Twistlock™

Modernizing Traditional Security:

How Containerization Makes Security and Compliance Easier

Keith Mokris

Sr. Product Marketing Manager

Joined Twistlock 18 months ago with previous security experience in the mobile app security space

Based in Portland, OR

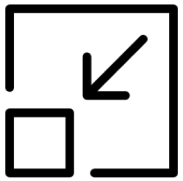
@keithmokris

What We'll Cover Today

- How containers change the security paradigm
- Use case from a "lift and shift" approach
- Container security best practices

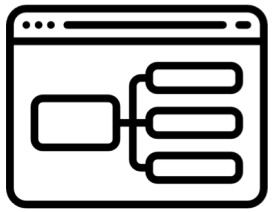
How Containers Change the Security Paradigm

Container Characteristics



Minimal

Typically single process entities



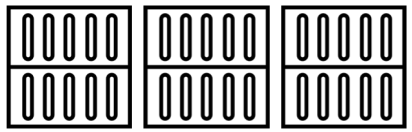
Declarative

Built from images that are machine readable

Predictable

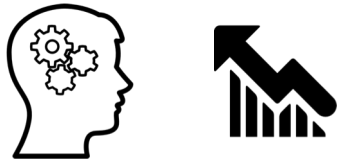
Do exactly the same thing from `run` to `kill`

What’s Different About Securing Containers?



Many more entities

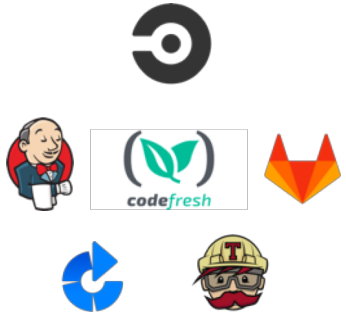
High rate of change, much more ephemeral

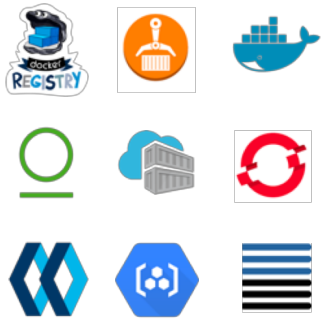


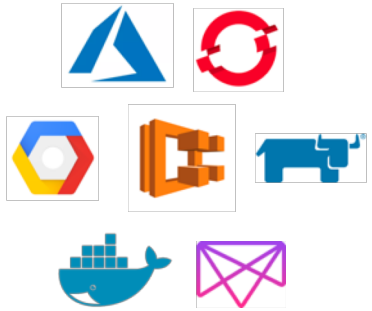
Security is largely in the hands of the developer

Security must be as portable as the containers

Steps and Tools Involved with Building, Shipping, and Running Containers







BUILD

Developer writes a Dockerfile, which includes a base image, maintainer, run instructions, etc., that is then built into a an image

SHIP

Images are stored in a repository which is leveraged to deployed images to run in various environments of your choosing

RUN

Teams can run containers in their own data center or on various cloud services like the ones shown here

An Example Dockerfile

Dockerfile

Users ▸ keithmokris ▸ Desktop ▸ Dockerfile

```
1  FROM ubuntu:18.04
2  COPY . /app
3  RUN make /app
4  CMD python /app/app.py
```

FROM creates a layer from the ubuntu:18.04 Docker image

COPY adds files from your Docker client's current directory

RUN builds your application with make

CMD specifies what command to run within the container

A Modernization Use Case

“Lift and shift”

Moving a monolithic application to containers – no rearchitecture required.

The Customer

- An environmental science and engineering consultancy serving some of the largest civil waterworks projects in the world.
- 12+ year old application — from IBM xSeries, to VMWare ESX on premise, now running in Azure.

The Challenges

- Maintaining consistency across dev, test and prod environments was difficult and resource intensive
- New builds could not be automated — leading to long deploy cycles
- Disaster Recovery was complicated — requiring days not minutes
- Governmental compliance / security standards required manual configuration, and security rules had devolved over 14 years of continual updates

To address these challenges, the application was repackaged as a container.

Security Benefits Across 3 Pillars

Vulnerability Management

Compliance

Threat Protection

Vulnerability Management

Traditional

- Vulnerability scanning done after application build, in a different workflow
- Ops had no visibility into vuln posture in running
- Reliant on point in time scans

Modern

- Integrated vulnerability scanning into Jenkins
- Automated blocking of builds exceeding vuln thresholds
- Dev + Ops visibility into vulnerability posture of running apps – in real time

Compliance

Traditional

- Ops-owned with manual feedback loop to dev teams
- Manually created compliance documentation and rules
- No version control of app components – point in time checks

Modern

- Shared ownership throughout lifecycle – ops defines rules, dev tools like Jenkins enforce
- Only trusted components are allowed to run in production
- Shared, central view into compliance state of all assets in environment

Threat Protection

Traditional

- IDS / IPS rules devolved over time – overly permissive
- Reliant on perimeter firewalls + limited anti-malware capabilities.
- Manual rule creation

Modern

- Whitelist based, tied to every build
- Layer 3,7 firewalling + container and host rules all created and deployed at scale
- Fully automated rule creation and enforcement

There are inherent security benefits from repackaging monolithic apps in containers

Container Security Requirements

Best practices for using containers

Know the Source and Content of Your Images

Do you know where your containers come from?

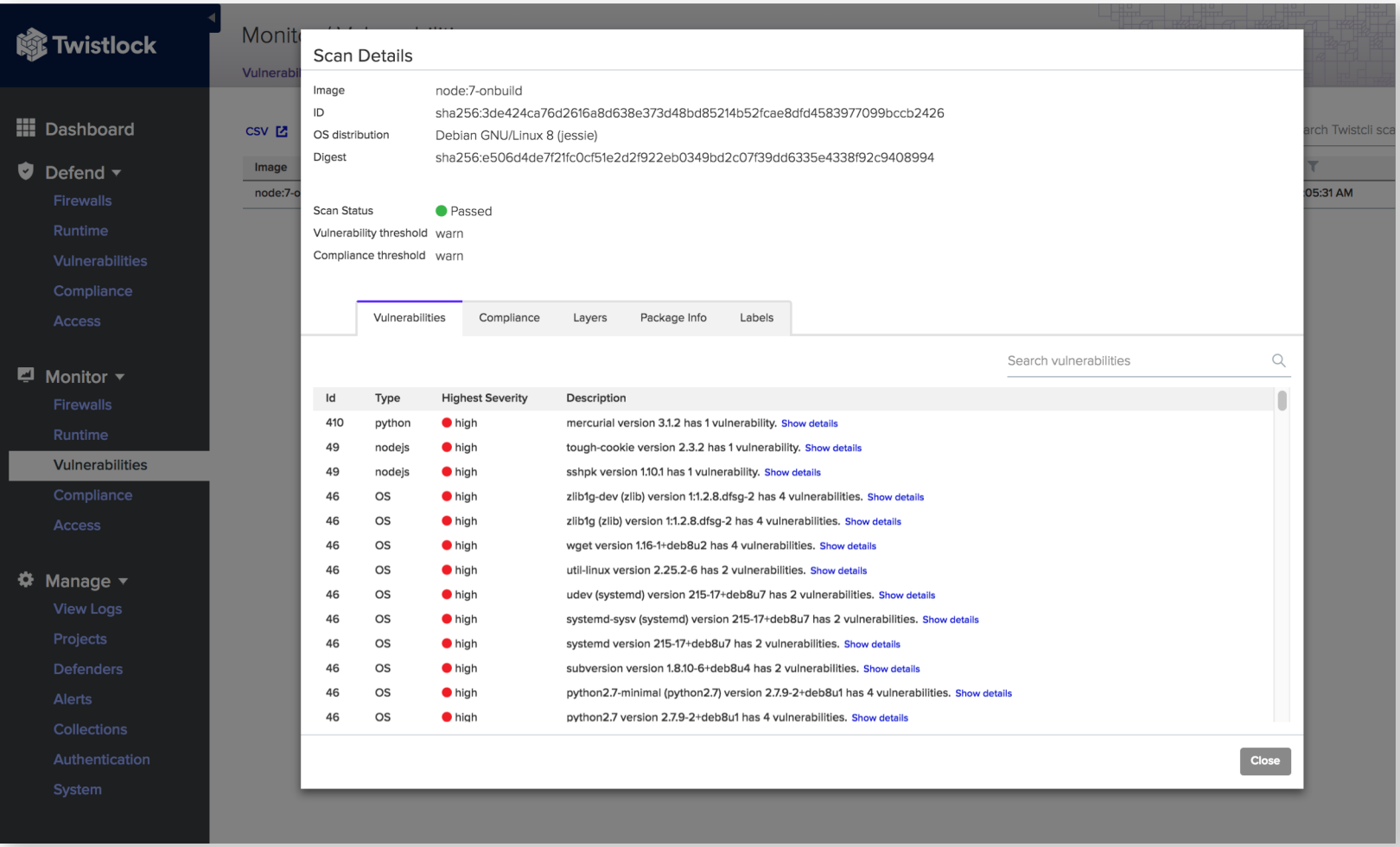
Are your developers downloading container images and libraries from unknown and potentially harmful sources?

Do the containers use third party library code that is obsolete or vulnerable?

Eradicate Vulnerabilities from Your Containers

Container images are typically built on some base image, which is itself built on top of other base images

A developer may grab a base image and other layers from public third party sources. These images and libraries may contain obsolete or vulnerable code, thereby putting your application at risk



Would you want this image running in your environment?

An example hellonode image using a GitHub tutorial

270 High Vulnerabilities
900 Medium Vulnerabilities

Yeah, no thanks!

Harden Your Images, Containers, Daemons and Hosts

Implementing the Docker and Kubernetes CIS Benchmarks is essential

Automation provides tremendous benefits

Example checks:

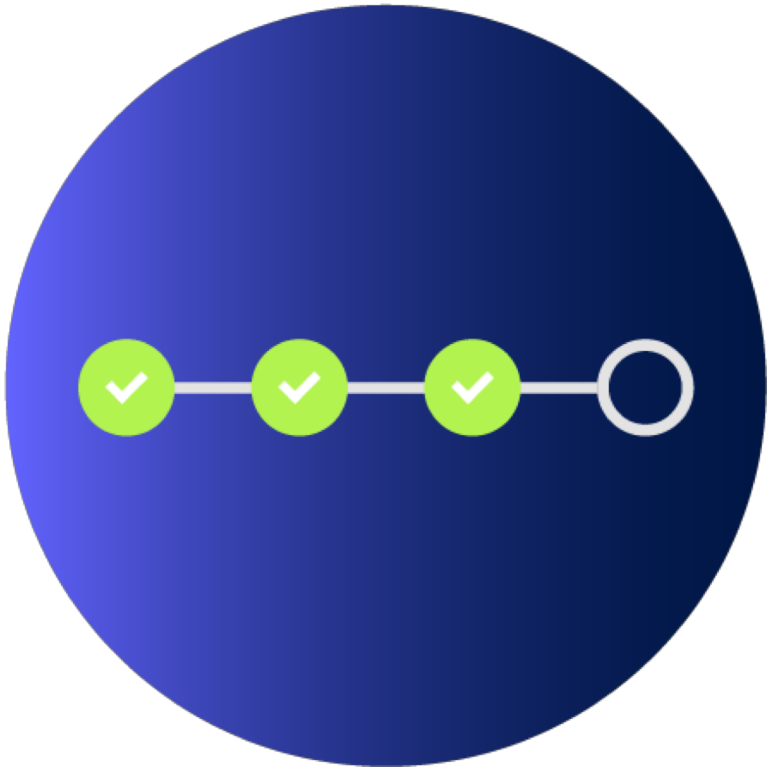
- Do not mount sensitive host system directories on containers
- Open only needed ports on container
- Only allow trusted users to control Docker daemon
- Don’t allow secrets in clear text environment variables

Leverage Your CI/CD Pipeline Tools

The best place to detect and fix security vulnerabilities is during development and as part of the CI/CD work flow

Use your CI/CD tools to initiate security scanning whenever a new image is constructed and consumes the results in the native CI/CD console

Twistlock offers the ability to fail a build and set grace periods based on granular policies



An Example with Jenkins

hellonode 28

Pipeline | Changes | Tests | Artifacts | Logout

Branch: master
Commit: 1985768

20s
a month ago

Changes by noreply
Started by user Keith Mokris

Start → Clone repository → Build image → Scan → **Publish** → Push image → End

Publish - 10s

Publish Twistlock analysis results 10s

```
[TWISTLOCK] Publishing images analysis results
[TWISTLOCK] Scanning images  on master
[TWISTLOCK] Waiting for scanner to complete


Please provide Console credentials:
No CA cert was specified, using insecure connection
Vulnerabilities
---------------
Image                               ID                 CVE                 Package             Version    Severity    Status
-----                               --                 ---                 -------             -------    --------    ------
docker.io/keylowe/sandbox:latest    311fbba058a4e9ba   NODE-SECURITY-583   http-proxy-agent    2.1.0      high        open
docker.io/keylowe/sandbox:latest    311fbba058a4e9ba   NODE-SECURITY-611   deep-extend         0.5.1      low         fixed in >=1.0.1
docker.io/keylowe/sandbox:latest    311fbba058a4e9ba   NODE-SECURITY-594   deep-extend         0.5.1      low         fixed in >=6.5.2
docker.io/keylowe/sandbox:latest    311fbba058a4e9ba   CVE-2018-3721       lodash              3.10.1     low         fixed in >=4.17.5

Compliance
----------
Image                               ID                 Severity    Description
-----                               --                 --------    -----------
docker.io/keylowe/sandbox:latest    311fbba058a4e9ba   high        Image should be created with a user
[TWISTLOCK] Found 5 vulnerabilities in 1 images
[TWISTLOCK] Computing warning deltas based on reference build #26
```

Enforce Role Segregation and Access Control for Your Docker Environment

Leveraging an access control tool that offers integration with enterprise directories, fine-grained access management, and also logging and auditing for all your Docker daemon accesses provides vital security benefits

Automate Anomaly Detection and Threat Defense in Container Runtime

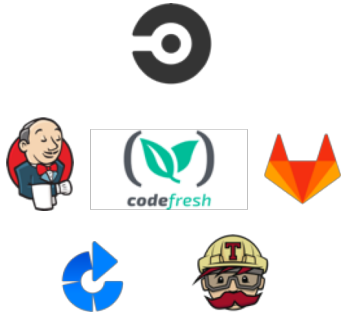
Containers should be minimal, declarative, and immutable. Those characteristics mean that it is actually possible to build a reliable baseline for the containerized application. Using this baseline in runtime you can detect anomalies and active threats much more accurately than with monolithic applications that change frequently.

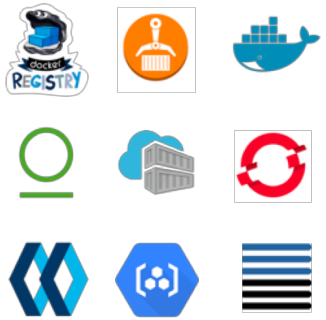
Automating the baselining process, the detection actions, as well as the enforcement, is the only way to scale up runtime security

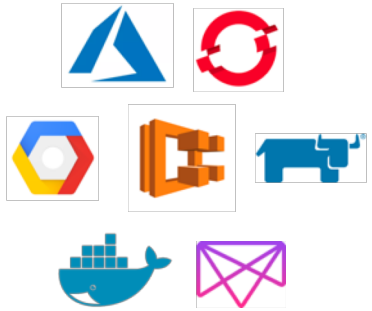
Perform Regular Audits to Prevent Image and Container Sprawl

Containers are flexible and easy to use, but that also means it's easy to end up with many instances of containers and images, some of which may be obsolete and risky -- performing regular audits to identify nonessential containers and images and eliminate them from your systems.

How Does Twistlock Provide Security Across the Application Lifecycle?







BUILD

Integrate Twistlock with any CI tool you are using to provide vulnerability and compliance scanning and enforcement for your container images, hosts, and serverless functions.

SHIP

Continuously monitor your registry to ensure you are always shipping secure code into production.

RUN

Protect your running applications with layer 3 and layer 7 cloud native firewalls, powerful runtime defense, and access control — providing defense in depth to prevent next generation attacks.

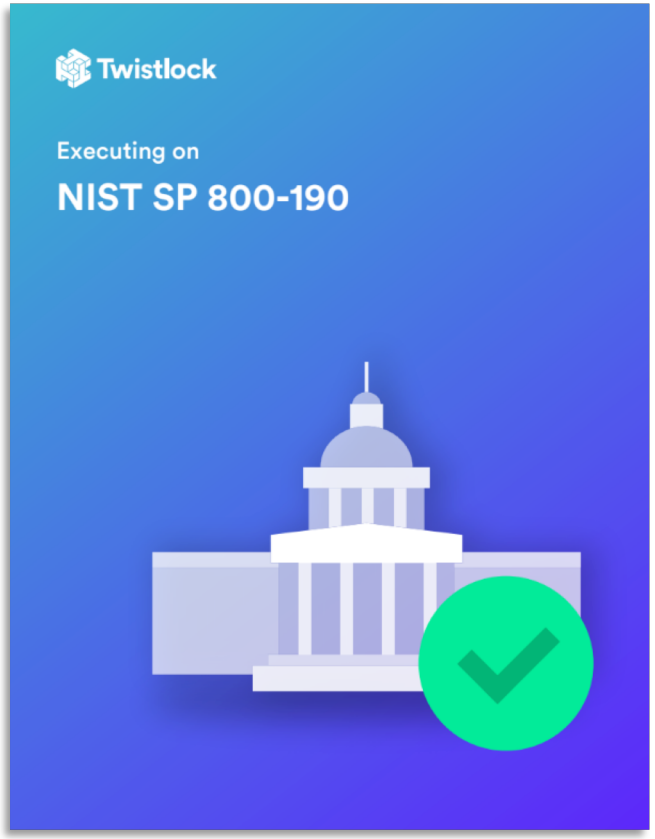
NIST SP 800-190, Application Container Security Guide

Co-authored by Twistlock CTO John Morello

Provides official guidance on securing cloud native applications

Download our companion guide at:

Twistlock.com/NIST



Questions?

@TwistlockTeam or keith@twistlock.com