistence
✅ Caching for performance
✅ Docker deployment

This alone is interview-worthy!

Phases 3-4 add polish, but Phase 2 is the core.

📝 DAILY COMMIT STRATEGY

Make this look like real development:

Day 1:

```
git commit -m "Initial project structure"
git commit -m "Add docker-compose for Redis and PostgreSQL"
git commit -m "Configure environment variables"
```

Day 2:

```
git commit -m "Implement Redis queue client"
git commit -m "Create basic API Gateway webhook receiver"
git commit -m "Add worker polling loop"
```

Day 3:

```
git commit -m "Integrate Ollama for LLM analysis"
git commit -m "Add basic code analyzer for pattern detection"
```

Never commit everything at once!

🚨 CRITICAL SUCCESS FACTORS

1. **Test each phase before moving on**
    - Don't add features until current ones work
    - Use curl to test APIs
    - Check logs frequently
2. **Commit often**
    - Real projects have 50+ commits
    - Show your development process
3. **Document as you go**
    - Write README sections after each phase
    - Comment your code while writing it
4. **Ask for help when stuck**
    - Come back to this chat
    - Google specific errors
    - Check Ollama/FastAPI docs
5. **Don't skip Phase 1!**
    - It's the foundation
    - Everything else builds on it

💡 TIPS FOR SUCCESS

When You Get Stuck:

1. Read the error message carefully
2. Check Docker logs: `docker-compose logs worker`
3. Verify services are running: `docker-compose ps`
4. Test components individually
5. Come back here for help!

Stay Motivated:

- Set a timer (Pomodoro: 25 min work, 5 min break)
- Celebrate each phase completion

- Show friends your progress
- Remember: This gets you the Cisco job!

Track Progress:

- [ ] Phase 1: Foundation ⏳
- [ ] Phase 2: Intelligence ⏳
- [ ] Phase 3: Production ⏳
- [ ] Phase 4: Demo Ready ⏳

🎉 READY TO START?

Begin with Phase 1, Task 1.1

1. Create the project folder
2. Set up Docker Compose
3. Get Redis + PostgreSQL running

First milestone: See "PONG" from Redis!

Come back when you complete Phase 1 or get stuck.

You got this! 🚀