Securing the Kubernetes software supply chain

Microsoft's Ratify proposal adds a verification workflow to Kubernetes container deployment.

By Simon Bisson
Columnist, InfoWorld

DEC 15, 2021 3:00 AM PST

Modern software development practices make securing the software supply chain more important than ever. Our code has dependencies on open source libraries which have dependencies on other libraries and so on—a chain of code that we didn't develop, didn't compile, and have little or no idea where it came from.

Some of that code is almost ubiquitous. The Log4Shell exploit that caused havoc across the industry was from an exploit resulting from an old bug in a common Java logging component, log4j. We're building an industry that stands not on the shoulders of giants, but on the shoulders of a handful of application and component maintainers who keep our global infrastructure working in their spare time and out of the goodness of their hearts.

[Also on InfoWorld: 4 reasons to get Kubernetes-certified, and 4 reasons not to]

Distributed development adds risks

This is not to minimize the work maintainers do; they're an essential part of the modern software supply chain, delivering everything from small modules to entire container-based platforms. They're undervalued and underpaid for how important their code is. Sadly, there have been several instances where bad actors have offered to take over maintaining code, only to add in malware, expecting the code to be installed automatically as it had a history of being trustworthy.

We can expect to see more attacks like these as more of our code becomes a group effort. How do we protect ourselves and our applications? First and foremost, we need to understand that the software supply chain does in fact exist, that we're not building code in isolation, and we haven't been doing so for a long time now, if we ever did. Open source and third-party libraries are an essential part of how we make software, and they're only going to get more important.

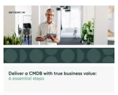
What steps can we take to secure the software supply chain? A lot of work has gone into providing the tools to manage the software bill of materials: scanning code for libraries, using static and dynamic analysis, adding digital signatures and hashes to code, and bringing it all into managed repositories. But one part of the picture is missing: How do we validate that work and validate the code we're using? After all, one of the age-old security adages remains "trust but verify."

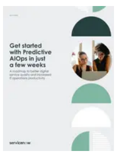
Securing the software supply chain

We need to trust the code we use, but we also need to verify that it's trustworthy. We also need to do it under time pressure, with cloud-native code shipping new builds as code in repositories changes. The automated nature of modern development is key here, with platforms like GitHub at the heart of our software life cycle, delivering continuous integration and continuous delivery (CI/CD).

Things get more complex when we're working with technologies like Kubernetes, which are designed to build on the mix-and-match philosophy of microservice architectures and containers. While our code may run in isolated containers, it runs in nested abstracted userlands, each dockerfile collecting a selection of dependencies, many of which aren't fully documented. How can we trust the bill of materials in the containers we use?

RECOMMENDED WHITEPAPERS


Deliver a CMDB with true business value: 6 essential steps


Get started with Predictive AIOps in just a few weeks

The many ways infrastructure and service visibility deliver benefits across IT

Introducing Ratify: an artifact verification workflow

Microsoft's cloud-native open source team has been working on a new specification that should help with this. Ratify is a verification framework that supports the various artifacts that come together in Kubernetes applications. It uses a set of trusted security metadata and a signed bill of materials to ensure that everything you deploy is what you're expecting to deploy.

Images and other components take advantage of the Notary V2 signing and verification tool and the ORAS (OCI Registry As Storage) Artifact specification. ORAS is part of the Open Container Initiative registry definition, extending it to storing anything, not just containers. It works well as a way of putting together a software bill of materials. Interestingly there's commonality between a Bindle-distributed application installer definition and an ORAS manifest, making it simple to go from an SBOM to a verified distributed application installer. Together the two deliver a supply chain graph that can be parsed and used as part of a verification scheme inside a Kubernetes cluster.

Ratify bundles these concepts together, adding a workflow engine to apply policies to your software bill of materials, verifying many different supply chains across your code and its dependencies. At its heart is a coordinator that manages the policy workflow across your container image. It's extensible, so it can work across public and private registries, using a familiar plug-in model that's similar to those used in Kubernetes.

Policy-powered ratification

The policy model used by Ratify is based on familiar tools, offering a way to quickly roll out basic policies using your own configurations as well as more complex policies built using the Open Policy Agent. It'll also work at different points in your application development life cycle, plugging into CI/CD systems to verify artifacts at build time and into Kubernetes to ensure that code hasn't been altered between build and run.

It’s important to have a verification mode that works across your stack, ensuring you’re avoiding supply chain attacks that can happen to code at build, in repositories and registries, and at runtime.

Having one tool handle verification across your software life cycle is important as it ensures you only need to write policies once. Different tools for different scenarios add the risk of transcription and translation errors in policies; having a single tool and a single policy format helps avoid that risk. You’re also able to have an external policy testing tool that can help you investigate “what ifs” before delivering your code.

Building your first Ratify policy

The Ratify team has some demo code in their GitHub repository that shows how to use Ratify with Gatekeeper in Kubernetes. Ratify installs using a Helm chart, bringing along some sample configuration templates. You can use these to test Ratify operations, for example, blocking all images that don’t have a signature. Gatekeeper will deny any container images that aren’t signed, blocking them from running.

Policy files are written in YAML, so you can edit them in Visual Studio Code or other tools, taking advantage of code formatting tools. They build up into a simple rules engine, stepping artifacts through a verification. Are they from an approved registry? Are you checking an artifact more than once for different signatures? Are artifacts in your own private registry more trusted than artifacts from public registries? Do all your verification engines agree when you’re running multiple checks? It turns out that the rules for a runtime verification are quite simple to define, though that’s unlikely to be the case for one running in a CI/CD system where you need to determine the state of many different artifacts and with signatures from many different roots of trust.

Ratify is currently an interesting initial proposal for a set of tools that can manage all the elements in a software bill of materials. While it won't prevent zero days impacting long-hidden bugs, it can quickly determine what code is affected, creating rules to prevent it being used and run.

With supply chain risks very much in the spotlight, it’s important for the industry to look carefully at proposals like this and work on them in public spaces. It’s good to see that Microsoft has already committed to sharing Ratify with the Cloud Native

Computing Foundation, where it should get the scrutiny from the wider Kubernetes development community that it needs.

Author of InfoWorld's Enterprise Microsoft blog, Simon Bisson has worked in academic and telecoms research, been the CTO of a startup, run the technical side of UK Online, and done consultancy and technology strategy.

Follow

How to choose a low-code development platform