

```
import asyncio
import json
import logging
import os
import signal
```

```
from websockets.asyncio.client import connect
from websockets.exceptions import ConnectionClosed
```

```
log = logging.getLogger("orchestrator")
```

```
class Orchestrator:
```

```
    def __init__(self, urls, token=None, subscribe=None, workers=4):
        self.urls = urls
        self.headers = {"Authorization": f"Bearer {token}"} if token else {}
        self.subscribe = subscribe
        self.workers = workers
        self.queue = asyncio.Queue(maxsize=10000)
        self.handlers = {}
        self.stop = None
```

```
    def on(self, event_type):
```

```
        def register(func):
            self.handlers.setdefault(event_type, []).append(func)
            return func
        return register
```

```
    async def listen(self, url):
```

```
        try:
            async for ws in connect(url, additional_headers=self.headers, ping_interval=20,
ping_timeout=20):
                log.info("connected to %s", url)
                try:
                    if self.subscribe:
                        await ws.send(self.subscribe)
                    async for raw in ws:
                        await self.queue.put((url, raw))
                except ConnectionClosed as e:
                    log.warning("connection to %s closed (%s), reconnecting", url, e)
            except asyncio.CancelledError:
                raise
        except Exception:
            log.exception("listener for %s gave up", url)
```

```

        self.stop.set()

    async def work(self):
        while True:
            url, raw = await self.queue.get()
            try:
                await self.dispatch(url, raw)
            except Exception:
                log.exception("handler failed for message from %s", url)
            finally:
                self.queue.task_done()

    async def dispatch(self, url, raw):
        try:
            msg = json.loads(raw)
        except ValueError:
            log.warning("non-JSON message from %s: %r", url, raw[:200])
            return
        if not isinstance(msg, dict):
            msg = {"type": "unknown", "data": msg}
        event_type = msg.get("type") or msg.get("eventType") or "unknown"
        for handler in self.handlers.get(event_type, self.handlers.get("*", [])):
            result = handler(msg)
            if asyncio.iscoroutine(result):
                await result

    async def run(self):
        self.stop = asyncio.Event()
        loop = asyncio.get_running_loop()
        for sig in (signal.SIGINT, signal.SIGTERM):
            try:
                loop.add_signal_handler(sig, self.stop.set)
            except NotImplementedError:
                pass

        listeners = [asyncio.create_task(self.listen(u)) for u in self.urls]
        workers = [asyncio.create_task(self.work()) for _ in range(self.workers)]
        await self.stop.wait()

        log.info("shutting down")
        for t in listeners:
            t.cancel()
        try:
            await asyncio.wait_for(self.queue.join(), timeout=10)

```

```

except asyncio.TimeoutError:
    log.warning("shutdown with %d unprocessed messages", self.queue.qsize())
for t in workers:
    t.cancel()
await asyncio.gather(*listeners, *workers, return_exceptions=True)

def build(urls, token, subscribe):
    orch = Orchestrator(urls, token, subscribe)

    @orch.on("alarm")
    async def handle_alarm(msg):
        log.warning("ALARM id=%s severity=%s source=%s", msg.get("id"), msg.get("severity"),
msg.get("source"))

    @orch.on("event")
    async def handle_event(msg):
        log.info("EVENT id=%s name=%s", msg.get("id"), msg.get("name"))

    @orch.on("")
    async def handle_other(msg):
        log.info("unhandled %s: %s", msg.get("type") or msg.get("eventType"),
json.dumps(msg)[:300])

    return orch

def main():
    logging.basicConfig(level=os.environ.get("LOG_LEVEL", "INFO"),
                        format="%(asctime)s %(levelname)s %(name)s %(message)s")
    urls = [u.strip() for u in os.environ.get("ERICSSON_WS_URLS", "").split(",") if u.strip()]
    if not urls:
        raise SystemExit("set ERICSSON_WS_URLS to one or more comma-separated ws:// or
wss:// URLs")
    orch = build(urls, os.environ.get("ERICSSON_API_TOKEN"),
os.environ.get("ERICSSON_WS_SUBSCRIBE"))
    try:
        asyncio.run(orch.run())
    except KeyboardInterrupt:
        pass

if __name__ == "__main__":
    main()

```

