We are starting implementation of the **Open Grant Network Community Compute** system.

Below is a list of tasks grouped by milestone.

How to claim a task:
• Reply in this chat with the task ID (for example: CC-03 claimed)
• Only claim tasks you have time to complete.
• When you finish, reply with CC-03 done and open a PR against the develop branch on GitHub.

All work for this feature should target the **develop branch** of the open-grant-network repo unless the task says otherwise.

Need access to our GitHub?

https://docs.google.com/forms/d/e/1FAIpQLSchvPz9gbv7xkkt0AkR17myvdv7704mM-sWI_yU6lBgfmlRkQ/viewform?usp=publish-editor

Ensure your NDA is complete:

https://ecoservantsproject.org/nda

2) Claimable tasks for Community Compute

Milestone 1 – Database and schema

[CC-01] Create community_nodes table

- Add community_nodes to dev/schema.sql with the columns from the plan
- Include node_public_id, api_token, consent_hash, consent_version, user_id, timestamps and is_active
- Make sure the migration is compatible with existing tables

[CC-02] Create community_node_sessions table

- Extend dev/schema.sql with community_node_sessions
- Include session_token, node_id, created_at, last_seen_at, user_agent, ip_hash, is_active
- Add foreign key back to community_nodes

[CC-03] Create community_jobs table

- Add community_jobs table with job_type, payload_json, status fields and timestamps
- Include claimed_by_node_id and foreign key to community_nodes

[CC-04] Create community_job_results table

- Add community_job_results table with job_id, node_id, result_json, result_checksum, timestamps
- Add foreign keys to community_jobs and community_nodes

[CC-05] Convert allow-list CSV into SQL schema

- Design one or more tables that replace Allow_List_Policy_v1.csv and robots_audit_summary.csv
- Add these table definitions to dev/schema.sql so the orchestrator can query allowed domains directly

---

Milestone 2 – Compute backend API (FastAPI)

These live next to phase2_api.py in dev/phase2.

[CC-10] Implement /compute/register-node endpoint

- Add endpoint to the FastAPI app
- Accept node_public_id and optional wp_user_id or nonce
- Create or look up community_nodes row, generate api_token and return it
- Return latest consent version and consent text URL

[CC-11] Implement /compute/consent endpoint

- Add endpoint to record consent from a node
- Accept node_public_id, api_token, consent_version, consent_hash
- Update community_nodes and set is_active = 1, clear opt_out_at

[CC-12] Implement /compute/opt-out endpoint

- Add endpoint that marks a node as opted out
- Set is_active = 0 and opt_out_at = now

- Ensure opted out nodes are excluded from job assignment queries

[CC-13] Implement /compute/job endpoint (job fetch)

- Auth via api_token
- Query for the next pending job that matches any constraints you add (job type filters later)
- Lock it as claimed with claimed_by_node_id and claimed_at
- Return a minimal JSON payload for the client to run

[CC-14] Implement /compute/job-result endpoint

- Accept job_id, result_json, optional result_checksum, and result status
- Insert into community_job_results
- Update community_jobs.status to done or failed with timestamps

[CC-15] Add basic rate limiting per node

- Add simple token bucket style or minimum time gap per node in the /compute/job logic
- Prevent a node from claiming more than N jobs per minute

Milestone 3 – Reference Node Client (Python CLI)

Create a simple CLI client in dev/clients/community_node_cli/ or similar.

[CC-20] Bootstrap Python node client skeleton

- Set up a small Python package that can:
    - Load config file (API URL, node_public_id, api_token)
    - Make authenticated requests to the compute API
- Include basic logging

[CC-21] Implement node registration flow

- On first run, generate a UUID as node_public_id
- Call /compute/register-node
- Store the returned api_token and consent version in a local config file

[CC-22] Implement consent flow in the client

- Add a simple prompt that shows the consent version and URL
- After user confirms, call /compute/consent
- Save that the node has consented so it does not prompt every time

[CC-23] Implement job polling loop

- Implement a loop that:
    - Calls /compute/job
    - If a job is returned, executes it via a local handler
    - Calls /compute/job-result with outcome
    - Sleeps between polls to respect rate limits

[CC-24] Implement local safety limits

- Add local enforcement for:
    - Max response size
    - Per request timeout
- Ensure the client fails fast and reports failed if safe limits are exceeded

---

Milestone 4 – WordPress bridge endpoints

Expose a basic UX for WP-authenticated users. This work happens in your WP plugin / theme code, not the Python repo.

[CC-30] Create /wp-json/esgt/v1/community-node/enable endpoint

- Authenticated WP REST endpoint
- Calls the Python /compute/register-node and /compute/consent indirectly (or writes to DB if shared)
- Stores node_public_id and api_token in user meta or secure storage

[CC-31] Create /wp-json/esgt/v1/community-node/disable endpoint

- Authenticated WP REST endpoint
- Calls /compute/opt-out

- Clears or flags the stored node credentials

[CC-32] Extend /wp-json/esgt/v1/status to include basic compute stats

- Add total active nodes, total jobs done, maybe last 24 hours jobs count
- Use this on /about-bot and related pages

[CC-33] Simple WP admin screen for node activity

- Add a minimal admin page that calls WP REST endpoints and shows:
    - Whether the current user has a node linked
    - Last few jobs associated with their node

Milestone 5 – First job types

Use your existing robots audit scripts and grant URLs.

[CC-40] Implement job type robots_recheck

- Add a job creator script that reads from the robots audit tables and creates community_jobs entries for robots_recheck
- Define the payload schema for this job type
- Implement server-side validator that ensures each job URL is a robots.txt URL from the allow-list

[CC-41] Implement node handler for robots_recheck

- In the Python node client, add handler code to:
    - Fetch robots.txt for the given domain
    - Record status and basic metadata
    - Return structured JSON to /compute/job-result

[CC-42] Implement job type link_health

- Add a script that generates link_health jobs for URLs from grants and organizations that are in the allow-list
- Payload should include url and expected_status

[CC-43] Implement node handler for link_health

- In the node client, implement small GET with timeout and limited redirects
- Return status code and simple flags like is_reachable, redirected

[CC-44] Implement server-side result aggregator for link health

- Add a routine that reads from community_job_results for link_health
- Updates grants or organizations with a last_verified and simple health flag

---

Milestone 6 – Policy, logging and docs

[CC-50] Migrate Allow_List_Policy_v1.csv into DB and loader script

- Write a script to load the CSV into the new allow-list tables
- Wire job generators to use the DB instead of CSV directly

[CC-51] Extend Ethical_Crawling_Guidelines for community nodes

- Update docs/Ethical_Crawling_Guidelines.md to include a clear section for Community Compute
- Cover what nodes will and will not do in concrete language

[CC-52] Write Community_Node_Developer_Guide.md

- New doc under docs/ or docs/phase2 with:
    - How to run the reference node client
    - How to add a new job type
    - Safety checklist for any new job type

[CC-53] Add basic monitoring metrics

- Add logging or metrics hooks in the Python API for:
    - Jobs created, claimed, completed, failed
    - Active nodes and last seen times
- Start with simple logs or a small metrics table