Telegram → n8n → AWS Lambda: EC2 Start/Stop/Status Bot – Step■by■Step Guide

This document walks you through building a Telegram bot that starts, stops, and checks the status of an EC2 instance using n8n and AWS Lambda. It includes IAM, Lambda code, n8n workflow wiring, and troubleshooting.

1) Architecture

```
Telegram Bot
■ (getUpdates)
▼
n8n (Docker)
■■ HTTP Request → Telegram getUpdates (with offset)
■■ Code (Python Beta) → parse /start | /stop | /status → build payload
■■ IF (payload exists?)
■ ■■ TRUE → Edit Fields (Set) → AWS Lambda (Invoke) → Merge → Telegram Send
(confirmation)
■ ■■ FALSE → Telegram Send (“Use /start|/stop|/status [instance-id]”)
■■ Schedule Trigger polls on an interval
```

2) Prerequisites

• AWS Account in ap-south-1 (Mumbai) • Docker Desktop • Telegram Bot token • EC2 Instance ID • n8n latest image.

3) Run n8n (Docker)

```
services:
n8n:
image: n8nio/n8n:latest
ports:
- "5678:5678"
environment:
- N8N_HOST=localhost
- N8N_PORT=5678
- N8N_PROTOCOL=http
- GENERIC_TIMEZONE=Asia/Kolkata
volumes:
- ./n8n_data:/home/node/.n8n

# Start:
docker compose up -d
Open http://localhost:5678 and create an account.
```

4) Telegram Setup

1. Create a bot with @BotFather → copy BOT_TOKEN.
2. Send any message to your bot so it has a chat.
3. Find chatId (optional check):
GET https://api.telegram.org/bot/getUpdates

5) AWS Credentials & IAM

Create a dedicated IAM user for n8n to invoke Lambda (not root). Attach this policy:

```
{
"Version": "2012-10-17",
"Statement": [
{"Effect":"Allow","Action":"lambda:InvokeFunction",
"Resource":"arn:aws:lambda:ap-south-1::function:RustDeskSchedulerStack-Ec2StartStopFn*"}
]
}
```

Create an **execution role** for the Lambda with EC2 permissions:

```
{
"Version": "2012-10-17",
"Statement": [
{"Effect":"Allow","Action":["ec2:DescribeInstances"],"Resource":"*"},
{"Effect":"Allow","Action":["ec2:StartInstances","ec2:StopInstances"],"Resource":"*"}
]
}
```

6) Lambda Function (Python 3.12)

Filename: app.py • Handler: app.lambda_handler • Region: ap-south-1

```
import json, os, boto3
from botocore.exceptions import ClientError

ec2 = boto3.client("ec2")
DEFAULT_INSTANCE_ID = os.getenv("DEFAULT_INSTANCE_ID") # optional

def _get_state(instance_id: str) -> str:
r = ec2.describe_instances(InstanceIds=[instance_id])
return r["Reservations"][0]["Instances"][0]["State"]["Name"]

def _start(instance_id: str):
state = _get_state(instance_id)
if state == "stopped":
ec2.start_instances(InstanceIds=[instance_id])
return {"ok": True, "state": "pending", "msg": "start requested"}
if state in ("running", "pending"):
return {"ok": True, "state": state, "msg": "already running/starting"}
return {"ok": False, "state": state, "msg": f"cannot start from state '{state}'"}

def _stop(instance_id: str):
state = _get_state(instance_id)
if state == "running":
ec2.stop_instances(InstanceIds=[instance_id])
return {"ok": True, "state": "stopping", "msg": "stop requested"}
if state in ("stopped", "stopping"):
return {"ok": True, "state": state, "msg": "already stopped/stopping"}
return {"ok": False, "state": state, "msg": f"cannot stop from state '{state}'"}

def _status(instance_id: str):
return {"ok": True, "state": _get_state(instance_id), "msg": "status"}

def lambda_handler(event, _ctx):
action = (event or {}).get("action")
```

```
instance_id = (event or {}).get("instance_id") or DEFAULT_INSTANCE_ID
if not action or not instance_id:
return {"ok": False, "msg": "missing action or instance_id", "event": event}
try:
if action == "start": return _start(instance_id)
if action == "stop": return _stop(instance_id)
if action == "status": return _status(instance_id)
return {"ok": False, "msg": f"unknown action '{action}'"}
except ClientError as e:
return {"ok": False, "msg": str(e), "action": action, "instance_id": instance_id}
```

7) Lambda Test Events

```
Start:
{"action": "start", "instance_id": "i-09c40c2e332da8c67"}

Stop:
{"action": "stop", "instance_id": "i-09c40c2e332da8c67"}

Status:
{"action": "status", "instance_id": "i-09c40c2e332da8c67"}
```

8) Build the n8n Workflow

8.1 Schedule Trigger: run every 5–10 seconds.

8.2 HTTP Request (getUpdates): URL must be an expression to fetch only NEW messages:

```
https://api.telegram.org/bot{{$env.TELEGRAM_BOT_TOKEN}}/getUpdates?timeout=10&allowed;_upd
ates=["message"]&offset;={{ $getWorkflowStaticData('global').lastUpdateId ?
($getWorkflowStaticData('global').lastUpdateId + 1) : 0 }}
```

8.3 Code in Python (Beta): parse /start | /stop | /status and update lastUpdateId

```
from typing import List, Dict
import re

DEFAULT_INSTANCE_ID = "i-09c40c2e332da8c67"
INSTANCE_MAP = {}
INSTANCE_RE = re.compile(r"^i-[0-9a-f]{8,17}$")

s = getWorkflowStaticData("global")
last = int(s.get("lastUpdateId", 0) or 0)
root = items[0].get("json", {})
updates = root.get("result", [])
out: List[Dict] = []

for upd in updates:
uid = int(upd.get("update_id", 0) or 0)
if uid <= last: continue
last = uid
msg = upd.get("message") or {}
text = (msg.get("text") or "").strip()
if not text: continue
chat_id = msg.get("chat", {}).get("id")
```

```
parts = text.split()
cmd = parts[0].lstrip("/").lower()
arg = parts[1] if len(parts) > 1 else None

if cmd in ("start","stop","status"):
instance_id = arg or INSTANCE_MAP.get(str(chat_id)) or DEFAULT_INSTANCE_ID
if not INSTANCE_RE.match(instance_id):
out.append({"json": {"chatId": chat_id, "reply":"Use /start i-xxxx | /stop i-xxxx |
/status i-xxxx"}})
continue
out.append({"json": {"chatId": chat_id, "payload": {"action": cmd, "instance_id":
instance_id}}})
else:
out.append({"json": {"chatId": chat_id, "reply": "Use /start|/stop|/status
[instance-id]"}})

# Optional: de-duplicate, keep newest per chat
latest_by_chat = {}
for it in out:
latest_by_chat[str(it["json"]["chatId"])] = it
out = list(latest_by_chat.values())

s["lastUpdateId"] = last
return out if out else [{"json": {"noop": True}}]
```

8.4 IF (Flow): Condition → exists; Value 1 (Expression) → {{ $json.payload }}.

8.5 Edit Fields (Set) – “Carry chatId” (TRUE path):

```
Mode: Manual Mapping
Include Other Input Fields: ON
Fields:
chatId = {{ $json.chatId }} (String or Number)
payload = {{ $json.payload }} (Object)
payloadStr = {{ JSON.stringify($json.payload) }} (String)
```

8.6 AWS Lambda (Invoke):

```
Credential: your AWS account (static keys)
Operation: Invoke
Function Name or ID: full Lambda ARN
Invocation Type: Wait for Results
JSON Input (Expression): {{ $json.payloadStr }}
```

8.7 Merge (combine):

```
Inputs: top = Carry chatId, bottom = AWS Lambda
Mode: Combine
Combine By: Position
Number of Inputs: 2
Output Type: Enrich Input 1
Result: each item has chatId, payload, payloadStr + result.ok/state/msg
```

8.8 Telegram → Send a text message (success path):

```
Chat ID (Expression):
{{ Number($json.chatId) || $json.chatId }}
```

```
Text (Expression):
{{
$json.result && $json.result.ok
? (
$json.payload.action === 'status'
? `■■ EC2 status for ${$json.payload.instance_id}: ${$json.result.state}`
: `■ EC2 ${$json.payload.action} requested for ${$json.payload.instance_id} →
${$json.result.state} (${ $json.result.msg })`
)
: `■ EC2 ${$json.payload.action} for ${$json.payload.instance_id} failed: ${$json.result ?
$json.result.msg : 'no result'}${$json.result?.state ? ' (state: ' + $json.result.state +
')' : ''}`
}}
```

8.9 Telegram → Send a text message (FALSE/help path):

```
Chat ID: {{ Number($json.chatId) || $json.chatId }}
Text: {{ $json.reply || 'Use /start|/stop|/status [instance-id]' }}
```

9) Troubleshooting

• Flood of messages: clear backlog once ⇒ call getUpdates to find largest update_id, then call getUpdates?offset=. Keep the offset expression in HTTP Request so it never replays.
• “Forbidden / Missing Authentication Token”: in Lambda node, JSON Input must be a string → use {{ JSON.stringify($json.payload) }} (we pass payloadStr).
• “UnauthorizedOperation ec2:DescribeInstances”: attach EC2 permissions to the Lambda execution role (not the n8n user).
• “IncorrectInstanceState”: Lambda is idempotent and returns informative messages; wait for transitions or call /status.

10) Security Notes

• Use a least-privilege IAM user for n8n (lambda:InvokeFunction only).
• Keep EC2 permissions on the Lambda’s role.
• Store tokens/keys only in n8n Credentials and rotate periodically.

That’s it—your Telegram bot now controls EC2 via n8n and AWS Lambda with confirmations for /start, /stop, and /status.