

# SOLVING THE DISTRIBUTED NAMING PROBLEM (!) IN APPLIED PACKAGE MANAGEMENT



Eric Myhre  
**@warpfork**



I have software I want to build!

I have software I want to ~~build~~  
use!  
compile!  
okay, *patch*!  
then compile and use!  
and share..!

(there are many different ways to go about this)

What's tricky isn't building.

We have build tools.

What's tricky is  
integrating.

What's tricky is connecting  
different build processes together.

What's tricky is composing systems.

What's tricky is  
**integrating.**

The best way to figure out how to *integrate* something is... to break it apart, first.

# *What is a “build”?*

Files go in...

```
{  
  "inputs": {  
    "/": "ipfs:Qm6q7G4hWr26q7G4hWTa5Lf89p"  
    "/app/gcc": "ipfs:Qmr283FpTaSL883FpL8"  
    "/task/src": "git:7c554c3ae7bee8e3b42"  
  }  
}
```

# *What is a “build”?*

Files go in...

Files go out...

```
{  
  "inputs": {  
    "/": "ipfs:Qm6q7G4hWr26q7G4hWTa5Lf89p"  
    "/app/gcc": "ipfs:Qmr283FpTaSL883FpL8"  
    "/task/src": "git:7c554c3ae7bee8e3b42"  
  },  
  
  "outputs": {  
    "/task/out": {"packtype": "ipfs"}  
  }  
}
```

# *What is a “build”?*

Files go in...

Files go out...

An action in the middle.

```
{  
  "inputs": {  
    "/": "ipfs:Qm6q7G4hWr26q7G4hWTa5Lf89p"  
    "/app/gcc": "ipfs:Qmr283FpTaSL883FpL8"  
    "/task/src": "git:7c554c3ae7bee8e3b42"  
  },  
  "action": {  
    "exec": [  
      "/app/gcc/bin/gcc",  
      "/task/src/cool.c",  
      "-o", "/task/out/cool.exe" ]  
  },  
  "outputs": {  
    "/task/out": {"packtype": "ipfs"}  
  }  
}
```

```
{  
  "inputs": {  
    "/": "ipfs:Qm6q7G4hWr26q7G4hWTa5Lf89p"  
    "/app/gcc": "ipfs:Qmr283FpTaSL883FpL8"  
    "/task/src": "git:7c554c3ae7bee8e3b42"  
  },  
  "action": {  
    "exec": [  
      "/app/gcc/bin/gcc",  
      "/task/src/cool.c",  
      "-o", "/task/out/cool.exe" ]  
  },  
  "outputs": {  
    "/task/out": {"packtype": "ipfs"}  
  }  
}
```

*What can we do if we...*

Do this with IPLD?

```
{  
  "inputs": {  
    "/" : "ipfs:Qm6q7G4hWr26q7G4hWTa5Lf89p"  
    "/app/gcc": "ipfs:Qmr283FpTaSL883FpL8"  
    "/task/src": "git:7c554c3ae7bee8e3b42"  
  },  
  "action": {  
    "exec": [  
      "/app/gcc/bin/gcc",  
      "/task/src/cool.c",  
      "-o", "/task/out/cool.exe" ]  
  },  
  "outputs": {  
    "/task/out": {"packtype": "ipfs"}  
  } }
```

“Computation-addressable”

```
{  
  "formulaID": "QmVdAD3v91JWPjZXGkTKD6",  
  "exitCode": 0,  
  "results": {  
    "/task/out": "ipfs:QmD9AjmrldCY4QUquhiQ"  
  } }
```

*This is a useful building block.*

- Share a complete build instruction easily.
- Rebuild later, or use in bug reports, etc.
- Audit logs; explanations of builds that happened previously.
- Check output hashes to easily check reproducibility.
- General forcing-function for sanity!

*Now let's try to integrate.*

*Now let's try to integrate.*

- Cool, we can address builds we've already drafted.

...

*Now let's try to integrate.*

- Cool, we can address builds we've already drafted.
- But nobody likes copy-pasting hashes;
- We'll need a way to describe multiple steps and their relationships in advance to be productive.

*Now let's try to integrate.*

“describe multiple steps and relate them in advance”.....

We have a **naming** problem.



*So break it down again.*

Distributed naming problems are hard.

So let's try to break down the problem into component parts...

(until suddenly none of them look like “naming” anymore!)

and then integrate again afterwards.

*So break it down again.*

- Naming for labeling (still immutable!)
- Naming for making a graph (can be locally-scoped!)
  - Naming for looking for ‘updates’
  - Naming for grouping and categorization
    - Naming for reputation
- ...

*So break it down again.*

- 
- Naming for labeling (still immutable!)
  - Naming for making a graph (can be locally-scoped!)
    - Naming for looking for ‘updates’
    - Naming for grouping and categorization
    - Naming for reputation
  - ...

Some of these use-cases are easier than others.

*So break it down again.*

- Naming for labeling (still immutable!)
- Naming for making a graph (can be locally-scoped!)
  - **Naming for looking for ‘updates’**
  - Naming for grouping and categorization
    - Naming for reputation
  - ...

Some of these use-cases entangle application logic.

# *locally scoped naming is easy*

Lesson for any distributed system design:

If you can turn a *distributed* naming problem into a *locally scoped* naming problem, do it. It's easier.  
(Then try to make it composable.)

```
"imports": {
  "root-builder": "catalog:early.polydawn.io/monolith/debian-gcc-plus:v1.2017.01.04:linux-amd64",
  "app-smsh": "catalog:early.radix.polydawn.io/smsh:candidate:bin-linux-amd64",
  "src-gzip": "catalog:early.hypae.polydawn.io/sources/gzip:v1.9:src",
  "autoconfesque": "ingest:pack:tar:.../shared/autoconfesque"
"steps": {
  "build": {
    "operation": {
      "inputs": {
        "/": "root-builder",
        "/app/smsh": "app-smsh",
        "/task/raw": "src-gzip",
        "/radix/autoconfesque": "autoconfesque"
      "action": {
        "userinfo": {"uid":0},
        "exec": [
          "/app/smsh/bin/smsh",
          "ln -s raw/* src",
          "ls -lah",
          "/radix/autoconfesque/build.sh",
          "mkdir out2"
          "mv out/usr/local/bin out2"
        "outputs": {
          "bin": "/task/out2"
        "busybash": {
          "operation": {
            "inputs": {
              "/": "root-busybash",
              "/app/smsh": "app-smsh",
              "/task": "subject"
            "action": {
              "exec": [
                "/app/smsh/bin/smsh",
                "ldd ./bin/gzip",
                "./bin/gzip -h"
            "rich": {
              "operation": {
                "inputs": {
                  "/": "root-debian",
                  "/app/smsh": "app-smsh",
                  "/task": "subject"
                "action": {
                  "exec": [
                    "/app/smsh/bin/smsh",
                    "ldd ./bin/gzip",
                    "./bin/gzip -h"
      "exports": {
        "bin-amd64-linux": "build.bin"
```



# *locally scoped naming is easy*

... so what do we do when  
we want to collaborate and  
integrate with other people  
and other authors?

```
"imports": {  
  "root-builder": "catalog:early.polydawn.io/monolith/debian-gcc-plus:v1.2017.01.04:linux-amd64",  
  "app-smsh": "catalog:early.radix.polydawn.io/smsh:candidate:bin-linux-amd64",  
  "src-gzip": "catalog:early.hypae.polydawn.io/sources/gzip:v1.9:src",  
  "autoconfesque": "ingest:pack:tar:....shared/autoconfesque"  
}  
"steps": {  
  "build": {  
    "operation": {  
      "inputs": {  
        "/": "root-builder",  
        "/app/smsh": "app-smsh",  
        "/task/raw": "src-gzip",  
        "/radix/autoconfesque": "autoconfesque"  
      }  
      "action": {  
        "userinfo": {"uid":0},  
        "exec": [  
          "/app/smsh/bin/smsh",  
          "ln -s raw/* src",  
          "ls -lah",  
          "/radix/autoconfesque/build.sh",  
          "mkdir out2",  
          "mv out/usr/local/bin out2"  
        ]  
        "outputs": {  
          "bin": "/task/out2"  
        }  
      }  
    }  
  }  
  "busybash": {  
    "operation": {  
      "inputs": {  
        "/": "root-busybash",  
        "/app/smsh": "app-smsh",  
        "/task": "subject"  
      }  
      "action": {  
        "exec": [  
          "/app/smsh/bin/smsh",  
          "ldd ./bin/gzip",  
          "./bin/gzip -h"  
        ]  
      }  
    }  
  }  
  "rich": {  
    "operation": {  
      "inputs": {  
        "/": "root-debian",  
        "/app/smsh": "app-smsh",  
        "/task": "subject"  
      }  
      "action": {  
        "exec": [  
          "/app/smsh/bin/smsh",  
          "ldd ./bin/gzip",  
          "./bin/gzip -h"  
        ]  
      }  
    }  
  }  
}  
"exports": {  
  "bin-amd64-linux": "build.bin"  
}
```



# *connect things with content-addressable (IPLD?) messages*

... so what do we do when  
we want to collaborate and  
integrate with other people  
and other authors?

Can we keep extending the  
defn of “locally scoped”  
gradually?

```
"imports": {
  "root-builder": "catalog:early.polydawn.io/monolith/debian-gcc-plus:v1.2017.01.04:linux-amd64",
  "app-smsh": "catalog:early.radix.polydawn.io/smsh:candidate:bin-linux-amd64",
  "src-gzip": "catalog:early.hypae.polydawn.io/sources/gzip:v1.9:src",
  "autoconfesque": "ingest:pack:tar:../../../../shared/autoconfesque"
"steps": {
  "build": {
    "operation": {
      "inputs": {
        "/": "root-builder",
        "/app/smsh": "app-smsh",
        "/task/raw": "src-gzip",
        "/radix/autoconfesque": "autoconfesque"
      "action": {
        "userinfo": {"uid":0},
        "exec": [
          "/app/smsh/bin/smsh",
          "ln -s raw/* src",
          "ls -lah",
          "/radix/autoconfesque/build.sh",
          "mkdir out2",
          "mv out/usr/local/bin out2"
        "outputs": {
          "bin": "/task/out2"
"busybash": {
  "operation": {
    "inputs": {
      "/": "root-busybash",
      "/app/smsh": "app-smsh",
      "/task": "subject"
    "action": {
      "exec": [
        "/app/smsh/bin/smsh",
        "ldd ./bin/gzip",
        "./bin/gzip -h"
    "rich": {
      "operation": {
        "inputs": {
          "/": "root-debian",
          "/app/smsh": "app-smsh",
          "/task": "subject"
        "action": {
          "exec": [
            "/app/smsh/bin/smsh",
            "ldd ./bin/gzip",
            "./bin/gzip -h"
"exports": {
  "bin-amd64-linux": "build.bin"
```

```
"formulaID": "QmVdAD3v91JWPjZXGkTKD6",
"exitCode": 0,
"results": {
  "/task/out": "ipfs:QmD9AjmrqdCY4QUquhiQ"
```

# Docs, more docs, and prototypes

[repeatr.io](https://repeatr.io)

[github.com/polydawn/timeless](https://github.com/polydawn/timeless)

The screenshot shows a web browser window with the URL <https://repeatr.io/design/layers/>. The page has a dark-themed header with the word "TIMELESS" in white. Below the header is a sidebar with a vertical navigation menu:

- 1. Welcome
- 2. Vision
- 3. Getting Started
- 4. Design (selected)
- Timeless Stack API Layers (sub-menu):
  - Catalogs & Releasing
  - Executors
- 5. Schema Reference
- 6. Tools Reference
- # Glossary
- # More
- Workflows

The main content area is titled "TIMELESS STACK API LAYERS". It contains the following text:

The Timeless Stack APIs are split into several distinct levels, based on their expressiveness. The lower level layers are extremely concrete references, and focus heavily on immutability and use of hashes as identifiers; these layers are the “timeless” parts of the stack, because they leave no ambiguity and are simple serializable formats. The higher level layers are increasingly expressive, but also require increasing amount of interpretation.

- **Layer 0: Identifying Content** — simple, static identifiers for snapshots of filesystems.
- **Layer 1: Identifying Computation** — scripts, plus explicit declarations of needed input filesystem snapshots, and selected paths which should be snapshotted and kept as outputs.
- **Layer 2: Computation Graphs** — statically represented pipelines, using multiple isolated computations (each with independent, declarative environments) to build complex outputs.
- **Layer 3: Planners** — use any tools you want to generate Layer 2 pipelines! The Timeless Stack has standard bring import and export APIs, and you can compute Layer 2 however you like!

At the bottom of the content area, there is a note about version control and a statement about the layers being the foundation for the system.

**The Layers, in detail**