The State of APIs in the Container Ecosystem

Phil Estes, Principal Engineer, AWS
QCon London 2022
Developers, Developers, Developers

Docker led the container UX for developers *circa 2013-2014

- Docker’s command line was (and still is) a lightweight client
  - A well-defined HTTP-based REST API connected the client to the Docker engine
  - Usually local, but could be remote (*TCP with certificate auth, or SSH tunneled*)
- Developers love the command line, but APIs enable integration & automation
  - CI/CD, security tools, telemetry/monitoring, vendor extension points
What’s behind the Docker API?

<table>
<thead>
<tr>
<th>A configuration</th>
<th>An image bundle</th>
<th>A registry protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>The heart of a container is the (JSON) representation of its configuration. The command to run; the resources (cgroups), the isolation settings (namespaces), volumes, env variables, ...</td>
<td>The configuration is paired with the image metadata and layers of filesystem content This is what is “built”, and then possibly “pushed” and “pulled” from container registries.</td>
<td>An HTTP API to query, inspect, fetch, and push content to a remote distribution endpoint. For many this equates to DockerHub, but has grown to include many OSS projects and hosted registries.</td>
</tr>
</tbody>
</table>
The OCI:

> Configuration = The runtime spec
> Image = The image spec
> Registry = The distribution spec
> Runtime implementation = runc
But what about an **API** for containers?
Docker provided a solid answer for a single node.

At scale, users need to orchestrate containers.

Cue: The orchestration wars..
But, really we only have time to talk about Kubernetes.
$$\text{kubectl apply -f podspec.yaml}$$

Kubernetes API over HTTP: `<endpoint>`:8081/api/v1/services

Kubernetes API Server

Node → kubelet → container runtime = gRPC endpoint implementing CRI
Kubernetes API

Key component of the control plane for K8s object interaction

- CRUD operations via a REST API over HTTP on common objects
  - Examples: Pods, Nodes, Services, Deployments, Secrets, DaemonSet
  - `etcd` distributed database and “object” watchers handle operational flow to workers
- The power of Kubernetes is in the extensibility of this declarative state system
  - Can create and operate on new resource objects (CRDs)
  - Custom controllers can handle operations related to a custom resource
Kubernetes: The Container Runtime Interface (CRI)

Allows an OCI compliant container runtime to service the kubelet

- Today the CRI is the (only?) common API for runtimes
  - However, CRI is not used outside of the Kubernetes ecosystem (today)
  - Implementing the CRI requires a runtime to represent a “pod sandbox”, not just containers
- The *dockershim* implementation is deprecated and removed as of K8s 1.24
  - All runtimes must provide a CRI-implementing endpoint:
    - Today: cri-o, containerd, Docker (provided by Mirantis), and Singularity
Kubernetes API Summary:

> HTTP/REST client API for K8s object model
> CRI over gRPC for kubelet - runtime
> Containers and images are OCI compliant
But what about an **API** for containers?
“Build. Ship. Run.”
(from an API perspective)
## Do APIs exist for “Build, Ship and Run”?

|--------|-------|------|
| **Dockerfile** is not a standard, but effectively a de-facto standard.  
*What really matters is the output: build tools take many inputs & provide unique capabilities but all produce OCI compliant images.* | **Yes! The registry/distribution protocol is an OCI standard today.**  
*Pushing and pulling images (and related artifacts) is standardized and the API is stable and well-understood.* | **In the Kubernetes context, yes; the K8s API is clearly defined and adopted. At the runtime layer only the formats are standardized**  
*CRI is the common factor among major runtimes. Underlying OCI types are standardized.* |

@estesp
Build

- Use of traditional Dockerfile + base image workflows remain significant
  - Docker build, BuildKit, buildah, and many others
  - The “API” is the Dockerfile de-facto syntax standard; innovation favors BuildKit
- Many tools now combine build workflows with K8s dev and deployment
  - Skaffold
  - Tekton
  - Kaniko
  - ..and many vendor tools that integrate CI/CD, GitOps, etc. with traditional build operations
- Interesting projects
  - “ko” - static Go binary build and push integrated with many other tools
  - Buildpacks (CNCF)
  - dagger.io
Given the common registry/distribution API and common format (OCI) most build tools directly handle the “ship” step to **any** OCI compliant registry

- All the build tools listed on the last slide support pushing images to cloud services, on-prem, or self-hosted registries

Innovations around “ship” will most likely come via “artifacts” support

- Signing support: cosign/sigstore & Notary v2
- Software Bill of Materials (SBOM)
- Bundling (images + Helm charts + ?)

Artifact and signing support are being collaborated on in various OCI and CNCF working groups, hopefully leading to common APIs and common formats
## Run

### User/Consumer

- **K8s or something else?**
  - With Kubernetes, you will have many options for additional abstractions, depending on managed service, roll-your-own, etc. APIs will be common across tools (e.g. K8s API, Knative, OpenFaaS, CF, ..)

- **Non-K8s container orchestration**
  - Google Cloud Run
  - AWS Fargate/AWS EC2
  - Hashicorp Nomad
  - Cycle.io, Azure Container Instances

### Builder/Vendor

- **Stay in the CNCF/K8s ecosystem?**
  - Build/extend/integrate with the Kubernetes API/control plane
  - Common API, broad adoption means lots of building blocks & integrations

- **Container runtime support:**
  - Easy path: CRI support/integration within K8s context
  - No clean option for integrating with >1 specific runtimes; potential at lowest layer (runc/OCI hooks) but has drawbacks
Decision Points

> Docker engine & API still a valid single node solution
  
  *Lots of tools/integrations, compose, and podman implements API*
  *Nerdctl+containerd providing a similar (client) experience sans Docker*

> Kubernetes provides the most adopted platform for tools/APIs
  
  *Allows for broad standardization from developer toolsuite to production*

> Most likely will adopt other APIs adjacent to K8s or containers
  
  *Cloud provider, infrastructure platform, cloud services (storage, network)*
The API Future

**Significant innovation around runtimes/APIs will stay in Kubernetes**

- SIG-Node, Kubelet, and OCI communities will innovate up through the stack to enhance container capabilities
  - KEPs for user namespaces, checkpoint/restore, swap support
  - Bonus that work done at these layers is implemented in all CRI runtimes and exposed via common APIs
- No clear path to commonality at the runtime layer itself
  - Effectively two main camps: Docker/containerd/runc or cri-o/podman/buildah/crun
  - Different design ideologies mean no real path to a common API for runtimes outside of CRI
Thank You!

Phil Estes, Principal Engineer

Container Runtime Leader
Amazon Web Services
Maintainer, containerd
OCI Technical Oversight
Board member

@estesp