Alright — here’s a **clean, mistake-resistant, copy-pasteable** guide that folds in everything we learned, fixes the pain points you hit (WIF provider confusion, Caddy upstream name mismatch, missing .env, firewall tags/rules, “old” Caddyfile inside the container), and — most importantly — **moves almost all config files out of the VM and into your repo(s)**.

If you drop this into any chat, it should be enough for someone with zero background to deploy your apps to a fresh GCP VM.

**0) What you’ll end up with**

* **Repo-first config**: Caddy compose.yml + Caddyfile, and each app’s compose.prod.yml, Dockerfile, and start.sh all live in git — not hand-edited on the VM.
* **CI builds** (GitHub Actions) → pushes images to **Artifact Registry**.
* **CI deploys** by **SSH’ing** once to the VM to (a) copy the compose/Caddyfile and (b) run docker compose up -d. No “deploy.sh” on the VM needed.
* Secrets come from **Secret Manager**. CI fetches them and writes .env remotely in /opt/apps/<app>/.env.
* **Caddy** is the single entry point (80/443), reverse-proxies to each app by its **compose service name** on a shared docker network web.

**1) One-time GCP project setup (Cloud Shell)**

Replace placeholders with your values and run as-is.

# ===== Vars =====

export PROJECT\_ID="your-project-id"

export REGION="europe-west3"

export ZONE="${REGION}-b"

export AR\_REPO="apps"

gcloud config set project "$PROJECT\_ID"

# ===== APIs =====

gcloud services enable \

artifactregistry.googleapis.com \

secretmanager.googleapis.com \

compute.googleapis.com

# ===== Service Accounts =====

gcloud iam service-accounts create ci-deployer --display-name="CI Deployer"

gcloud iam service-accounts create vm-runtime --display-name="VM Runtime"

# CI builds/pushes images -> needs AR writer

gcloud projects add-iam-policy-binding "$PROJECT\_ID" \

--member="serviceAccount:ci-deployer@${PROJECT\_ID}.iam.gserviceaccount.com" \

--role="roles/artifactregistry.writer"

# VM pulls images & reads secrets at runtime

gcloud projects add-iam-policy-binding "$PROJECT\_ID" \

--member="serviceAccount:vm-runtime@${PROJECT\_ID}.iam.gserviceaccount.com" \

--role="roles/artifactregistry.reader"

gcloud projects add-iam-policy-binding "$PROJECT\_ID" \

--member="serviceAccount:vm-runtime@${PROJECT\_ID}.iam.gserviceaccount.com" \

--role="roles/secretmanager.secretAccessor"

# ===== Artifact Registry (Docker) =====

gcloud artifacts repositories create "$AR\_REPO" \

--repository-format=docker \

--location="$REGION" \

--description="App containers" || true

**(Optional but best) GitHub → GCP auth with WIF (no keys)**

**Do this in the Console UI (simplest and avoids the gcloud condition errors you saw):**

1. IAM & Admin → **Workload Identity Federation** → **Create pool**
   * Name: gh-pool
   * Provider type: **OIDC**
   * Issuer: https://token.actions.githubusercontent.com
   * Attribute mapping:
     + google.subject=assertion.sub
     + attribute.repository=assertion.repository
     + attribute.ref=assertion.ref
   * Finish and copy the **provider resource name**, e.g.  
     projects/1234567890/locations/global/workloadIdentityPools/gh-pool/providers/gh-provider
2. Allow your repo to impersonate ci-deployer:

PROJECT\_NUMBER="$(gcloud projects describe "$PROJECT\_ID" --format='value(projectNumber)')"

gcloud iam service-accounts add-iam-policy-binding \

"ci-deployer@${PROJECT\_ID}.iam.gserviceaccount.com" \

--role="roles/iam.workloadIdentityUser" \

--member="principalSet://iam.googleapis.com/projects/${PROJECT\_NUMBER}/locations/global/workloadIdentityPools/gh-pool/attribute.repository/<YourGitHubUser>/<YourRepo>"

⚠️ The earlier error INVALID\_ARGUMENT: The attribute condition must reference one of the provider's claims happens when trying to **add a condition** during provider creation via CLI. Using the **UI with just mapping (no condition)** avoids this. Conditions go on **bindings**, not the provider.

**If you can’t use WIF yet**: create a JSON key for ci-deployer and store it as GCP\_SA\_KEY in GitHub Secrets. (Rotate regularly.)

**2) Create the VM once (Cloud Shell)**

# ===== Firewall rules and tag =====

gcloud compute firewall-rules create allow-http-80 \

--network=default --direction=INGRESS --priority=1000 \

--action=ALLOW --rules=tcp:80 \

--source-ranges=0.0.0.0/0 --target-tags=http-8080 || true

gcloud compute firewall-rules create allow-https-443 \

--network=default --direction=INGRESS --priority=1000 \

--action=ALLOW --rules=tcp:443 \

--source-ranges=0.0.0.0/0 --target-tags=http-8080 || true

# ===== VM =====

gcloud compute instances create vm-runtime \

--zone="$ZONE" --machine-type=e2-micro \

--boot-disk-size=10GB \

--image-family=debian-12 --image-project=debian-cloud \

--service-account="vm-runtime@${PROJECT\_ID}.iam.gserviceaccount.com" \

--scopes=https://www.googleapis.com/auth/cloud-platform \

--tags=http-8080 \

--metadata=startup-script='

set -e

apt-get update

apt-get install -y ca-certificates curl gnupg lsb-release jq

# Docker

install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian $(. /etc/os-release && echo $VERSION\_CODENAME) stable" > /etc/apt/sources.list.d/docker.list

apt-get update && apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin

# gcloud CLI on VM (handy)

echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] http://packages.cloud.google.com/apt cloud-sdk main" > /etc/apt/sources.list.d/google-cloud-sdk.list

curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | gpg --dearmor > /usr/share/keyrings/cloud.google.gpg

apt-get update && apt-get install -y google-cloud-cli

# Shared docker network for Caddy + apps

docker network create web || true

# Size journald + auto-prune docker

mkdir -p /etc/systemd/journald.conf.d

printf "[Journal]\nSystemMaxUse=200M\nRuntimeMaxUse=100M\n" >/etc/systemd/journald.conf.d/size.conf

systemctl restart systemd-journald

cat >/etc/cron.daily/ci-clean <<EOF

#!/usr/bin/env bash

docker system prune -af --volumes >/dev/null 2>&1 || true

EOF

chmod +x /etc/cron.daily/ci-clean

'

You’ll set up **Caddy** from the repo in the next step via CI — not by hand on the VM.

**3) Repo layout (all files in git)**

You can keep infra with the app (mono-repo) or split into an **infra** repo. Below is mono-repo for simplicity:

your-app-repo/

├─ infra/

│ └─ caddy/

│ ├─ compose.yml

│ └─ Caddyfile

├─ deploy/

│ └─ compose.prod.yml

├─ Dockerfile

├─ start.sh

├─ package.json

├─ frontend/ # if you have a React SPA

│ ├─ package.json

│ └─ ...

└─ .github/

└─ workflows/

└─ ci-cd.yml

**infra/caddy/compose.yml**

services:

caddy:

image: caddy:2

restart: unless-stopped

ports: ["80:80","443:443"]

volumes:

- ./Caddyfile:/etc/caddy/Caddyfile

- caddy\_data:/data

- caddy\_config:/config

networks: [web]

volumes:

caddy\_data:

caddy\_config:

networks:

web:

external: true

**infra/caddy/Caddyfile**

First route one app. Add more later.

# Plain HTTP while testing; add domains later

:80 {

# Helpful during debug

log {

output stdout

format console

}

# Default site -> your app on port 3000

reverse\_proxy myapp:3000

}

# When you add domains:

# myapp.yourdomain.com {

# reverse\_proxy myapp:3000

# }

# api.yourdomain.com {

# reverse\_proxy myapp:3001

# }

**Why keep this in repo?** You’ll copy it to the VM with CI on first deploy, then just reload Caddy. No in-container edits, no “why does it still say myapp:3000?” surprises.

**deploy/compose.prod.yml (your app runner)**

services:

myapp: # <== service name used by Caddy DNS (IMPORTANT)

image: ${REGION}-docker.pkg.dev/${PROJECT\_ID}/${AR\_REPO}/myapp:${TAG:-latest}

restart: unless-stopped

env\_file: .env

networks: [web]

healthcheck:

test: ["CMD", "wget", "-qO-", "http://localhost:${PORT:-3000}/"]

interval: 20s

timeout: 3s

retries: 5

networks:

web:

external: true

**Pitfall you hit earlier:** Caddy could not reach myapp because the service was named nine-tones-app. **Caddy upstream must equal the compose service name** (myapp) and both must be on network web.

**Dockerfile (multi-stage, small)**

# ---- build backend ----

FROM node:20-alpine AS backend

WORKDIR /app

COPY package\*.json ./

RUN npm ci

COPY . .

# e.g. transpile TS backend -> dist/

RUN npm run build:backend

# ---- build frontend ----

FROM node:20-alpine AS frontend

WORKDIR /app

COPY frontend/package\*.json frontend/

RUN cd frontend && npm ci && npm run build # -> frontend/build

# ---- runtime ----

FROM node:20-alpine

WORKDIR /app

RUN adduser -D -H nodejs

COPY --from=backend /app/dist ./dist

COPY --from=frontend /app/frontend/build ./frontend/build

COPY start.sh ./start.sh

RUN chmod +x start.sh

USER nodejs

ENV NODE\_ENV=production PORT=3000 API\_PORT=3001

EXPOSE 3000 3001

CMD ["./start.sh"]

**start.sh**

#!/usr/bin/env sh

set -e

# Start API if you have one

if [ -f "dist/index.js" ]; then

node dist/index.js &

fi

# Serve the SPA

npx -y serve -s frontend/build -l ${PORT:-3000} --single

wait -n

**4) Secrets (Secret Manager)**

Create a secret **once** per app. Use newline env file format:

NODE\_ENV=production

PORT=3000

API\_PORT=3001

# FIREBASE\_\* = ...

SOAP\_SU=...

SOAP\_SP=...

printf 'NODE\_ENV=production\nPORT=3000\nAPI\_PORT=3001\n' | \

gcloud secrets create myapp-env --data-file=- --replication-policy=automatic || \

gcloud secrets versions add myapp-env --data-file=-

**5) GitHub Actions (build → push → deploy, all files from repo)**

Add **repo variables** (Settings → Variables):

* GCP\_PROJECT\_ID, GCP\_REGION, GCP\_AR\_REPO=apps
* VM\_HOST = your VM public IP
* VM\_SSH\_USER = your Linux username on the VM (e.g. borissolomonia)

Add **repo secrets**:

* If **WIF**: none for GCP (you’ll store GCP\_WIF\_PROVIDER and GCP\_DEPLOY\_SA as **variables**)
* If **JSON key**: GCP\_SA\_KEY (full JSON)
* SSH: VM\_SSH\_KEY (your OpenSSH private key for that VM user)

**.github/workflows/ci-cd.yml**

name: ci-cd

on:

push:

branches: [ main ]

jobs:

build-and-deploy:

runs-on: ubuntu-latest

permissions:

contents: read

id-token: write # needed only for WIF

env:

PROJECT\_ID: ${{ vars.GCP\_PROJECT\_ID }}

REGION: ${{ vars.GCP\_REGION }}

AR\_REPO: ${{ vars.GCP\_AR\_REPO }}

APP: myapp

TAG: ${{ github.sha }}

IMAGE: ${{ vars.GCP\_REGION }}-docker.pkg.dev/${{ vars.GCP\_PROJECT\_ID }}/${{ vars.GCP\_AR\_REPO }}/myapp:${{ github.sha }}

steps:

- uses: actions/checkout@v4

# ==== Auth to GCP ====

# ---- Option A: WIF (no key) ----

- name: Auth (WIF)

if: ${{ vars.GCP\_WIF\_PROVIDER && vars.GCP\_DEPLOY\_SA }}

uses: google-github-actions/auth@v2

with:

workload\_identity\_provider: ${{ vars.GCP\_WIF\_PROVIDER }}

service\_account: ${{ vars.GCP\_DEPLOY\_SA }}

project\_id: ${{ vars.GCP\_PROJECT\_ID }}

# ---- Option B: JSON key ----

- name: Auth (JSON key)

if: ${{ !vars.GCP\_WIF\_PROVIDER }}

uses: google-github-actions/auth@v2

with:

credentials\_json: ${{ secrets.GCP\_SA\_KEY }}

- uses: google-github-actions/setup-gcloud@v2

- name: Docker auth to Artifact Registry

run: gcloud auth configure-docker $REGION-docker.pkg.dev --quiet

# ==== Build & Push ====

- name: Build and push image (sha + latest)

run: |

docker build -t "$IMAGE" .

docker push "$IMAGE"

docker tag "$IMAGE" "$REGION-docker.pkg.dev/$PROJECT\_ID/$AR\_REPO/$APP:latest"

docker push "$REGION-docker.pkg.dev/$PROJECT\_ID/$AR\_REPO/$APP:latest"

# ==== Deploy files from repo to VM ====

# Copy Caddy and compose files (infra)

- name: Copy Caddy files

uses: appleboy/scp-action@v0.1.7

with:

host: ${{ vars.VM\_HOST }}

username: ${{ vars.VM\_SSH\_USER }}

key: ${{ secrets.VM\_SSH\_KEY }}

source: "infra/caddy/\*"

target: "/opt/apps/caddy"

overwrite: true

strip\_components: 2

# Pull Secret Manager -> .env on VM and bring up caddy + app compose

- name: Deploy on VM

uses: appleboy/ssh-action@v1.0.3

with:

host: ${{ vars.VM\_HOST }}

username: ${{ vars.VM\_SSH\_USER }}

key: ${{ secrets.VM\_SSH\_KEY }}

script: |

set -euo pipefail

# Ensure shared docker network exists

sudo docker network create web 2>/dev/null || true

# --- CADDY up-to-date ---

cd /opt/apps/caddy

# sanity: show mounts (avoid "old" Caddyfile surprise)

sudo docker compose down || true

sudo docker compose up -d

sudo docker compose exec caddy caddy reload --config /etc/caddy/Caddyfile

# --- APP deploy ---

APP\_DIR="/opt/apps/myapp"

sudo mkdir -p "$APP\_DIR"

# write compose from repo to VM

cat >"$APP\_DIR/compose.yml" <<'YML'

services:

myapp:

image: '"$REGION"'-docker.pkg.dev/'"$PROJECT\_ID"'/'"$AR\_REPO"'/myapp:'"$TAG"'

restart: unless-stopped

env\_file: .env

networks: [web]

healthcheck:

test: ["CMD", "wget", "-qO-", "http://localhost:${PORT:-3000}/"]

interval: 20s

timeout: 3s

retries: 5

networks:

web:

external: true

YML

# fetch env from Secret Manager

TMPENV="$(mktemp)"

gcloud secrets versions access latest --secret=myapp-env > "$TMPENV"

sudo mv "$TMPENV" "$APP\_DIR/.env"

sudo chmod 600 "$APP\_DIR/.env"

# pull & run

sudo docker login -u oauth2accesstoken -p "$(gcloud auth print-access-token)" "$REGION-docker.pkg.dev"

cd "$APP\_DIR"

sudo docker compose pull

sudo docker compose up -d --remove-orphans

# quick smoke test via Caddy

curl -fsSI http://127.0.0.1:80/ | head -n 10 || (echo "Caddy smoke test failed" && exit 1)

# ==== Post-deploy verification from CI ====

- name: Verify from the internet

run: |

IP="${{ vars.VM\_HOST }}"

curl -fsSI "http://$IP/" | tee /dev/stderr | grep -q "200 OK"

# Adjust filename if your JS bundle name changes

curl -fsSI "http://$IP/static/js/main.b07b262c.js" | tee /dev/stderr | grep -q "200 OK"

**Why we write compose.yml on the VM instead of SCP that file too?**  
We *can* also SCP deploy/compose.prod.yml to /opt/apps/myapp/compose.yml. If you prefer that, add another scp-action step for deploy/compose.prod.yml → /opt/apps/myapp/compose.yml and remove the here-doc in the SSH step. Either way is fine — both are repo-first.

**6) First deploy & test**

1. Commit the files and push to main.
2. In GitHub, watch the **ci-cd** workflow:
   * Auth → Build → Push → Copy Caddy → Write .env (from Secret Manager) → Bring up app → Caddy reload → Verify.
3. Browse to http://<VM\_IP>/. If blank:
   * Open browser **DevTools → Console/Network** for errors.
   * On the VM (or from CI logs) check:
   * # app healthy?
   * sudo docker logs --tail=100 myapp
   * sudo docker compose -f /opt/apps/caddy/compose.yml logs --tail=80 caddy
   * # caddy can reach myapp?
   * sudo docker compose -f /opt/apps/caddy/compose.yml exec caddy sh -lc \
   * 'apk add --no-cache curl >/dev/null 2>&1 || true; curl -I http://myapp:3000/'
   * **Most common fixes**
     + Service name mismatch (Caddyfile vs compose) → must be identical.
     + App not on network web.
     + Firewall tag missing (http-8080) → run gcloud compute instances add-tags ....
     + CORS or frontend calling http://localhost:3001 → change frontend to /api/... and proxy in Caddy:
     + handle\_path /api/\* {
     + reverse\_proxy myapp:3001
     + }

**7) Add another app later**

* Create newapp-env in Secret Manager.
* Add deploy/compose.prod.yml equivalent for newapp (service name newapp!).
* Add Caddy routes in infra/caddy/Caddyfile:
* # Example: route by subpath
* handle /newapp/\* {
* uri strip\_prefix /newapp
* reverse\_proxy newapp:3000
* }
* CI: duplicate the “APP/myapp” bits (or parameterize with a matrix).
* Push. CI builds newapp, copies updated Caddyfile, and brings up newapp.

**8) Production polish (HTTPS, domains)**

When you have DNS:

myapp.example.com {

reverse\_proxy myapp:3000

}

api.example.com {

reverse\_proxy myapp:3001

}

* Keep firewall rule for 443.
* Caddy auto-provisions certificates (Let’s Encrypt). If you saw **“rejectedIdentifier … example.com forbidden by policy”**, that’s because example.com is blocked at the CA — use real domains.

**9) Runbooks & quick commands**

**Disk & cleanup**

docker system df

docker image prune -af

journalctl --disk-usage

sudo du -xh /var /home --max-depth=2 | sort -h | tail -20

**Doctor**

# Is Caddy listening?

ss -ltnp | awk '$4 ~ /:(80|443)$/'

# App healthy?

sudo docker logs --tail=100 myapp

# Can Caddy reach app DNS?

sudo docker compose -f /opt/apps/caddy/compose.yml exec caddy sh -lc \

'apk add --no-cache curl >/dev/null 2>&1 || true; curl -I http://myapp:3000/'

# Are both in 'web'?

sudo docker network inspect web --format '{{range $id,$c := .Containers}}{{$c.Name}}{{"\n"}}{{end}}'

**Rollback**

# Redeploy a previous image tag (sha)

# Edit ci step to pass TAG=<oldsha>, or run manually on VM:

cd /opt/apps/myapp

sudo sed -i "s~:latest~:<oldsha>~" compose.yml

sudo docker compose pull && sudo docker compose up -d

**Why your past errors happened & how this guide avoids them**

* **WIF provider “attribute condition” error**  
  You tried to set a condition at provider creation via CLI. This guide uses the **Console UI** with only **attribute mapping**, then scopes the **binding** to your repo — the safe, supported path.
* **Caddy saw myapp:3000 but your service was nine-tones-app**  
  This guide enforces service name myapp and keeps both **Caddyfile and compose** in the repo; CI copies both together so they stay in sync. We also reload Caddy after copying and show the mount mapping on start (to rule out stale configs).
* **Blank page / 502**  
  We added **browser DevTools** check + **Caddy logs** + **curl from inside Caddy to the service**. The compose includes a **healthcheck** to surface bad boots early.
* **Firewall “took too long to respond”**  
  We make the VM carry the http-8080 tag at creation and create the allow-http-80 / allow-https-443 rules once. There’s also a quick curl verification in CI after deploy.
* **Missing .env**  
  CI now fetches Secret Manager and writes .env on the VM before compose up. You won’t see the “env file … not found” problem again.
* **Old Caddyfile inside container**  
  We bind ./Caddyfile:/etc/caddy/Caddyfile from the repo, **restart** Caddy container on change, and run caddy reload. The Action also prints container **Mounts** once (sanity).

**10) “Ask me these first” (so a helper can tailor this)**

1. What’s your **GCP project id**, region (e.g., europe-west3), and VM zone?
2. Do you want **WIF** (no keys) for CI auth, or a **JSON key** for speed?
3. Do you have **domains** now, or will you run on the VM IP first?
4. What’s the **compose service name** you want for the app? (I defaulted to myapp.)
5. List your **environment variables** (for Secret Manager), and which ports does your app expose (frontend/API)?

Answer those, commit this layout, and you’ll be able to rinse-and-repeat on any new VM or app with minimal edits.

If you want, I can generate these files into a starter repo scaffold for you (with your exact PROJECT\_ID, REGION, AR\_REPO, etc.) so you just fill in the secret values and push.