



---

# SUPPORTING CONTAINER IMAGES ON SEL4

---

USING KRY10 OS

---

**Alison Felizzi**

Software Engineer - Kry10



0

# WHAT IS A CONTAINER?

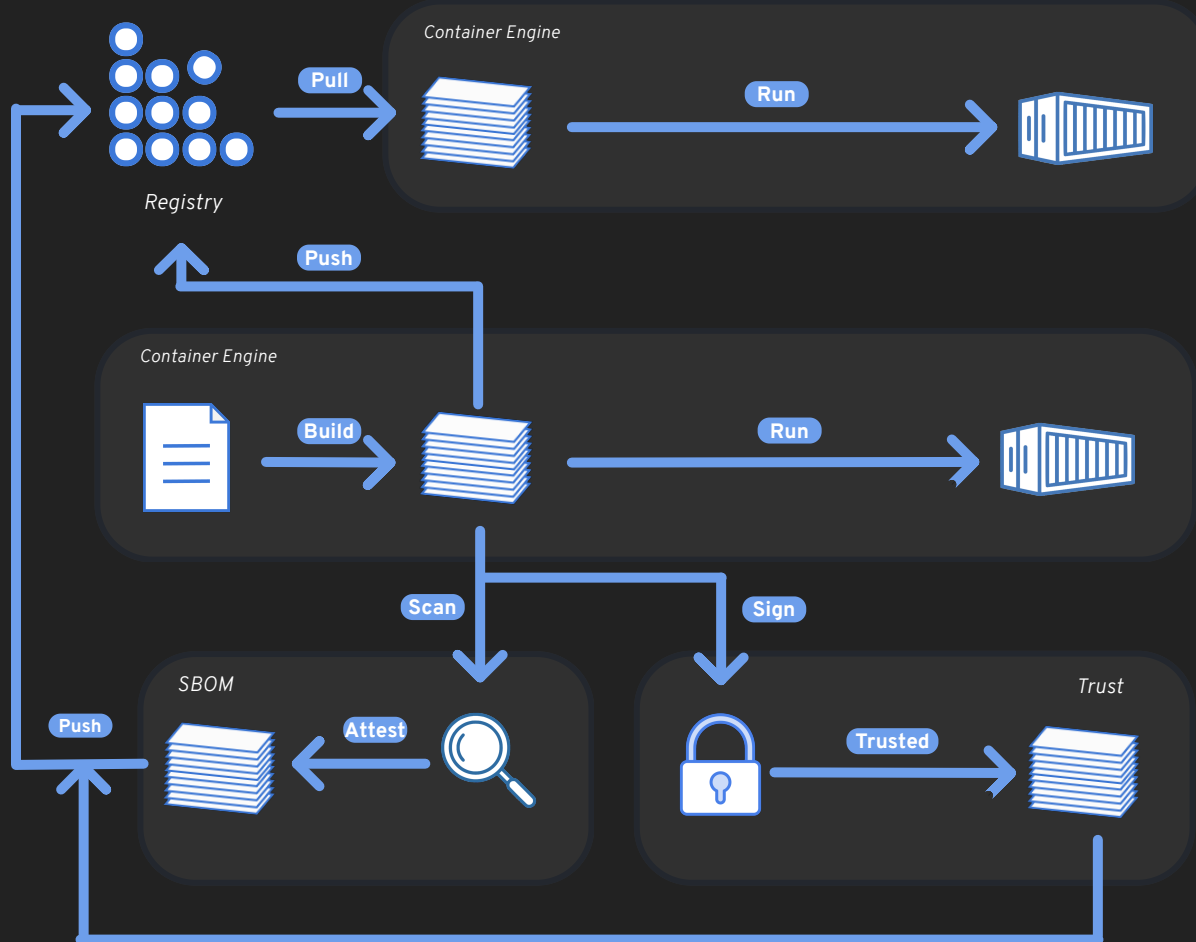
1. BUILD SYSTEM

2. PACKAGING SYSTEM

3. OS VIRTUALISATION MECHANISM

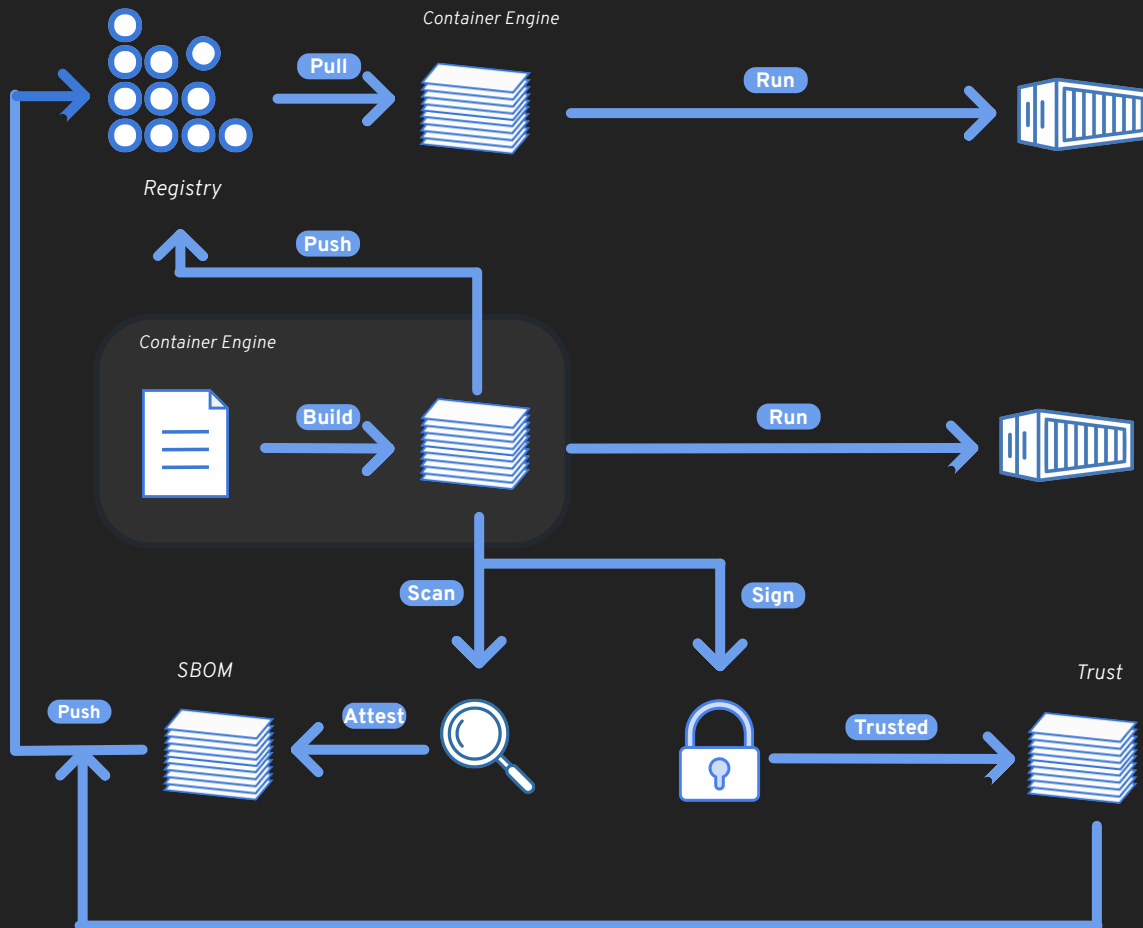
# WHAT IS A CONTAINER?

## A CONTAINER LIFECYCLE



# WHAT IS A CONTAINER?

## A CONTAINER LIFECYCLE

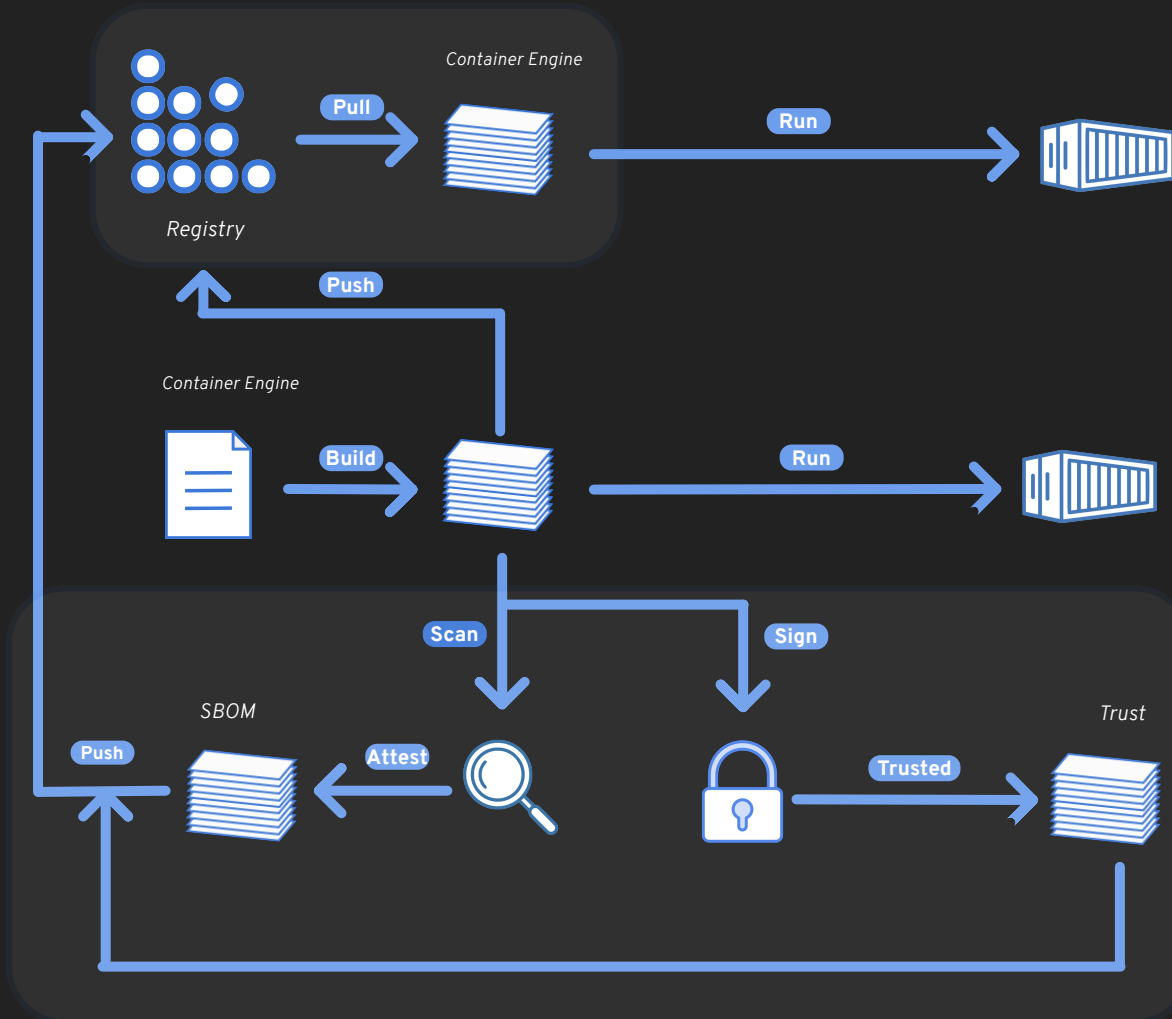


## 1. BUILD SYSTEM

A mechanism for assembling images with their build and runtime dependencies.

# WHAT IS A CONTAINER?

## A CONTAINER LIFECYCLE

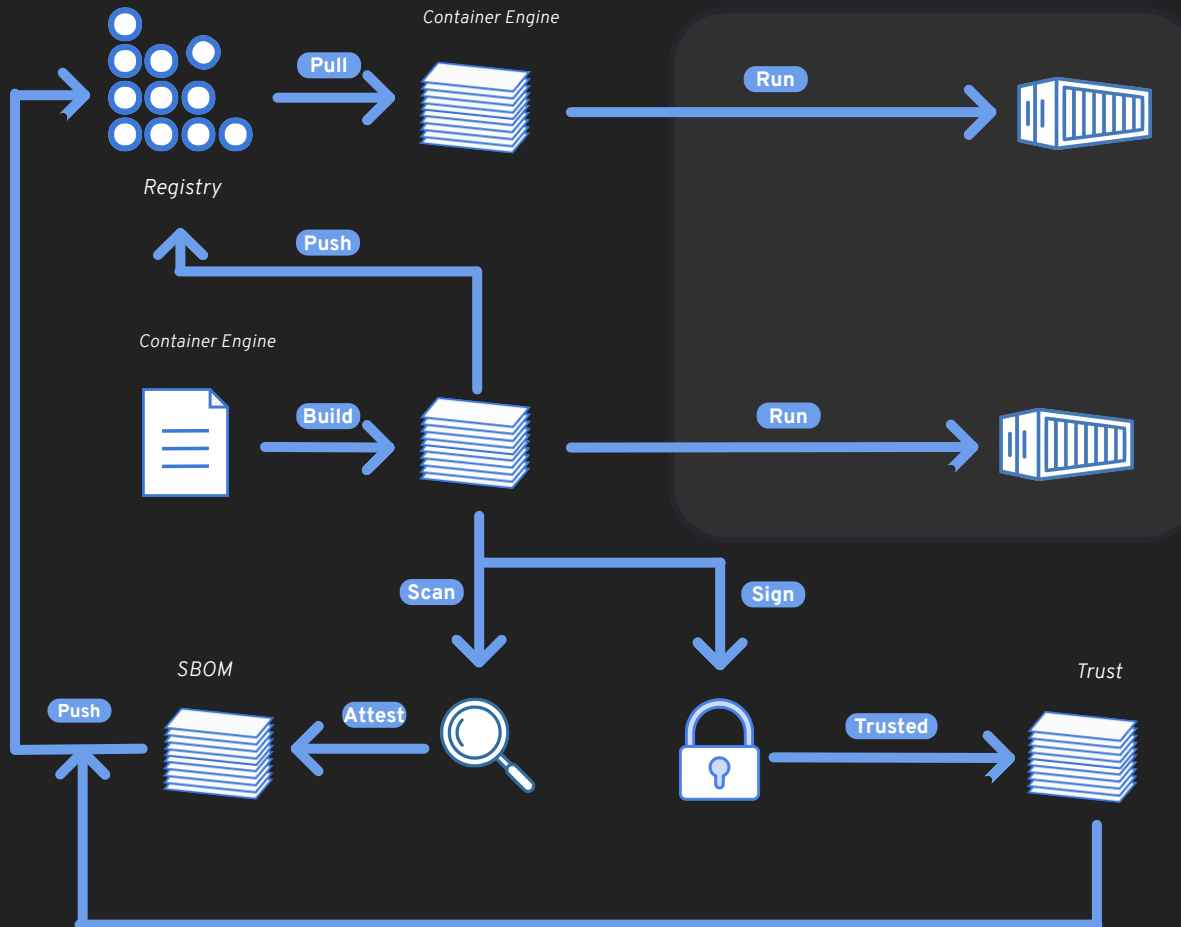


## 2. PACKAGING SYSTEM

A mechanism to push and pull images across local, remote, development and production environments.

# WHAT IS A CONTAINER?

## A CONTAINER LIFECYCLE

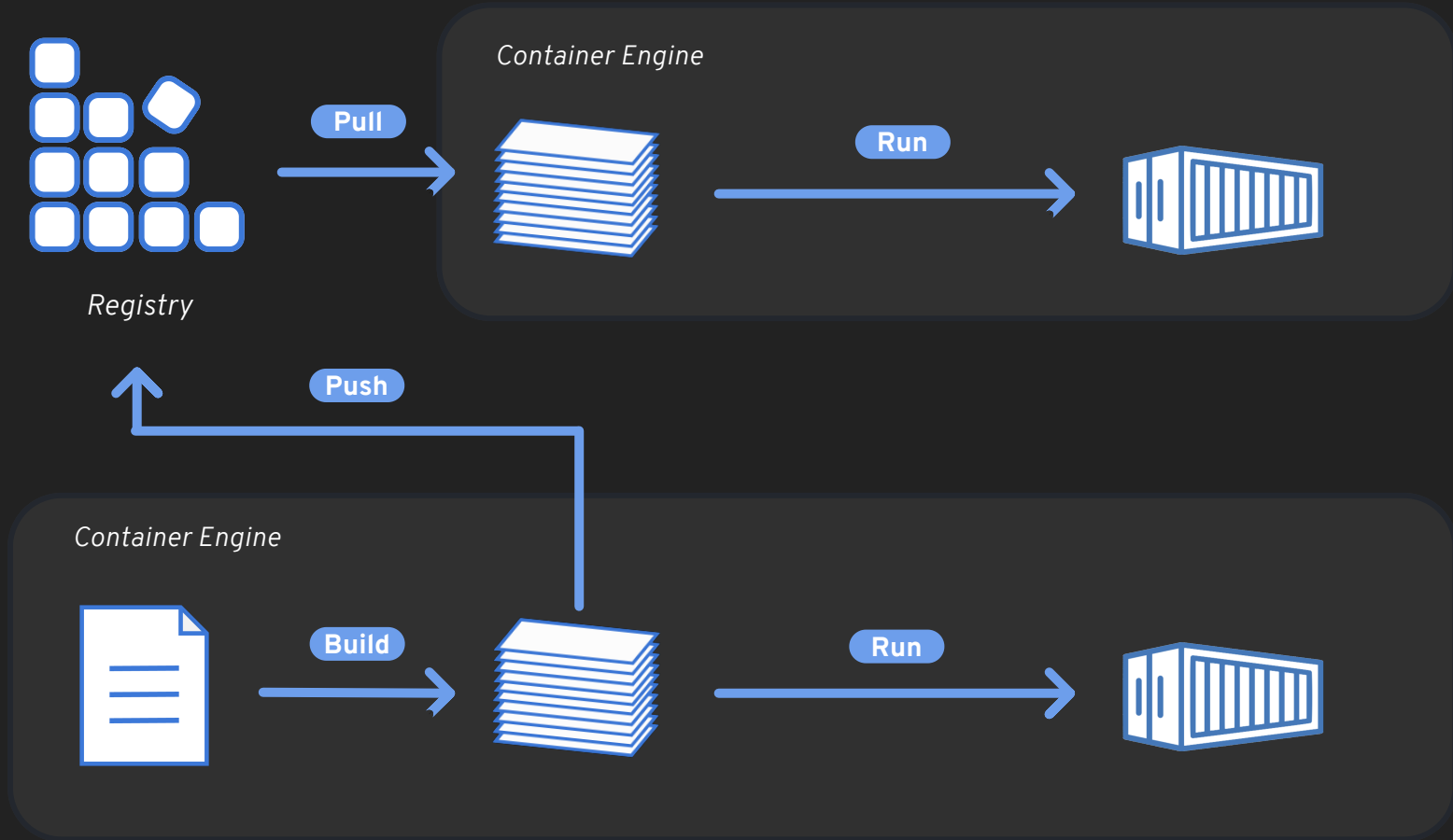


## 3. OS VIRTUALISATION

A mechanism that bundles an application, its configurations, libraries and dependencies into its own isolated namespace, keeping it independent of other environments running in parallel.

# WHAT IS A CONTAINER?

## A CONTAINER LIFECYCLE



**WHY?**



# WHY?

## FRICTIONS WITH CURRENT SEL4 APPROACHES

### BUILD SYSTEM



### CURRENT BUILD SYSTEM SOLUTIONS

- CAMKES CMAKE
- MICROKIT BYO BUILD SYSTEM
- DOCKER SHELLS
- NIX

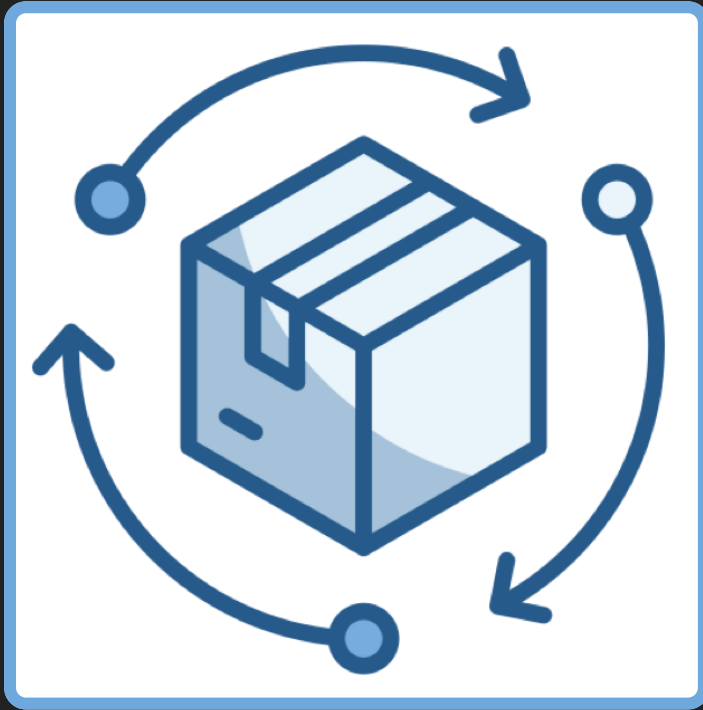
### CHALLENGES

- CHALLENGING MANAGING DIFFERENT TOOLING PRE-REQS
- BUILDS EXIST WITHIN THE CONTEXT OF A FULL SYSTEM SOURCE TREE
- BUILD SYSTEMS SIGNIFICANTLY DIFFER

# WHY?

## FRICTIONS WITH CURRENT SEL4 APPROACHES

### PACKAGING SYSTEM



### CURRENT PACKAGING SYSTEM SOLUTIONS

- SOURCE TREES
- BINARIES?
- DOESN'T REALLY EXIST

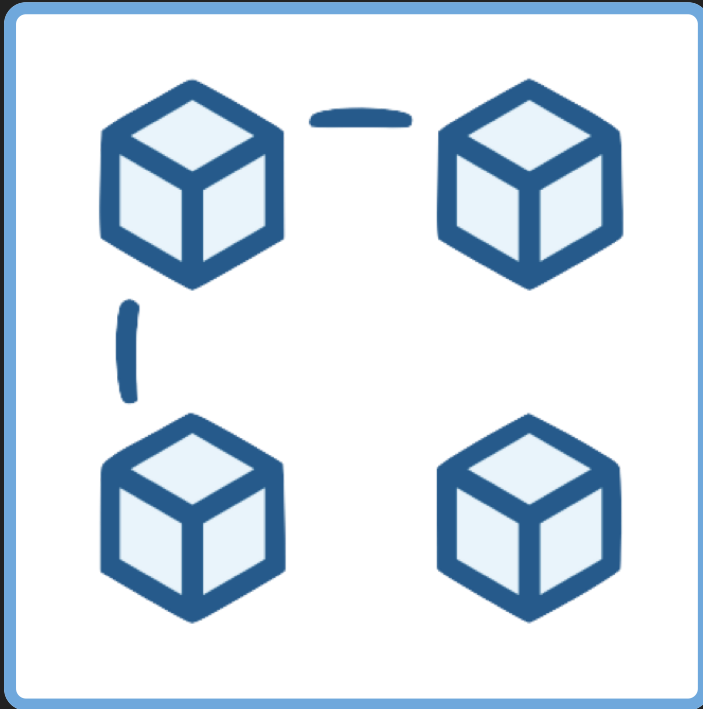
### CHALLENGES

- PASSING AROUND SOURCE TREES CAN BE FRAGILE
- PASSING AROUND BINARIES LACKS RUNTIME REQUIREMENT & DEPENDENCIES

# WHY?

## FRICCTIONS WITH CURRENT SEL4 APPROACHES

### OS VIRTUALISATION MECHANISM



### CURRENT VIRTUALISATION SOLUTIONS

- CAMKES VMM / MICROKIT VMM
- BYO ROOTFS

### CHALLENGES

- CRAFTING YOUR OWN CUSTOM VMM CAN BE DAUNTING
- COMPLEXITY MANAGING A FULL LINUX SUBSYSTEM WHEN BUILDING A CUSTOM ROOTFS
- SLOW ITERATION CYCLE & ERROR PRONE

1

# WHY?

@ KRY10

## MOUMOU TE HURUHURU

Do not waste the feathers of the bird

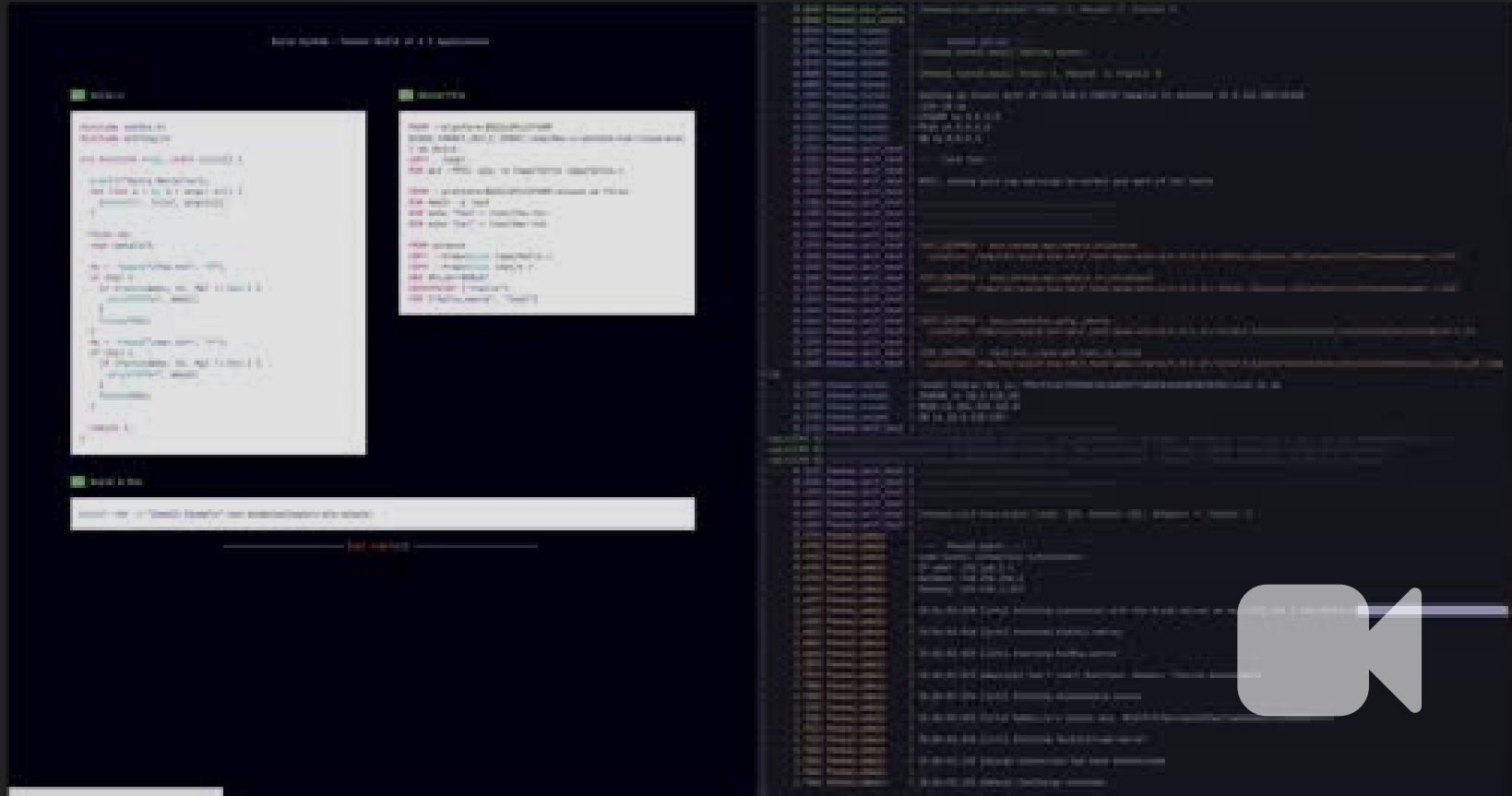
The container ecosystem is a well established and standardised framework that solves these problems.

**SO WHAT  
WOULD THIS  
LOOK LIKE?**

**DEMO**

# DOCKER BUILD OF A C APPLICATION

## BUILD SYSTEM



# DOCKER BUILD OF AN ELIXIR APPLICATION

## BUILD SYSTEM

```
root@b7d7e24037c2:~/elixir-app# docker build -t elixir-app .
[+] Building 0s (1)
[+] --> [Elixir] FROM elixir:1.15.5
[+] --> [Elixir] WORKDIR /app
[+] --> [Elixir] COPY . .
[+] --> [Elixir] RUN mix deps.get
[+] --> [Elixir] RUN mix compile
[+] --> [Elixir] EXPOSE 8080
[+] --> [Elixir] CMD mix run --no-halt
[+] --> [Elixir] COMMIT docker.io/elixir-app:latest
root@b7d7e24037c2:~/elixir-app# docker push elixir-app
The push refers to repository docker.io/elixir-app
[+] elixir-app:latest
root@b7d7e24037c2:~/elixir-app# docker run -p 8080:8080 elixir-app
mix run --no-halt
** (CompileError) lib/elixir_app.ex:3:1: undefined variable 'x'
```

# COMPOSE BUILD OF MULTIPLE APPLICATIONS

## BUILD SYSTEM





# DOCKER BUILD OF A NANOMQ MQTT PACKAGE PACKAGING SYSTEM

The image displays a Docker build process for a Nanomq MQTT package packaging system, split into two panels. The left panel shows the Dockerfile instructions, and the right panel shows the corresponding build output.

**Left Panel (Dockerfile):**

```
FROM ubuntu:20.04 AS base
RUN apt-get update && apt-get install -y \
    curl \
    git \
    nano \
    && rm -rf /var/lib/apt/lists/*

FROM base AS builder
WORKDIR /app
COPY . .
RUN curl -sL https://github.com/nanomq/nanomq/releases/download/v0.10.0/nanomq_0.10.0_Linux_armv7.tar.gz | tar -xzf - && mv nanomq /usr/local/bin/

FROM base AS final
COPY --from=builder /app /app
```

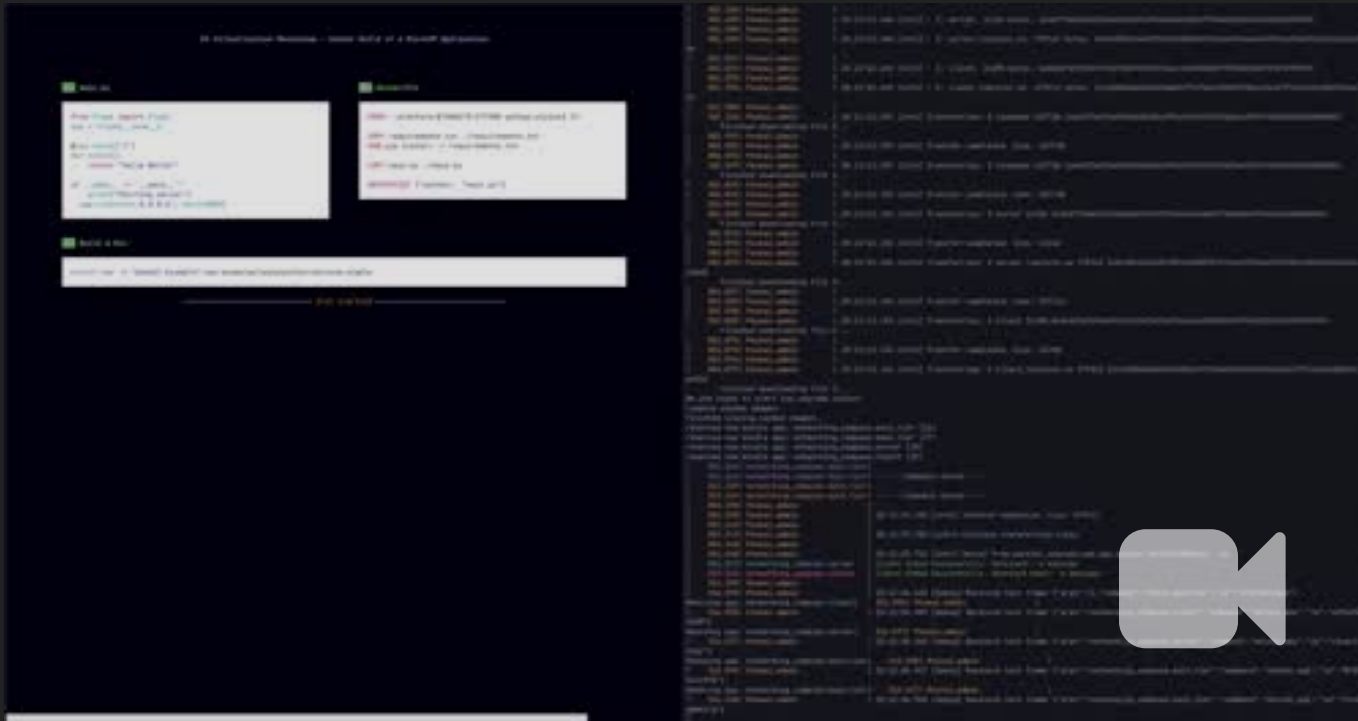
**Right Panel (Build Output):**

```
Step 1/3: FROM ubuntu:20.04 AS base
--
Step 2/3: FROM base AS builder
--
Step 3/3: FROM base AS final
--
```

The output shows the successful completion of the Docker build process, with the final image ready for use. A large white video camera icon is overlaid on the bottom right corner of the screenshot.

# DOCKER BUILD OF A PYTHON APPLICATION

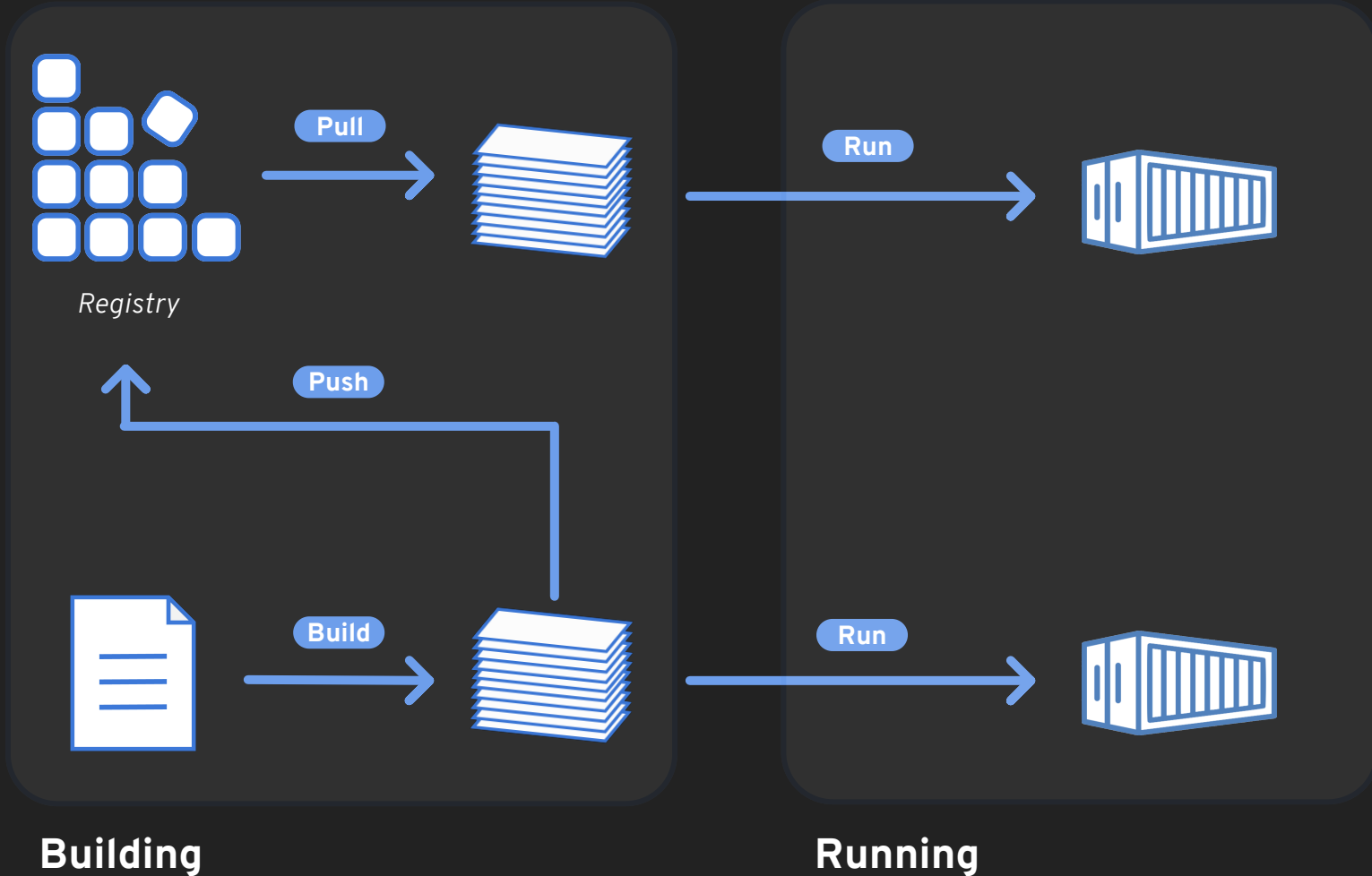
## OS VIRTUALISATION MECHANISM



# **RUNNING AND BUILDING A CONTAINER**

# RUNNING & BUILDING A CONTAINER

## A CONTAINER LIFECYCLE

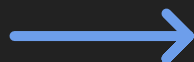
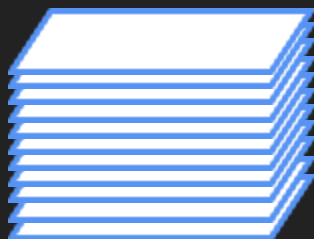


# BUILDING A CONTAINER

```

1 FROM --platform=$BUILDPLATFORM kos/dev-c-aarch64-kos-linux-musl as build
2 COPY . /app/
3 RUN gcc -fPIC -pie -o /app/hello /app/hello.c
4
5 FROM --platform=$BUILDPLATFORM alpine as files
6 RUN mkdir -p /out
7 RUN echo "foo" > /out/foo.txt
8 RUN echo "bar" > /out/bar.txt
9
10 FROM scratch
11 COPY --from=build /app/hello /
12 COPY --from=files /out/* /
13 ENV HELLO="WORLD"
14 ENTRYPOINT ["/hello"]
15 CMD ["hello_world", "test"]

```



```

hello/
├── blobs
│   └── sha256
│       ├── 30a76a4a2be556c8ff8cff68a3e6a68
│       ├── 3e6080001d7b2e588ba7bd7c83b4fe5
│       ├── 5b5387c8849f2140efd643840a2af98
│       ├── 8b895fab1ad99d2ce5a24b59e302974
│       ├── da30198d6af810de0ead95a824b42b1
│       └── ...
└── index.json
oci-layout

```

```

{
  "architecture": "arm64",
  "os": "linux",
  "config": {
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    ],
    "Entrypoint": [
      "hello"
    ],
    "Cmd": [
      "hello_world", "test"
    ],
    "WorkingDir": "/",
    "Layers": {
      "fs": {
        "Type": "layers",
        "DiffIDs": [
          "sha256:30a76a4a2be556c8ff8cff68a3e6a68...",
          "sha256:3e6080001d7b2e588ba7bd7c83b4fe5...",
          "sha256:5b5387c8849f2140efd643840a2af98...",
          "sha256:5b5387c8849f2140efd643840a2af98...",
          "sha256:8b895fab1ad99d2ce5a24b59e302974...",
          "sha256:da30198d6af810de0ead95a824b42b1..."
        ]
      }
    }
  }
}

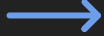
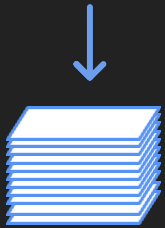
```

# BUILDING A CONTAINER

```

1 FROM --platform=SBUILDPLATFORM kos/dev-c-aarch64-kos-linux-musl as build
2 COPY ./app/.
3 RUN gcc -fPIC -pie -o /app/hello /app/hello.c
4
5 FROM --platform=SBUILDPLATFORM alpine as files
6 RUN mkdir -p /out
7 RUN echo "foo" > /out/foo.txt
8 RUN echo "bar" > /out/bar.txt
9
10 FROM scratch
11 COPY --from=build /app/hello /
12 COPY --from=files /out/* /
13 ENV HELLO="WORLD"
14 ENTRYPOINT ["/hello"]
15 CMD ["hello_world", "test"]

```



```

hello/
├── sha256
│   ├── 36a7f6a42ba556c8ff8c1f58ba3e4e8
│   ├── 3e4080001d7b2e588ba7bd7c83b4fe5
│   ├── 5b5387c8e849f2140e0d62880a2af98
│   ├── 6b895fab1ed9d2c0e5a24b99e302974
│   └── da301986a8f10de0ead95a824b42b1
│   └── ...
├── index.json
└── oci-layout

```

```

"workdir": "/bin",
"cmd": "test",
"args": [
  "test"
],
"entrypoint": [
  "/bin"
],
"mounts": [
  "test"
],
"oci-layout": {
  "mediaType": "application/vnd.oci.image.manifest.v1+json",
  "artifactType": "application/vnd.oci.image.manifest.v1+json",
  "annotations": {
    "org.opencontainers.image.title": "test",
    "org.opencontainers.image.description": "test",
    "org.opencontainers.image.created": "2023-08-24T10:00:00Z",
    "org.opencontainers.image.version": "1.0.0",
    "org.opencontainers.image.revision": "1",
    "org.opencontainers.image.source": "https://github.com/containers/buildah",
    "org.opencontainers.image.url": "https://github.com/containers/buildah",
    "org.opencontainers.image.documentation": "https://github.com/containers/buildah",
    "org.opencontainers.image.vendor": "Red Hat",
    "org.opencontainers.image.base.name": "alpine",
    "org.opencontainers.image.base.version": "3.18.0",
    "org.opencontainers.image.base.digest": "sha256:36a7f6a42ba556c8ff8c1f58ba3e4e83e4080001d7b2e588ba7bd7c83b4fe55b5387c8e849f2140e0d62880a2af986b895fab1ed9d2c0e5a24b99e302974da301986a8f10de0ead95a824b42b1"
  },
  "manifests": [
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "digest": "sha256:36a7f6a42ba556c8ff8c1f58ba3e4e83e4080001d7b2e588ba7bd7c83b4fe55b5387c8e849f2140e0d62880a2af986b895fab1ed9d2c0e5a24b99e302974da301986a8f10de0ead95a824b42b1",
      "platform": {
        "architecture": "aarch64",
        "os": "linux"
      }
    }
  ]
}

```

## CHALLENGES

### BUILD SYSTEM

- Not interfering with user builds
- Supporting a multi-platform approach to tooling

### PACKAGING SYSTEM

- Capturing the runtime dependencies of the application e.g. rootfs, command line, environment variables

# RUNNING A CONTAINER

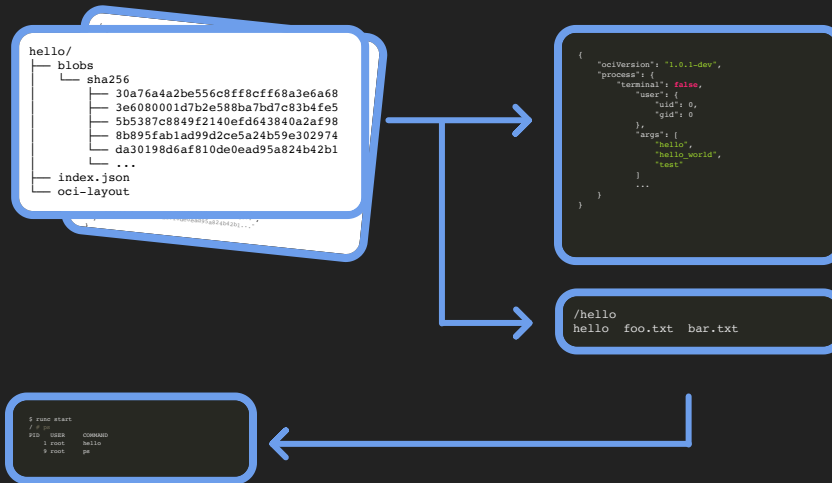
```
hello/  
├── blobs  
│   └── sha256  
│       ├── 30a76a4a2be556c8ff8cff68a3e6a68  
│       ├── 3e6080001d7b2e588ba7bd7c83b4fe5  
│       ├── 5b5387c8849f2140efd643840a2af98  
│       ├── 8b895fab1ad99d2ce5a24b59e302974  
│       └── da30198d6af810de0ead95a824b42b1  
│       └── ...  
└── index.json  
    └── oci-layout
```

```
{  
  "ociVersion": "1.0.1-dev",  
  "process": {  
    "terminal": false,  
    "user": {  
      "uid": 0,  
      "gid": 0  
    },  
    "args": [  
      "hello",  
      "hello_world",  
      "test"  
    ]  
  },  
  ...  
}
```

```
/hello  
hello foo.txt bar.txt
```

```
$ runc start  
/ # ps  
PID  USER  COMMAND  
1   root   hello  
9   root   ps
```

# RUNNING A CONTAINER



## CHALLENGES

### PACKAGING SYSTEM

- Ensuring the runtime state can be unravelled and appropriately loaded when executing the image.

### OS VIRTUALISATION MECHANISM

- Respecting the original container specification whilst injecting any of our own runtime functionality.



# RUNNING AND BUILDING A CONTAINER

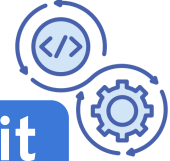
MAPPING A CONTAINER IMAGE TO AN  
SEL4 APPLICATION



# BUILDING A CONTAINER

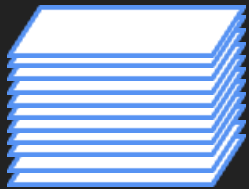
Native

```
1 FROM --platform=$BUILDPLATFORM kos/dev-c-  
  kos-linux-musl as build  
2 COPY ./app/.  
3 RUN gcc -fPIC -pie -o /app/hello /app/hell  
4  
5 FROM --platform=$BUILDPLATFORM alpine as  
6 RUN mkdir -p /out  
7 RUN echo "foo" > /out/foo.txt  
8 RUN echo "bar" > /out/bar.txt  
9  
10 FROM scratch  
11 COPY --from=build /app/hello /  
12 COPY --from=files /out/* /  
13 ENV HELLO="WORLD"  
14 ENTRYPOINT ["/hello"]  
15 CMD ["hello_world", "test"]
```



koskit

buildkit

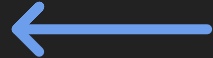
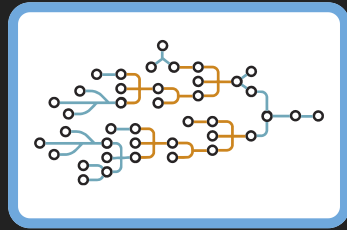


```
{  
  "architecture": "arm64",  
  "os": "linux",  
  "config": {  
    "env": [  
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",  
    ],  
    "entrypoint": [  
      "hello",  
    ],  
    "cmd": [  
      "hello_world", "test"  
    ],  
    "workingdir": "/"  
  },  
}
```

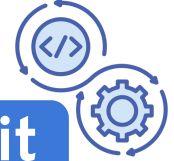
```
hello/  
  blobs  
    sha256  
      30a76a4a2be556c8ff8c8ff68a3e6a68  
      3e6080001d7b2e588ba7bd7c83b4fe5  
      5b5387c8849f2140efd643840a2af98  
      8b895fab1ad99d2ce5a24b59e302974  
      da30198d6af810de0ead95a824b42b1  
    ...  
  index.json  
  oci-layout
```

# BUILDING A CONTAINER

Native

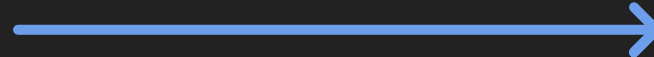
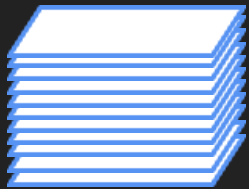


- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts



**koskit**

**buildkit**

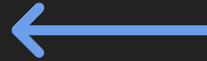
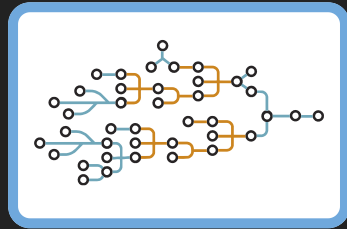


```
{
  "architecture": "arm64",
  "os": "linux",
  "config": {
    "env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    ],
    "entrypoint": [
      "hello"
    ],
    "cmd": [
      "hello_world", "test"
    ],
    "workingdir": "/"
  },
}
```

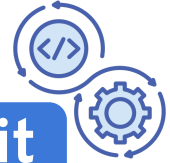
```
hello/
├── blobs
│   ├── sha256
│   │   ├── 30a76a4a2be556c8ff8c8ff68a3e6a68
│   │   ├── 3e6080001d7b2e588ba7bd7c83b4fe5
│   │   ├── 5b5387c8849f2140efd643840a2af98
│   │   ├── 8b895fab1ad99d2ce5a24b59e302974
│   │   └── da30198d6af810de0ead95a824b42b1
│   └── ...
└── index.json
oci-layout
```

# BUILDING A CONTAINER

Native

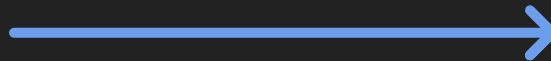
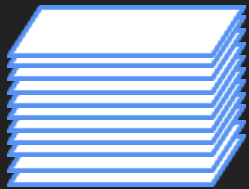


- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts



**koskit**

**buildkit**

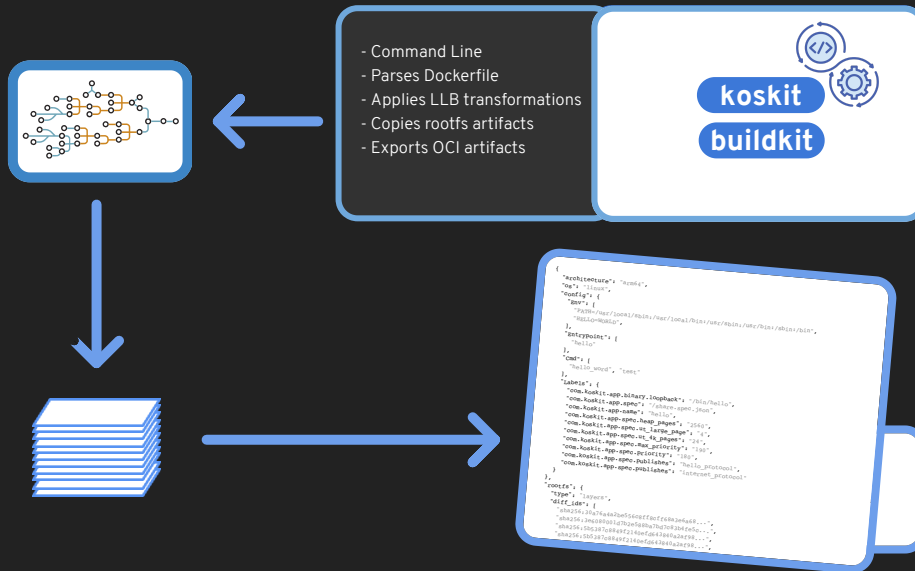


```
{
  "architecture": "arm64",
  "os": "linux",
  "config": {
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "HELLO=WORLD",
    ],
    "Entrypoint": [
      "hello"
    ],
    "Cmd": [
      "hello_word", "test"
    ],
    "Labels": {
      "com.koskit.app.binary.loopback": "/bin/hello",
      "com.koskit.app.spec": "/share.spec.json",
      "com.koskit.app.name": "hello",
      "com.koskit.app.spec.heap_pages": "2560",
      "com.koskit.app.spec.ut_large_page": "4",
      "com.koskit.app.spec.ut_4k_pages": "24",
      "com.koskit.app.spec.max_priority": "190",
      "com.koskit.app.spec.priority": "180",
      "com.koskit.app.spec.publishes": "hello_protocol",
      "com.koskit.app.spec.publishes": "internet_protocol"
    }
  },
  "rootfs": {
    "type": "layers",
    "diff_ids": [
      "sha256:30a76a4a2be556c8ff8cff68a3e6a68...",
      "sha256:3e608001d7b2e588ba7bd7c83b4fe5c...",
      "sha256:5b5387c8849f2140efd643840a2af98..."
    ]
  }
}
```

e6a68  
b4fe5  
2af98  
02974  
b42b1

# BUILDING A CONTAINER

Native



## BUILD SYSTEM

### CHALLENGES

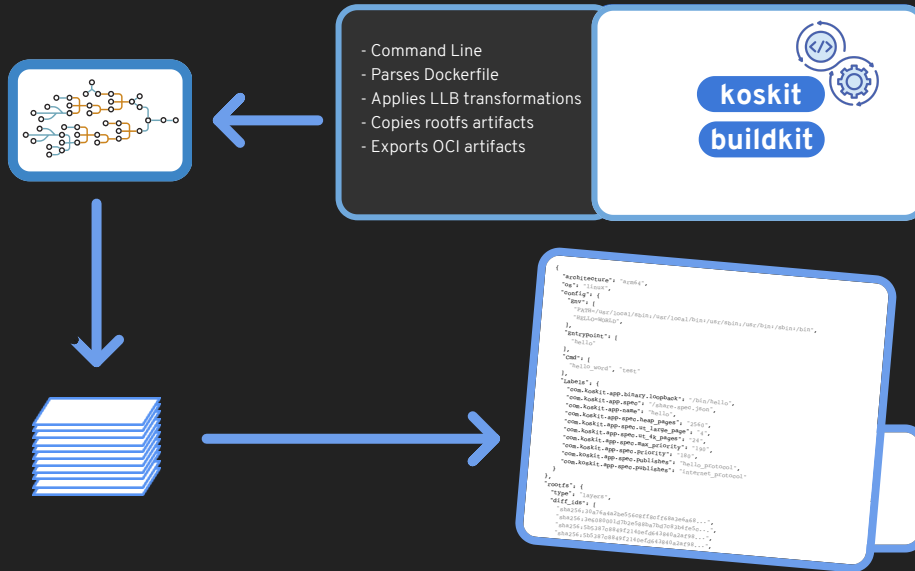
- Not interfering with user builds
- Supporting a multi-platform approach to tooling

### SOLUTIONS

- Uses buildkit's LLB abstraction to transparently construct seL4-geared build definitions.
- koskit automatically imports a platform's SDK toolchain into the build

# BUILDING A CONTAINER

Native



## PACKAGING SYSTEM

### CHALLENGES

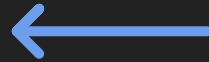
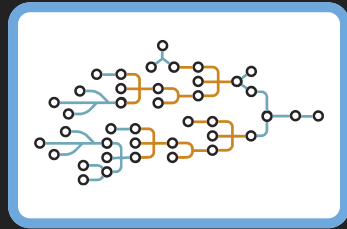
- Capturing the runtime dependencies of the application e.g. rootfs, command line, environment variables
- Ensuring the runtime state can be unravelled and appropriately loaded when executing the image.

### SOLUTIONS

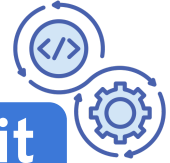
- koskit injects seL4-specific configuration under namespaced metadata labels and attributes.

# BUILDING A CONTAINER

Native

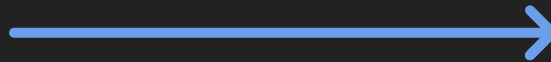
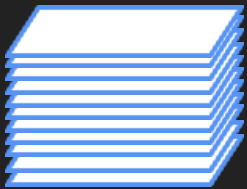


- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts



**koskit**

**buildkit**

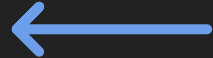
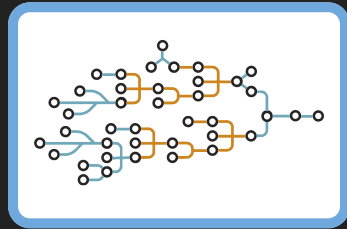


```
{
  "architecture": "arm64",
  "os": "linux",
  "config": {
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "HELLO=WORLD",
    ],
    "Entrypoint": [
      "hello"
    ],
    "Cmd": [
      "hello_word", "test"
    ],
    "Labels": {
      "com.koskit.app.binary.loopback": "/bin/hello",
      "com.koskit.app.spec": "/share.spec.json",
      "com.koskit.app.name": "hello",
      "com.koskit.app.spec.heap_pages": "2560",
      "com.koskit.app.spec.ut_large_page": "4",
      "com.koskit.app.spec.ut_4k_pages": "24",
      "com.koskit.app.spec.max_priority": "190",
      "com.koskit.app.spec.priority": "180",
      "com.koskit.app.spec.publishes": "hello_protocol",
      "com.koskit.app.spec.publishes": "internet_protocol"
    }
  },
  "rootfs": {
    "type": "layers",
    "diff_ids": [
      "sha256:30a76a4a2be556c8ff8cff68a3e6a68...",
      "sha256:3e608001d7b2e588ba7bd7c83b4fe5c...",
      "sha256:5b5387c8849f2140efd643840a2af98..."
    ]
  }
}
```

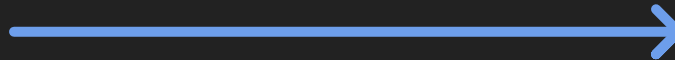
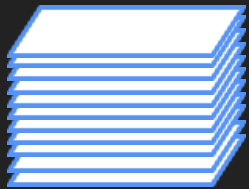
e6a68  
b4fe5  
2af98  
02974  
b42b1

# BUILDING A CONTAINER

Native



- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts



<b>Untyped</b>	0x200000
<b>VSpace</b>	1
<b>TCB</b>	1
<b>CNode</b>	6
<b>Frames</b>	42
<b>Cmd</b>	[/hello, hello_word, test ]
<b>Binary</b>	hello

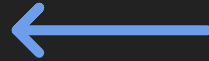
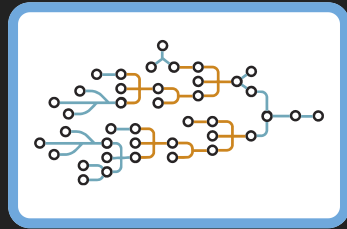
hello.elf

initrd.cpio

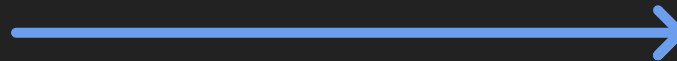
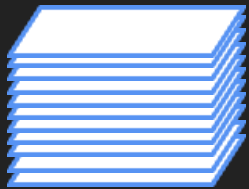


# BUILDING A CONTAINER

microvm



- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts



## initrd.squahfs

```
/bin/kosinit /bin/runc
/container/rootfs
/container/config.json
```

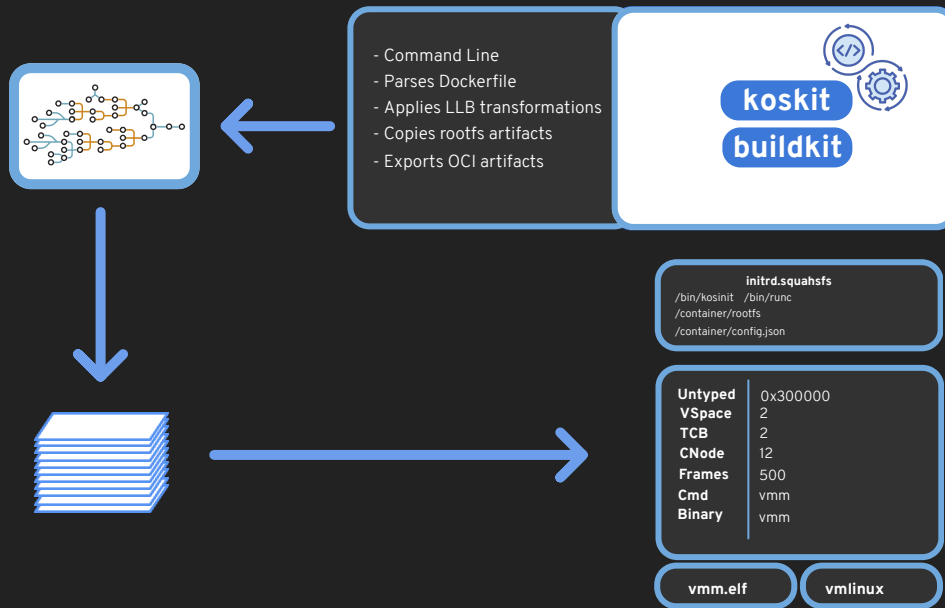
Untyped	0x300000
VSpace	2
TCB	2
CNode	12
Frames	500
Cmd	vmm
Binary	vmm

vmm.elf

vmlinux

# BUILDING A CONTAINER

microvm



## OS VIRTUALISATION MECHANISM

### CHALLENGES

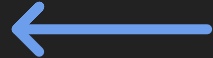
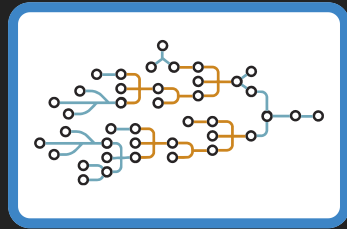
- Respecting the original container specification whilst injecting any of our own runtime functionality.

### SOLUTIONS

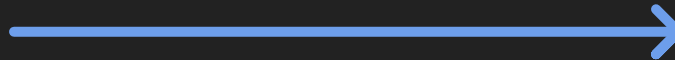
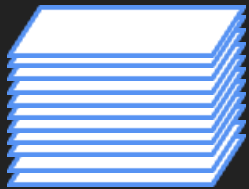
- koskit bundles a custom Linux initrd with its own init task and OCI-compliant JSON configuration.

# BUILDING A CONTAINER

microvm



- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts



## initrd.squahfs

```
/bin/kosinit /bin/runc
/container/rootfs
/container/config.json
```

Untyped	0x300000
VSpace	2
TCB	2
CNode	12
Frames	500
Cmd	vmm
Binary	vmm

vmm.elf

vmlinux

4

# RUNNING A CONTAINER

Native



SEL4

# RUNNING A CONTAINER

Native

ROOT  
SERVER

<b>VSpace</b>	0
<b>TCB</b>	1
<b>CNode</b>	6
<b>Frames</b>	42
<b>Untyped</b>	0x200000
<b>Cmd</b>	[/hello, hello_word, test ]
<b>Binary</b>	hello

hello.elf

initrd.cpio

4

## RUNNING A CONTAINER

Native

ROOT  
SERVER



4

## RUNNING A CONTAINER

Native

ROOT  
SERVER

*Stack*

*Memory Mapped Initrd*

*Heap*

*BSS*

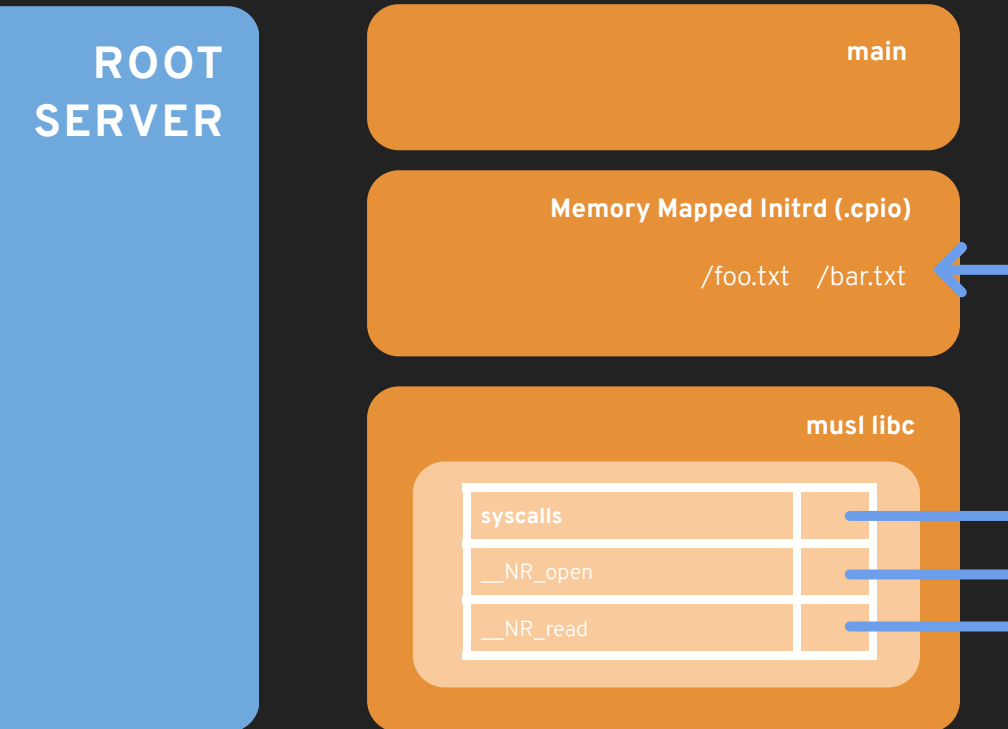
*Data*

*Text*

SEI 4

# RUNNING A CONTAINER

Native





4

## RUNNING A CONTAINER

Native

ROOT  
SERVER

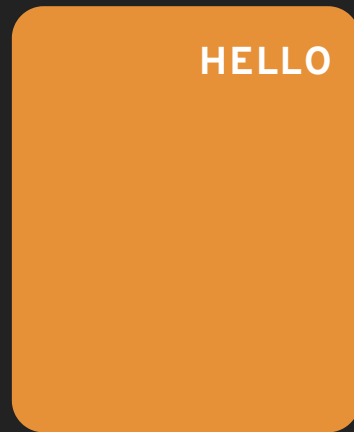
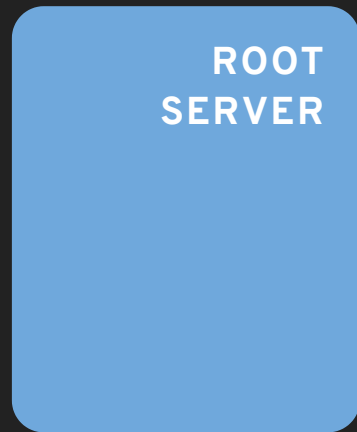
HELLO

SEI 4

4

# RUNNING A CONTAINER

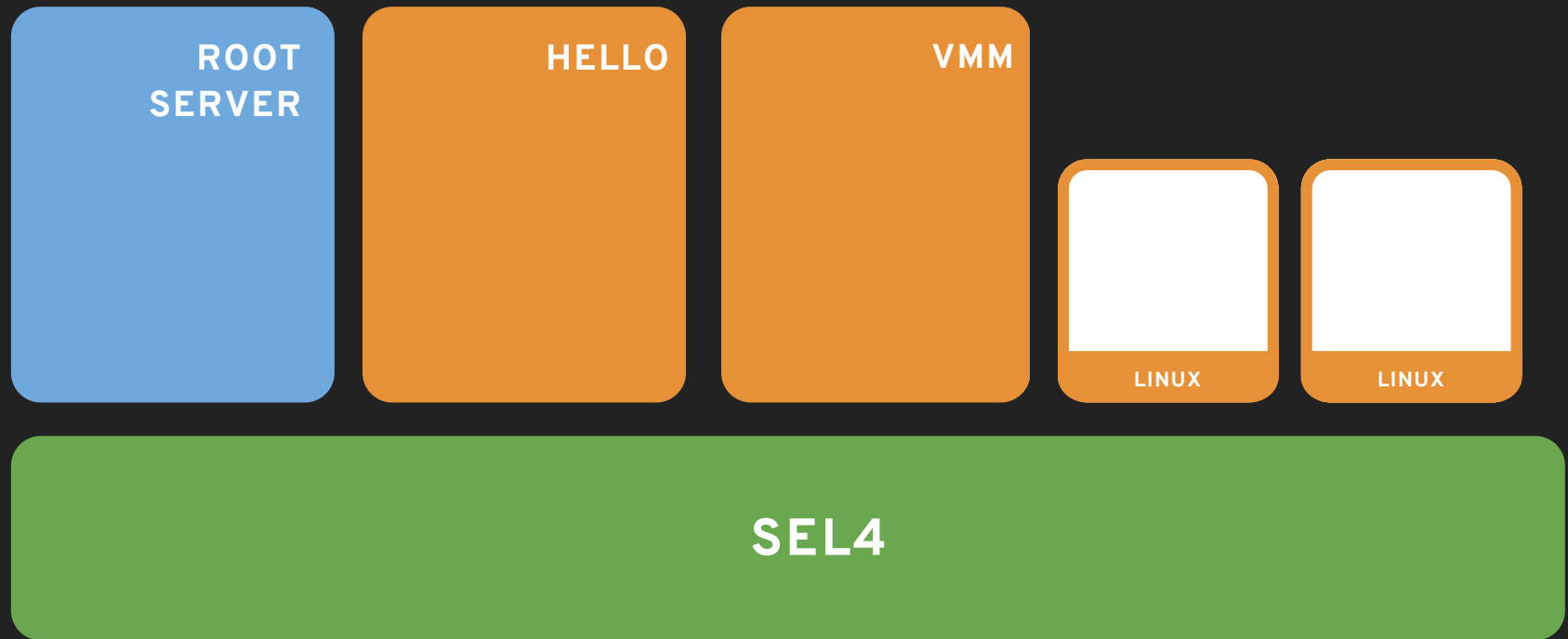
Native



4

# RUNNING A CONTAINER

microvm



# RUNNING A CONTAINER

microvm

ROOT  
SERVER

HELLO

VMM

virtio-blk  
mounts

linux1.squash

linux2.squash

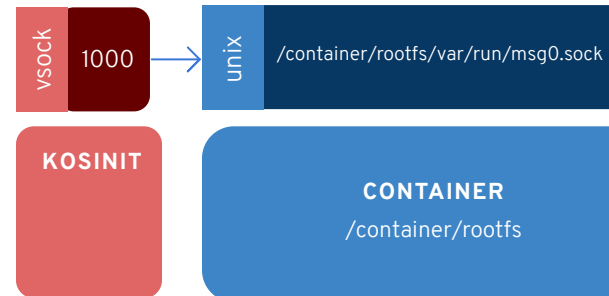
virtio-net  
interfaces

eth0

vsock ports

1000

1001



LINUX 1

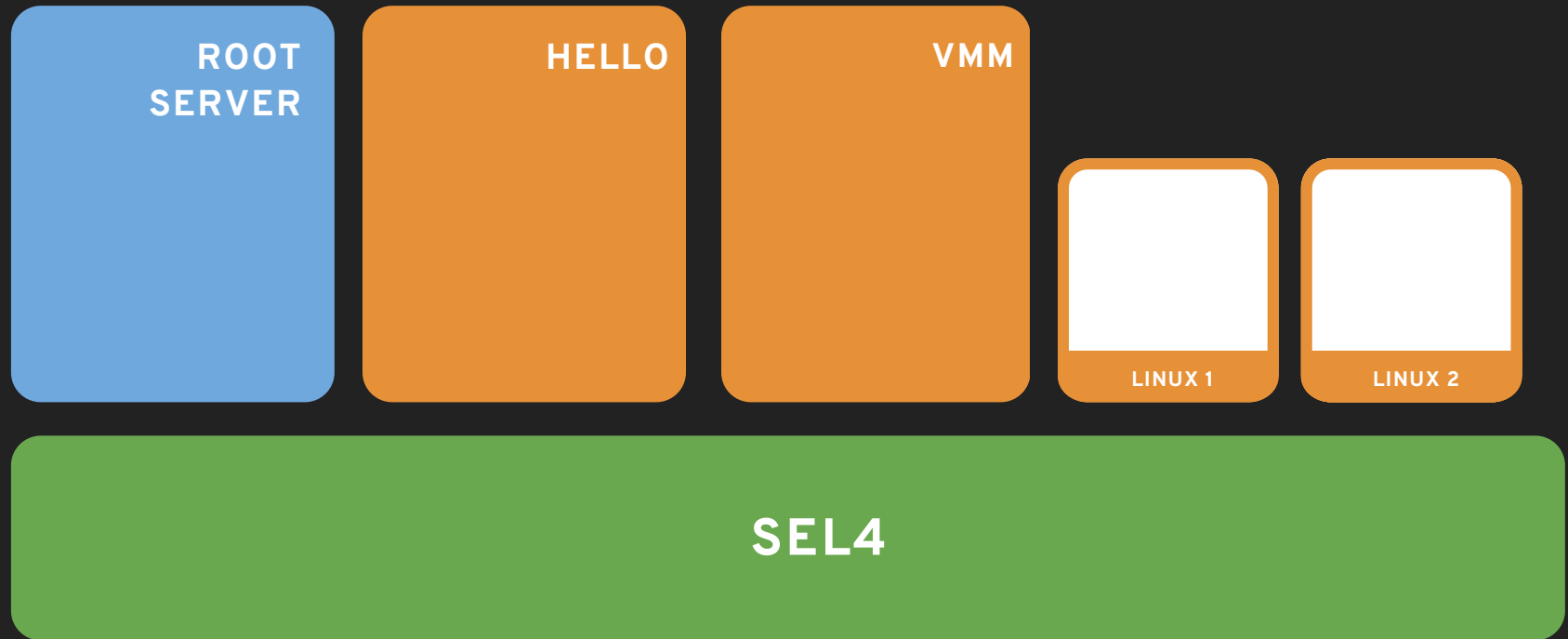
vsock 1001

KOSINIT

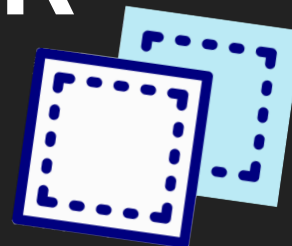
4

# RUNNING A CONTAINER

microvm



# STITCHING IT TOGETHER



# COMPOSE SYSTEMS

```
1 name: example_compose
2
3 services:
4   client:
5     image: kos:apps/client
6     command: ["client", "5050"]
7     depends_on:
8       - server
9     networks:
10      back-tier:
11   server:
12     image: kos:apps/server
13     command: ["server", "5050"]
14     networks:
15      back-tier:
16
17 networks:
18   back-tier:
19     x-msg-server: "Poukai.msg0"
```

ROOT  
SERVER

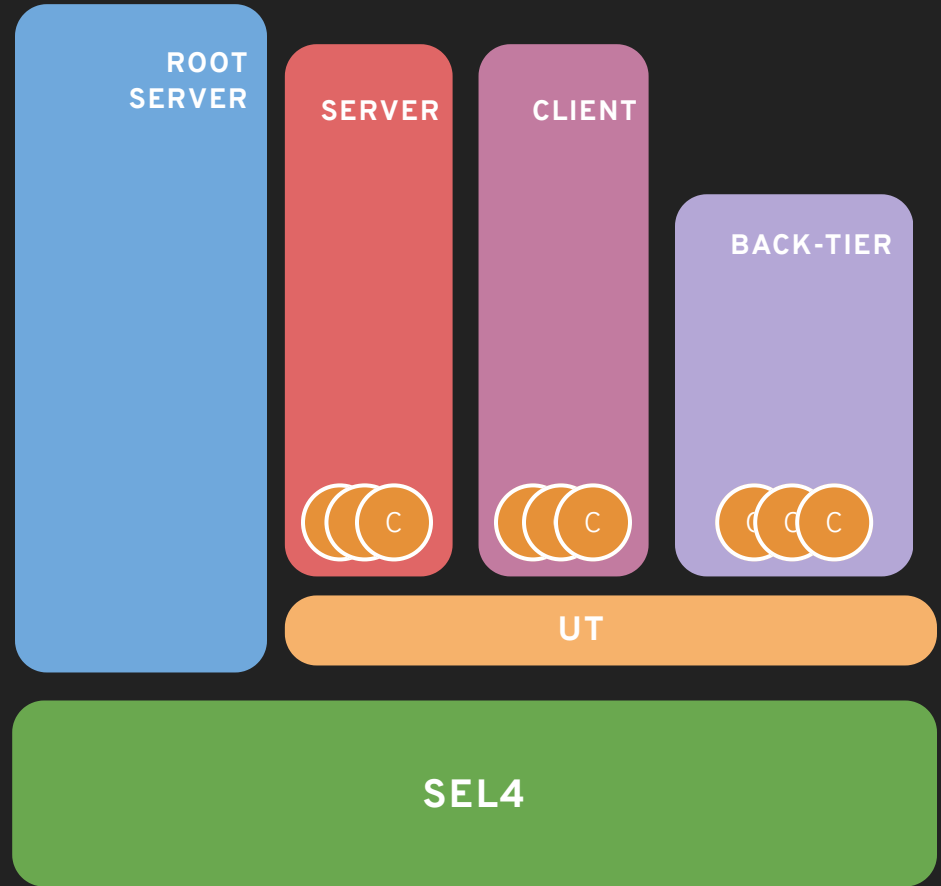
SEL4

5

# COMPOSE SYSTEMS

UP

```
1 name: example_compose
2
3 services:
4   client:
5     image: kos:apps/client
6     command: ["client", "5050"]
7     depends_on:
8       - server
9     networks:
10      back-tier:
11   server:
12     image: kos:apps/server
13     command: ["server", "5050"]
14     networks:
15      back-tier:
16
17 networks:
18   back-tier:
19     x-msg-server: "Poukai.msg0"
```



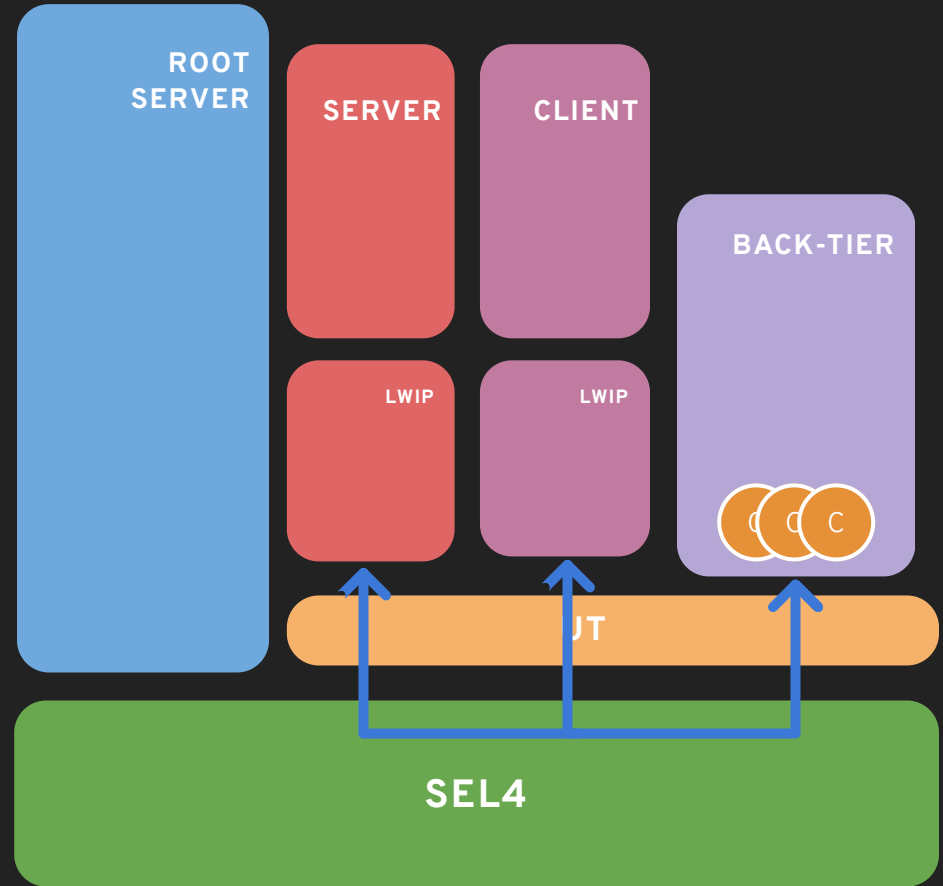


5

# COMPOSE SYSTEMS

UP

```
1 name: example_compose
2
3 services:
4   client:
5     image: kos:apps/client
6     command: ["client", "5050"]
7     depends_on:
8       - server
9     networks:
10      back-tier:
11  server:
12    image: kos:apps/server
13    command: ["server", "5050"]
14    networks:
15      back-tier:
16
17 networks:
18   back-tier:
19     x-msg-server: "Poukai.msg0"
```



5

# COMPOSE SYSTEMS

## DOWN

```
1 name: example_compose
2
3 services:
4   client:
5     image: kos:apps/client
6     command: ["client", "5050"]
7     depends_on:
8       - server
9     networks:
10      back-tier:
11   server:
12     image: kos:apps/server
13     command: ["server", "5050"]
14     networks:
15      back-tier:
16
17 networks:
18   back-tier:
19     x-msg-server: "Poukai.msg0"
```

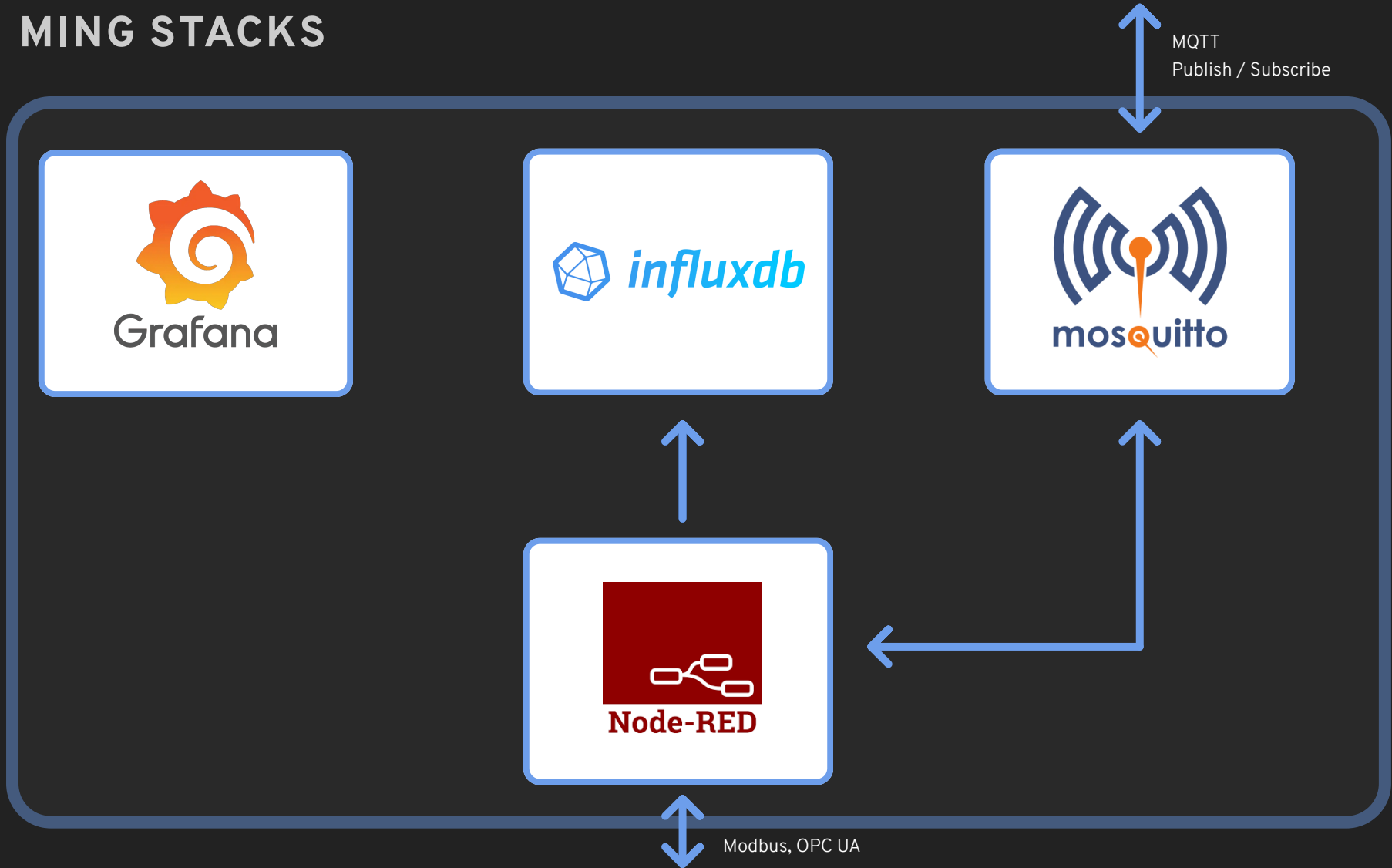
ROOT  
SERVER

SEL4

**NEXT STEPS**

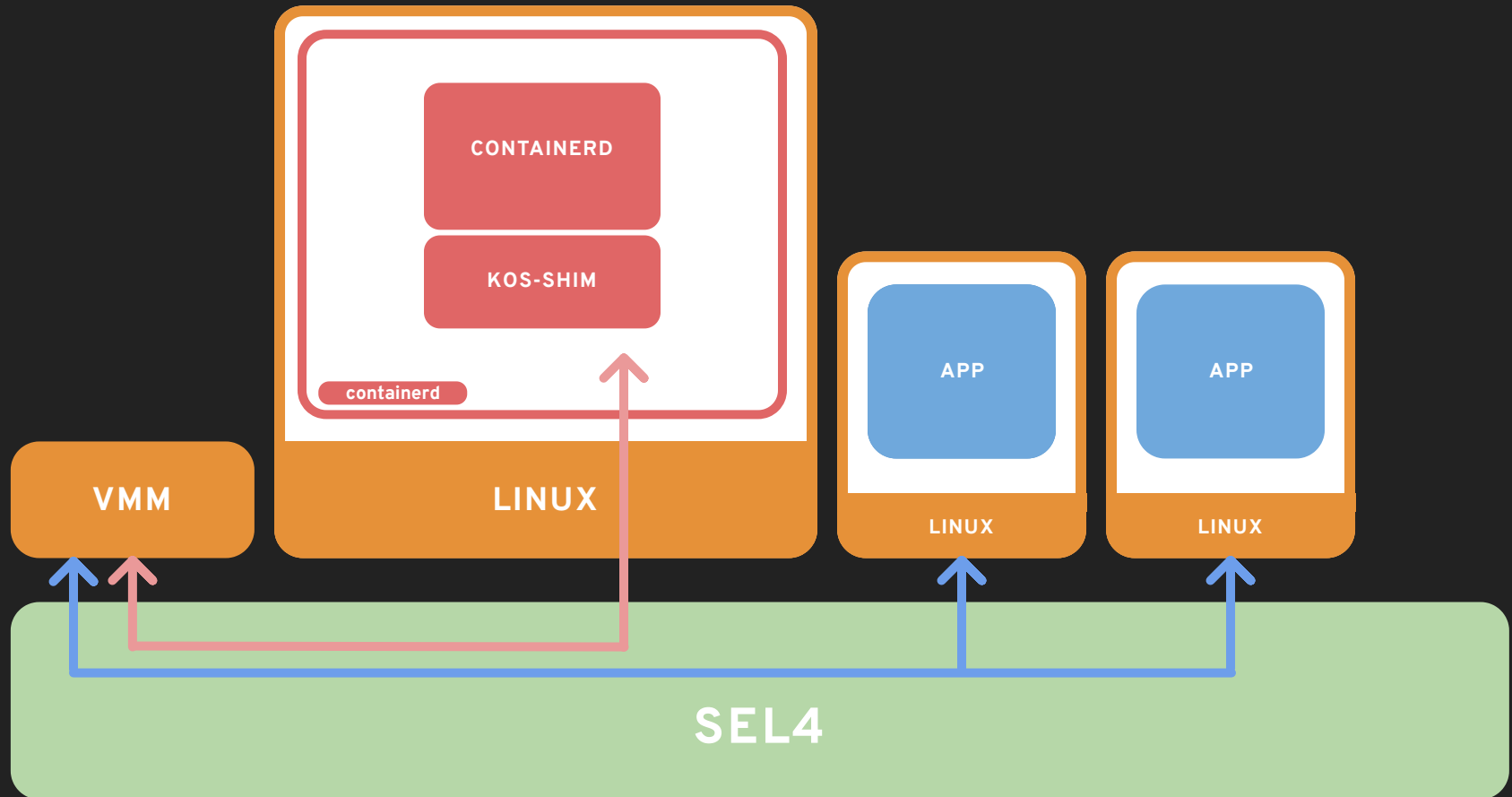
# NEXT STEPS

## MING STACKS



# NEXT STEPS

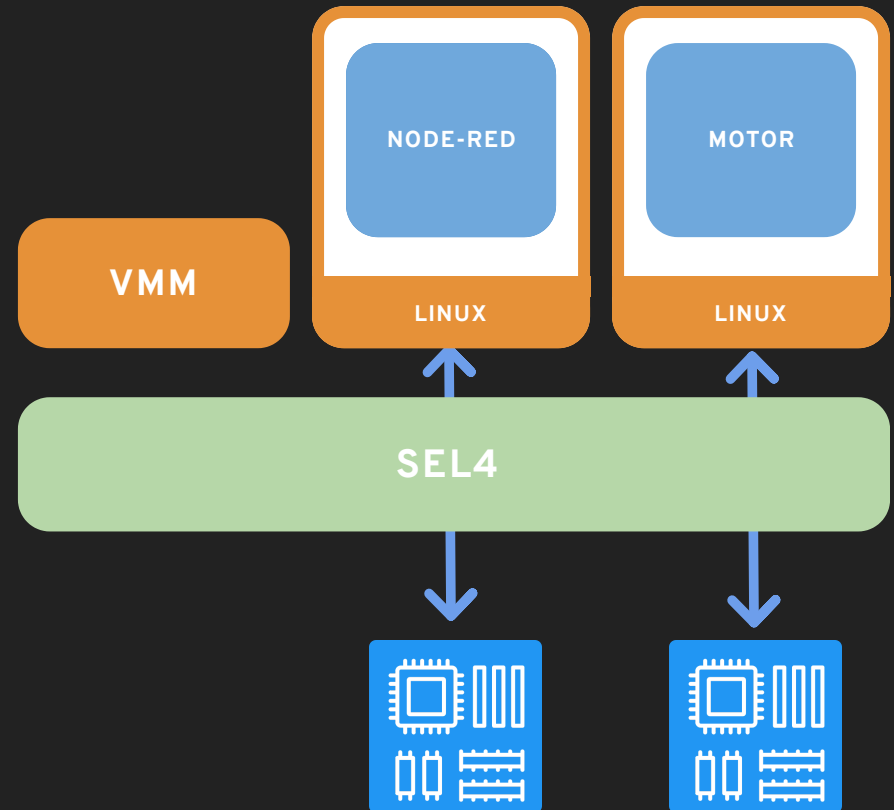
## CONTAINERD CONTROL PLANES



# NEXT STEPS

## DRIVER CONTAINERS

```
1 name: devices-compose
2
3 services:
4   node-red:
5     image: docker:nodered/node-red
6     privileged: true
7     cap_add:
8       - SYS_RAWIO
9     devices:
10      - "/dev/mem:/dev/mem"
11      - "/dev/gpiomem:/dev/gpiomem"
12      - "/dev/i2c-1:/dev/i2c-1"
13   motor:
14     image: kos:my_motor_app/.
15     privileged: true
16     cap_add:
17       - SYS_MODULE
```



**END**