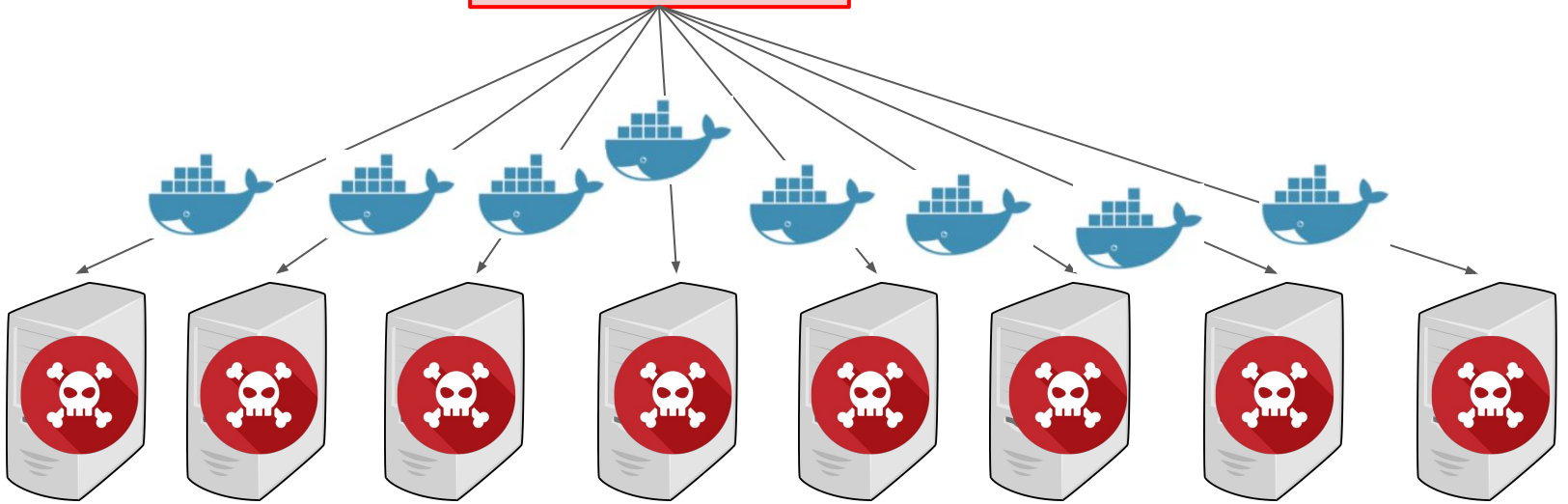
Docker Image Provenance

All Day DevOps 2018

Adam Lewis
Chief Security Architect
Motorola Solutions

@lewiada



How do we stop this?

assures connection to the right server (assuming no TLS vuln)

Authentication / Access Control (who can push Image to registry)

Sign Images in location not reachable from Internet

Docker Content Trust

The Update Framework

A ***specification*** for securing the distribution of software:
Application updaters, Library package managers, System package managers

CLOUD NATIVE COMPUTING FOUNDATION

Designed by security researchers Justin Samuel and Justin Cappos in 2009, open sourced in 2010 (https://theupdateframework.github.io), and given to the Linux Foundation's Cloud Native Computing Foundation in 2017

Builds upon and improves upon Thandy, the updater used to secure the distribution of the Tor browser - which had a threat model which included nation-state attackers; robust!

Security Guarantees

TUF **uses a combination of crypto keys** (both online and offline) and **metadata** to defend against *variety of attacks,* both against the software AND the signing keys

PROVENANCE

prov·e·nance

/ˈprävənəns/

noun

the place of origin or earliest known history of something.
"an orange rug of Iranian provenance"
synonyms: origin, source, place of origin; More

- the beginning of something's existence; something's origin.
  "they try to understand the whole universe, its provenance and fate"
- a record of ownership of a work of art or an antique, used as a guide to authenticity or quality.
  plural noun: **provenances**
  "the manuscript has a distinguished provenance"

FRESHNESS

Arbitrary Image The attacker provides an image they created in place of a package the user wants to install.

Replay Attack An attacker replays older versions of correctly signed image or metadata, causing clients to install an old image with security vulnerabilities the attacker can exploit. The attacker can then compromise the deployment by exploiting the vulnerable container.

Freeze Attack Similar to a replay attack, a freeze attack works by providing metadata that is not current. However, in a freeze attack, the attacker freezes the information a client sees at the current point in time to prevent the client from seeing updates, rather than providing the client older versions than the client has already seen. As with replay attacks, the attacker's goal is ultimately to compromise a deployment that has vulnerable versions of image installed. A freeze attack may be used to prevent updates in addition to having an installed image be out of date.

Extraneous Dependencies The attacker rewrites the image metadata to have additional packages installed alongside a package the user intends to install. For example, the attacker provides metadata that incorrectly states that package foo depends on bar. This will cause bar to be installed when it is not desired or needed. If bar has a security vulnerability, this allows an attacker to compromise the client's system.

Endless Data This attack is performed by returning an endless stream of data in response to any download request. This may cause the package manager to fill up the disk or memory on the client and crash the client's system.

SURVIVABLE KEY COMPROMISE

Threat Model begins with a super important assumption:
ASSUME COMPROMISE

Solves the Key Management problem

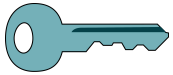
Separates responsibilities across a hierarchy of keys (online and offline) → loss of any of the online keys is not fatal to the security of the system.

Enables Transparent Key Rotation

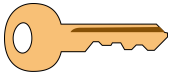
Understanding TUF keys and Roles

The Update Framework

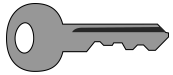
ROOT:
- Lists all other keys to be trusted by host
- ideally kept offline (only need for initialization and rotation of the other keys)
- Is what enables recovery from compromise of other keys



TARGET:
- signed by the targets key (the workhorse key)
- lists hashes of the actual files that clients want to download



SNAPSHOT:
- Lists hashes of the targets metadata file and root metadata file that should be trusted by the client



TIMESTAMP:
- lists the hash of the snapshot.json file to be trusted by the client
- frequently resigned

It is the combination of these keys that enables provenance + freshness + survivable key compromise

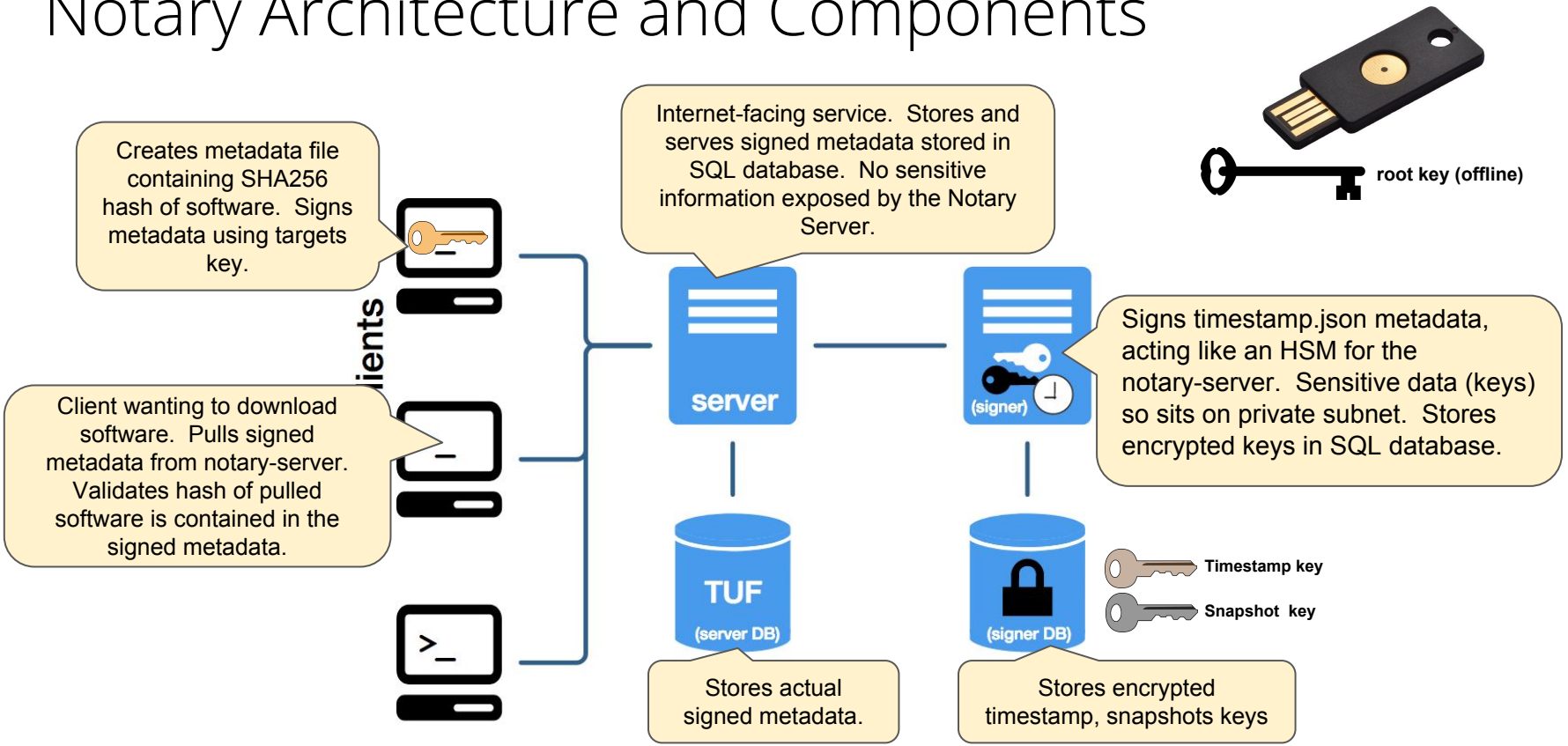
Notary

- Opensource Implementation of TUF written in Go by security engineers at Docker**
  - *comprises a metadata server, online signer, and client for running and interacting w/trusted collections (either as a signer or a verifier)*
  - *Implementing TUF, provides high levels of trust over digital content using strong cryptographic signatures.*

- Most widely adopted implementation of the TUF specification
- Along with TUF, contributed to the CNCF

** but intentionally independent of Docker

Notary Architecture and Components



Docker Content Trust

- Seamless Integration of notary client library into Docker client, introduced in Docker v1.8

```
export DOCKER_CONTENT_TRUST=1
```

- Once enabled, no docker operation that you can do that is not based on cryptographic signatures
- Every docker push will sign the image, every docker pull/run will verify the image
- Designed with UX in mind - make security usable!
- Docker already signing all their official images on Docker Hub
    - Goal is to make default, currently “opt in” … goal is every docker pull/build/run/push is secure by default

What TUF/Notary/DCT is **NOT**

- TUF / Notary / Content Trust validates the publisher of content (origin authenticity), and that it has not been tampered with since being built and placed in the registry (integrity)
- It does NOT inspect the code inside the package to see if it is safe vs. malicious
- (and therefore) provides ZERO guarantees around the security, vulnerabilities, or lack of vulnerabilities within a Docker Image
- There might still be insecure applications libraries, critical vulnerabilities in the image base layer, vulnerable packages, etc. inside the Docker Image

These things also need to be addressed. Notary / DCT is one tool in the security toolbox. You should also be doing continuous scanning of your images as you build them before pushing them to your registry, and also while in the registry as new CVE might be discovered post-build

PUSH

docker push <image>:<tag>

hash images digest, create and sign targets.json file using target private key (on disk)

Upload the signed targets.json metadata to the Notary server

SERVER DB

NOTARY SERVER

SQL

SIGNER DB

NOTARY SIGNER

SQL

docker

DOCKER REGISTRY

Generate timestamp metadata

submit metadata for signing

get timestamp key

sign timestamp metadata

signed timestamp metadata

store all metadata, including timestamp

Upload the actual Docker Image to the Docker Registry

PULL

SERVER DB

SIGNER DB

NOTARY SERVER

SQL

NOTARY SIGNER

SQL

DOCKER REGISTRY

docker pull <image>:<tag>

request all signed metadata for <image>:<tag>

retrieve signed metadata files from notary-server's DB

send signed metadata files back to client

Validate signatures on metadata files, parse targets.json for hash of requested image digest

pull by hash (using hash of <image>:<tag> listed in targets.json metadata

<image>:<tag>

compare hash of Image digest against hash listed in targets.json

*** Appropriate sacrifices to the demo gods were made within the past 24 hours.*

Try it Out!

```
$ export DOCKER_CONTENT_TRUST=1
```

Any push/pull to/from dockerhub.com will be cryptographically signed/ cryptographically verified

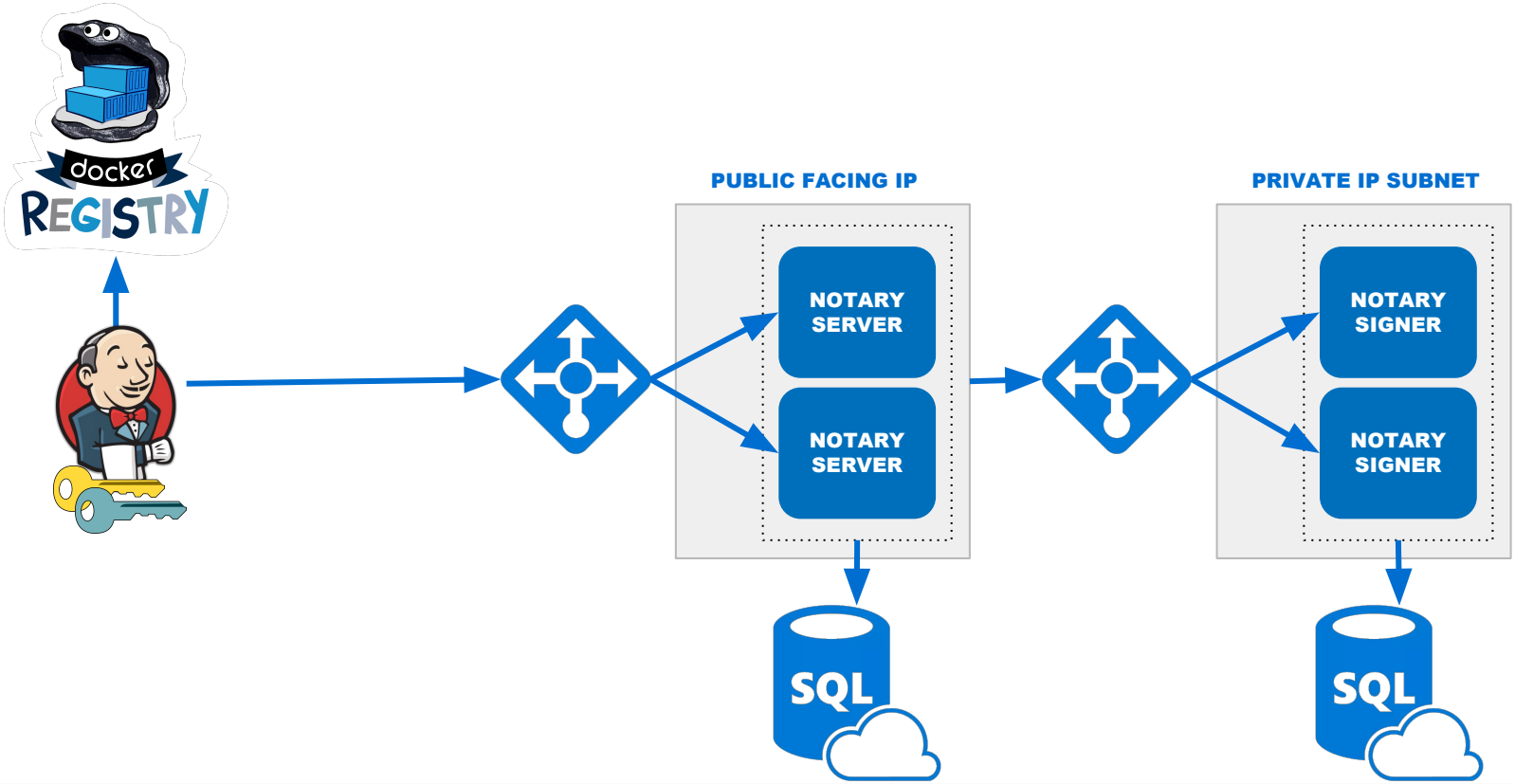
- Private keys (for pushing) will be generated in ~/.docker/trust/private
- Signed metadata (from pulling) can be found in ~/.docker/trust/tuf

If this is your first time, new keys will be generated - you will be asked for a passphrase to encrypt/decrypt them, no sensitive data stored in plaintext on disk!

Bypass prompts for passwords:

```
$ export DOCKER_CONTENT_TRUST_ROOT_PASSPHRASE="alldaydevops2018"
$ export DOCKER_CONTENT_TRUST_REPOSITORY_PASSPHRASE="alldaydevops2018"
```

CONTINUOUS INTEGRATION, CONTINUOUS SIGNING



Summary of Take Aways

- Your registry represents an attractive and single point of attack for adversaries with a massive blast radius
- TUF is a specification. Notary is an open source implementation of TUF written by security engineers at Docker (but can also be used to sign digital content other than Docker images). Docker Content Trust is a seamless integration of the Notary client library into the Docker client.
- The combination of signing keys enables not only Image Provenance, but also Freshness and the ability to Survive a Key Compromise
- Usability was treated as a first class citizen, seamless integration into familiar Docker workflows, including support for pipeline integrations. Continuous Integration + Continuous Signing!
- Don't' forget to automate other Docker security functions to , like continuous scanning

THANK YOU

Adam Lewis
Chief Security Architect
Motorola Solutions

@lewiada

Thank You All Day DevOps Sponsors

Platinum Sponsors

Sonatype

cloudbees

Gold Sponsors

puppet

GitLab

XebiaLabs
Enterprise DevOps

GENERAL DYNAMICS
Information Technology

cisco DEVNET
developer.cisco.com

Carnegie Mellon University
Software Engineering Institute

CHEF

SCALED AGILE

Media Sponsors

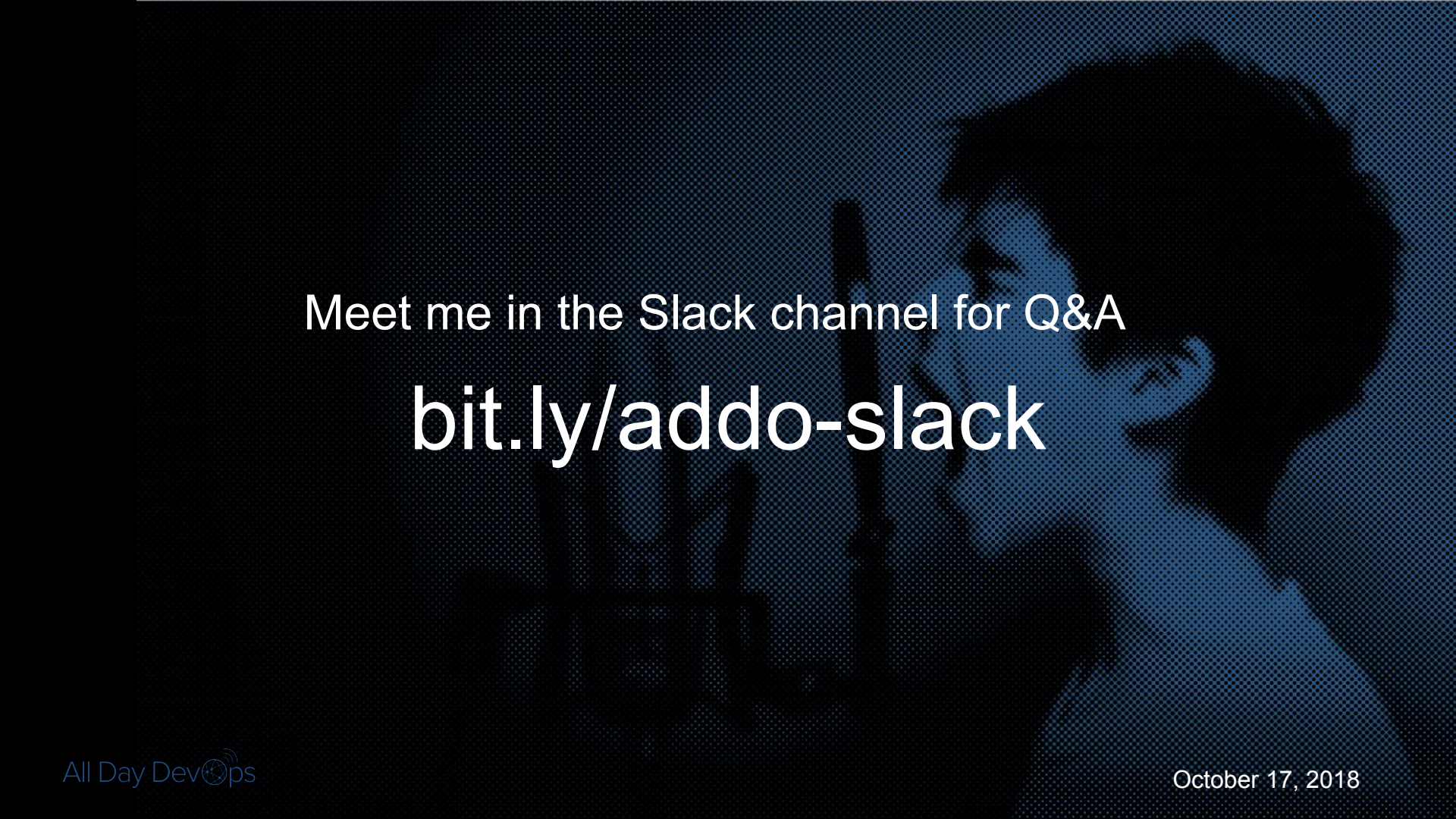
TechBeacon

DevOps.com

DZone

Solutions Review

Thank You All Day DevOps Supporters


Meet me in the Slack channel for Q&A
bit.ly/addo-slack
All Day DevOps
October 17, 2018