This is the **Reference Architecture v2026.5 ("Universal Mac-AMD64")**.

It fulfills your requirement to run **AMD64 (x86\_64) containers on an Apple Silicon Mac** using Docker Desktop's built-in Rosetta 2 emulation. This ensures binary parity with your remote EPYC server while allowing local development.

### 🍎 Mac AMD64 Emulation Strategy

1. **Forced Platform:** The local DevContainer config (.devcontainer/mac-amd64/) explicitly forces "runArgs": ["--platform=linux/amd64"]. This triggers Docker Desktop to use Rosetta 2 (which is ~10x faster than QEMU) to run the container.
2. **Context Awareness:** The setup script validates that you are using the desktop-linux context (Docker Desktop) to ensure VM optimizations like virtiofs are active.
3. **Build Compatibility:** A custom build script detects if you are running locally on a Mac and forces docker buildx bake to load *only* the linux/amd64 image (since multi-arch images cannot be loaded into the local daemon).

### 🚀 Usage

1. mkdir cpp-universal && cd cpp-universal
2. Save the code below as **generate\_project.py**.
3. Run: python3 generate\_project.py
4. Install: pixi install
5. **Configure:** pixi run setup-dev (Select **"Local Mac"** or **"Remote EPYC"**).

### 📄 The Project Generator

Python

import os  
import stat  
import platform  
import subprocess  
import json  
from pathlib import Path  
  
FILES = {  
 # -------------------------------------------------------------------------  
 # 1. Pixi Manifest (Bleeding Edge + Dev Tools)  
 # -------------------------------------------------------------------------  
 "pixi.toml": r"""  
[project]  
name = "cpp-universal"  
version = "2026.5.0"  
platforms = ["linux-64", "osx-arm64"]  
# 🟢 Bleeding Edge Channels for GCC 15 / LLVM 21  
channels = ["conda-forge", "conda-forge/label/gcc\_dev", "conda-forge/label/llvm\_rc"]  
  
[pypi-options]  
resolve-dependencies-with-uv = true  
  
[dependencies]  
cmake = "\*"  
ninja = "\*"  
python = "3.12.\*"  
sccache = "\*"  
bun = "\*"  
  
# --- Automation Suite (Zero Tolerance) ---  
[feature.automation.dependencies]  
docker-py = "\*"  
rich = "\*"  
ruff = "\*"  
vulture = "\*"  
deptry = "\*"  
hadolint = "\*"  
actionlint = "\*"  
check-jsonschema = "\*"  
typos = "\*"  
checkov = "\*"  
pytest = "\*"  
pytest-testinfra = "\*"  
  
[feature.automation.pypi-dependencies]  
ty = "\*"  
zizmor = "\*"  
  
# --- Compilers ---  
[feature.gcc15.dependencies]  
gcc = "15.\*"  
gxx = "15.\*"  
  
[feature.llvm-head.dependencies]  
clang = "21.\*"  
clangxx = "21.\*"  
lld = "21.\*"  
lldb = "21.\*"  
clang-tools = "21.\*" # clang-tidy, clang-format  
llvm-tools = "21.\*" # llvm-mc, objdump  
llvm-bolt = "21.\*"  
  
# --- Environments ---  
[environments]  
automation = ["automation"]  
stable = ["gcc15", "llvm-head"]  
dev\_container = ["stable", "dev", "automation"]  
  
[feature.dev.dependencies]  
gdb = "\*"  
starship = "\*"  
direnv = "\*"  
opentelemetry-collector = "\*"  
  
[tasks]  
validate = { cmd = "python -m scripts.validate", env = { PYTHONUNBUFFERED = "1" } }  
build = { cmd = "python -m scripts.build", env = { PYTHONUNBUFFERED = "1" } }  
setup-dev = { cmd = "python -m scripts.setup\_dev", env = { PYTHONUNBUFFERED = "1" } }  
init-container = "python -m scripts.lib.container\_init"  
""",  
  
 # -------------------------------------------------------------------------  
 # 2. Docker Infrastructure  
 # -------------------------------------------------------------------------  
 "docker/docker-bake.hcl": r"""  
variable "REGISTRY" { default = "ghcr.io/my-org/cpp" }  
variable "CONFIG\_HASH" { default = "local" }  
  
group "default" { targets = ["build"] }  
  
target "base" {  
 dockerfile = "docker/Dockerfile"  
 # 🟢 Multi-Arch: Builds for both Server (AMD64) and Mac Native (ARM64)  
 # Note: Local Mac Dev will force-pull AMD64 to match Server.  
 platforms = ["linux/amd64", "linux/arm64"]  
   
 cache-from = ["type=gha"]  
 cache-to = ["type=gha,mode=max"]  
 attest = ["type=provenance,mode=max", "type=sbom"]  
}  
  
target "build" {  
 inherits = ["base"]  
 matrix = {  
 # 🟢 Ubuntu Matrix  
 os = ["focal", "noble"]  
 env = ["stable"]  
 }  
 name = "${os}-${env}"  
 args = {  
 BASE\_IMAGE = "ghcr.io/prefix-dev/pixi:${os}"  
 PIXI\_ENV = "${env}"  
 }  
 tags = ["${REGISTRY}:${os}-${env}-${CONFIG\_HASH}"]  
}  
""",  
  
 "docker/Dockerfile": r"""  
# syntax=docker/dockerfile:1  
ARG BASE\_IMAGE  
FROM ${BASE\_IMAGE}  
  
WORKDIR /app  
COPY pixi.toml pixi.lock ./  
  
ARG PIXI\_ENV  
# ⚡️ Cache Mounts  
RUN --mount=type=cache,target=/root/.cache/pixi \  
 --mount=type=cache,target=/root/.cache/uv \  
 pixi install --frozen --environment ${PIXI\_ENV}  
  
# ❄️ Freeze Env to JSON  
RUN pixi run -e ${PIXI\_ENV} python -c "import os, json; print(json.dumps(dict(os.environ)))" > /app/pixi\_env.json  
  
# Symlink Python & S3 Pack  
RUN ln -sf /app/.pixi/envs/${PIXI\_ENV}/bin/python /app/python\_runtime  
RUN pixi global install pixi-pack && \  
 pixi-pack pack -e ${PIXI\_ENV} --platform linux-64 -o /app/environment.tar.gz  
  
COPY docker/entrypoint.py /app/entrypoint.py  
ENTRYPOINT ["/app/python\_runtime", "/app/entrypoint.py"]  
CMD ["/bin/bash"]  
""",  
  
 "docker/entrypoint.py": r"""  
import os, sys, json  
def main():  
 if os.path.exists("/app/pixi\_env.json"):  
 with open("/app/pixi\_env.json") as f: os.environ.update(json.load(f))  
 args = sys.argv[1:] or ["/bin/bash"]  
 try: os.execvpe(args[0], args, os.environ)  
 except FileNotFoundError: sys.exit(f"Error: Command '{args[0]}' not found.")  
if \_\_name\_\_ == "\_\_main\_\_": main()  
""",  
  
 # -------------------------------------------------------------------------  
 # 3. Dev Containers (The Split Strategy)  
 # -------------------------------------------------------------------------  
   
 # Option A: Local Mac (AMD64 Emulated)  
 ".devcontainer/mac-amd64/devcontainer.json": r"""  
{  
 "name": "Local Mac (AMD64 Emulation)",  
   
 // 🟢 Target the Noble (Ubuntu 24.04) image  
 "image": "ghcr.io/my-org/cpp:noble-stable-latest",  
   
 "remoteUser": "vscode",  
 "updateRemoteUserUID": true,  
  
 // 🟢 FORCE PLATFORM: Use Docker Desktop Rosetta Emulation  
 "runArgs": [  
 "--platform=linux/amd64",   
 "--cap-add=SYS\_PTRACE",   
 "--security-opt", "seccomp=unconfined"  
 // Note: --network=host is NOT supported on Mac Docker Desktop  
 ],  
  
 "features": {  
 "ghcr.io/devcontainers/features/common-utils:2": { "installZsh": true },  
 "ghcr.io/devcontainers/features/sshd:1": { "version": "latest" },  
 "ghcr.io/devcontainers-contrib/features/bun:1": { "version": "latest" }  
 },  
  
 "postCreateCommand": "pixi run init-container",  
   
 "customizations": {  
 "vscode": { "extensions": ["ms-vscode.cpptools", "sst.opencode"] }  
 }  
}  
""",  
  
 # Option B: Remote EPYC (Hybrid)  
 ".devcontainer/remote-hybrid/devcontainer.json": r"""  
{  
 "name": "Remote EPYC Server",  
 "image": "ghcr.io/my-org/cpp:noble-stable-latest",  
 "remoteUser": "vscode",  
 "updateRemoteUserUID": true,  
 "features": {  
 "ghcr.io/devcontainers/features/common-utils:2": { "installZsh": true },  
 "ghcr.io/devcontainers/features/sshd:1": { "version": "latest" },  
 "ghcr.io/devcontainers-contrib/features/bun:1": { "version": "latest" }  
 },  
 "runArgs": ["--network=host"],  
 "postCreateCommand": "pixi run init-container"  
}  
""",  
  
 # -------------------------------------------------------------------------  
 # 4. Automation Scripts  
 # -------------------------------------------------------------------------  
 "scripts/setup\_dev.py": r"""  
import os  
import shutil  
import platform  
import subprocess  
import json  
from rich.prompt import Prompt, Confirm  
from rich.console import Console  
from rich.panel import Panel  
  
console = Console()  
  
def check\_docker\_desktop():  
 """Validates Docker Desktop Context."""  
 if platform.system() != "Darwin": return  
   
 try:  
 # Check Context  
 ctx = subprocess.check\_output(["docker", "context", "show"], text=True).strip()  
 if ctx == "desktop-linux":  
 console.print("[green]✅ Using Docker Desktop (desktop-linux)[/green]")  
 else:  
 console.print(f"[yellow]⚠️ Current Context: {ctx}. 'desktop-linux' recommended for Mac.[/yellow]")  
   
 # Check Architecture  
 if platform.machine() == "arm64":  
 console.print("[cyan]🍎 Apple Silicon detected. Configuring AMD64 Emulation...[/cyan]")  
 console.print(" Ensure [bold]Use Rosetta for x86/amd64 emulation[/bold] is enabled in Docker Settings.")  
 except:  
 console.print("[red]❌ Docker CLI not working[/red]")  
  
def main():  
 console.print(Panel.fit("Universal C++ Environment Setup"))  
 check\_docker\_desktop()  
   
 mode = Prompt.ask("Select Mode", choices=["local", "remote"], default="local")  
   
 if mode == "local":  
 console.print("[green]✅ Setup Complete[/green]")  
 console.print("1. Open VS Code")  
 console.print("2. Run 'Reopen in Container'")  
 console.print("3. Select [bold]Local Mac (AMD64 Emulation)[/bold]")  
 else:  
 alias = Prompt.ask("Remote Alias", default="epyc")  
 ip = Prompt.ask("Remote IP")  
 user = Prompt.ask("Remote User", default="ubuntu")  
   
 cfg = os.path.expanduser("~/.ssh/config")  
 entry = f"\nHost {alias}\n HostName {ip}\n User {user}\n ForwardAgent yes\n"  
 if Confirm.ask(f"Add {alias} to SSH config?"):  
 with open(cfg, "a") as f: f.write(entry)  
   
 if shutil.which("mutagen") and Confirm.ask("Start Mutagen Sync?"):  
 subprocess.run(["mutagen", "sync", "terminate", "cpp-univ"], stderr=subprocess.DEVNULL)  
 subprocess.run(["mutagen", "sync", "create", "--name", "cpp-univ", "--mode", "two-way-safe",   
 "--ignore", "build/", "--ignore", ".pixi/", ".", f"{alias}:/home/{user}/workspace/cpp-univ"])  
  
if \_\_name\_\_ == "\_\_main\_\_": main()  
""",  
  
 "scripts/build.py": r"""  
import os, hashlib, subprocess  
from rich.console import Console  
console = Console()  
BASE\_IMAGES = { "focal": "ghcr.io/prefix-dev/pixi:focal", "noble": "ghcr.io/prefix-dev/pixi:noble" }  
  
def main():  
 console.rule("[bold blue]Universal Build")  
 digests = {k: "latest" for k in BASE\_IMAGES} # Add lookup logic if needed  
   
 hasher = hashlib.sha256()  
 for f in ["pixi.lock", "docker/Dockerfile", "docker/docker-bake.hcl"]:  
 if os.path.exists(f):   
 with open(f, "rb") as file: hasher.update(file.read())  
 config\_hash = hasher.hexdigest()[:12]  
   
 if "GITHUB\_OUTPUT" in os.environ:  
 with open(os.environ["GITHUB\_OUTPUT"], "a") as f: f.write(f"HASH={config\_hash}\n")  
  
 env = os.environ.copy()  
 env.update({"CONFIG\_HASH": config\_hash})  
   
 # 🟢 CRITICAL FIX for Local Builds:  
 # "docker buildx bake --load" fails if multiple platforms are defined.  
 # We must force a single platform when building locally on Mac.  
 target = "--push" if os.getenv("CI") else "--load"  
   
 if target == "--load":  
 print("⚠️ Local Build detected: Forcing linux/amd64 load (Emulation Mode)")  
 # Overrides the platforms list in docker-bake.hcl to just one  
 subprocess.run(["docker", "buildx", "bake", "--set", "\*.platforms=linux/amd64", target], env=env, check=True)  
 else:  
 # CI pushes both amd64 and arm64  
 subprocess.run(["docker", "buildx", "bake", target], env=env, check=True)  
  
if \_\_name\_\_ == "\_\_main\_\_": main()  
""",  
  
 "scripts/validate.py": r"""  
import subprocess, sys, shutil  
from concurrent.futures import ThreadPoolExecutor  
  
CHECKS = [  
 ("Ruff", ["ruff", "check", "."]),  
 ("Ty", ["ty", "check", "."]),  
 ("Hadolint", ["hadolint", "docker/Dockerfile"]),  
 ("Checkov", ["checkov", "-d", "docker", "--quiet", "--compact"]),  
 ("Zizmor", ["zizmor", ".github/workflows"]),  
 ("Schema", ["check-jsonschema", "--schemafile", "https://raw.githubusercontent.com/devcontainers/spec/main/schemas/devContainer.schema.json", ".devcontainer/mac-amd64/devcontainer.json"])  
]  
  
def run(c):  
 if not shutil.which(c[1][0]): return (False, c[0], "Missing")  
 r = subprocess.run(c[1], capture\_output=True, text=True)  
 return (r.returncode==0, c[0], r.stdout+r.stderr)  
  
def main():  
 with ThreadPoolExecutor() as exe:  
 for ok, name, out in exe.map(run, CHECKS):  
 print(f"{'✅' if ok else '❌'} {name}")  
 if not ok:   
 print(out)  
 sys.exit(1)  
  
if \_\_name\_\_ == "\_\_main\_\_": main()  
""",  
  
 "scripts/lib/container\_init.py": r"""  
import os, json, subprocess  
def main():  
 if os.path.exists("/app/pixi\_env.json"):  
 with open("/app/pixi\_env.json") as f: data = json.load(f)  
 with open(os.path.expanduser("~/.zshrc"), "a") as f:  
 f.write("\n# Pixi Hydration\n")  
 for k,v in data.items():   
 if k not in ["PATH", "HOME"]: f.write(f'export {k}="{v}"\n')  
   
 # Verify Architecture  
 arch = subprocess.check\_output(["uname", "-m"]).decode().strip()  
 if arch == "x86\_64": print("✅ Running in AMD64 (Emulated/Native)")  
 else: print("⚠️ Running in ARM64")  
   
 subprocess.run(["bun", "install", "--global", "@google/gemini-cli", "opencode"], check=False)  
  
if \_\_name\_\_ == "\_\_main\_\_": main()  
""",  
  
 ".github/workflows/ci.yml": r"""  
name: Build  
on: [push]  
jobs:  
 build:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v4  
 - uses: prefix-dev/setup-pixi@v0.8.3  
 with: { environments: automation }  
 - uses: docker/setup-qemu-action@v3  
 - uses: docker/setup-buildx-action@v3  
 - uses: docker/login-action@v3  
 with: { registry: ghcr.io, username: ${{ github.actor }}, password: ${{ secrets.GITHUB\_TOKEN }} }  
 - run: pixi run -e automation build  
""",  
  
 "scripts/\_\_init\_\_.py": "",  
 "scripts/lib/\_\_init\_\_.py": "",  
 "scripts/tests/\_\_init\_\_.py": "",  
 "scripts/tests/test\_placeholder.py": "def test\_ok(): assert True",  
   
 "pyproject.toml": r"""  
[tool.ruff]  
target-version = "py312"  
[tool.ty]  
check-untyped-defs = true  
[tool.vulture]  
exclude = ["scripts/tests"]  
[tool.deptry]  
ignore\_obsolete = ["pytest", "pytest-testinfra", "rich", "docker", "check-jsonschema"]  
"""  
}  
  
def generate\_project():  
 root = Path("cpp-universal")  
 root.mkdir(exist\_ok=True)  
 for filename, content in FILES.items():  
 filepath = root / filename  
 filepath.parent.mkdir(parents=True, exist\_ok=True)  
 with open(filepath, "w", encoding="utf-8") as f: f.write(content.strip() + "\n")  
 print(f"📄 Created {filepath}")  
   
 for script in (root / "scripts").glob("\*\*/\*.py"):  
 os.chmod(script, os.stat(script).st\_mode | stat.S\_IEXEC)  
   
 print("\n✅ Project Generated! Run 'pixi install' inside.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 generate\_project()