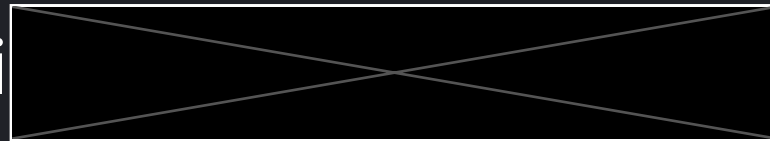


ARM Serial Experiments

Heterogeneous cluster experiments

- Ric Tsui ric.tsui
- Marco Njo




Objective

This project aims to provide insights on multiarch support for our images, CI pipelines, and running them on hetero-k8s cluster.

But not

- Using emulation to run already compiled artifacts
- A fully migrated cluster

Incentives

- Cheap Computing power
- Codebase  is largely platform independent
- Already supported by existing cloud infra

Azure Node Cost

Let's talk about \$

- In VOS, a default controller node size is **Standard_D16_v3**, billing \$0.7680/hr
- Equivalent ARM instance **Standard_D16ps_v6** uses \$0.5620/hr
- $(0.7680 - 0.5620) / 0.7680 = 27\%$
- an ARM instance is 27% cheaper

Default VOS Cluster Controller Workload Table

Name	Replicas	CPU Request	CPU Limit	Memory Request	Memory Limit	Notes
Elasticsearch	3	1	12	4096Mi	5120Mi	Log storage
ETCD	3	500m	4	512Mi	4096Mi	storage
Logstash	3	1	4	2048Mi	2560Mi	Log processing
Kibana	1	0.3	1	512Mi	1024Mi	Log UI
PostgreSQL (PGDB)	2	500m	2	512Mi	2048Mi	Database

Approximate cost saving for each cluster per year

A default VOS controller node count is 3

```
(Price diff) * node count * hours * days:  
(0.7680 - 0.5620) *3 *24* 365 = 5413.68
```

If we can migrate 300 nodes across all VOS env (dev,
stag, prod.) = \$541368

Projects other than VOS can benefit

For example, the  project has over 500 nodes running in the staging environment in Linode.

If it were to be deployed on Azure using ARM64 nodes, it could save approximately \$902,280 per year.

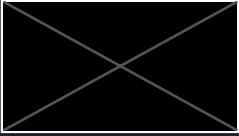
Scenario

- As a Dev:
 - Reducing cost in dogfooding, able to use instances longer???
 - native Mac M1 debugging & execution!!!
 - with existing cloud infra
- As an Ops:
 - using with existing cloud infra
 - not another platform to manage

- As a project owner:
 - Future-proofing our CI
 - flexibility
 - minimum change
- As a SaaS provider:
 - Opaque cost cuts
 - Able to serve weird req?

Low-effort, huge payout

Current support

- : Yes, with containerd manually enabled in each execution
- Local Docker: Yes with buildx
- Kubernetes: Supports multiarch
- Azure ACR: Support multiarch

Milestones

1. Create exemplary arm64 workload in x64 cluster
2. Able to build multiarch image in CI pipeline
3. Migrate some workloads which are already using portable languages

How AI helped?

debug, explain, evolve

How AI helped? 1

```
184.5 Segmentation fault (core dumped)
184.5 qemu: uncaught target signal 11 (Segmentation fault) - core dumped
193.7 dpkg: error processing package libc-bin (--configure):
```

```
-----
Dockerfile:27
```

```
-----
26 | # Install system packages
27 | >>> RUN set -eux; \
28 | >>> apt-get update; \
...
39 | >>> dnsutils \
...
```

```
-----
ERROR: failed to solve: process ... exit code: 100
```

How AI helped? 2

I am trying to build multiarch image with docker buildx on x64 host, and it failed at L26 .
Could you see what is wrong?

✓ Optimizing tool selection...

I can see you're encountering a segmentation fault during the Docker build process when trying to build a multiarch image. This is a common issue when building ARM images on x64 hosts using QEMU emulation. Let me first examine the current Dockerfile to understand the context better.

✓ Read `Dockerfile`, lines 1 to 50

Now let me see the rest of the file to get the complete picture:

✓ Read `Dockerfile`, lines 51 to 69

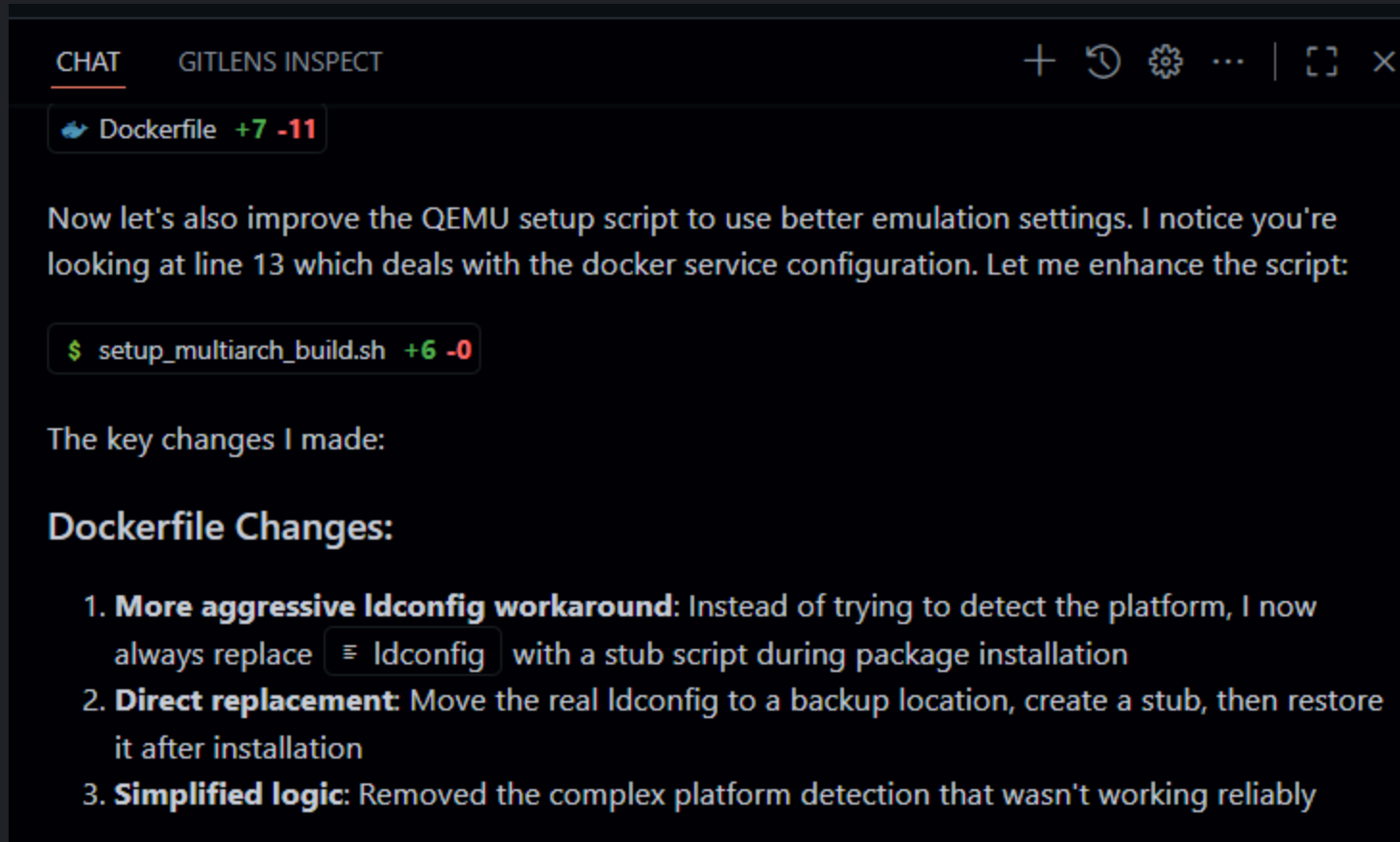
Now let me check if there are any build scripts or configurations related to multiarch builds:

✓ Read `setup_multiarch_build.sh`

The issue you're experiencing is a known problem when building multiarch images using QEMU emulation. The segmentation fault occurs during the `libc-bin` package configuration step because of QEMU emulation issues with certain system calls, particularly related to the `ldconfig` command.

Here are several solutions to fix this issue:

How AI helped? 3



The screenshot shows a chat interface with two tabs: 'CHAT' and 'GITLENS INSPECT'. The 'CHAT' tab is active. At the top right of the chat area, there are icons for adding a new chat, undo, settings, and a menu. Below the tabs, a file named 'Dockerfile' is shown with a change summary of '+7 -11'. The main text of the chat says: 'Now let's also improve the QEMU setup script to use better emulation settings. I notice you're looking at line 13 which deals with the docker service configuration. Let me enhance the script:'. Below this, a file named 'setup_multiarch_build.sh' is shown with a change summary of '+6 -0'. The chat continues with the text: 'The key changes I made:'. This is followed by a section header 'Dockerfile Changes:' and a list of three items: 1. 'More aggressive Idconfig workaround: Instead of trying to detect the platform, I now always replace `≡ Idconfig` with a stub script during package installation', 2. 'Direct replacement: Move the real Idconfig to a backup location, create a stub, then restore it after installation', and 3. 'Simplified logic: Removed the complex platform detection that wasn't working reliably'.

CHAT GITLENS INSPECT

Dockerfile +7 -11

Now let's also improve the QEMU setup script to use better emulation settings. I notice you're looking at line 13 which deals with the docker service configuration. Let me enhance the script:

\$ setup_multiarch_build.sh +6 -0

The key changes I made:

Dockerfile Changes:

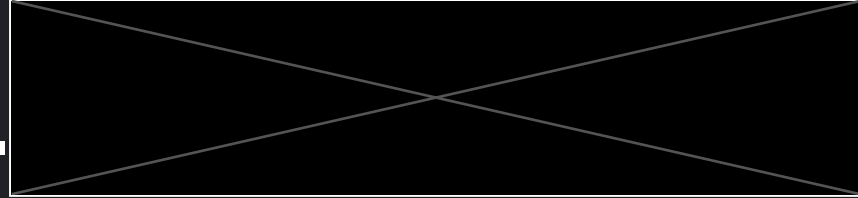
- More aggressive Idconfig workaround:** Instead of trying to detect the platform, I now always replace `≡ Idconfig` with a stub script during package installation
- Direct replacement:** Move the real Idconfig to a backup location, create a stub, then restore it after installation
- Simplified logic:** Removed the complex platform detection that wasn't working reliably

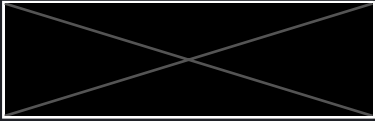
DEMO: vos_zookeeper_middleware

```
> k get pod -A -l app=zookeeper -o wide
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
ngde-arm64	zookeeper-0	1/1	Running	0	42m	10.244.0.4	aks-armpool-40809183-vmss000001	<none>	<none>
ngde-arm64	zookeeper-1	1/1	Running	0	18m	10.244.2.10	aks-generic-91644194-vmss000007	<none>	<none>
ngde-arm64	zookeeper-2	1/1	Running	0	18m	10.244.4.7	aks-generic-91644194-vmss000008	<none>	<none>

DEMO: vos_app_



- a VOS app widely used by 
- consists of a Java VOS app which launches K8s job of Golang
- the Golang binary for ARM64 is built and added to the cross build docker image
- tested deployable on ARM64 node pool

How to CRUD Multiarch Image

- Create

```
docker buildx build -t myimage:tag --platform linux/amd64,linux/arm64 .
```

- Read simply pull and inspect

```
docker manifest inspect myimage:tag
```

- Update `myimage:tag` from foreign, or simply
rebuild with `docker buildx build` from local


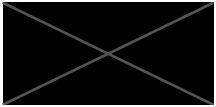
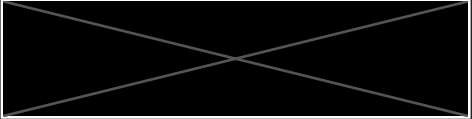
```
az acr import --subscription RND -g harmonicinc-acr-resource-group --name harmonicinc --source docker.io/minio... --image minio:... --force
```

How to on k8s

```
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          preference:
            matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values: ["arm64"]
```

```
spec:
  tolerations:
    - key: node-role.vos/arm_controller
      operator: Equal
      value: "true"
      effect: NoSchedule
```

Summary

- Discovered multiarch support in our CI
- Achieved: Migration of VOS Base Image, Zookeeper, 
- Our platform agnostic codebase allows a lot more to happen
 -  (Golang)
 - ELK (Java)
 - VOSHA (Java)
 - Teleport (Go prebuilt)
 -  (Rust)

For More

- performance review [Ampere Altra vs IceLake and Epyc for Java applications \(ARM/Intel/AMD\) | by Роман Близнец | Medium](#)
- doc [Multi-arch build and images, the simple way](#)

Thank you.