```
import io
import json
import os
import logging
from datetime import datetime
import requests
from requests.auth import HTTPBasicAuth

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)

ORDS_BASE = os.getenv("ORDS_BASE")
ORDS_USER = os.getenv("ORDS_USER")
ORDS_PWD = os.getenv("ORDS_PWD")
ORDS_TABLE = os.getenv("ORDS_TABLE")

def handler(ctx, data: io.BytesIO = None):
    # ---- config validation ----
    missing = [k for k, v in {
        "ORDS_BASE": ORDS_BASE,
        "ORDS_USER": ORDS_USER,
        "ORDS_PWD": ORDS_PWD,
        "ORDS_TABLE": ORDS_TABLE
    }.items() if not v]
    if missing:
        raise ValueError(f"Missing env vars: {',
'.join(missing)}")
    def safe_str(v, maxlen=1500):
        if v is None:
            return "None"
        s = str(v)
        return s if len(s) <= maxlen else s[:maxlen] +
"...(truncated)"
    def normalise_ts_for_ords(ts: str) -> str:
        if not ts:
            return datetime.utcnow().strftime('%Y-%m-
%dT%H:%M:%S') + "+00:00"
        s = ts.strip()
        # Z -> +00:00
```

```
        if s.endswith('Z'):
            return s[:-1] + "+00:00"
        # already has an offset?
        if '+' in s[10:] or s[10:].count('-') > 0:
            return s
        # no offset: append UTC
        return s + "+00:00"

    # ---- parse payload ----
    try:
        payload = json.loads(data.getvalue()) if data else
{}
    except Exception as e:
        logger.info("ERROR parsing JSON payload: %s",
safe_str(e))
        raise

    data_section = payload.get("data", {}) or {}
    add = data_section.get("additionalDetails", {}) or {}

    # ---- build row ----
    # NOTE on ID:
    # If your DB column ID is NUMBER, do NOT send UUID.
Remove "id" below and let DB generate it.
    # If your ID is VARCHAR2, UUID is fine.
    row = {
        "id": payload.get("eventID", ""),  # use eventID
        "event_time":
normalise_ts_for_ords(payload.get("eventTime", "")),
        "event_type": payload.get("eventType", "") or "n/a",
        "resource_name": data_section.get("resourceName") or
"n/a",
        "request_type": add.get("requestType", "") or "n/a",
        "status": add.get("status", "") or "n/a",
        "compartment_id": data_section.get("compartmentId")
or "n/a",
        "source": payload.get("source", "") or "n/a",
    }
```

```
    # ---- insert ----
    url =
f"{ORDS_BASE.rstrip('/')}/{ORDS_TABLE.strip().strip('/')}/"
    headers = {"Content-Type": "application/json", "Accept":
"application/json"}

    try:
        resp = requests.post(
            url,
            json=row,
            headers=headers,
            auth=HTTPBasicAuth(ORDS_USER, ORDS_PWD),
            timeout=15,
            allow_redirects=False
        )
        logger.info("POST status=%s content-type=%s
location=%s",
                    resp.status_code,
resp.headers.get("content-type"),
resp.headers.get("location"))
        if resp.status_code in (200, 201):
            logger.info("Insert OK")
            return {"ok": True, "status": resp.status_code,
"id": row.get("id")}
        # Log error body (JSON if possible; otherwise text)
        try:
            logger.info("ORDS error body(json)=%s",
safe_str(resp.json()))
        except Exception:
            logger.info("ORDS error body(text first
2000)=%s", safe_str((resp.text or ""), 2000))
        return {"ok": False, "status": resp.status_code,
"id": row.get("id")}
    except requests.exceptions.Timeout:
        logger.info("POST timeout after 15s")
        return {"ok": False, "error": "TIMEOUT", "id":
row.get("id")}
    except Exception as exc:
```

```
        logger.info("POST exception: %s: %s",
type(exc).__name__, safe_str(exc))
        return {"ok": False, "error":
f"{type(exc).__name__}: {str(exc)[:200]}", "id":
row.get("id")}
```