SUPPORTING CONTAINER IMAGES ON SEL4

USING KRY10 OS

Alison Felizzi

Software Engineer - Kry10

WHAT IS A CONTAINER?

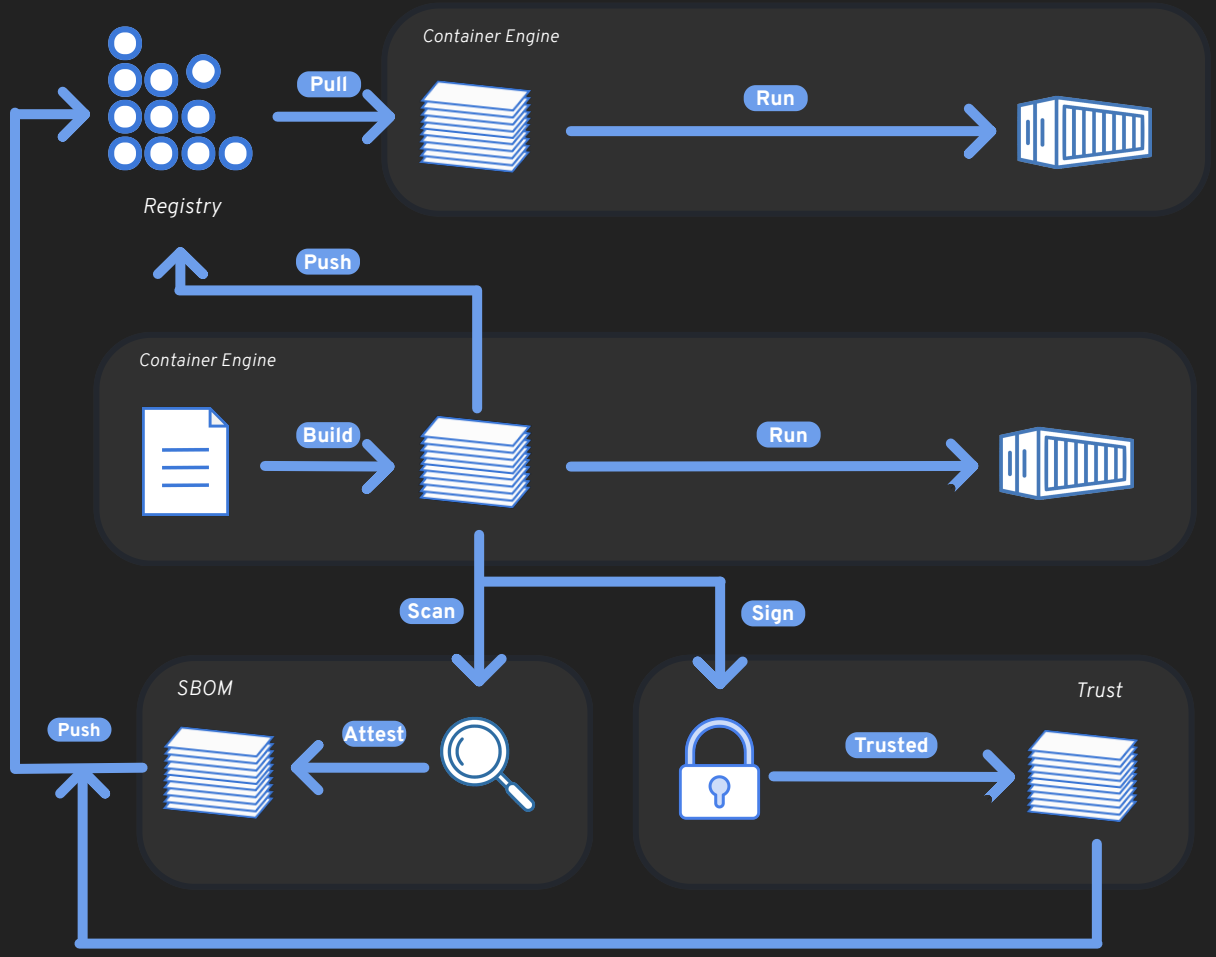
1. BUILD SYSTEM

2. PACKAGING SYSTEM

3. OS VIRTUALISATION MECHANISM

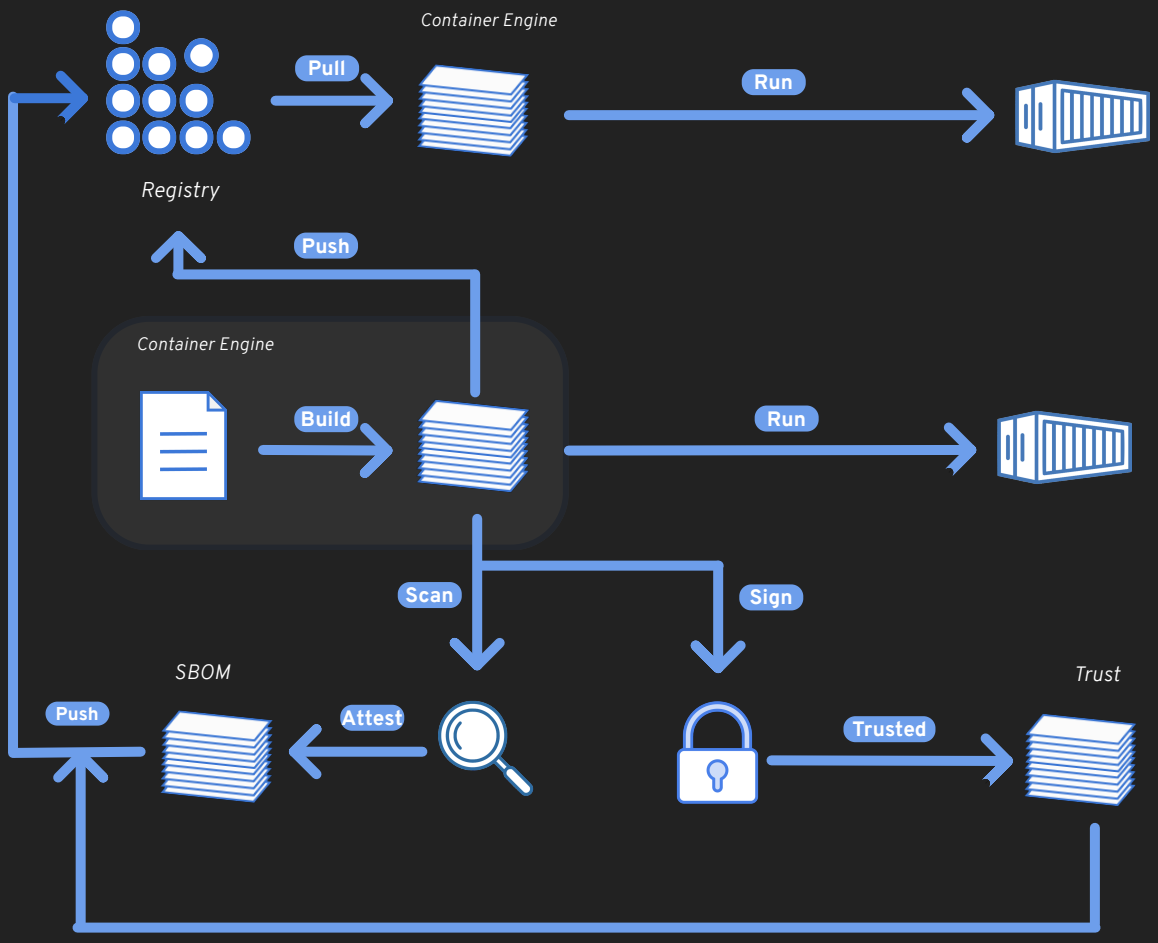
WHAT IS A CONTAINER?

A CONTAINER LIFECYCLE



WHAT IS A CONTAINER?

A CONTAINER LIFECYCLE

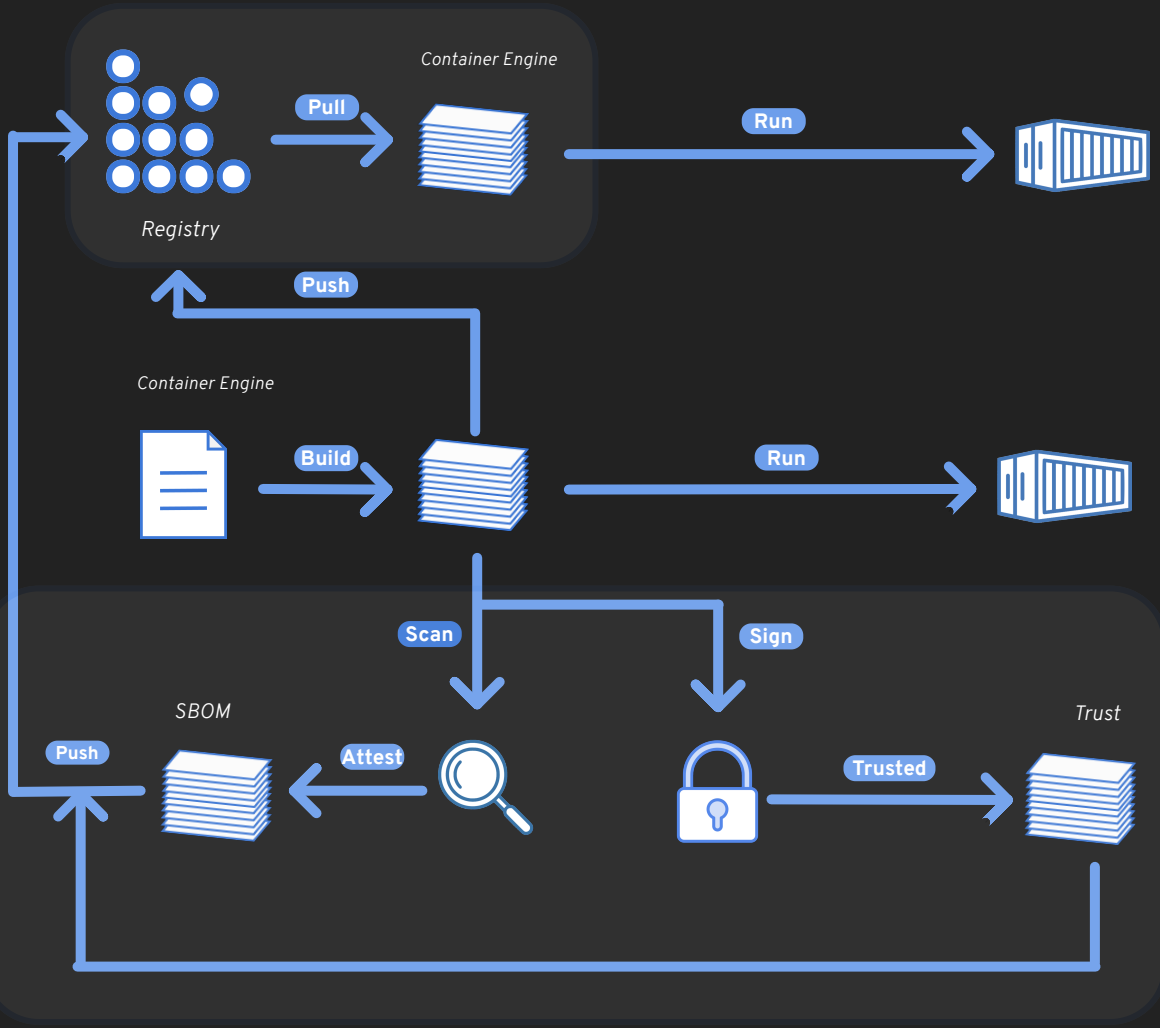


1. BUILD SYSTEM

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

WHAT IS A CONTAINER?

A CONTAINER LIFECYCLE

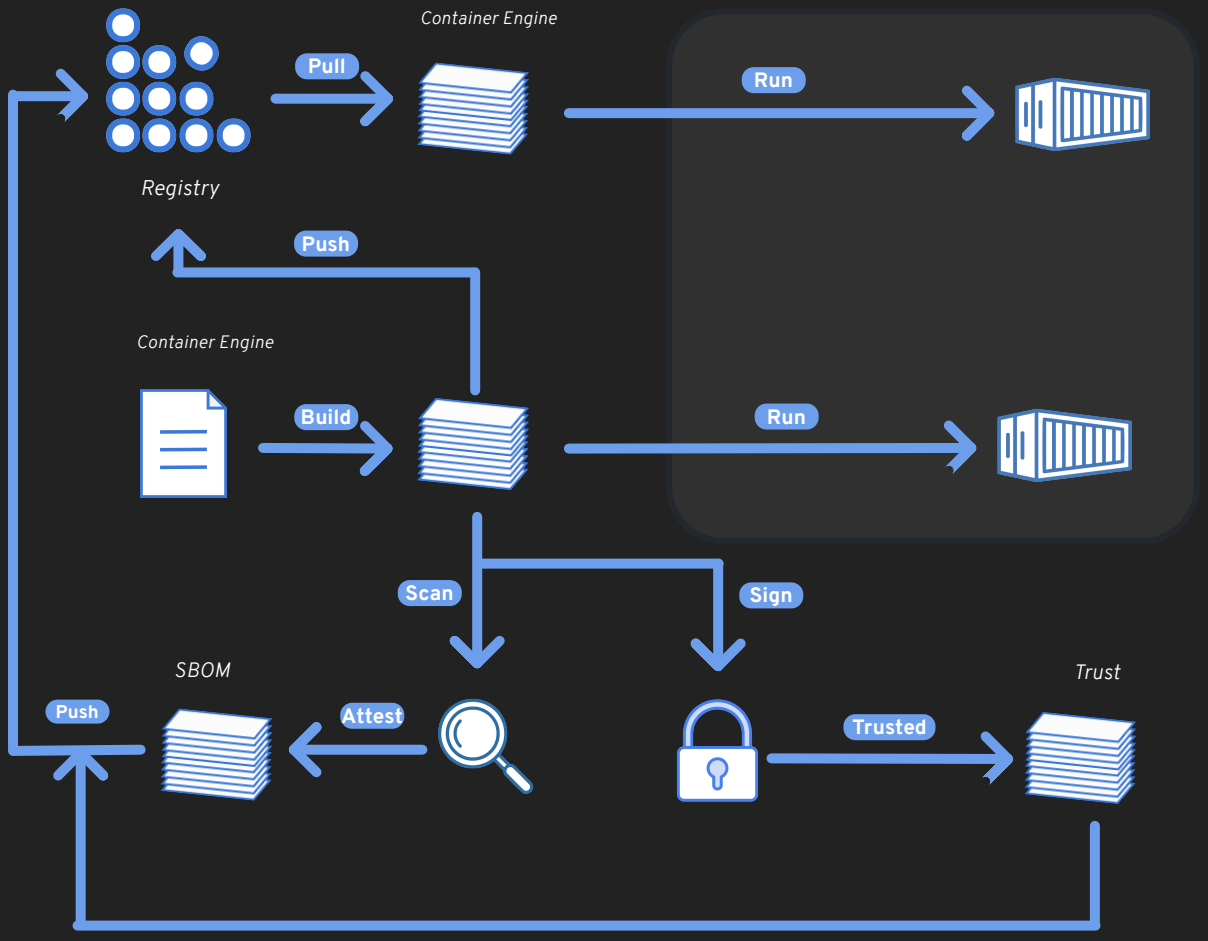


2. PACKAGING SYSTEM

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

WHAT IS A CONTAINER?

A CONTAINER LIFECYCLE

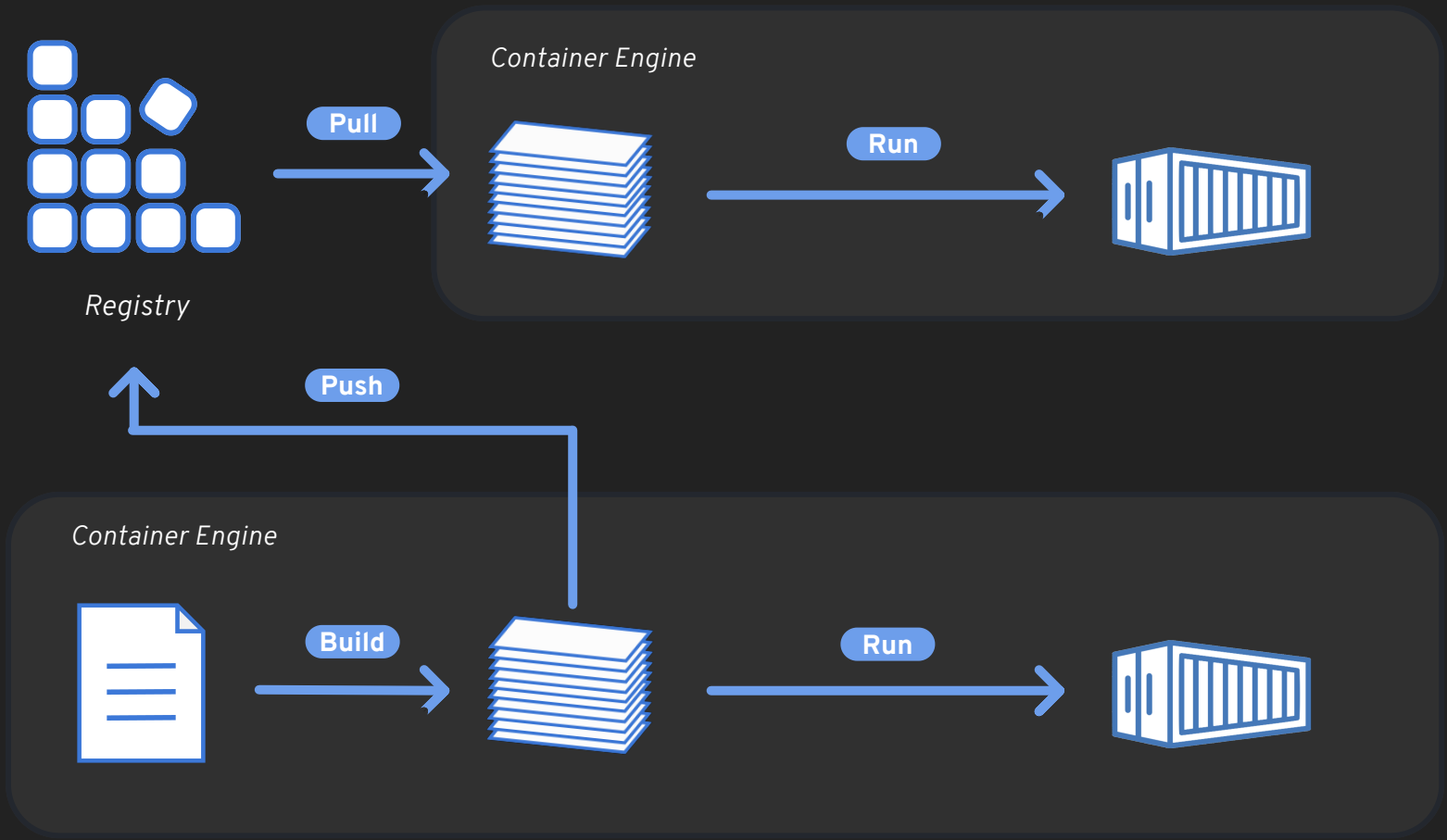


3. OS VIRTUALISATION

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

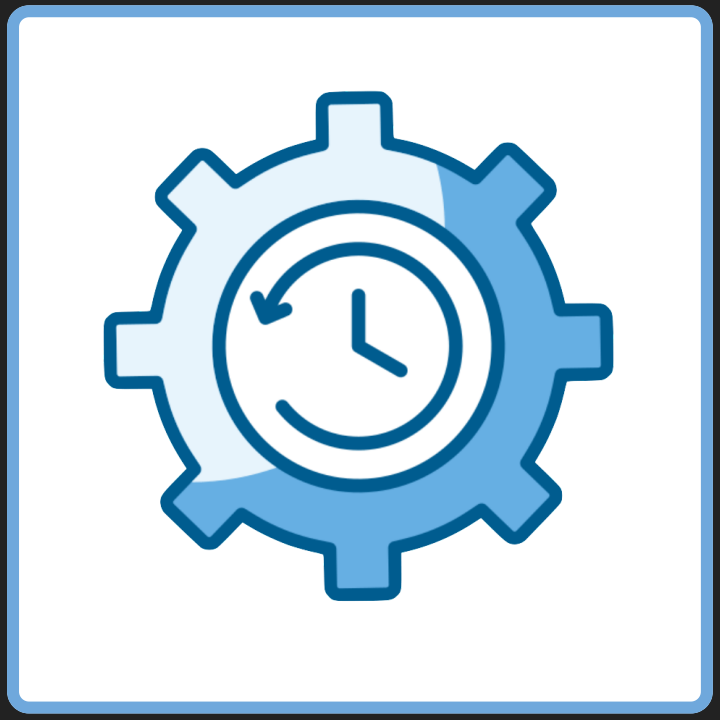


WHY?

WHY?

FRICTIONS WITH CURRENT SEL4 APPROACHES

BUILD SYSTEM



CURRENT BUILD SYSTEM SOLUTIONS

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

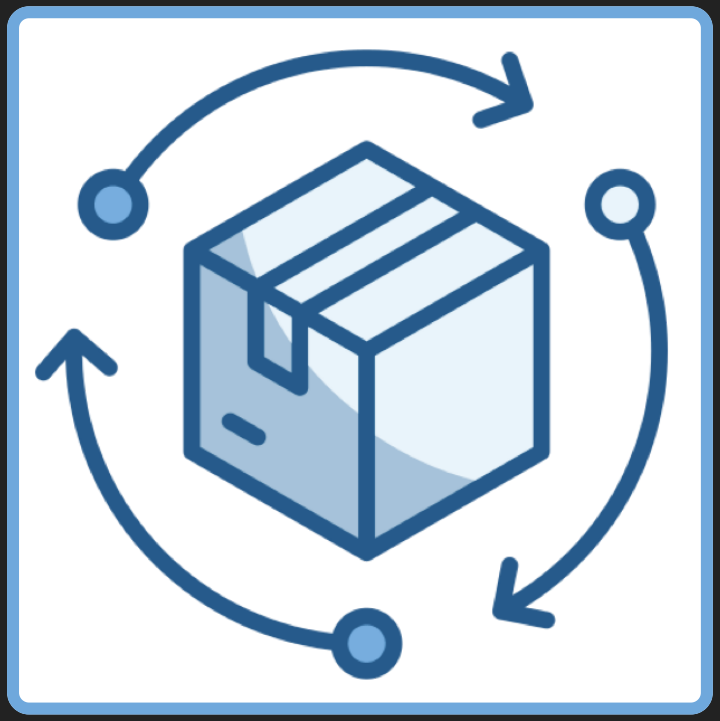
CHALLENGES

- 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

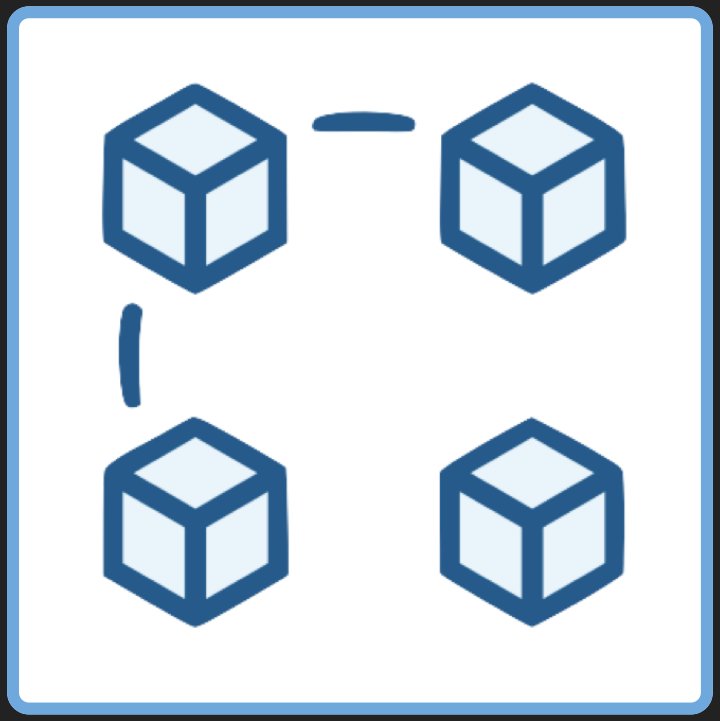
CHALLENGES

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

WHY?

FRICTIONS 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

WHY?

@ KRY10

> **MOUMOU TE HURUHURU**
>
> Do not waste the feathers of the bird

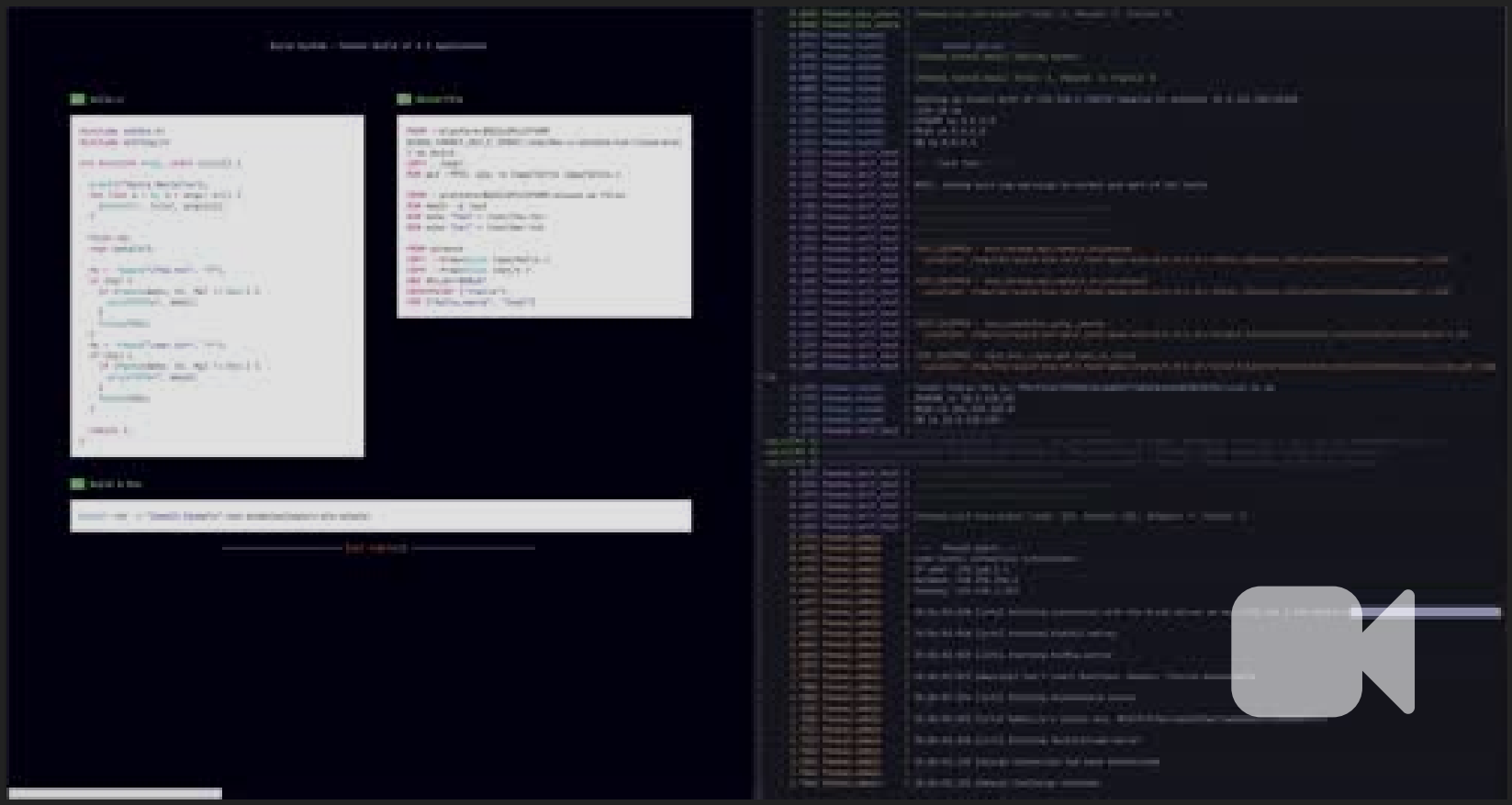
The container ecosystem is a well established and standardised framework that solves these problems.

SO WHAT WOULD THIS LOOK LIKE?

DEMO

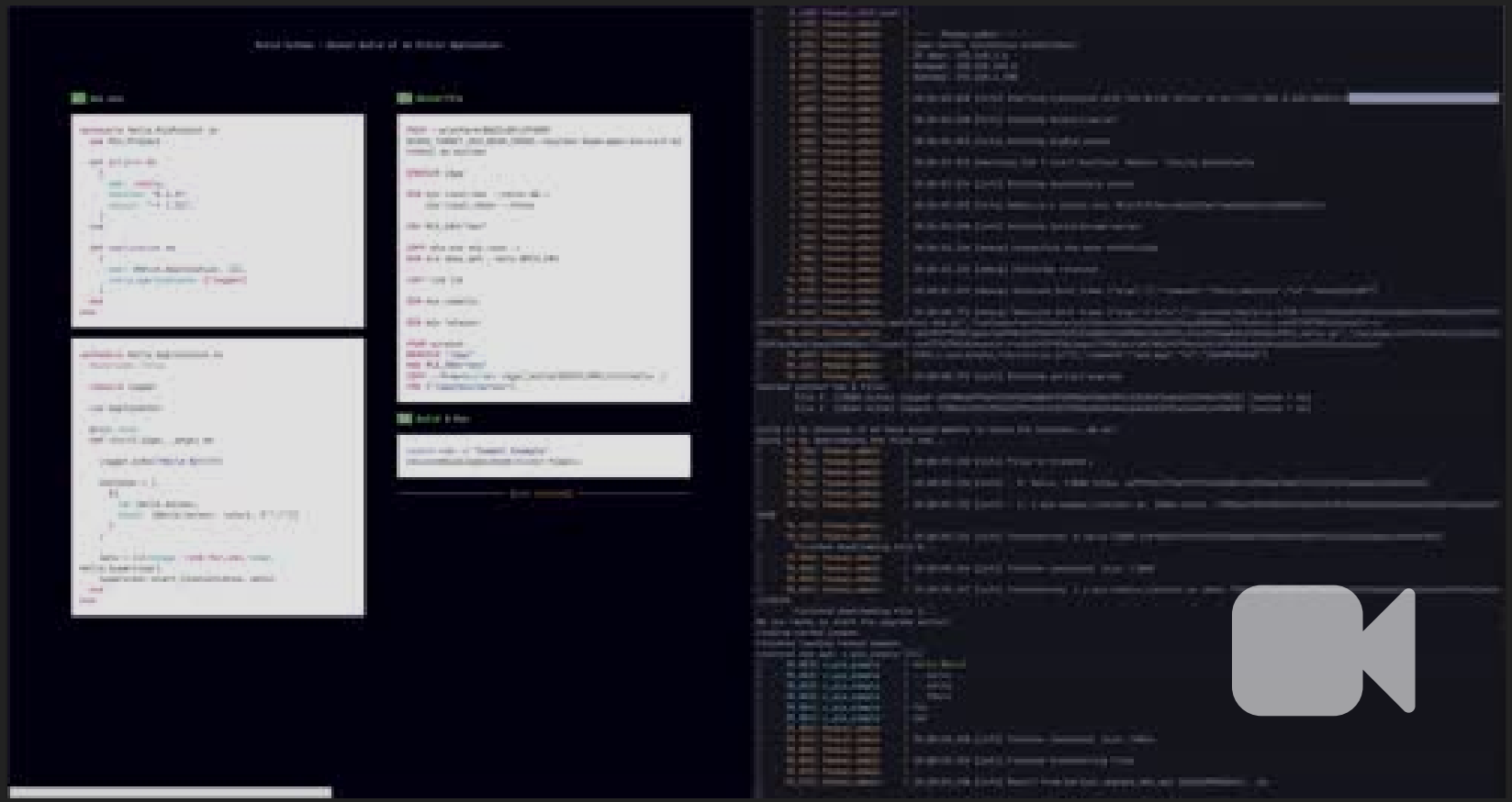
DOCKER BUILD OF A C APPLICATION

BUILD SYSTEM



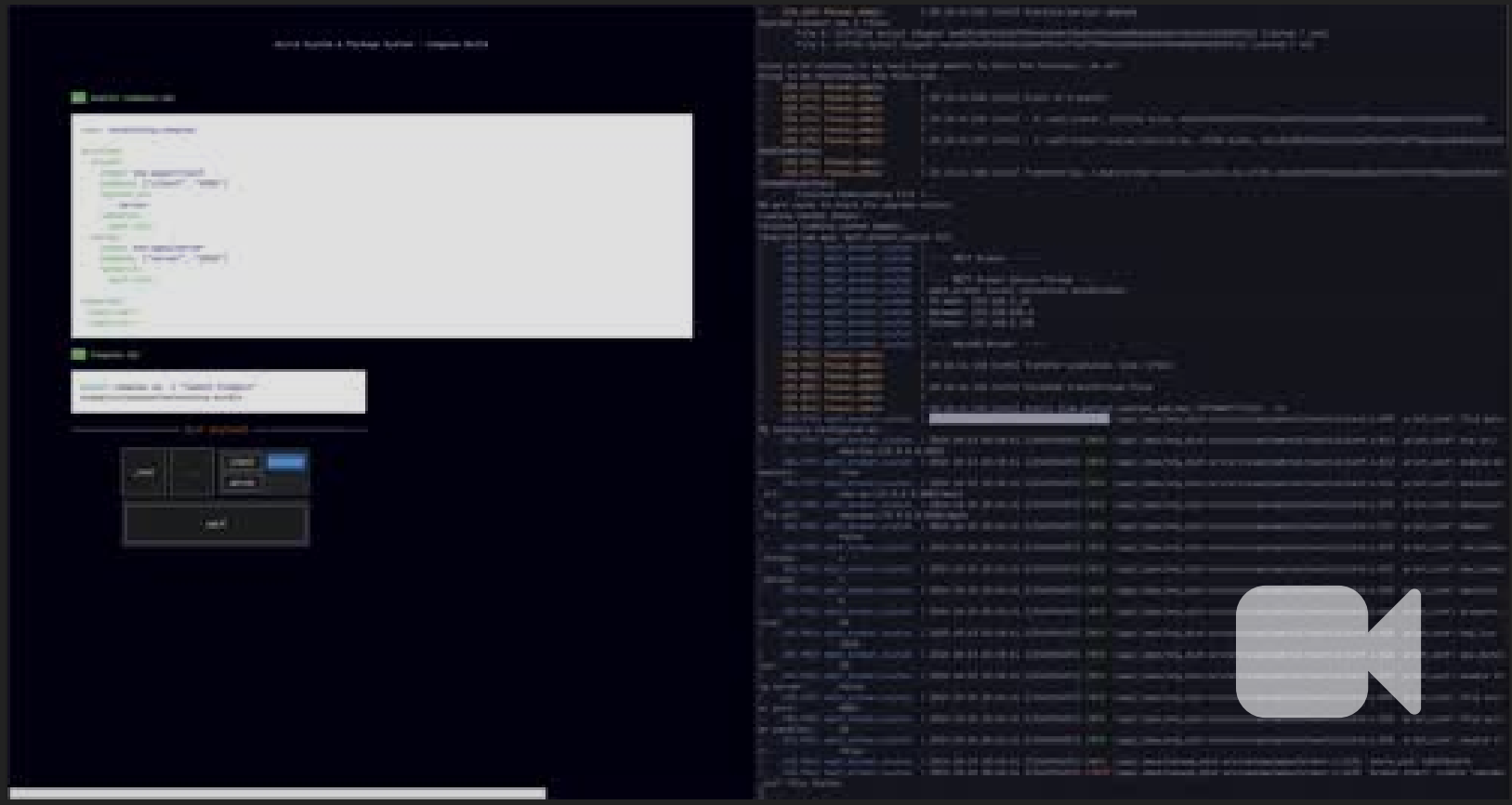
DOCKER BUILD OF AN ELIXIR APPLICATION

BUILD SYSTEM



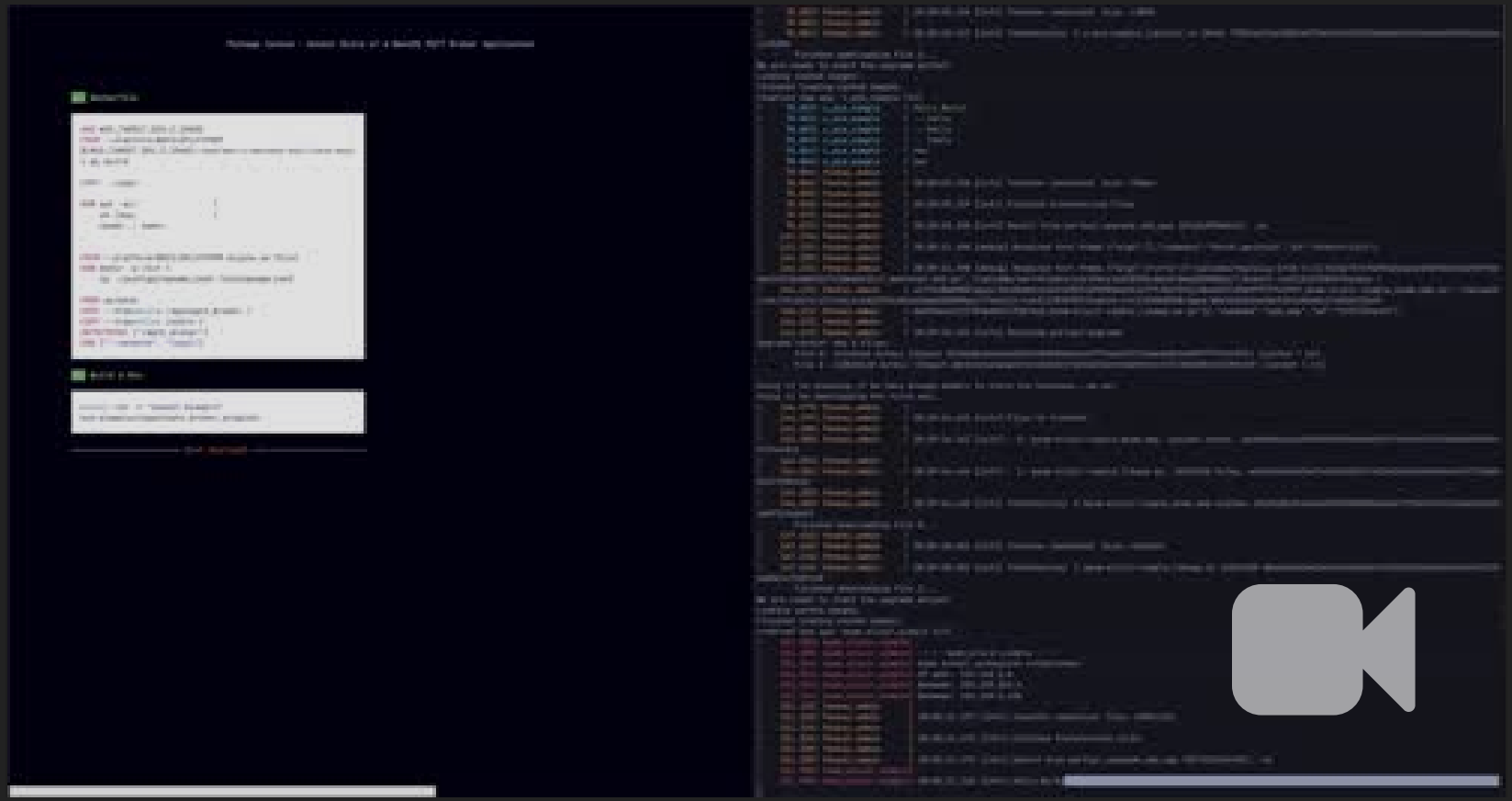
COMPOSE BUILD OF MULTIPLE APPLICATIONS

BUILD SYSTEM



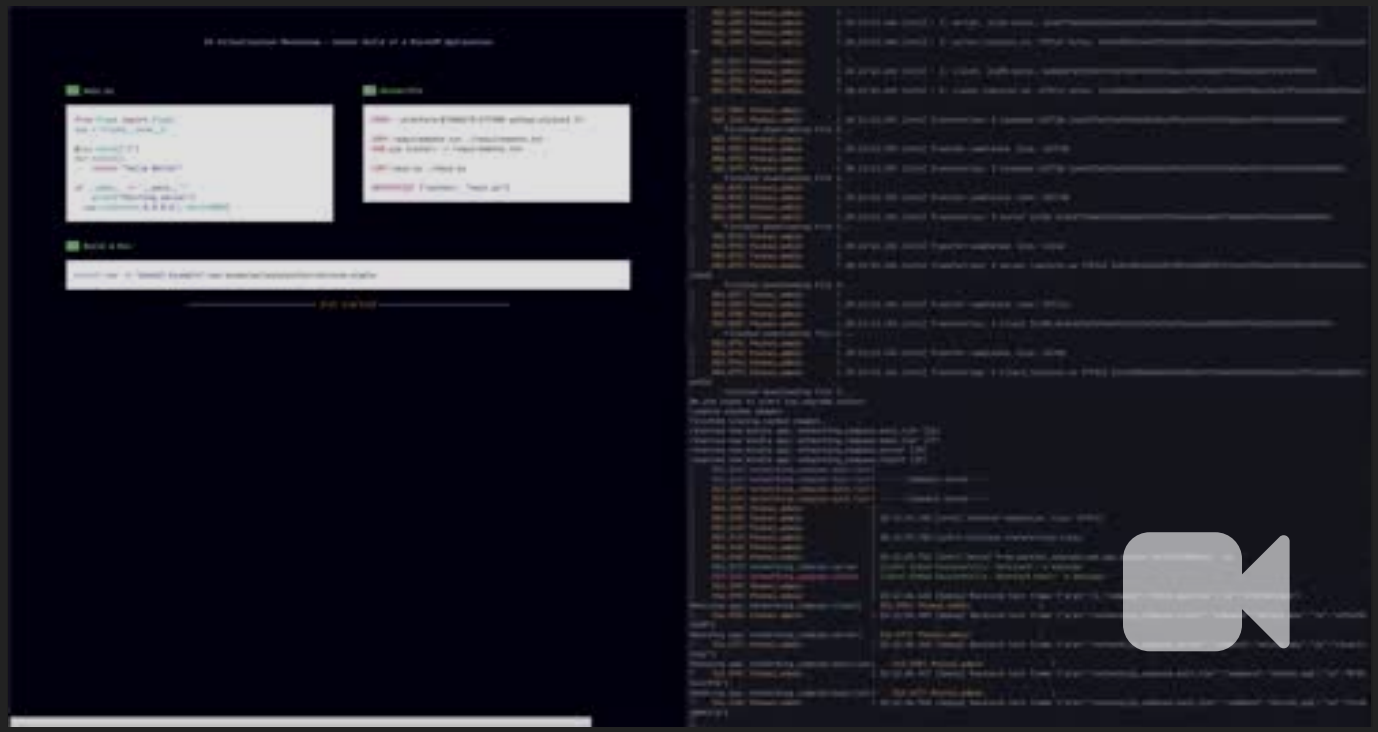
DOCKER BUILD OF A NANOMQ MQTT PACKAGE

PACKAGING SYSTEM



DOCKER BUILD OF A PYTHON APPLICATION

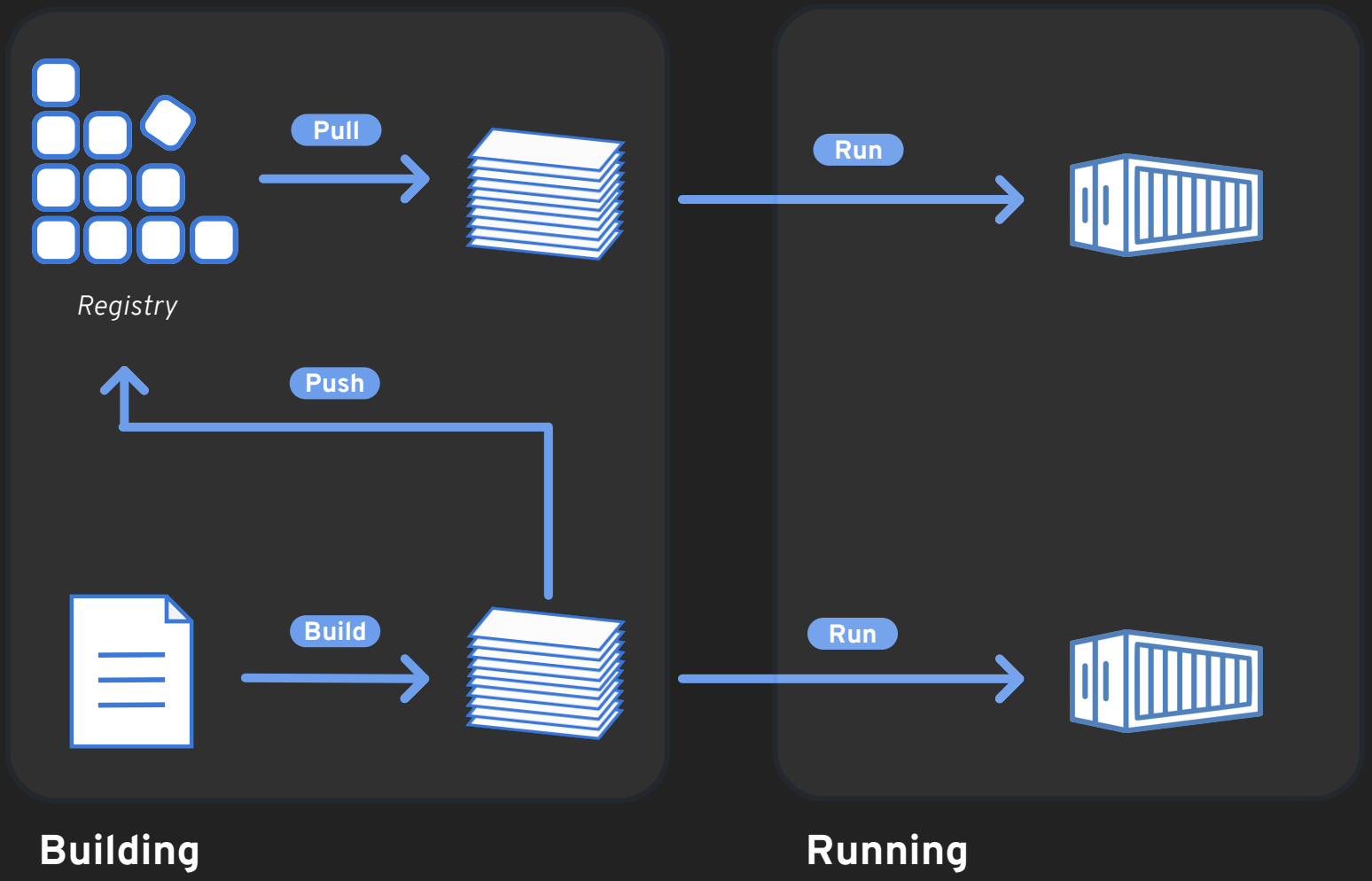
OS VIRTUALISATION MECHANISM



RUNNING AND BUILDING A CONTAINER

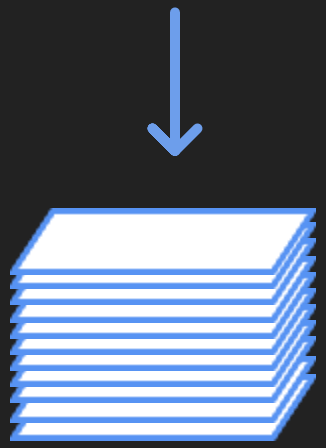
RUNNING & BUILDING A CONTAINER

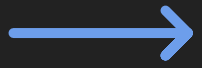
A CONTAINER LIFECYCLE



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
```

BUILDING A CONTAINER

```
FROM --platform=$BUILDPLATFORM kos/dev-c-aarch64-kos-linux-musl as build
COPY . /app/.
RUN gcc -fPIC -pie -o /app/hello /app/hello.c

FROM --platform=$BUILDPLATFORM alpine as files
RUN mkdir -p /out
RUN echo "foo" > /out/foo.txt
RUN echo "bar" > /out/bar.txt

FROM scratch
COPY --from=build /app/hello /
COPY --from=files /out/* /
ENV HELLO="WORLD"
ENTRYPOINT ["/hello"]
CMD ["hello_world", "test"]
```

CHALLENGES

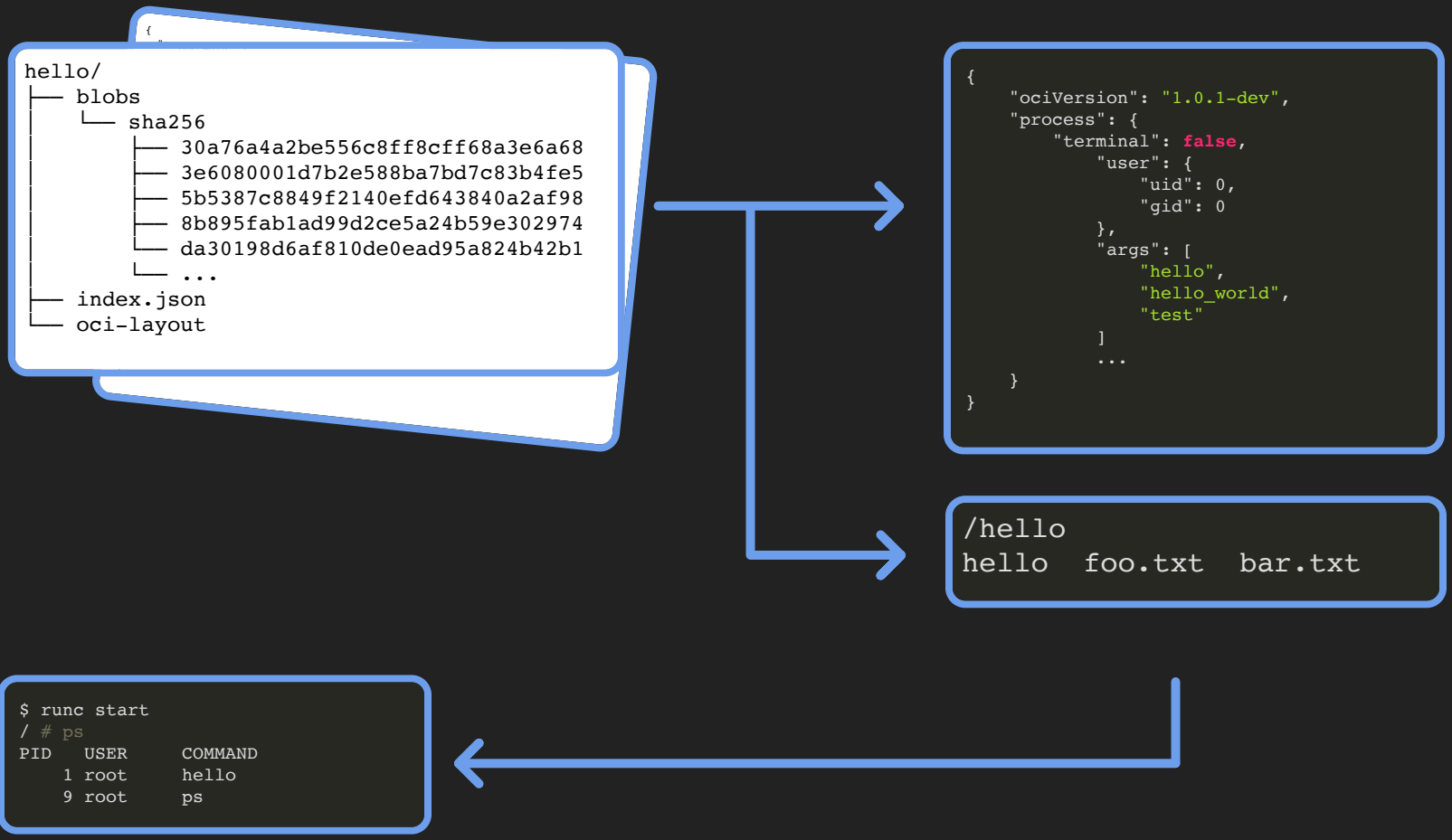
BUILD SYSTEM

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

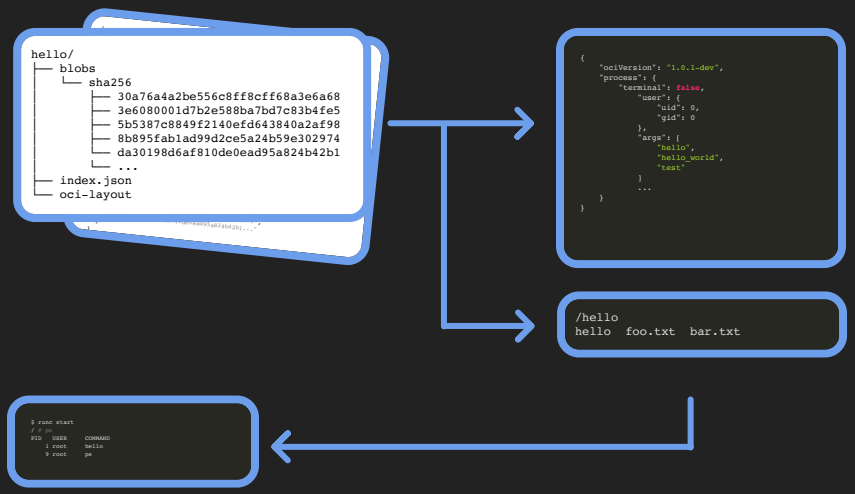
PACKAGING SYSTEM

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

RUNNING A CONTAINER



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

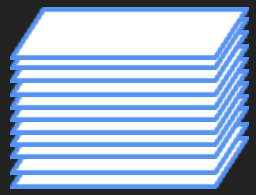
BUILDING A CONTAINER

Native

```
1  FROM --platform=$BUILDPLATFORM kos/dev-c-
   kos-linux-musl as build
2  COPY . /app/.
3  RUN gcc -fPIC -pie -o /app/hello /app/hel
4
5  FROM --platform=$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"]
```

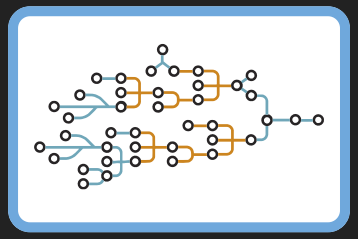
koskit

buildkit

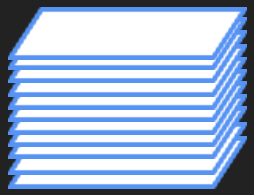


BUILDING A CONTAINER

Native

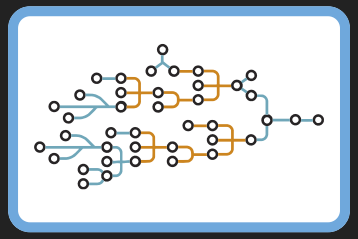


- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts

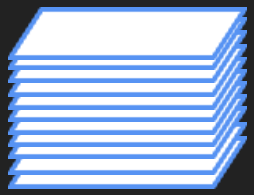


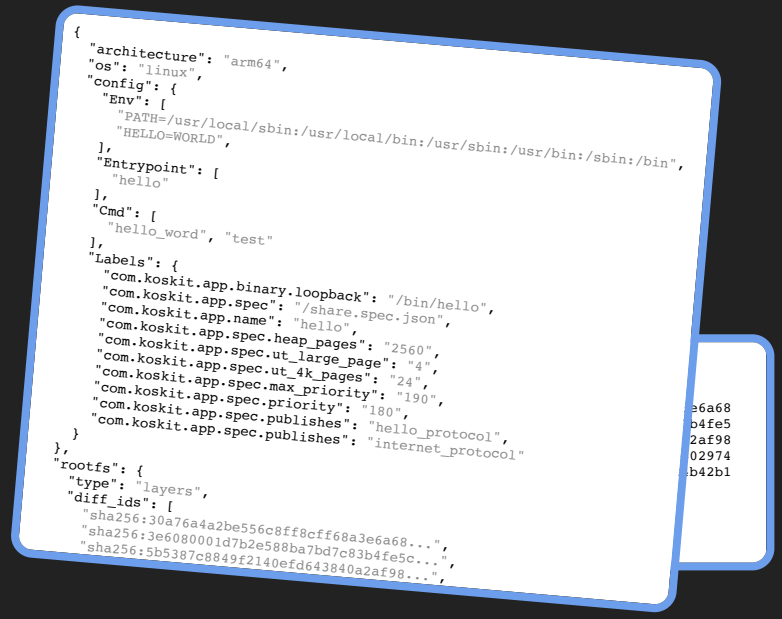
BUILDING A CONTAINER

Native



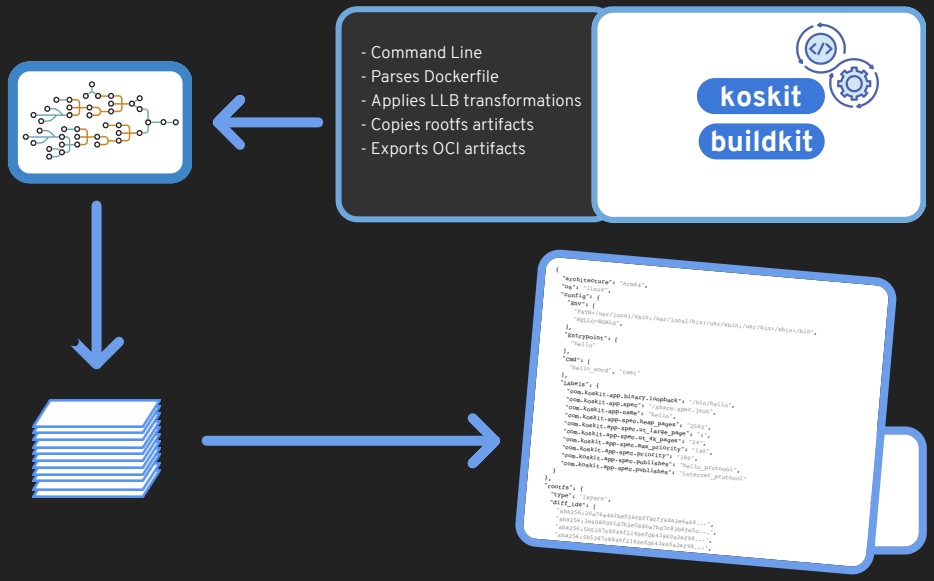
- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts





BUILDING A CONTAINER

Native



BUILD SYSTEM

CHALLENGES

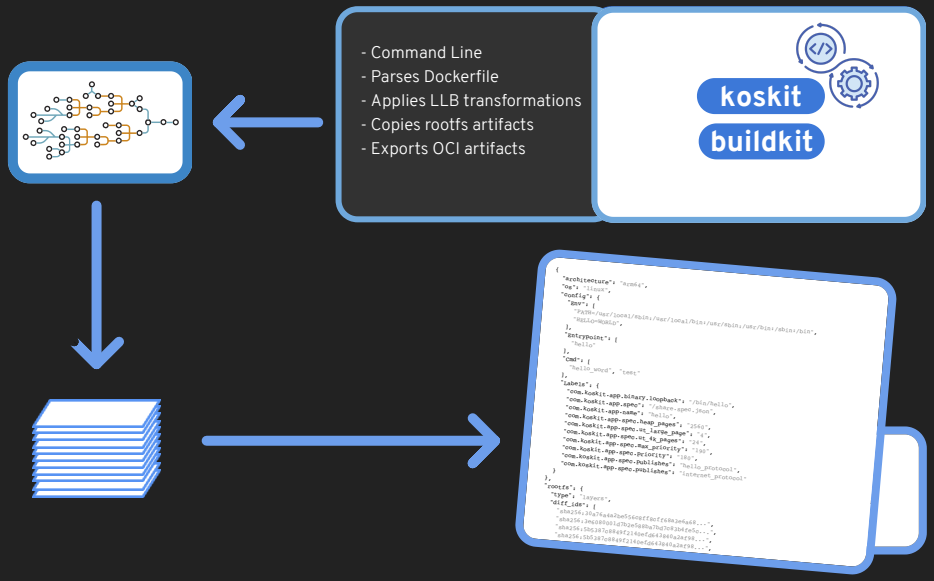
- Not interfering with user builds
- Supporting a multi-platform approach to tooling

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

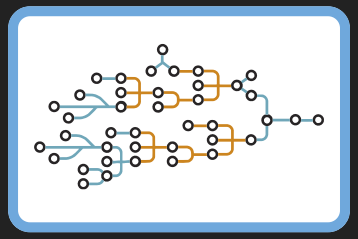
- Capturing the runtime dependencies of the application e.g. rootfs, command line, environment variables
- Ensuring the runtime state can be unravelled and appropriately loaded when executing the image.

SOLUTIONS

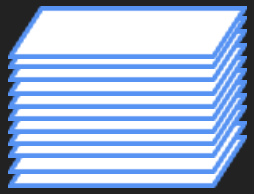
- koskit injects seL4-specific configuration under namespaced metadata labels and attributes.

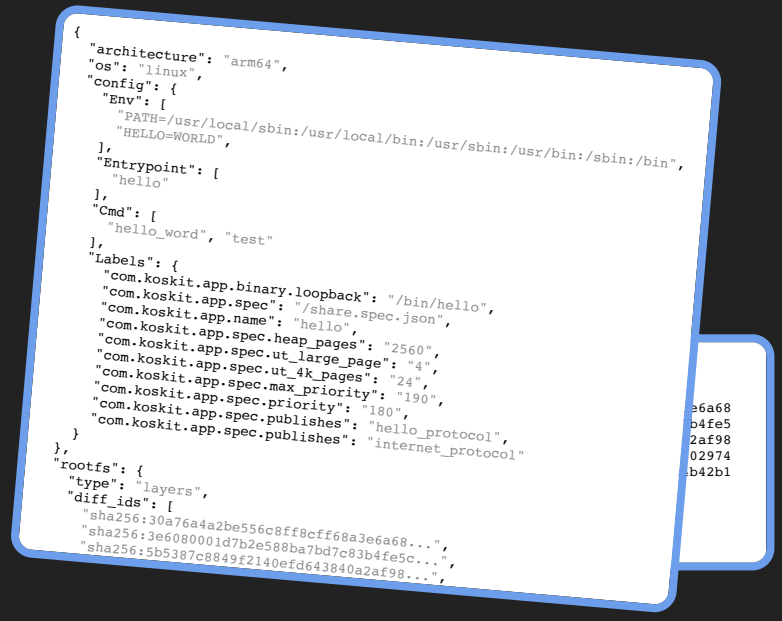
BUILDING A CONTAINER

Native



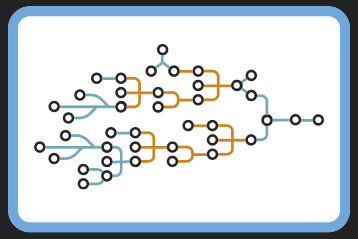
- Command Line
- Parses Dockerfile
- Applies LLB transformations
- Copies rootfs artifacts
- Exports OCI artifacts



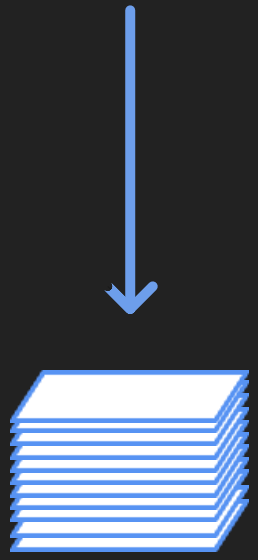


BUILDING A CONTAINER

Native



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

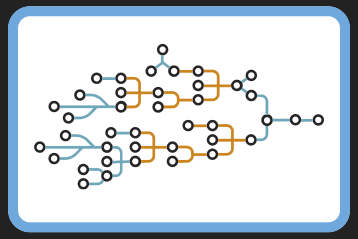


Untyped	0x200000
VSpace	1
TCB	1
CNode	6
Frames	42
Cmd	[/hello, hello_word, test]
Binary	hello

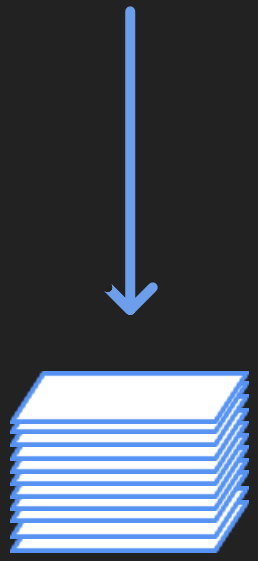
hello.elf | **initrd.cpio**

BUILDING A CONTAINER

microvm



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



initrd.squahsfs

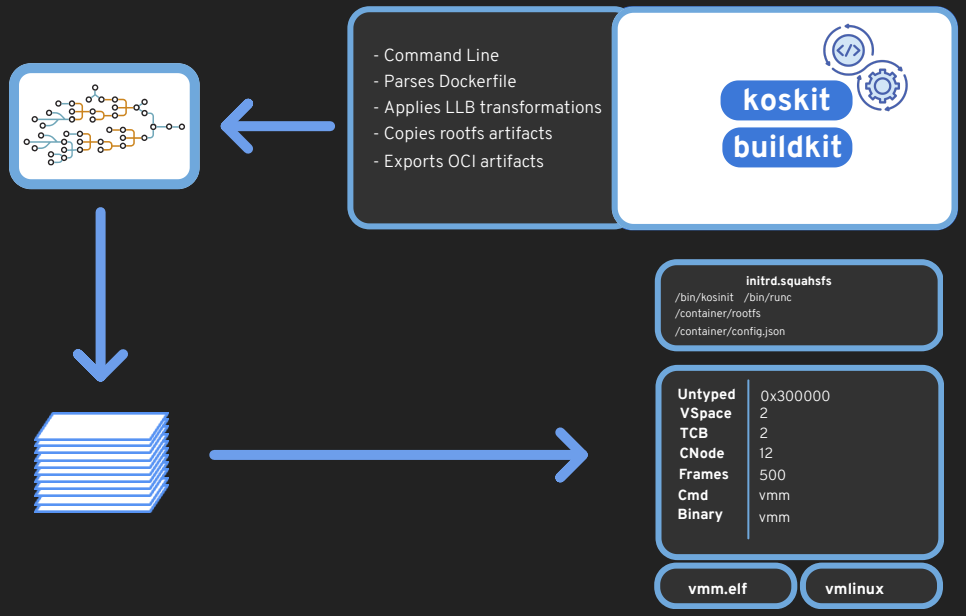
/bin/kosinit /bin/runc
/container/rootfs
/container/config.json

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

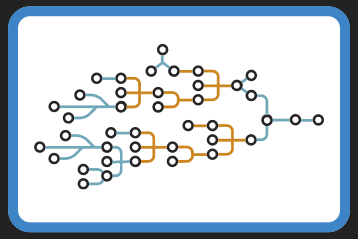
- Respecting the original container specification whilst injecting any of our own runtime functionality.

SOLUTIONS

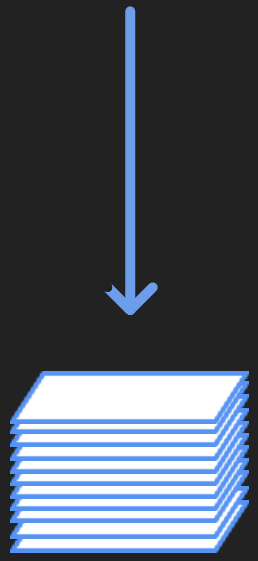
- koskit bundles a custom Linux initrd with its own init task and OCI-compliant JSON configuration.

BUILDING A CONTAINER

microvm



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



initrd.squahsfs

/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**

RUNNING A CONTAINER

Native

RUNNING A CONTAINER

Native

ROOT SERVER

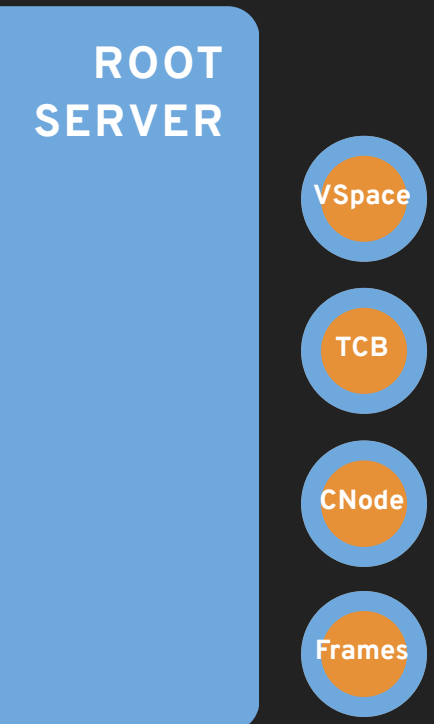
VSpace	0
TCB	1
CNode	6
Frames	42
Untyped	0x200000
Cmd	[/hello, hello_word, test]
Binary	hello

hello.elf

initrd.cpio

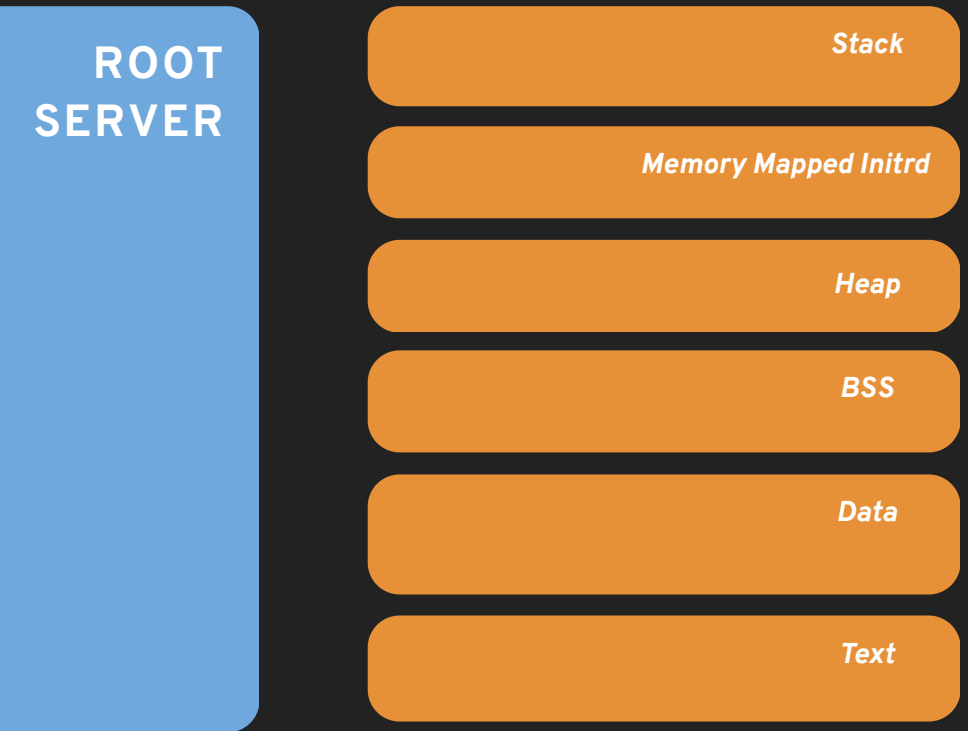
RUNNING A CONTAINER

Native



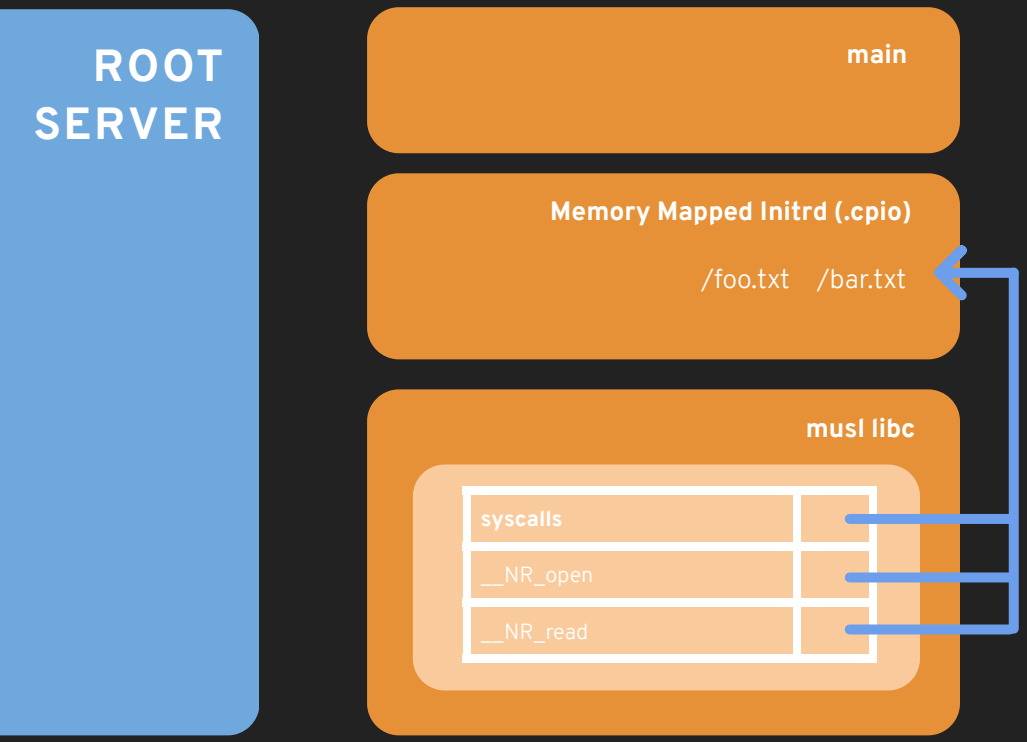
RUNNING A CONTAINER

Native



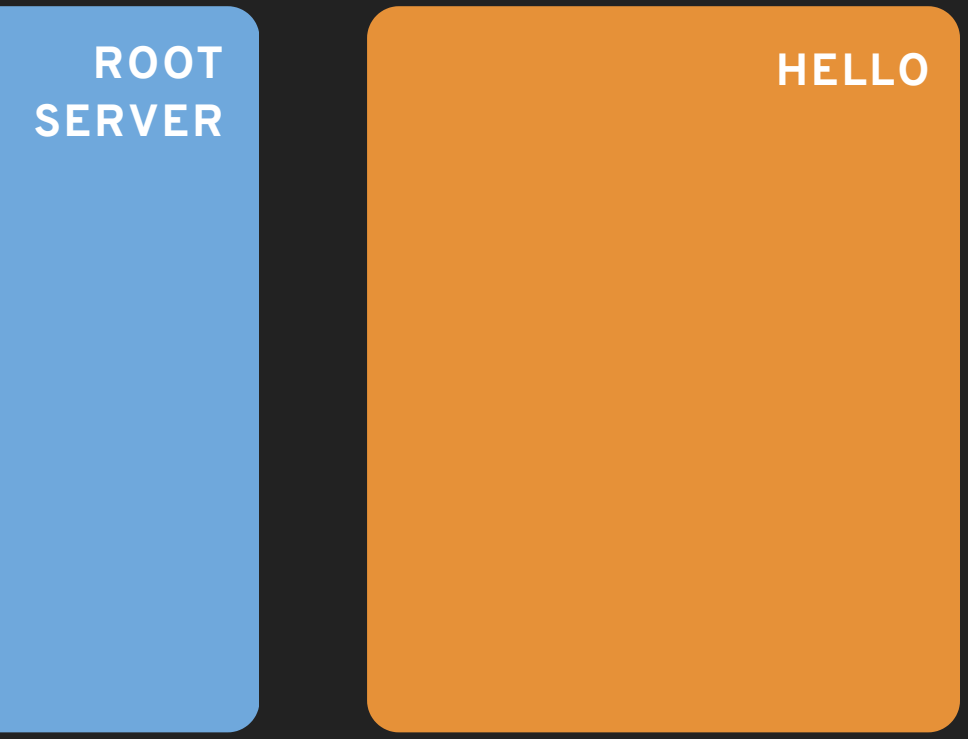
RUNNING A CONTAINER

Native



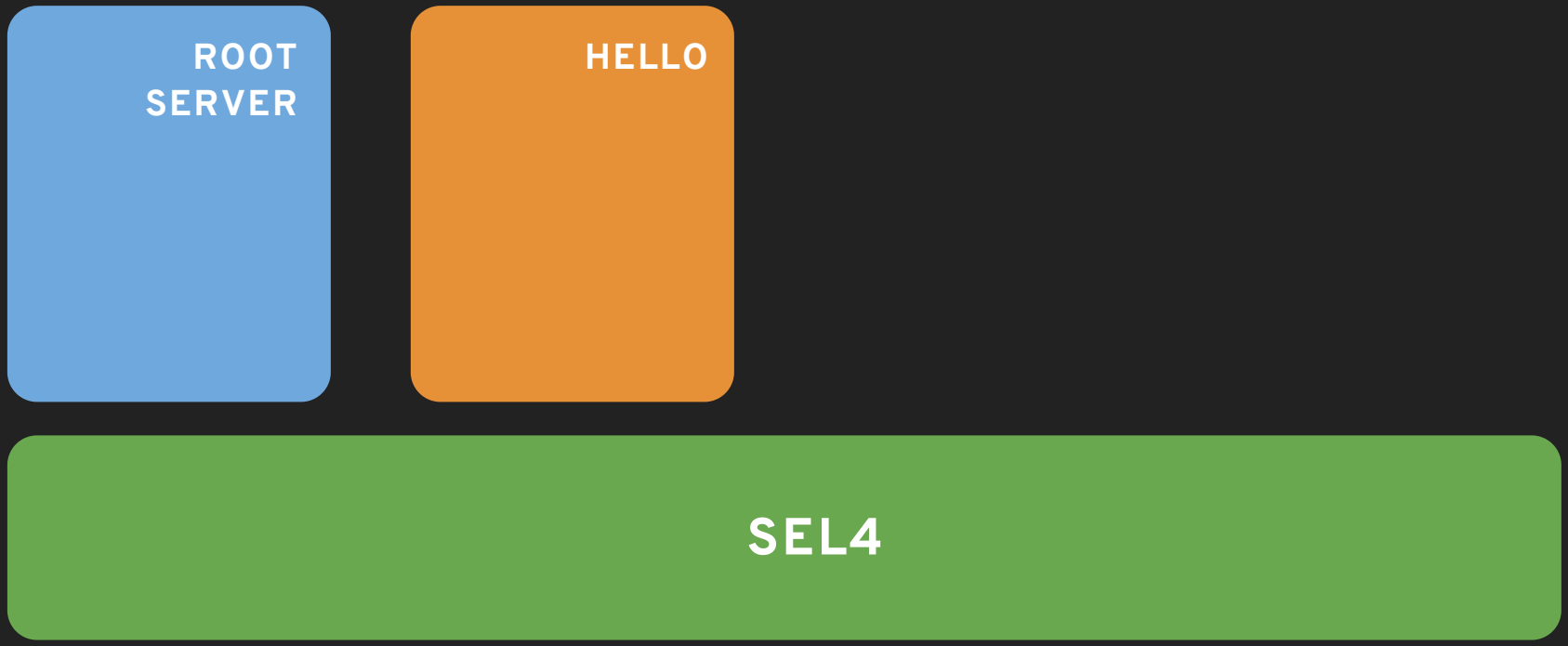
RUNNING A CONTAINER

Native



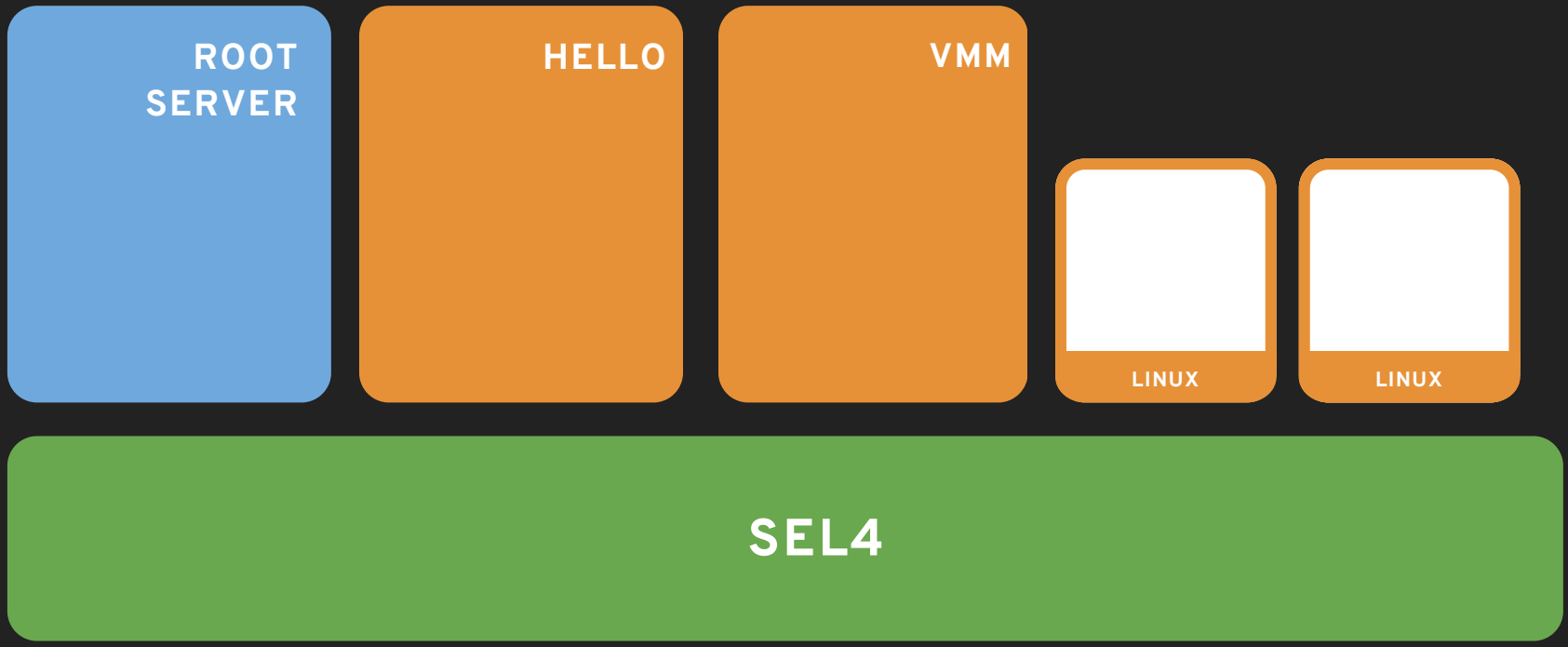
RUNNING A CONTAINER

Native



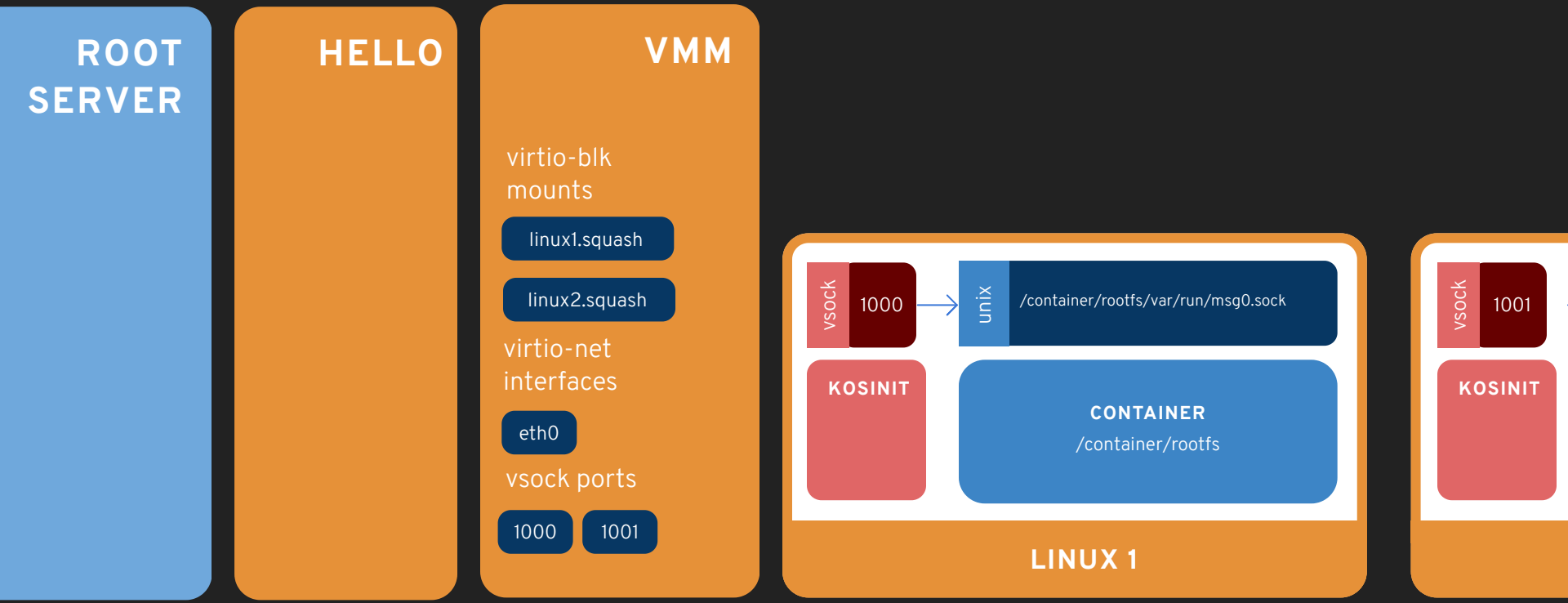
RUNNING A CONTAINER

microvm



RUNNING A CONTAINER

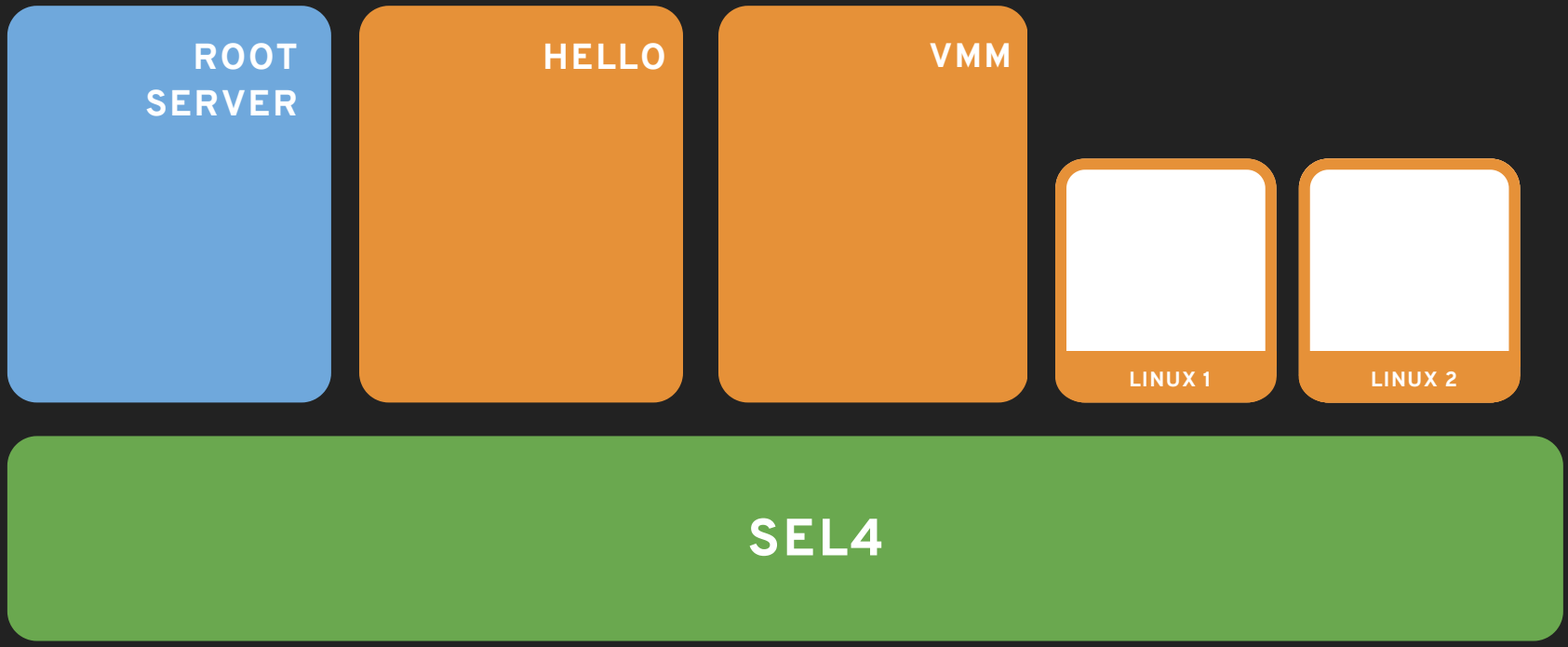
microvm



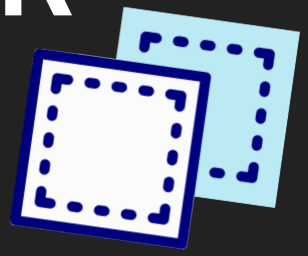
SEL4

RUNNING A CONTAINER

microvm



STITCHING IT TOGETHER



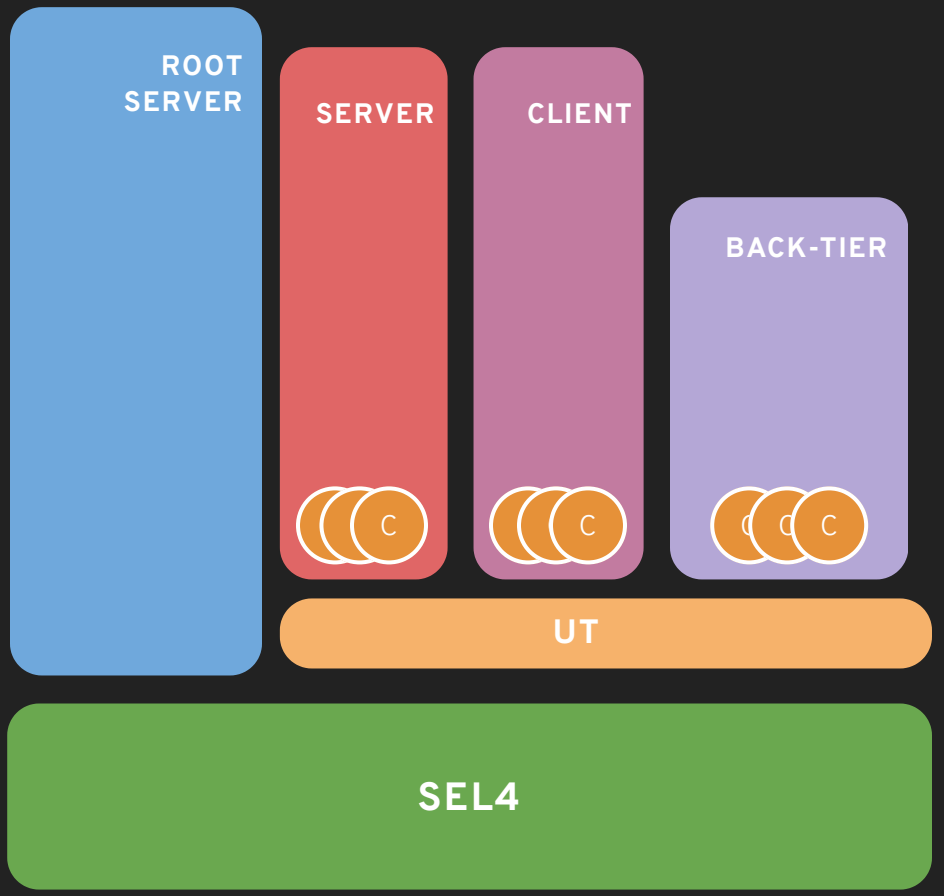
COMPOSE SYSTEMS

```yaml
name: example_compose

services:
  client:
    image: kos:apps/client
    command: ["client", "5050"]
    depends_on:
      - server
    networks:
      back-tier:
  server:
    image: kos:apps/server
    command: ["server", "5050"]
    networks:
      back-tier:

networks:
  back-tier:
    x-msg-server: "Poukai.msg0"
```

COMPOSE SYSTEMS

UP

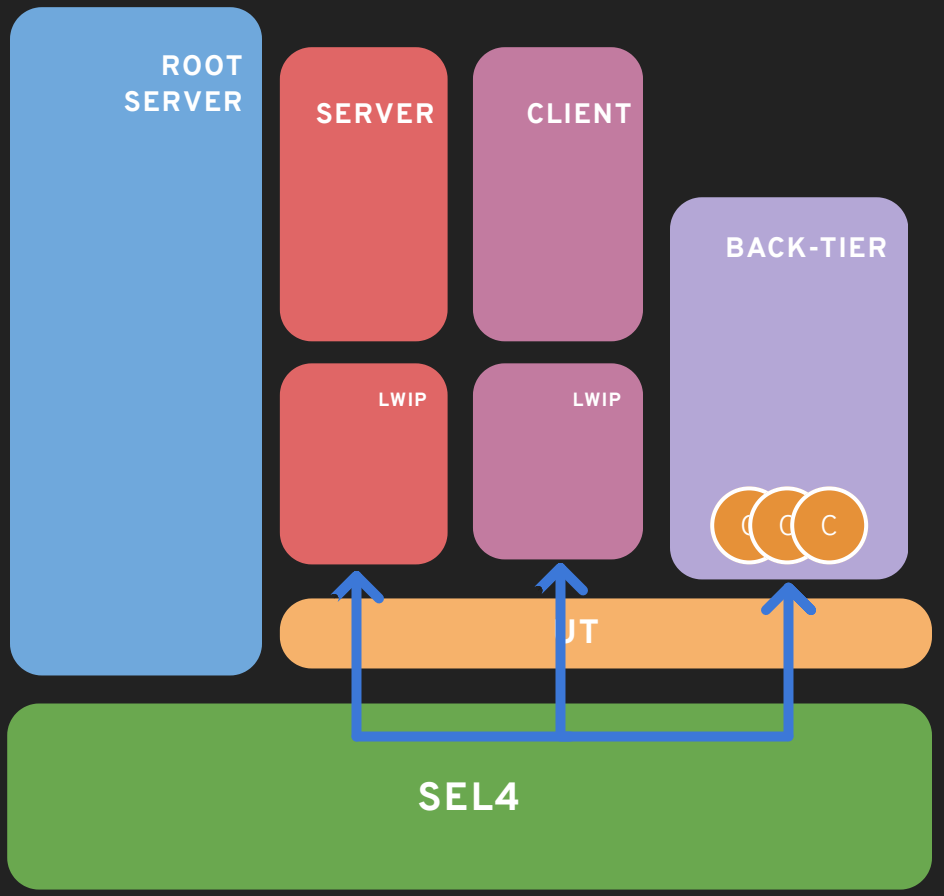
```yaml
name: example_compose

services:
  client:
    image: kos:apps/client
    command: ["client", "5050"]
    depends_on:
      - server
    networks:
      back-tier:
  server:
    image: kos:apps/server
    command: ["server", "5050"]
    networks:
      back-tier:

networks:
  back-tier:
    x-msg-server: "Poukai.msg0"
```



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"
```



COMPOSE SYSTEMS

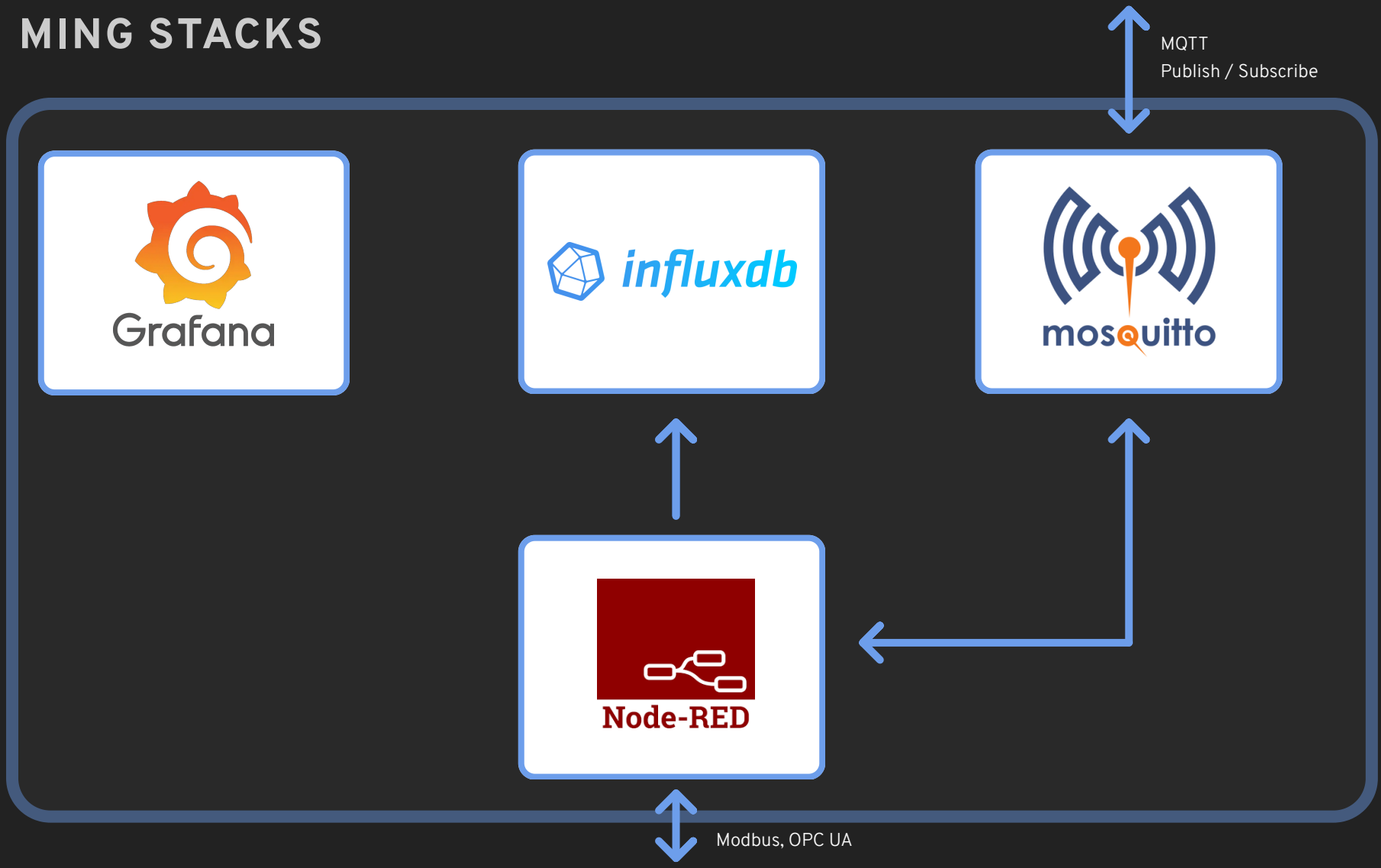
DOWN

```
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"
```

NEXT STEPS

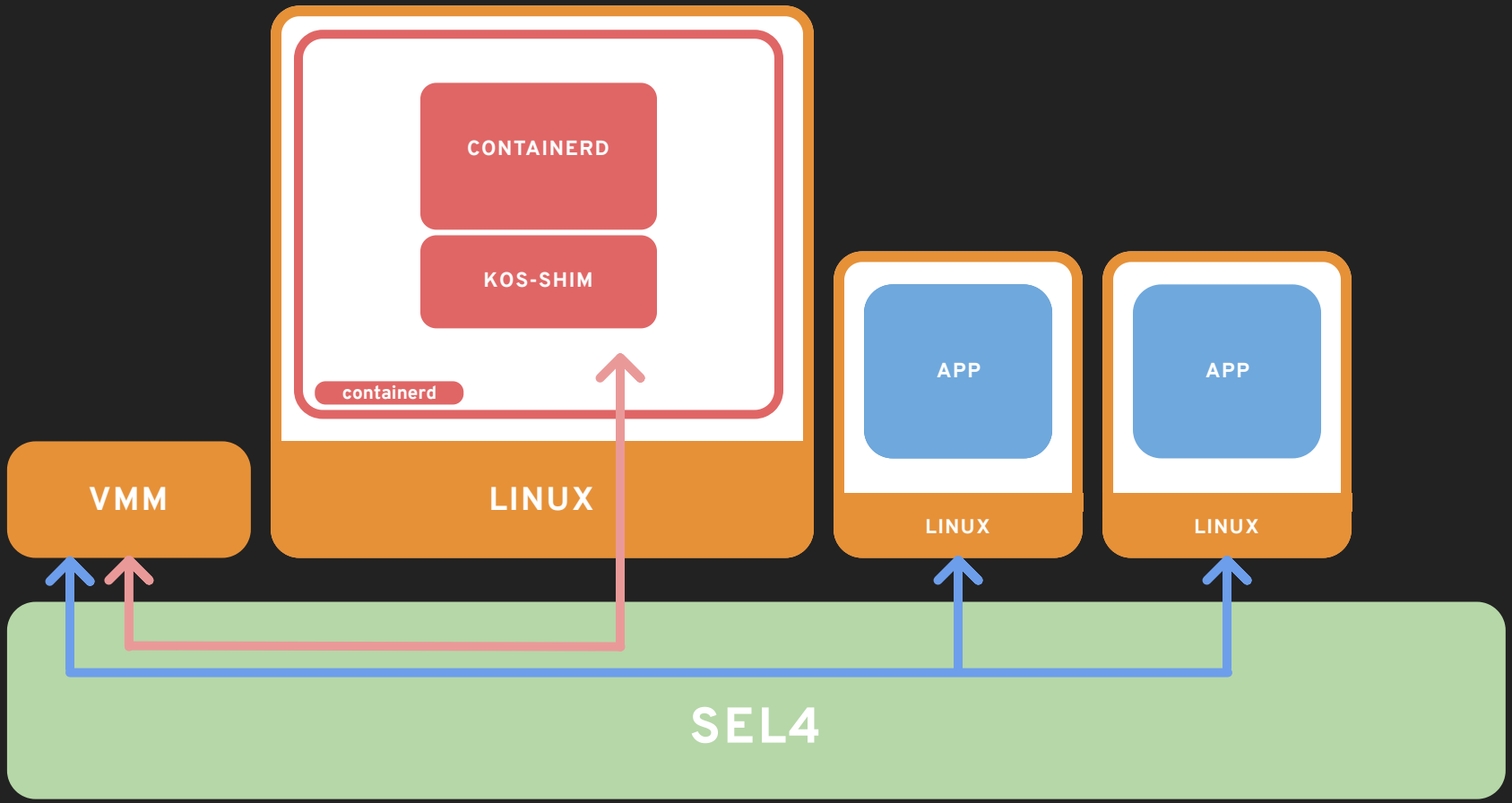
NEXT STEPS

MING STACKS



NEXT STEPS

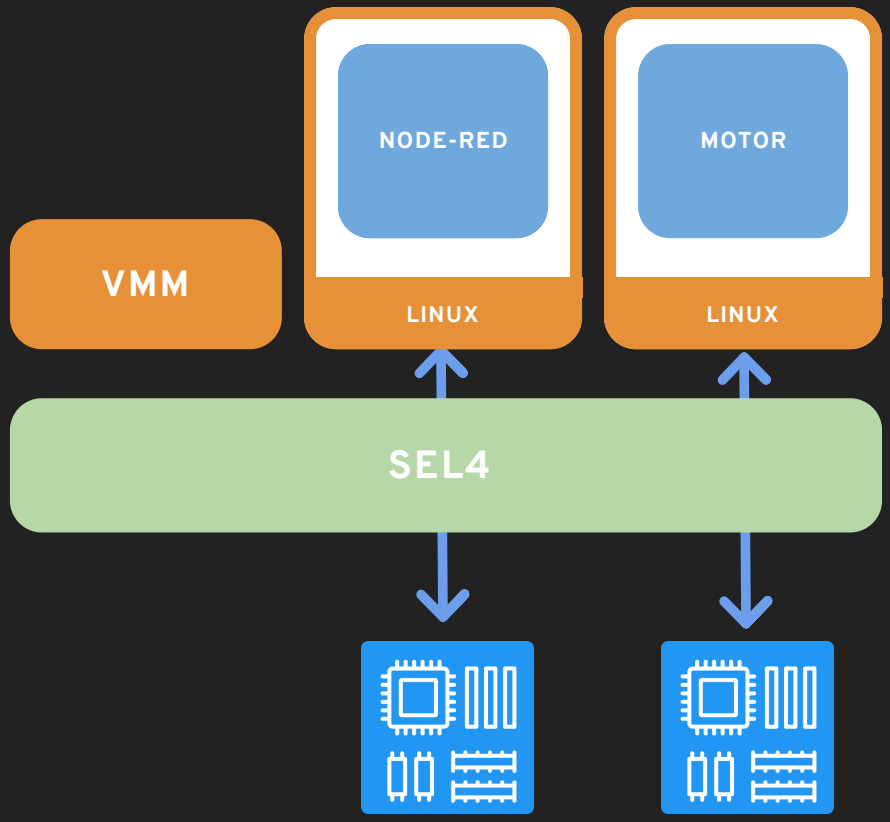
CONTAINERD CONTROL PLANES



NEXT STEPS

DRIVER CONTAINERS

```
name: devices-compose

services:
  node-red:
    image: docker:nodered/node-red
    privileged: true
    cap_add:
        - SYS_RAWIO
    devices:
        - "/dev/mem:/dev/mem"
        - "/dev/gpiomem:/dev/gpiomem"
        - "/dev/i2c-1:/dev/i2c-1"
  motor:
    image: kos:my_motor_app/.
    privileged: true
    cap_add:
        - SYS_MODULE
```



END