

STEVE LASKER

Steve Lasker / June 27, 2018 / Azure, Container Registry, Docker, Tips, Uncategorized

Key Differences between VM and Container Vulnerability Scanning

As we explore the various options for integrating vulnerability scanning into containers running in Azure, we encountered many different approaches and requests. I was initially writing a document for internal discussions, outlining some of our internal goals for how we want to think about container security, and why we might take one approach or another. While I'd suggest Azure has the best breadth and depth of containers in any of the major clouds, we are a large company, and not everyone is familiar with containers. In our busy lives, we only have a few seconds of mental processing, before we look to apply what we've just learned to the problems we have. Containers solve many problems, some more short term than others. However, until you've inverted your thought process, and think about a container-first mindset, you'll find yourself using a new tech as a better tool, but not nearly unlocking its potential.

As I wrote the document, there are lots of details and examples I wanted to outline to help people think about vulnerability from a container first mindset. Many of which represent opinions, or "best practices". As I wrote the Docker tagging best practices for tagging and versioning docker-images post, I thought about what makes a best practice for a new and rapidly evolving technology?

I don't consider myself a security expert, and I certainly don't play one on Twitter. I have worked in the industry for many years and have watched the balance of secure, usable and practical. We don't always get these right. And I don't believe the concept that something must be hard to be secure. I believe in good, obvious processes and realistic expectations for making something secure. After all, what's the first thing developers do when security gets in the way? ...we disable it. Not a very secure result, now is it?

To put some context to this paradigm shift, below are a number of concepts to consider when you think about adopting a container scanning solution. Just because something has been done a certain way for years, doesn't mean it's still relevant.

Virtual Machine and Container Differences

Below are several concepts we think about when running compute. I've provided a **VM** centric and **Container** centric view to each one.

“

Container Bigot Disclaimer: For those working with products like Vagrant, Terraform and other awesome products that automate VM creation; this isn't meant to suggest the challenges couldn't/aren't addressed with them. And while I'm far to say anything you can do in a VM, I can do better in a container, I think we all recognize containers, and specifically Docker has brought about the next foundational shift in how we package and run software.

Where Is Fast a Priority

VM

VMs are typically measured in the length of up-time. How long was the VM running before it had to be rebooted or retired? While startup time is always important, VM startup time is typically measured in minutes. As long as they come up in single digits, they're considered “the norm”.

Container

Containers are assumed to be relatively short-lived processes. They may be long running services, like websites or APIs, but the expectation is they get moved, updated, scaled with far more frequency.

Containers are measured in the number of seconds, or preferably milliseconds from the time of docker run to processing requests. Customers are also looking for docker pull to complete in seconds, reducing the amount of time to run a new image on a new node.

Considering containers are becoming the base of functions, where a workflow kicks off, and each step of the process may be its own container that's pulled, instanced and tossed in the matter of seconds, anything that can slow down the instancing must be changed.

Configuring and Applying Changes

VM

While projects/products like Terraform, Spinnaker, Vagrant and others aim to automate the final configuration of VMs, they are most commonly configured post instancing. Teams routinely

remote into the machine, applying software or patches. Even when it's not intended, many teams still do, and these long running nodes need to be protected.

Container

Containers are considered immutable. Meaning, once an image is instanced, it's not changed. One would likely argue a running container shouldn't be changed as it defeats the ability for a new instance to reflect the exact behavior.

Rather than update a running container, the image dockerfile is changed to implement the configuration change or software. The image is rebuilt, pushed to a registry, and awaits its time to be instanced.

Applying OS & Framework Patching

VM

Like **configuring and applying changes**, teams work to update templates for deployments. However, most operations teams will monitor and update their running VMs. These are updated, without necessarily understanding what apps are running, as it's just too difficult for companies to track which VMs could be impacted.

Container

Also, like **configuring and applying changes**, a container isn't patched while running. The image is rebuilt with the OS and framework patches. In this model, the image can be scanned and tested before its deployed.

Receiving Data

VM

As VMs are designed as long running instances, VMs are designed to host data. It may involve downloading EDI files to be processed, customer binaries to signed, pictures to be processed. In each case, the VM has the ability to download malicious content. If the VM is found to be vulnerable, the data on the VM must be scrubbed and typically recovered.

Container

Containers are considered immutable processes that may be run for seconds or hours. Any data that may be placed in a container would/should be considered disposable. The container must be assumed to fail at any point.

Containers store their data externally, using volume drivers or data services like DocDB, Relational or Blob storage.

An image that becomes suspect could/should be deleted without any worry for what temporary storage that may have been written.

Multiple Lines of Defense

VM

In the VM world, the VM itself has multiple lines of defense. It may be secured within a VNNet, locked down to where it may communicate through firewalls, which also run on the VM. Scanning is placed within each running VM.

Container

In the container world, securing the process within the container is the last line of defense. Containers themselves do not actively run scanning. They are lightweight processes. The container host may likely run scanning for what images are being deployed. In the best world, the image being requested to run is verified against a list of pre-scanned images. If the image has been already, and recently scanned, it's allowed to run.

The first line of defense for containers is the source where they are kept, before deployment. This is referred to as the container registry. Each company running containers will host their own private registry. The scanner keeps an inventory of all images in the company's private registries. Scanning at the source allows offline scanning, before the image has been requested to be deployed. This makes instantiating a container fast as it's just verified to have been seen prior. The host monitoring will either block or scan images it hasn't previously seen.

Vulnerability Detection and Impact

VM

In the VM world, for an enterprise solution, scan results are reported back to a central server. However, the report indicates what was found and the action. It doesn't necessarily trigger a recursive action for other like VMs.

Container

In the container world, the registry is actively scanned. Each protected node may report back what images it's running. As the scanner finds existing vulnerabilities in newly pushed images to the container registry, or it finds new vulnerabilities in existing images in the registry, the scanner determines the impact. It can use the data for what images are running (referenced by their image digest) to understand the impact of the vulnerability.

If the scanner happens to find a vulnerability that was in a single image that was never deployed, the risk is non-impactful. The action is as simple as either quarantining the image, or deleting it.

If the scanner finds a new, critical vulnerability in a base image that has 50 other derived images, which are deployed on 220 production nodes, the impact is far more severe.

The fundamental difference between containers and VMs is where the scanning occurs, with the analysis and action taken. In the container world, a single scan and detection can account for hundreds to thousands of nodes being remediated.

Remediating Vulnerabilities

VM

Remediating vulnerabilities in a VM typically involves applying a patch to a running VM. The intent involves the patch removing or replacing the initial offending binary or configuration.

Container

When vulnerabilities are found with containers, a new, remediated image is built, pushed and deployed. Some of the key differences being the original configuration and/or binary shouldn't have any traits in the newly built image. When the vulnerability is found in a top-level image, meaning an image that copied the offending binary in its dockerfile, it's built once and deployed to the intended nodes. To remediate a base image, all the subsequently derived images must be rebuilt. A shameless plug for ACR Build. In the VM environment, without products like Terraform, each node must be individually updated.

The Scanning Paradigm Shift

With the above differences, we can see the approach for scanning containers would be very different from Virtual Machines. The basic approach of containers represents a paradigm shift from working with virtual machines. While Virtual Machines were a major change in 2001, improving many challenges of installations of the OS and software, installed on dedicated hardware, they are now nearly two decades old. We've learned a lot since. We've changed the workloads from machines under someone's desk, in a closet with hopefully enough air conditioning, or a bigger closet in the basement of an enterprise. Expectations of 3 months to get infrastructure acquired and provisioned are long gone. Customers spin up instances in an instant, on-demand, with expectations of seconds in a cloud they have no care as to where it runs.

Containers aren't just a better mousetrap, they account for many of the lessons the industry has learned and adjusted to as the workloads have grown and transitioned from assisting humans at work. Compute now accounts for most of the work, and the humans control and interact with the computers. As a result, it's no longer just an annoyance when the excel spreadsheet is locked or not available. Having a system down, just for a few minutes, can kill the reliability of the business, cost them millions of dollars in fines or lost revenue, or cause complete failure of the business.

As with any new tech, taking the time to digest its impact, understanding the opportunity for things it can resolve, and things it can't are just as important as understanding the calorie count of your favorite new dessert.

Posted in Azure, Container Registry, Docker, Tips, Uncategorized and tagged Azure, Containers, Docker, Security. Bookmark the permalink.

1 Comment Add yours

1



Dave Rendón on August 6, 2018 at 12:45 pm

Very informative Steve, thanks for sharing!

Reply

 Like

+ Leave a comment

Write a comment...

Log in or provide your name and email to leave a comment.

☐ Email (Address never made public)

Name

Website (Optional)

☐ Email me new posts

☐ Instantly ☐ Daily ☐ Weekly

☐ Email me new comments

☐

Save my name, email, and website in this browser for the next time I comment.

Comment

Search ...

Search

Blog at WordPress.com.