SECURITY IN MULTI-TENANT CONTAINER AS A SERVICE PLATFORMS

Klarrio
STREAMING AHEAD

OVERVIEW

Containers are executable packages of software that can easily run on many different machines. For the most part, they can run independent of the distribution, software, libraries and processes that are present on the machine they are running on.

To do this, they bundle all the needed executables, libraries and files in one package and they take advantage of a form of operating system virtualization (on linux: namespaces and cgroups) to isolate the processes running in the container from the host machine.

So when we discuss container security and building a multi-tenant container as a service platform, it is natural to think that escaping that isolation is the thing to worry about most. In reality there are many other attack vectors. The most important ones are insecure container configuration, insecure networking and bad container images.

We will first discuss these before we discuss the actual container escapes.

INSECURE CONTAINER CONFIGURATIONS

Let's start with insecure container configurations. Stock container orchestrators are basically insecure by default. They allow very loose configuration of the containers to support situations where you want to partly break the isolation: things like containers in containers or containers that talk directly to hardware devices (e.g. GPUs or disks). Providing these configuration options to users is like giving them the key to their own jail: getting out is trivial.

In some container orchestrators you can (at least partly) disable these insecure configuration options. As an example, Kubernetes has had a mechanism to accomplish just this since release 1.8 (September 2017): Pod Security Policy (PSP). Using this system a Kubernetes admin can e.g. disable privileged mode (a mode wherein a container can access all devices on the host without restrictions). Historically, it has proven quite hard to get these policies right.

First of all, understanding the impact of the different options can be pretty complex and it is easy to accidentally forget to disallow an option that can lead to security breaches.

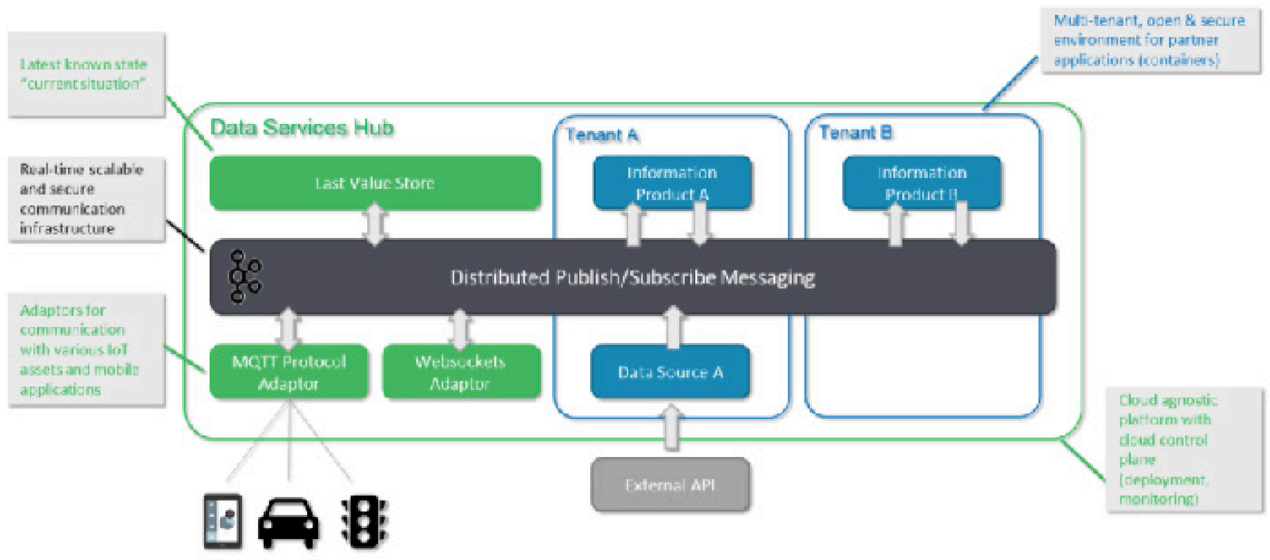
Second, PSP could not be extended, so you could not reject options for which there were no provisions. Partly because of this PSP itself never left beta and is now being deprecated.

There are alternatives: Open Policy Agent (OPA) + Gatekeeper or Kyverno are the ones with the most traction, but they are also quite complex to configure and difficult to get completely right.

Furthermore, to secure the container orchestrator you typically end up prohibiting a large part of the configuration options and requiring a fixed set of values for many of the remaining configuration options (or even worse, configuration options get silently replaced by the required options) which results in a confusing and frustrating end user experience.

DSH works on top of Marathon which has no provisions to limit configuration options out of the box. Instead of adding a validation/admission step via a marathon plugin - which would have created the same problems as listed above - an alternative approach was chosen: a proxy is put in front of the actual container orchestrator and direct access to the orchestrator is prohibited for tenants. This proxy allows a tenant to schedule applications using a custom container configuration description format. This format contains only a very basic but safe subset of all the container configurations options. We designed it from scratch, only adding options when a tenant actually needed them.

This default deny policy allowed us to create something that was pretty secure from the start while improving the user experience. Given the good results we had with this, it is definitely something we would consider for other projects even in the presence of (and on top of) newer solutions.


Latest known state "current situation"
Real-time scalable and secure communication infrastructure
Adaptors for communication with various IoT assets and mobile applications
Data Services Hub
Last Value Store
Tenant A
Information Product A
Tenant B
Information Product B
Distributed Publish/Subscribe Messaging
MQTT Protocol Adaptor
Websockets Adaptor
Data Source A
External API
Multi-tenant, open & secure environment for partner applications (containers)
Cloud agnostic platform with cloud control plane (deployment, monitoring)

INSECURE NETWORKING

By default most container runtimes allow containers to talk to other containers. In a multi-tenant system that is not the desired situation: one tenant should not automatically be able to access the services of another tenant. While all these services could (should?) be secured on their own (tls + authentication) it makes sense to control the network access between tenants from the container or container orchestrator level. On DSH (and in most of our kubernetes clusters) we use calico for that, but any system that implements networks policies can be used. In DSH we configure calico in such a way that tenants can never talk directly to other tenants. Basically, we only allow communication with kafka and a few platform services.
In DSH we do this to make sure all useful data ends up on kafka but other platforms might of course selectively allow some communication between tenants.

Container runtimes typically also allow containers to talk to the outside world. Unfortunately, that often also means that containers can talk to services on the host or in the cloud provider. Most of these services are well protected from the outside world but they are not always secured and hardened against attacks from within. To prevent attacks on these services from containers we can first of all turn off all services that are not needed. For the services that cannot be turned off and that are not sufficiently secured we can again use network policies to block access from inside the container. On DSH we block all access to the hosts & the cloud provider using calico.

BAD CONTAINER IMAGES

Container images are a very easy way to check out and deploy a new (version of an) application and if you want to try something brand new, you're more likely to find a container image than a native package.

But unlike packages in your distro (which typically get verified by the company behind the distro) not all of the available images are harmless. Malicious actors are always looking to put bad containers in publicly available container registries.

To do this they look for popular applications that do not have a standard container image and they create their own container images that bundle these applications together with malicious content. If a standard container image already exists they use techniques similar to domain name squatting to trick people into deploying the wrong container. An attacker can even put a clean container image in a container registry and later on (when people have started using it) replace it with a malicious image: when a container restarts it will pull in the bad image.

Once such a bad container image gets deployed many things can happen: it can be used to automatically attack the container orchestrator or the services running on the machines from the inside, it can lie dormant but provide a backdoor that attackers can use later on, it can open a reverse shell to another system controlled by an attacker or it can simply run some extra applications that steal cpu cycles from the cluster.

Because of the inflation of crypto prices attacks like these have become more and more popular. Attackers focus on mining crypto currencies that are hard to mine on GPUs or ASICs like monero. For these, simply stealing cpu cycles (and memory) can result in considerable financial gain for an attacker.

In DSH we deliberately disallowed tenants from pulling in containers from public container registries: all containers need to come from registries associated with and trusted by DSH. These registries are controlled by the tenants which at the minimum requires them to explicitly copy over images before they get deployed and because of this attackers cannot simply replace images in the registry.

Next to that, having dedicated registries makes it easier to scan images for known vulnerabilities, known exploits and even for known crypto-currency mining tools. Partly because of this DSH is moving over to Harbor (https://goharbor.io) as a container registry in order to improve our container scanning options.

CONTAINER ESCAPES

Usually a container escape is possible because of vulnerabilities in the kernel or the container runtime: the container can still write to some files it shouldn't be able to write to, it can still do system calls it isn't supposed to be able to do, ... But the vulnerabilities can basically reside in any service that interacts with the docker container (as an example: even a bug in a log collector could be exploited by crafting the right log message).

To prevent these attacks we need to harden the host, the runtime and the orchestrator and we need to keep everything up to date. For many of the systems we use the Center for Internet Security publishes the CIS benchmarks: a set of configuration guidelines that result in a more secure configuration. For DSH we tried to apply the relevant benchmarks which is not always an easy task as some OS security guidelines might actually conflict with running containers and services.

LIMIT THE BLAST RADIUS

Everything we talked about until now was related to preventing attacks but no matter how much effort we put into securing everything, the security will never be absolute: attackers are constantly on the lookout for new ways of escaping containers and exploiting services and because of this new attacks are uncovered on a regular basis. Tenants can be tricked in deploying these new attacks and container scanners will typically not pick them up. So we still need to prepare for container escapes.

When such an escape happens there are still some things we can do to limit the blast radius: when an attacker escapes the container it would be preferable if attackers could not access files and devices on the host machine. One thing we can do to prevent that is making sure that when something escapes that it is running with a user id that is not used on the host. User namespaces can be used to map user ids in containers to (non-existing) user ids on the host. Preferably we would like a different mapping per tenant (or even per application) to also make sure that no two tenants (or even applications) use the same user id (otherwise an escaped application could read files from another tenant). Unfortunately, multiple namespaces are not always easy to implement. Because of this in DSH we force tenants to use fixed, non-overlapping sets of user ids.

One other thing we could do is disallow outgoing traffic by default (whitelist outgoing traffic). This would prevent an attacker from setting up a reverse shell or applications inside the container from accessing a mining pool. Unfortunately whitelisting all destinations requires a lot of management. Because of this we did not yet implement it in DSH but it is definitely something to consider for the future.

WRAPPING UP

This text described the most important things to take into account with regards to security when designing a multi-tenant container as a service platform and how this applies to DSH. However, there is much more that can be said about the topic. If you want to learn more, a good book about the subject is “Container Security” by Liz Rice.

CONTACT US

BELGIUM
Tel: +32 (0)3 331 99 33
Email: info@klarrio.com

NETHERLANDS
Tel: +31 (0)10 313 25 24
Email: info.nl@klarrio.com

GERMANY
Tel: +49 2407 50 23 180
Email: info.de@klarrio.com

UNITED STATES
Tel: +1 919 649 2997
Email: info.usa@klarrio.com

AUSTRALIA/PACIFIC RIM
Tel: +61 402 850 059
Email: info.aus@klarrio.com

WWW.KLARRIO.COM

Klarrio is a one-stop professional services and cloud-native integrator and prototype developer for organizations that need deep expertise in Open Source technology. We help you define and accelerate your transformation to new business models, disruptive technologies and cloud-native value-added services and vendors.

We do it securely and cost effectively, while providing you with a team of cloud-native experts to help transfer our knowledge to your own, internal IT resources.

@klarr_io
@Klarrio
linkedin.com/company/klarrio

Klarrio
STREAMING AHEAD