SAP SuccessFactors

Container Security:
It's All About the Supply Chain

Michele Chubirka, SAP
October 28, 2021

THE BEST RUN SAP

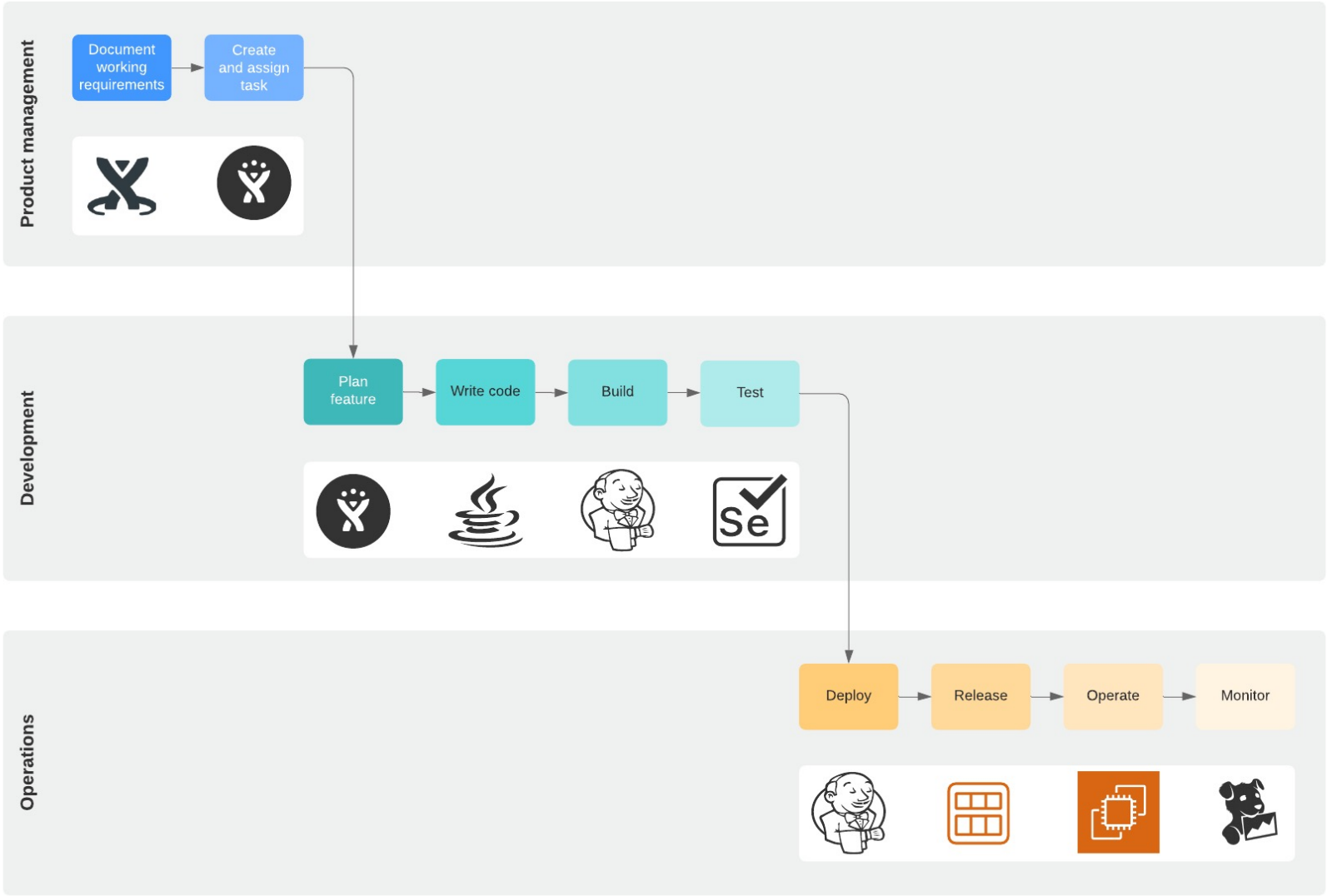
Who Am I?

- Michele Chubirka, SuccessFactors Chief Security Architect
- Creator of the Healthy Paranoia Security Podcast
- Former analyst, researcher, blogger, B2B writer, Unix and Network Engineer

The Software Supply Chain

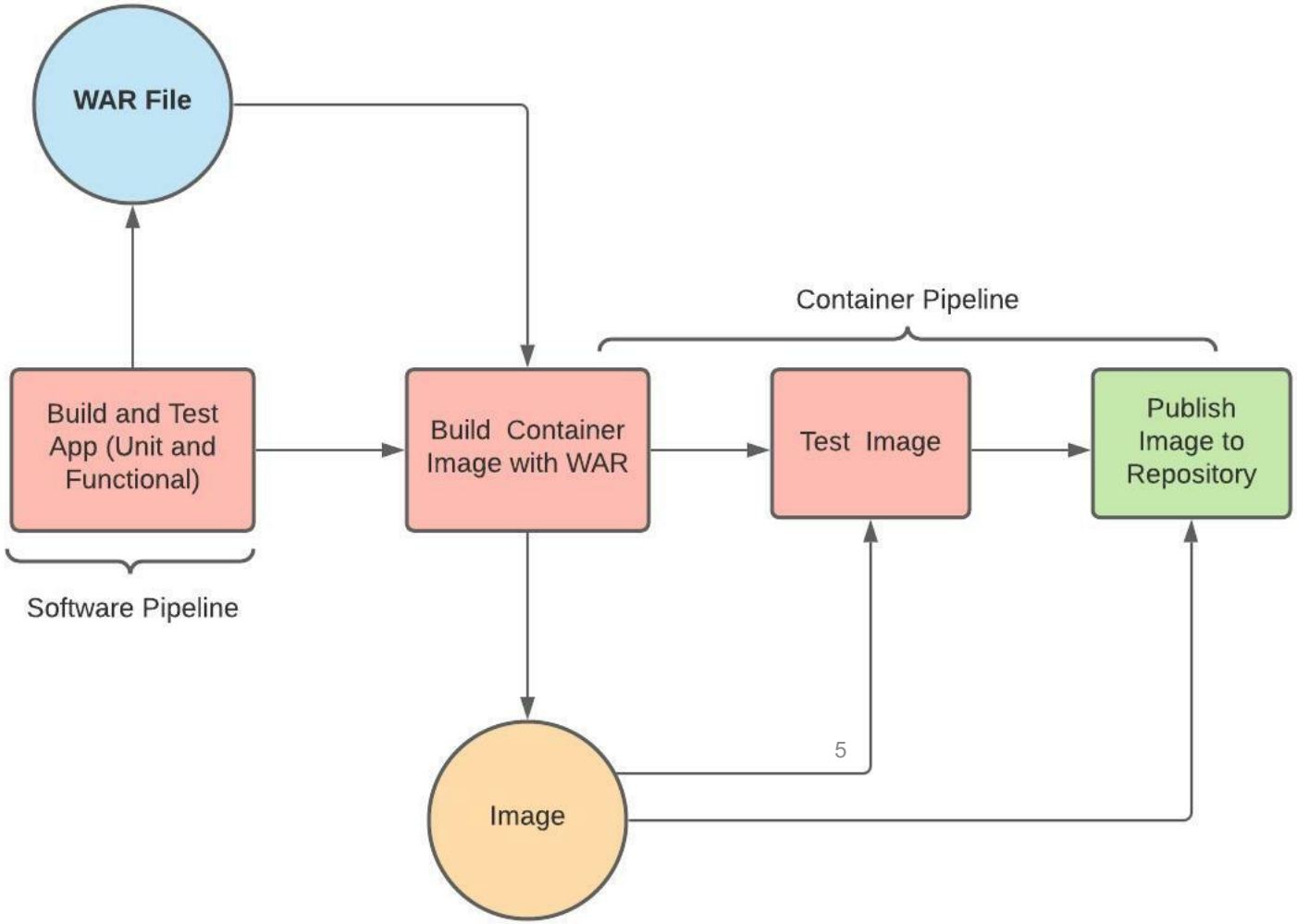
Wait, Containers are Software?

What is a Software Supply Chain?

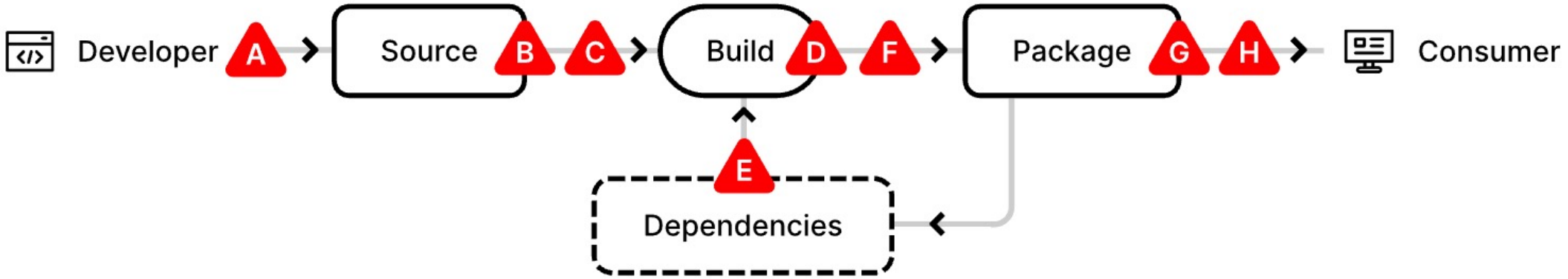


A set of processes that build a product.

Container Supply Chain



Supply Chain Threats



A Bypassed code review

B Compromised source control system

C Modified code after source control

D Compromised build platform

E Using a bad dependency

F Bypassed CI/CD

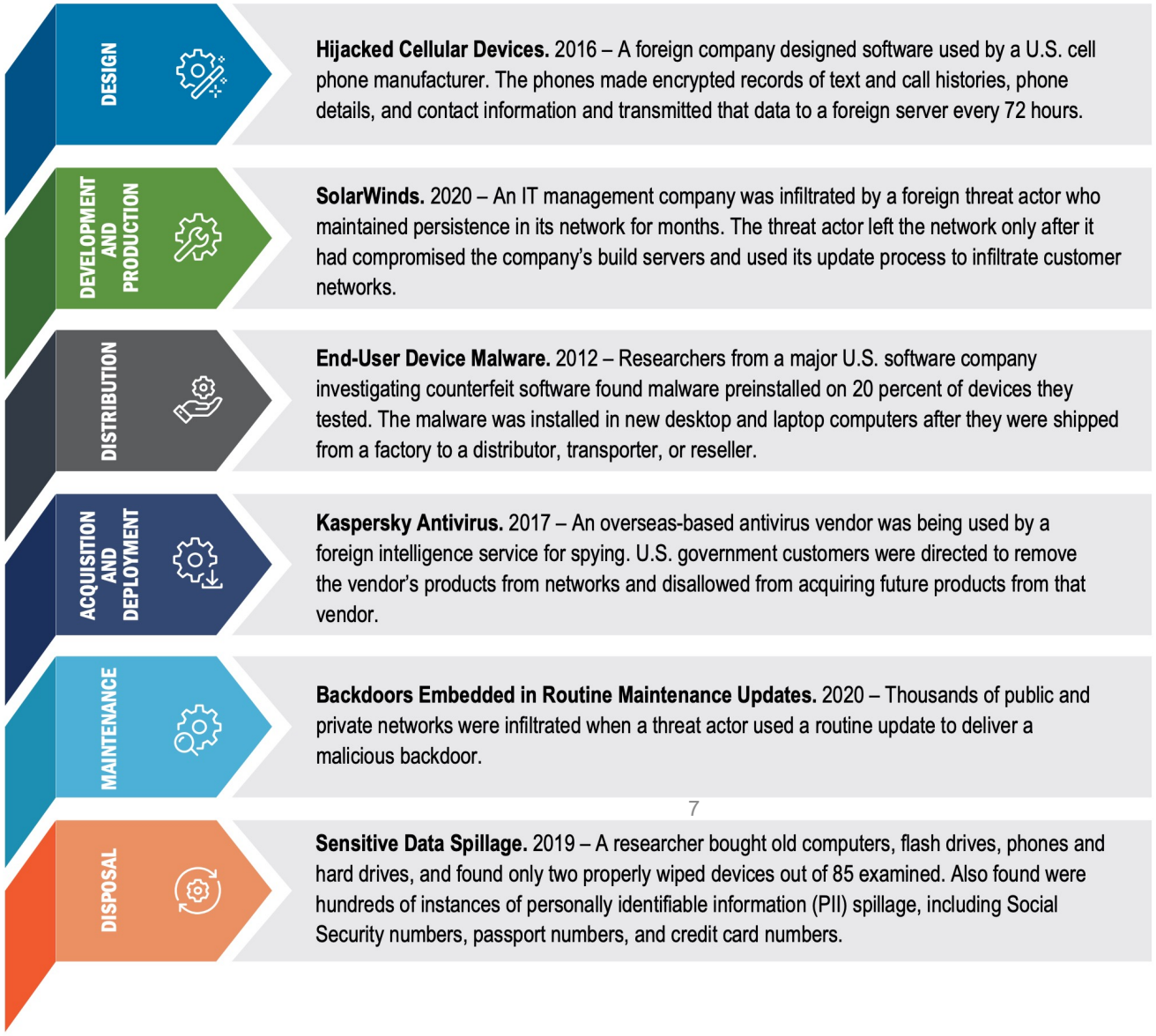
G Compromised package repo

H Using a bad package

From the SLSA Security Framework Project https://slsa.dev/

Software Supply Chain Risks

Table 1: ICT Supply Chain Lifecycle and Examples of Threats



From NIST "Defending Against Software Supply Chain Attacks." 2021

Supply Chain Security

Adding assurance to the software development process.

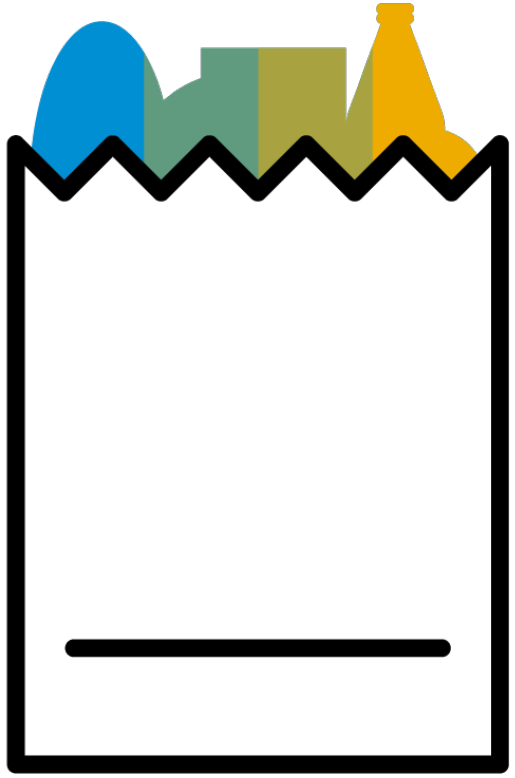
- Creating confidence and trust in the source material and practices used.
- Holistic view for protecting each stage in the software development lifecycle.
- Approaching the SDLC as a set of business processes.
- Validating the final product meets a reasonable set of security criteria to ensure it isn’t vulnerable.
- Although not new, recent high-impact attacks (i.e. SolarWinds) have heightened the attention on software supply chains by governments and large private-sector organizations.

What Is a Container?

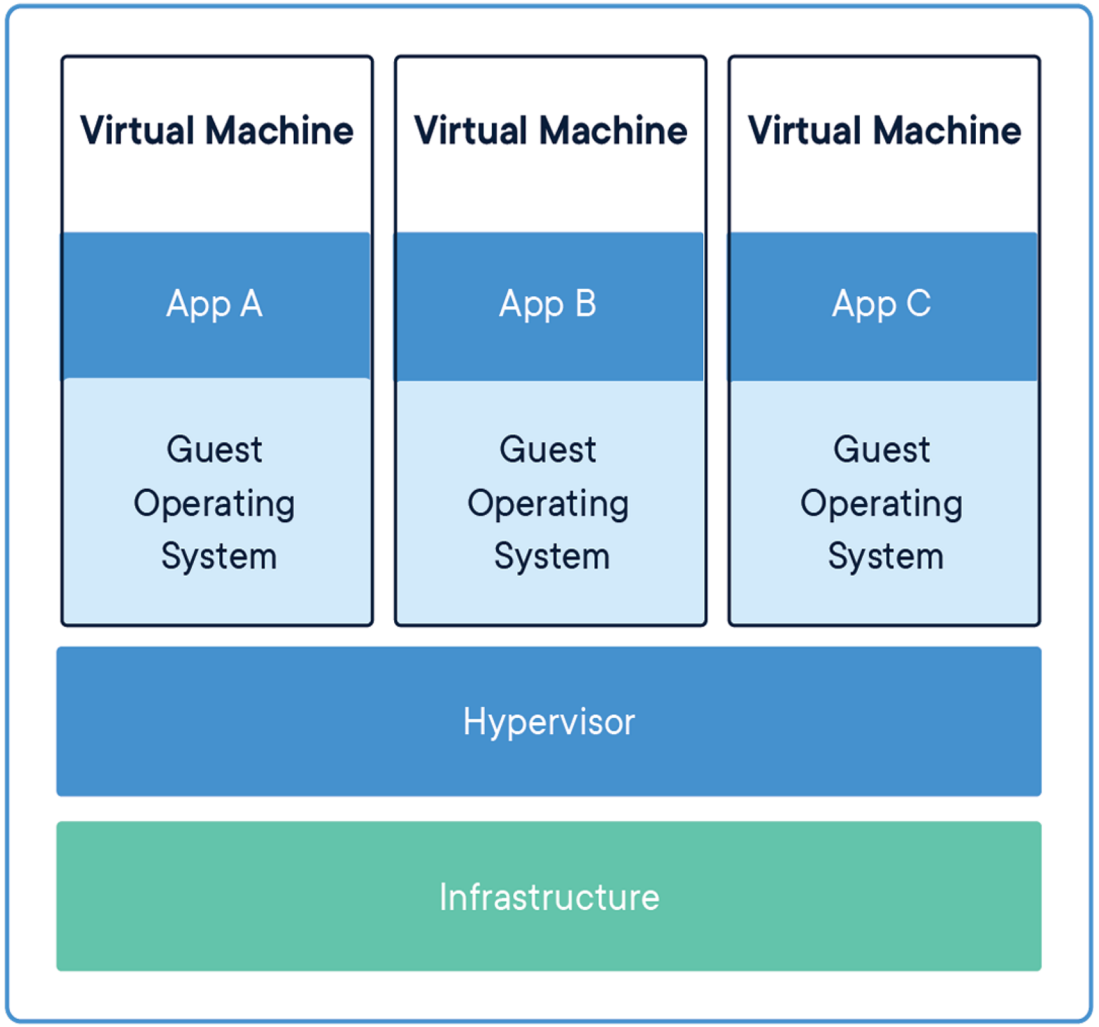
A Bunch of Software Ingredients

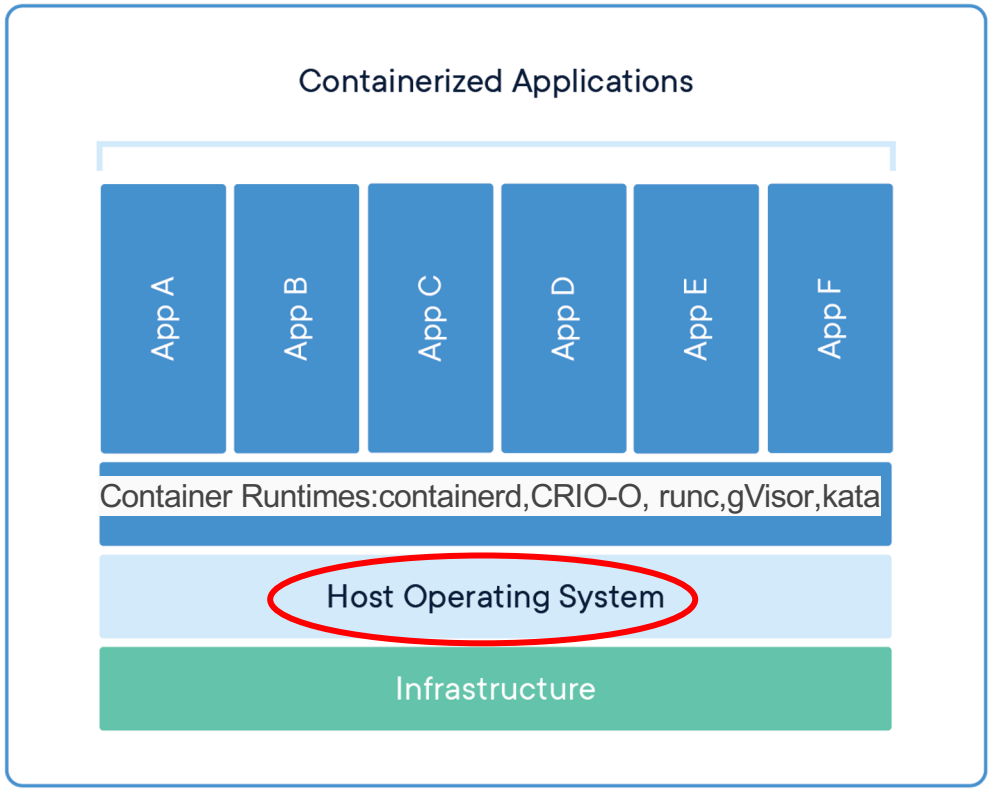
*A container is a **standard unit of software that packages up code** and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker **container image is a lightweight, standalone, executable package of software** that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.*

https://www.docker.com/resources/what-container



Past and Future: Virtual Machines vs. Containerization



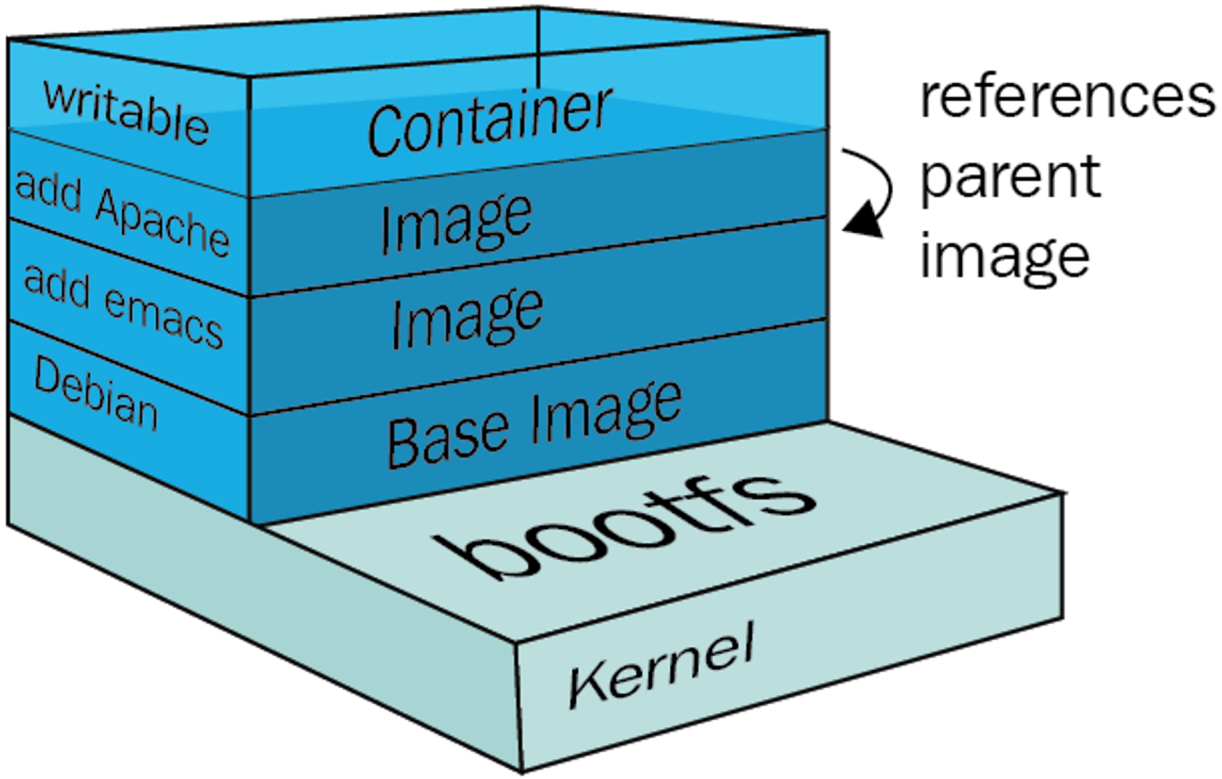


A container runtime is responsible for: running a container and other associated tasks such as downloading or unpacking the image.

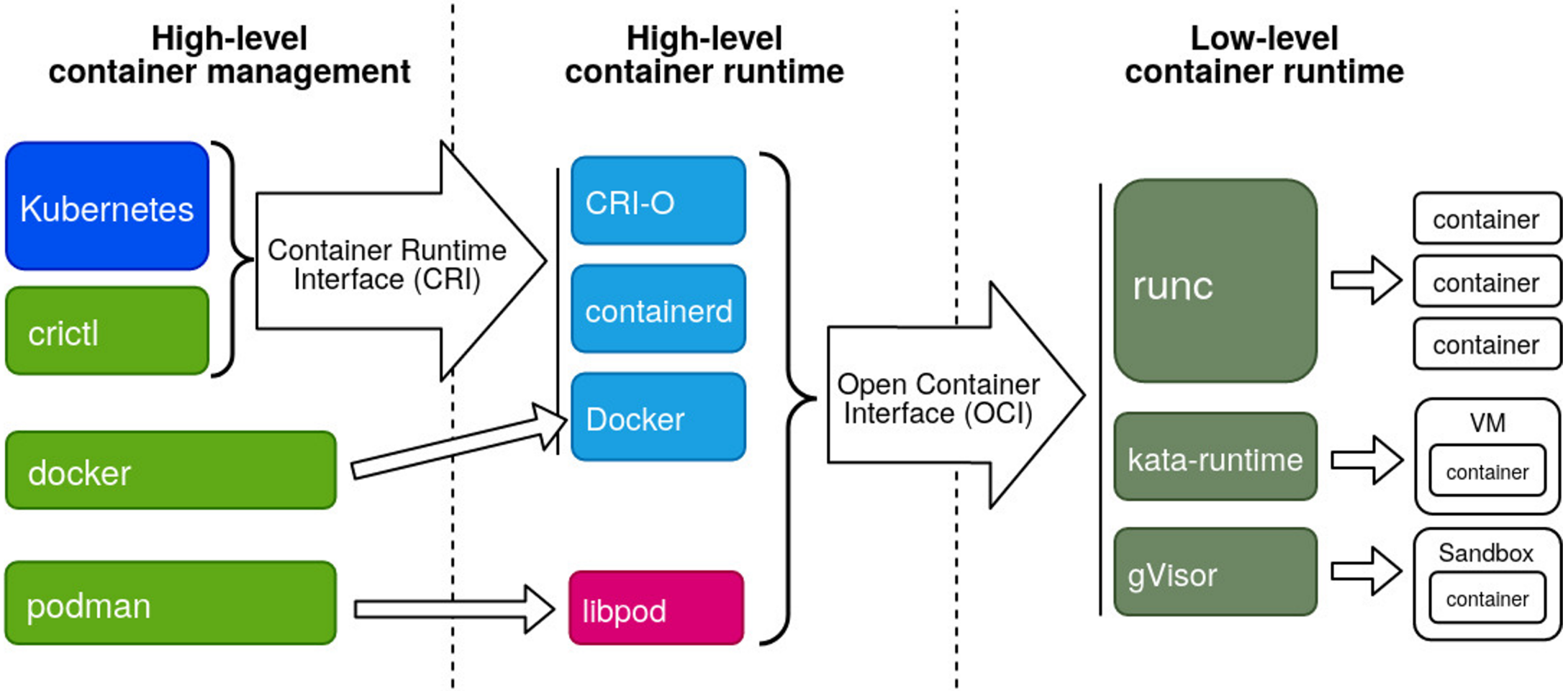
https://www.docker.com/resources/what-container

What's Cooking? User Space vs Kernel Space

- Virtual Machines run their own isolated kernels. There is no shared memory or execution space between these "guests" which run applications.
- Without the addition of a virtual machine technology (e.g. Kata, Firecracker or gVisor), ***containers share the same host kernel.****
- Linux Namespaces, Capabilities, Seccomp, SELinux, Apparmor and Cgroups can be used to enhance segmentation between running container instances on the host.



Images are filesystem bundles, built of layers corresponding to instructions in scripts or a dockerfile.



https://merlijn.sebrechts.be/blog/2020-01-docker-podman-kata-cri-o/

Huh?

Container Misconceptions

- Container = full virtual machine or an AMI
- You can put your entire monolithic application stack in a container
- The running instance != an image. Most use the term “container” interchangeably, but it’s important to know the difference.

*A container is just a software package, including its dependencies (libraries and application runtime) that will **run in a dedicated area of user space**. This is why the security of the supply chain is critical.*

Where Do You Get Your Container?

Recipes for Container Images

- Pull a base image from a public or private registry, add your code and configuration (i.e. layers).
- Use a multi-stage build, selectively copying what you need in the image.
- Without using a base or parent image, build an entire OCI-compliant image with a tool like Buildah.
- Create a single-layer image “from scratch” using a statically compiled binary.
- Make a distroless image that only includes the application and runtime dependencies.

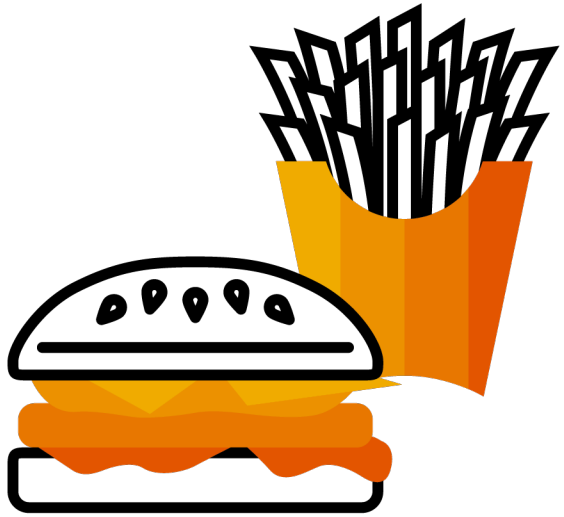
Use the Best Ingredients

Trust	Use a trusted source for base images.
Validate	Validate the image with a container security tool and/or Software Composition Analysis (SCA) tool to identify vulnerabilities.
Economize	Only add elements necessary for your microservice.
Reduce	Follow least privilege: don’t run root processes, don’t run the container as privileged.
Limit	Limit syscalls.
Parameterize	Don’t hardcode configs or embed credentials/secrets in the image, parameterize instead.
Immutable	Don’t add a shell, you shouldn’t login to a running instance. It should be immutable and ephemeral.
Minimize	Use techniques like Distroless or “from scratch” to eliminate unnecessary elements and layers.
Automate	Use the automation of a DevOps pipeline to ensure security validation is automatically performed with every build or change to the image.

Fry, Bake, Buy

- Fry - Custom built or extended 3rd party images, deployed but *changed during bootstrapping activity* (eg. apt-get, curl), cannot attest
- Bake - Custom built or extended 3rd party images, fully created with all dependencies as part of supply chain, attested and immutable
- Buy - 3rd party or vendor images, deployed without changes, attested and immutable

Bake or Buy, but never fry



Container Security Validation Process

Container must meet security criteria to receive passing score.

1a. 3rd-party images downloaded or mirrored from public repository to internal repository. Can be used as a base image, then "forked" with customer layer

1b. Custom images built internally, uploaded to internal repository

2. Container security validation of images. Checks for vulnerabilities and hardening of image. Validation occurs during upload of image to internal registry, daily scheduled scan of internal registry and during CI/CD.

3. Container is deployed to a runtime environment from a repository through a pipeline. A runtime control component ensures that only security-validated, signed images are deployed and can run.

Validation criteria could include:

- ***CVSS score below certain threshold (e.g. >=7)***
- ***No root-owned processes in container image***
- ***Image doesn't run as privileged***
- ***No shell installed***

What Are Container Orchestrators?

- An orchestrator handles the deployment and runtime lifecycle of running instances.
- Scales instances up and down to meet demand.
- Offers redundancy and availability options for running instances and hosts.
- Load balancing, service discovery and health monitoring of container hosts and instances.
- Ability to create virtual clusters with granular management of resources
- Provides multi-tenant segregation through network segmentation and resource restrictions
- Examples include Kubernetes, OpenShift, Mesos, Nomad
- Cloud provider managed orchestrators: Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS)
- Capability to use policies with an entrypoint that ensures only instances that pass are allowed to run (Admission Control)

Orchestrators provide the last "gate" for a container prior to runtime. With Kubernetes, various policy mechanisms (i.e. admission control) can validate and enforce attestation of the image.

CNCF – Principles of Supply Chain Security

- Establish and verify “trust” at every step in the process through a combination of code-signing, metadata, and cryptographic validation.
- Everything that can be automated should be automated and documented. This helps to prevent accidental errors and makes it easier to spot when things have gone wrong.
- Provide clarity:
  - Every step in the software build and supply chain process should be clearly defined with a precise, limited scope.
  - Every actor within the supply chain (whether human or machine) must have a clearly defined role. This allows us to limit the permissions of these actors to exactly those needed.
- Every entity in a system must engage in “mutual authentication.” This means that no human, software process, or machine should be trusted to be who they say they are, they must demonstrate it.

Tiers of Container Technology Architecture

1. Developer systems (generate images and send them to testing and accreditation)
2. Testing and accreditation systems (validate and verify the contents of images, sign images, and send images to the registry)
3. Registries (store images and distribute images to the orchestrator upon request)
4. Orchestrators (convert images into containers and deploy containers to hosts)
5. Hosts (run and stop containers as directed by the orchestrator)

From NIST SP 800-190 “Application Container Security Guide”

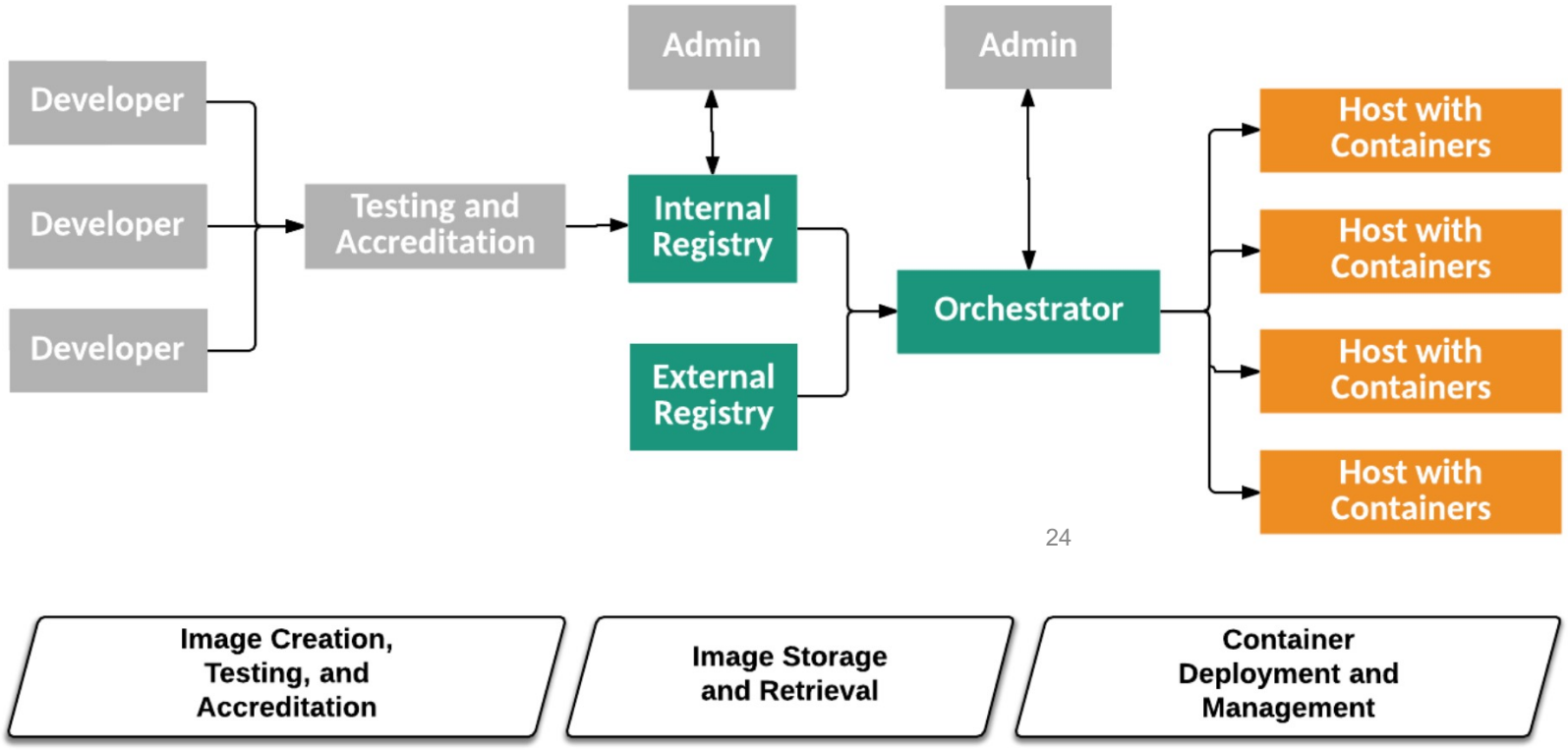


Figure 3: Container Technology Architecture Tiers, Components, and Lifecycle Phases

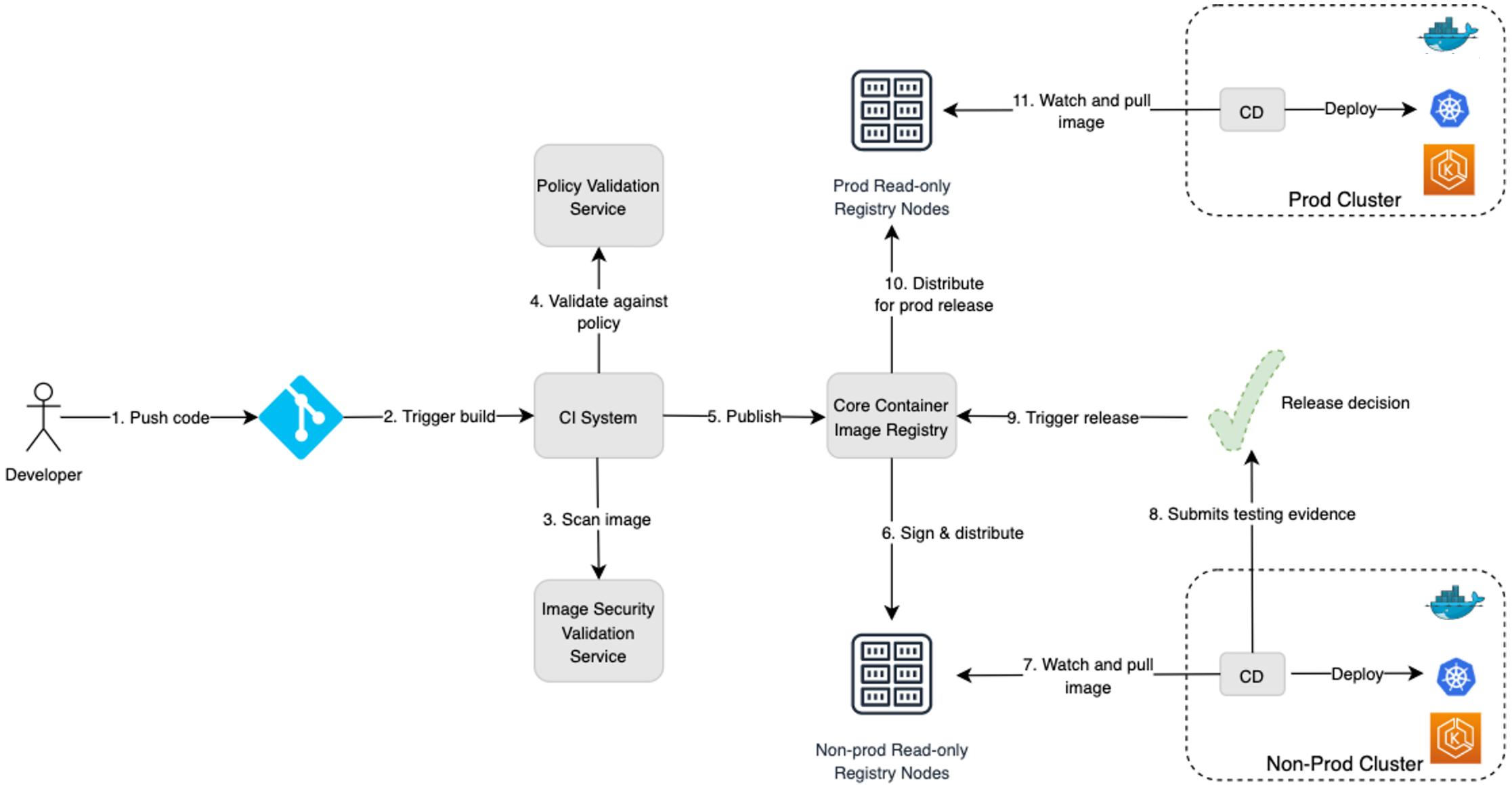
BOMs, SBOMs, and DBOM, Oh My…

- An SBOM (software bill of materials) is a formal list of your software ingredients, including open source components and licenses.
- In response to recent software supply chain attacks, White House Executive Order 14028 defined software supply chain security criteria for federal information systems.
- This includes providing an SBOM (software bill of materials) to the purchaser for each product or publishing it on a web site.
- You can generate an SBOM for your software before packaging into a container or after, using open source tools such as Syft or Tern.

DevSecOps Decisioning Principles

- Security tools should integrate as ***decision points*** in a pipeline.
- DevSecOps tool(s) should have a policy engine that can respond with a pass/fail decision for the pipeline.
  - This optimizes response time.
  - Supports separation of duties (SoD) by externalizing security decisions outside the pipeline.
  - “Fast and frugal” decisioning is preferred over customized scoring to better support velocity and consistency.
  - Save contextual risk scoring for release decisioning.
  - Does not exclude the need for detailed information provided as pipeline output.
- Full inspection of the supply chain element to be decisioned, aka “slow path,” should be used when an element is unknown to the pipeline decisioner.
- Minimal or incremental inspection of the supply chain element to be decisioned, aka “fast path,” should be used when an element is recognized (e.g. hash) by the pipeline decisioner.
- Decision points should have a “fast path” available, where possible, to minimize any latency introduced from security decisioning.
- Security policy engines should not be managed by the pipeline team, but externally by a security SME, to comply with SoD and reduce opportunities for subversion of security policy decisions during automation.
- The goal is to ***optimize*** coverage.

What's in Your Container Kitchen? A Real-World Use-Case



Reminders

- Small is beautiful: make your images compact, leaving out unnecessary libraries or binaries you don’t need for your individual microservice.
- Who do you trust? What are the “trusted registries” that supply your base images?
- Validate, Validate, Validate your images. They should be free of vulnerable libraries and components.
- Container images shouldn’t need to run as privileged or have processes that run as root.
- Don’t put credentials, keys or tokens into your image.
- Reduce the listening ports needed in your container image, it just increases the attack surface and resources that need to be secured.
- Make sure to use a standard runtime. Your current security tools might not support every possible flavor.
- A container isn’t a host, it’s an ephemeral drop of rain. You shouldn’t be logging in and trying to change it, because it’s going to evaporate.
- Only attested, immutable container images should run in your environment.
- Automate, automate, automate your container builds and deploys.
- Constantly re-evaluate your supply chain: verify source code, verify dependencies, sign and protect pipelines and artifacts.

References

Container Basics

https://washraf.gitbooks.io/the-docker ecosystem/content/Chapter%201/docker_vs_virtual_machines.html
https://www.docker.com/resources/what-container
https://opencontainers.org/
https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

Unikernels, Microkernels, MicroVMs, container sandboxes

https://nabla-containers.github.io/
https://gvisor.dev/docs/
https://katacontainers.io/
https://ubuntu.com/blog/what-is-kata-containers
https://jornfranke.wordpress.com/2019/09/14/unikernels-software-containers-and-serverless-architecture-road-to-modularity/
https://firecracker-microvm.github.io/

Building Containers

https://hub.docker.com/_/scratch#:~:text=0%20(specifically%2C%20docker%2Fdocker,1%2Dlayer%20image%20instead).&text=You%20can%20use%20Docker's%20reserved,starting%20point%20for%20building%20containers.
https://buildah.io/
https://github.com/opencontainers/image-spec
https://medium.com/better-programming/how-to-harden-your-containers-with-distroless-docker-images-c2abd7c71fdb
https://github.com/GoogleContainerTools/distroless

References

User space vs kernel space with containers:
https://www.redhat.com/en/blog/architecting-containers-part-1-why-understanding-user-space-vs-kernel-space-matters#:~:text=Kernel%20Space%20Matters,-July%2029%2C%202015&text=While%20containers%20are%20sometimes%20treated,resources%20they%20need%20access%20to.&text=These%20files%20and%20programs%20make%20up%20what%20is%20known%20as%20user%20space.
Linux Namespaces: https://opensource.com/article/19/10/namespaces-and-containers-linux
SELinux is a MAC mechanism used to provide fine-grained access control to a Linux host:
https://www.projectatomic.io/docs/docker-and-selinux/
https://www.youtube.com/watch?v=_WOKRaM-HI4
Composition of a container image:
https://dzone.com/articles/docker-layers-explained
https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/
Union Filesystem:
https://washraf.gitbooks.io/the-docker-ecosystem/content/Chapter%201/Section%203/union_file_system.html
https://blog.knoldus.com/unionfs-a-file-system-of-a-container/
https://containerd.io/scope/
https://docs.docker.com/storage/storagedriver/
https://github.com/opencontainers/runtime-spec/blob/master/bundle.md
Hardening and minimizing a container image:
https://www.secjuice.com/how-to-harden-docker-containers/
https://www.thoughtworks.com/radar/techniques/distroless-docker-images
https://github.com/docker-slim/docker-slim
https://medium.com/faun/how-to-build-a-docker-container-from-scratch-docker-basics-a-must-know-395cba82897b
Recommended secrets patterns with containers:
https://aws.amazon.com/blogs/database/design-patterns-to-access-cross-account-secrets-stored-in-aws-secrets-manager/
Injecting Vault Secrets into K8s Pods via a Sidecar
https://www.hashicorp.com/blog/injecting-vault-secrets-into-kubernetes-pods-via-a-sidecar/

References

The difference between a high-level and low-level container runtime: low-level runtimes focus on mechanics of actually running a container, while high-level runtimes add features such as management of container images, unpacking the image, and hand-off to the low-level runtime.
https://merlijn.sebrechts.be/blog/2020-01-docker-podman-kata-cri-o/
https://www.ianlewis.org/en/container-runtimes-part-1-introduction-container-r

While Docker was a key player in helping to popularize and standardize the image and runtime formats, it has since donated documentation and code to the OCI to further the mission of containerization as a vendor agnostic technology.
https://opencontainers.org/faq/
Think of Docker to containers as Kleenex is to tissue.
https://www.ianlewis.org/en/container-runtimes-part-1-introduction-container-r

What is Kubernetes?
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
What is OpenShift?
https://www.openshift.com/learn/what-is-openshift
What is Mesos?
http://mesos.apache.org/documentation/latest/

Good case study for container takeover
https://unit42.paloaltonetworks.com/azure-container-instances/

References

White House Executive Order 14028 https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Dept of Commerce and NTIA The Minimum Elements For a Software Bill of Materials (SBOM) https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

Generate a Software Bill of Materials for a Container Image with Syft https://thenewstack.io/generate-a-software-bill-of-materials-for-a-container-image-with-syft/

SLSA framework https://slsa.dev/

BUILDING RESILIENT SUPPLY CHAINS, REVITALIZING AMERICAN MANUFACTURING, AND FOSTERING BROAD-BASED GROWTH https://www.whitehouse.gov/wp-content/uploads/2021/06/100-day-supply-chain-review-report.pdf

CNCF – Evaluating Your Supply Chain Security https://github.com/cncf/tag-security/blob/main/supply-chain-security/supply-chain-security-paper/secure-supply-chain-assessment.md

CNCF – Software Supply Chain Best Practices https://github.com/cncf/tag-security/blob/main/supply-chain-security/supply-chain-security-paper/CNCF_SSCP_v1.pdf

NIST Cyber Supply Chain Risk Management C-SCRM https://csrc.nist.gov/projects/cyber-supply-chain-risk-management

NIST SP 800-190 Application Container Security Guide https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

SAP SuccessFactors

Thank you.

Contact information:

Michele Chubirka
Chief Security Architect
Michele.Chubirka@sap.com

DEV TOBER FEST

THE BEST RUN SAP