Configuration

Broadcast Groups

Status: Experimental
Version: Added in 2026.1.9

Overview

Broadcast Groups enable multiple agents to process and respond to the same message simultaneously. This allows you to create specialized agent teams that work together in a single WhatsApp group or DM — all using one phone number.

Current scope: **WhatsApp only** (web channel).

Broadcast groups are evaluated after channel allowlists and group activation rules. In WhatsApp groups, this means broadcasts happen when OpenClaw would normally reply (for example: on mention, depending on your group settings).

Use Cases

1. Specialized Agent Teams

Deploy multiple agents with atomic, focused responsibilities:

```
Group: "Development Team"
Agents:
  - CodeReviewer (reviews code snippets)
  - DocumentationBot (generates docs)
  - SecurityAuditor (checks for vulnerabilities)
  - TestGenerator (suggests test cases)
```

Each agent processes the same message and provides its specialized perspective.

2. Multi-Language Support

```
Group: "International Support"
Agents:
  - Agent_EN (responds in English)
  - Agent_DE (responds in German)
  - Agent_ES (responds in Spanish)
```

3. Quality Assurance Workflows

```
Group: "Customer Support"
Agents:
  - SupportAgent (provides answer)
  - QAAgent (reviews quality, only responds if issues found)
```

4. Task Automation

```
Group: "Project Management"
Agents:
  - TaskTracker (updates task database)
  - TimeLogger (logs time spent)
  - ReportGenerator (creates summaries)
```

Configuration

Basic Setup

Add a top-level `broadcast` section (next to `bindings`). Keys are WhatsApp peer ids:

- group chats: group JID (e.g. `120363403215116621@g.us`)
- DMs: E.164 phone number (e.g. `+15551234567`)

```
{
  "broadcast": {
    "120363403215116621@g.us": ["alfred", "baerbel", "assistant3"]
  }
}
```

Result: When OpenClaw would reply in this chat, it will run all three agents.

Processing Strategy

Control how agents process messages:

Parallel (Default)

All agents process simultaneously:

```
{
  "broadcast": {
    "strategy": "parallel",
    "120363403215116621@g.us": ["alfred", "baerbel"]
  }
}
```

Sequential

Agents process in order (one waits for previous to finish):

```
{
  "broadcast": {
    "strategy": "sequential",
    "120363403215116621@g.us": ["alfred", "baerbel"]
  }
}
```

Complete Example

```
{
  "agents": {
    "list": [
      {
        "id": "code-reviewer",
        "name": "Code Reviewer",
        "workspace": "/path/to/code-reviewer",
        "sandbox": { "mode": "all" }
      },
      {
        "id": "security-auditor",
        "name": "Security Auditor",
        "workspace": "/path/to/security-auditor",
        "sandbox": { "mode": "all" }
      },
      {
        "id": "docs-generator",
        "name": "Documentation Generator",
        "workspace": "/path/to/docs-generator",
        "sandbox": { "mode": "all" }
      }
    ]
  },
  "broadcast": {
    "strategy": "parallel",
    "120363403215116621@g.us": ["code-reviewer", "security-auditor", "docs-genera
    "120363424282127706@g.us": ["support-en", "support-de"],
    "+15555550123": ["assistant", "logger"]
  }
}
```

How It Works

Message Flow

1. **Incoming message** arrives in a WhatsApp group
2. **Broadcast check**: System checks if peer ID is in `broadcast`

3. **If in broadcast list**:
   - All listed agents process the message
   - Each agent has its own session key and isolated context
   - Agents process in parallel (default) or sequentially
4. **If not in broadcast list**:
   - Normal routing applies (first matching binding)

Note: broadcast groups do not bypass channel allowlists or group activation rules (mentions/commands/etc). They only change *which agents run* when a message is eligible for processing.

Session Isolation

Each agent in a broadcast group maintains completely separate:

- **Session keys** (`agent:alfred:whatsapp:group:120363...` vs `agent:baerbel:whatsapp:group:120363...`)
- **Conversation history** (agent doesn't see other agents' messages)
- **Workspace** (separate sandboxes if configured)
- **Tool access** (different allow/deny lists)
- **Memory/context** (separate IDENTITY.md, SOUL.md, etc.)
- **Group context buffer** (recent group messages used for context) is shared per peer, so all broadcast agents see the same context when triggered

This allows each agent to have:

- Different personalities
- Different tool access (e.g., read-only vs. read-write)
- Different models (e.g., opus vs. sonnet)
- Different skills installed

Example: Isolated Sessions

In group `120363403215116621@g.us` with agents `["alfred", "baerbel"]`:

Alfred's context:

```
Session: agent:alfred:whatsapp:group:120363403215116621@g.us
History: [user message, alfred's previous responses]
Workspace: /Users/pascal/openclaw-alfred/
Tools: read, write, exec
```

Bärbel's context:

```
Session: agent:baerbel:whatsapp:group:120363403215116621@g.us
History: [user message, baerbel's previous responses]
Workspace: /Users/pascal/openclaw-baerbel/
Tools: read only
```

Best Practices

1. Keep Agents Focused

Design each agent with a single, clear responsibility:

```
{
  "broadcast": {
    "DEV_GROUP": ["formatter", "linter", "tester"]
  }
}
```

✅ **Good:** Each agent has one job

❌ **Bad:** One generic "dev-helper" agent

2. Use Descriptive Names

Make it clear what each agent does:

```
{
  "agents": {
    "security-scanner": { "name": "Security Scanner" },
    "code-formatter": { "name": "Code Formatter" },
    "test-generator": { "name": "Test Generator" }
  }
}
```

3. Configure Different Tool Access

Give agents only the tools they need:

```
{
  "agents": {
    "reviewer": {
      "tools": { "allow": ["read", "exec"] } // Read-only
    },
    "fixer": {
      "tools": { "allow": ["read", "write", "edit", "exec"] } // Read-write
    }
  }
}
```

4. Monitor Performance

With many agents, consider:

- Using `"strategy": "parallel"` (default) for speed
- Limiting broadcast groups to 5-10 agents
- Using faster models for simpler agents

5. Handle Failures Gracefully

Agents fail independently. One agent's error doesn't block others:

```
Message ⟶ [Agent A ✓, Agent B ✗ error, Agent C ✓]
Result: Agent A and C respond, Agent B logs error
```

Compatibility

Providers

Broadcast groups currently work with:

- ✅ WhatsApp (implemented)
- 🚧 Telegram (planned)
- 🚧 Discord (planned)
- 🚧 Slack (planned)

Routing

Broadcast groups work alongside existing routing:

```
{
  "bindings": [
    {
      "match": { "channel": "whatsapp", "peer": { "kind": "group", "id": "GROUP_A
      "agentId": "alfred"
    }
  ],
  "broadcast": {
    "GROUP_B": ["agent1", "agent2"]
  }
}
```

`GROUP_A` : Only alfred responds (normal routing)

`GROUP_B` : agent1 AND agent2 respond (broadcast)

Precedence: `broadcast` takes priority over `bindings`.

Troubleshooting

Agents Not Responding

Check:

1. Agent IDs exist in `agents.list`
2. Peer ID format is correct (e.g., `120363403215116621@g.us`)
3. Agents are not in deny lists

Debug:

```
tail -f ~/.openclaw/logs/gateway.log | grep broadcast
```

Only One Agent Responding

Cause: Peer ID might be in `bindings` but not `broadcast`.

Fix: Add to broadcast config or remove from bindings.

Performance Issues

If slow with many agents:

Reduce number of agents per group

Use lighter models (sonnet instead of opus)

Check sandbox startup time

Examples

Example 1: Code Review Team

```
{
  "broadcast": {
    "strategy": "parallel",
    "120363403215116621@g.us": [
      "code-formatter",
      "security-scanner",
      "test-coverage",
      "docs-checker"
    ]
  },
  "agents": {
    "list": [
      {
        "id": "code-formatter",
        "workspace": "~/agents/formatter",
        "tools": { "allow": ["read", "write"] }
      },
      {
        "id": "security-scanner",
        "workspace": "~/agents/security",
        "tools": { "allow": ["read", "exec"] }
      },
      {
        "id": "test-coverage",
        "workspace": "~/agents/testing",
        "tools": { "allow": ["read", "exec"] }
      },
      { "id": "docs-checker", "workspace": "~/agents/docs", "tools": { "allow": [
    ]
  }
}
```

User sends: Code snippet

Responses:

code-formatter: “Fixed indentation and added type hints”

security-scanner: “⚠ SQL injection vulnerability in line 12”

test-coverage: “Coverage is 45%, missing tests for error cases”

docs-checker: “Missing docstring for function `process_data`”

›

Example 2: Multi-Language Support

```
{
  "broadcast": {
    "strategy": "sequential",
    "+15555550123": ["detect-language", "translator-en", "translator-de"]
  },
  "agents": {
    "list": [
      { "id": "detect-language", "workspace": "~/agents/lang-detect" },
      { "id": "translator-en", "workspace": "~/agents/translate-en" },
      { "id": "translator-de", "workspace": "~/agents/translate-de" }
    ]
  }
}
```

API Reference

Config Schema

```
interface OpenClawConfig {
  broadcast?: {
    strategy?: "parallel" | "sequential";
    [peerId: string]: string[];
  };
}
```

Fields

`strategy` (optional): How to process agents

`"parallel"` (default): All agents process simultaneously

`"sequential"` : Agents process in array order

`[peerId]` : WhatsApp group JID, E.164 number, or other peer ID

Value: Array of agent IDs that should process messages

Limitations

1. **Max agents:** No hard limit, but 10+ agents may be slow
2. **Shared context:** Agents don't see each other's responses (by design)
3. **Message ordering:** Parallel responses may arrive in any order
4. **Rate limits:** All agents count toward WhatsApp rate limits

Future Enhancements

Planned features:

- [ ] Shared context mode (agents see each other's responses)
- [ ] Agent coordination (agents can signal each other)
- [ ] Dynamic agent selection (choose agents based on message content)
- [ ] Agent priorities (some agents respond before others)

See Also

Multi-Agent Configuration

Routing Configuration

Session Management

‹ Groups Channel Routing ›