Deploying on the Edge

featuring Patrick Earl, Doug Reynolds, and Jeffrey Hamed as interviewed by Jose Moralas

Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Jose Morales: Hi, and welcome to the SEI Podcast Series. My name is Jose Morales. I am a senior researcher here at the Software Engineering Institute. I am joined today by three of my colleagues, Patrick Earl, Doug Reynolds, and Jeffrey Hamed, and we're going to be talking about a recent blog post that they have on research dealing with Kubernetes and edge devices. But before we start, I'd like each of you to introduce yourselves and sort of tell us a little bit about your background. Let's start with Patrick.

Patrick Earl: I'm a DevOps engineer here at the SEI. I've been here for about almost three years coming up in May. I do a lot of different things, mostly focused on Kubernetes, working on deploying stuff inside of a Kubernetes cluster and so forth. And that's me. I'll hand it off to Doug.

Doug Reynolds: Hi, I'm Doug Reynolds. I'm a senior DevOps engineer over here at the SEI. A lot of times, we get stuck in a DevOps world. We get to do a little bit of everything, as Patrick mentioned. But my background is infrastructure, software development, and engineering, and as well DevOps,

and mainly infrastructure as code and automation. Pretty much all the things. Jeff, why don't you take it away?

Jeffrey Hamed: Thanks, Doug. Hi, everybody, I'm Jeff Hamed. I'm a senior DevOps engineer at the Software Engineering Institute. I've been at the SEI for 15 years now, started as actually a forensic video analyst on a research project, and then I did a lot of .NET development, a lot of web development, and now, I'm doing the DevOps thing. And I love the amount of collaboration that we have here at the SEI that allows me to work with many different groups of people that have different expertise. Thank you.

Jose: All right. Excellent. Thank you, all three of you. The blog post you guys wrote deals with using cloud-centric technologies on edge devices with some resource constraints. But before we get into the depth of that, can one of you, Doug, maybe we can start with you, tell us a little bit more about these edge environments and what kind of challenges they pose to interacting with them, especially when you're deploying infrastructure?

Doug: Yes, sure. I guess *edge* is almost now like a buzzword, like *cloud*. But what you're looking at is a device that's not working in a data center. You could have a Raspberry Pi sitting out in the middle of nowhere. Some edge deployments I've run into are a small server rack with two servers sitting on a T1 or frame relay line that's the only connectivity. Or you could have like an Amazon Outpost or, even some Azure devices that kind of mimic the cloud, but they have limited uplinks. A lot of times when you do have an edge deployment, you might not have your modern infrastructure tooling like you can't just jump in there with using VMware's GovC or maybe some cloud provisioning or other types of automation tools that would just make your life super easy to jump in there and configure your resources and software. The other thing that really kind of nails down the edge is you have limited resources. You might have two servers online. You know, you're talking maybe 16 cores, or you might have more of an edge type processor like a Raspberry Pi or a Jetson or something, a more specialized like that. You really can't just go out there and be like, *Cloud provider, please, give me the largest CPU that you have available in your rack with all the GPUs and everything*. Your scaling is really limited to what you have on the edge, aka what's sitting somewhere, hopefully not getting rained on. a

Jose: What kind of connectivity can you expect to have on these edge devices? And I'm not just talking internet. I'm assuming that there's no

internet connectivity. What about intranet to some other server or resource? Do you ever see anything like that, or is it just that one Raspberry Pi or that one server blade by itself not connected to anything?

Doug: Well, I've seen a lot of different things. When I say T1 line, that doesn't necessarily mean internet connectivity. That could be just point to point T1 from a downstream to the upstream. That's limited just maybe to another site or another network. I've seen like radio devices that listen for radio signal. Sometimes, people just are out there, and they have an LTE modem and they're VPNed over into a real network. And then even on other devices, you're just hooked up to a satellite downlink. So maybe you can receive data from a network, but you can't really send it. Or maybe even if you're looking at, a lot of people now are relying on something like Starlink from a low orbit satellite resource where you can get connectivity. And a lot of times, that does provide the internet, because a lot of people, the internet's kind of been synonymous with how you get back to your private network, but it is not always the case. There are numerous ways, maybe point to point wireless like a mesh network. It really depends on what the problem you're trying to solve out in the field is or wherever your edge location is.

Jose: It strikes me that these edge environments, they have to be self-reliant. And anything that you deploy onto them has to be highly optimized to be able to function effectively within these constraints.

Doug: You have to have something that's as self-healing as possible. Obviously, if it gets struck by a bolt of lightning, that's probably not going to be super self-healing. Sometimes in a location, you might have some sort of a failover, where you have two servers running or two devices running, and one will pick up the slack if the other one goes offline. But you also have to worry about other things like generator power or battery backup. Maybe your site is just powered by solar panels and a battery backup, you have to have something that's definitely robust that if it does go down, it is able to come up cleanly instead of just coming up in a state that's unreliable. There are a lot of variables if you have a true edge device. I've seen also some edge devices where they're stuck in somewhere, somebody's kind of closet. It is filled with dust. There's no good HVAC. It might be 150 degrees in the summertime or like today and where we're talking 15 degrees outside. If you want to have something reliable, you have to really put together that engineering piece for that edge deployment, because there can be a very lot of variables that could affect your deployment.

Jose: Yes, I was thinking about that. The power issue is important. I'm assuming these systems have to be able to shut down gracefully in case of an unexpected situation. This blog post, as we said, deals with a prototype of an edge device using cloud-native technologies to deploy onto it. It is a very interesting concept. And I'd like to know how you came up with it, what's the origin of this idea? Patrick, you're one of the lead authors on it. Can you give us that insight?

Patrick: Oh, yes, sure. The main reason we wanted to move over to looking at Kubernetes in an edge environment is, as we've probably seen in the cloud or on-prem situations, there's movement to deploying software with containers, and Kubernetes has kind of just become the de facto standard for managing the deployments of said containers. A partner we work with was looking for a way to deploy a hypervisor onto these edge devices, so some small servers to replace a well-known hypervisor and bring cloud technologies to it so that they could automate the deployment as much as possible. We started off by looking into this and we wanted to use the tools we use in the cloud such as, Terraform, Kubernetes, Ansible, maybe throw some GitOps in there once we have the Kubernetes stack up. To start off, we were looking for a replacement hypervisor. We landed on Harvester, which is an interesting new HCI or hyper-converged infrastructure hypervisor project. Basically, it allows you to install a base OS onto your bare metal and it spins up a RKE2 cluster that then uses KubeVirt to allow you to host virtual machines. And then we would build on top of that. The main thing was to replace this hypervisor with another one that supports various features, which we'll get into more later on.

Jose: All right, and Patrick, I want to stay with you with my follow-up question here. You mentioned a lot of the tools that were used in the prototype. And could you now tell us about the overview of how the prototype actually functions and how the deployment scenario actually occurs?

Patrick: Yes, sure. Harvester allows you to deploy it using this age-old technology of PXE boot that's been around forever. Basically, you configure your system when it starts up for the first time, or if it detects there's no operating system to look to the network for an ISO install and so forth. Harvester uses an extension of it called iPXE, which is improved PXE boot environment. And with that, we can also configure it, so you don't even have to be there for the install. You can give it a configuration file. It will follow that

and then just install everything for you automatically. So that was a big target to us. Say you're out in the field and for some reason, maybe your hard drives fail, all of them, and you need to reprovision this machine. You could have a smaller machine that has the necessary resources like the ISO config files connected to a local network that the server is on and redeploy it in a simple fashion without the operator needing to configure it that much. Additionally, Harvester stood out to us because it comes with a Terraform provider that is also managed by the Harvester team. That was our main thing. And then Kubernetes, we were just installing VMs on top. There's a weird thing with Harvester where they don't want you deploying directly to the RKE2 cluster it spins up. You want to add another layer on top. From the minimal testing we did, we didn't see any big performance hits from that, but that's definitely something we might want to look into more in the future.

Jose: All right. Doug or Jeff, is there anything you'd like to add? Doug, maybe you want to go first?

Doug: Sure. Patrick brought up the Terraform and the provider that we use to configure Harvester. The one thing now with Terraform and now, some, most of the industry is moving over to OpenTofu, which is the totally open source work of Terraform. It gives you the ability to write declarative code to provision something. That's the nicety about these infrastructure as code tools. A lot of times when Terraform came out originally, you're like, *Oh, well, I'm going to go to the cloud and I'm going to spin up 10 VMs and provision 15 storage buckets, and I'm going to do that all with this automated platform*. You're not just looking at a cloud resource, you're looking at literally hundreds of different providers you can run in Terraform. And the other thing that's kind of neat about it is say you internally you have a piece of software you write for your organization. Terraform, OpenTofu, you can just go out and write a provider for your platform and then integrate it directly into Terraform and then use Terraform to provision your product or your enterprise solution or what kind of custom software development you have. And that really gives you a lot of flexibility, because a lot of folks now, they're doing API-first code. You start out and you do some simple provisioning, and you just have maybe a bunch of Postman scripts or maybe a couple cURL commands and just provision up some resources, or users, or whatever you're using to do with your provider. Well, that's great, and it is automated, and you don't have to hand jam a thousand things into a web console. But it is not really reproducible for other engineers that easily or it couldn't be done in automated fashions using some sort of CI/CD products. Terraform kind of

brings that all together. Now, you have something that you can do by CI/CD. And since it is almost like an industry standard now that if you're automating platforms, you're using Terraform or OpenTofu. For the base infrastructure, we wanted to leverage that same knowledge base around that to provide the automation that is needed for spinning up these edge devices. As Patrick mentioned, if the hard drives go down or something gets wiped or you send something out, you want to have a pretty seamless model deploy something because obviously, you might not have a DevOps engineer driving out of the field, or all your system admins going out in the field to provision something. With this, Git allows you to free up and use people who that's their job, to go around and fix things or do mechanical things or place things, to go out and run that with limited things they have to do other than just running a couple commands. Jeff, why don't you go into a little bit on what we did. Right now, we're talking about Terraform and Kubernetes, but we're also on the edge. We don't have any connectivity to pull the software down from. Why don't you give us a little bit of info on your part?

Jeffrey: Thanks, Doug. Yes. Great segue. I appreciate that. One of the challenges we had in our research here was how do we deploy software, the latest patches, the latest container images, the latest versions to an air-gapped environment, an environment that isn't going to contain, isn't guaranteed a lot of bandwidth. There were a few tools out there that we discovered. The first one is called Zarf. Zarf is a great air gap delivery tool made by a company called Defense Unicorns. And what Zarf does is that it is a binary that, essentially, you use to deploy Zarf packages to a Kubernetes cluster that is an air gap cluster or has low or not guaranteed bandwidth. Zarf is a binary, and we have it deployed in a GitLab CI/CD runner. And what we're doing is we're running GitLab pipelines that have what's called a Zarf YAML file in it. And when you list a bunch of container images in that YAML file, when you run the build, it pulls down the latest images. We can do things like scan the images from there for security vulnerabilities, and it creates a Zarf package. Now, at that point, you want to push that to some sort of shared storage solution. These packages can get relatively large, things like Big Bang, Platform One, the DevSecOps platform. If you're deploying Big Bang via Zarf, you can end up with a pretty big package. That's something to be mindful of. What we do is we run the pipeline, and it pulls down any container images it needs. Additionally, if we are deploying an actual application, it'll have a home chart or a Kubernetes manifest. And we take that, we create a Zarf package, we push it to shared storage. And from there, you have to take it over the air gap. Now, a lot of different things to consider.

A lot of groups we work with, their only option is Sneakernet, literally taking a disk over into a closed environment, and you're copying that disk over. There are other things that you hear bandied about these days, like cross-domain diodes in the cloud and that sort of thing. How you get it over is up to you. But what we're doing is we're saving a lot of time on what we call the low side, the fully connected, internet-connected environment, because we're using tooling and automation to create one package that can be deployed in a closed network. I took you up to the air gap. Now, we're going to go over the air gap. And once we're in the closed, what we call the high side environment, Zarf is already initialized in our Kubernetes cluster on the edge. And when we create a Zarf package and we bring it over, we can say, *Zarf, deploy this package for me*. And then it'll provision all the resources, create all the pods, services, persistent volumes, that sort of thing. Now, how that works with Zarf is, they use what's called a mutating webhook. And what that does is, Zarf has its own container registry it comes with, right? In the air-gapped environment, there's a container registry running that Zarf stands up for you. Now, when you deploy something with a Zarf package, when you say, *Zarf package deploy*, it is going to pull from that registry, so the cluster will pull what it needs, the container images it needs from that registry. You don't have to hard code that registry path anywhere. Zarf does it with what they call mutating webhook, which is pretty cool. I think that's basically covers the CI/CD part. Jose?

Jose: Well, that was excellent. There's a lot of good information there. I have a follow-up question for you, Jeff. These Zarf packages, did you ever have to consider what would be the appropriate size of any individual Zarf package? Did that matter?

Jeffrey: There's limitations, potentially anything over 20 gigs, maybe. It depends. It handles large deployments really well. But then some things you want to look out for is Zarf uses a cache. When you're redeploying an application, say, you change one image, that cache is going to get filled up in your CI/CD runner quickly. So that's something you'd consider. You want the Zarf lifecycle to fully complete so you have a nicely maintained list of Zarf deployments in your cluster, meaning what could happen is it'll time out in 20 minutes, right? If you have the time out of a Zarf package to deploy the time out in 20 minutes, Zarf thinks, *Hey, this failed*. *I'm going to say this Zarf package isn't whole, right?* However, Kubernetes is resilient, right? It'll continue to try to spin up all the resources it needs and maybe eventually deploys everything, like a Big Bang deployment. It just takes a little bit longer than 20

minutes. Now, your Zarf package or your Zarf history is not accurate, and that will affect subsequent deployments. You want to look out for that.

Doug: What you're saying, Jeff, is if you have a lot of pieces of software to deploy it might pay to break those packages up so Zarf doesn't sit there and spin its wheels for an hour.

Jeffrey: Exactly. Yes.

Doug: That would make sense because a lot of those things do have timeouts, and at least with Helm, you can adjust some of the timeouts with that. There might be some additional tuning to get all that stuff in place.

Jeffrey: Right. It is not a one size fits all approach. You have to do whatever works for you. But yes, absolutely. Small chunks are better but you want to complete deployment. And sometimes, those just tend to have a lot of resources, something like Longhorn, which is block storage, Kubernetes is block storage, right? There are a lot of dependencies. That's going to be a large package.

Jose: Now, going back to that, you did mention Sneakernet. We may have to Sneakernet some of these things across. Is Zarf the appropriate tool where your packages must be limited to a certain size to be put onto a DVD or a CD or some other media that you can actually Sneakernet over and bring into the high side?

Jeffrey: Yes, absolutely. That's a good tool. What they're doing is a lot of work behind the scenes to package everything and make it into a compressed file. It is absolutely a great choice for that.

Jose: OK. In this prototype that you guys built out, was the concept of self-healing ever considered, since this machine is what I would call self-reliant because it is resource constrained? It is not maybe connected to anything, it has to rely on itself for everything, for all of its functionality under all conditions. Was self-healing ever considered here or not part of the prototype? Maybe it is a future work, let's say.

Patrick: I would say the biggest consideration there is what kind of edge environment we are working with. Some cases it is only going to be one server and having some sort of self-healing there would be basically

impossible. That server goes down, the whole thing is down. Harvester in its deployment configuration does support a high availability. If you have a minimum of three different nodes, you could have some sort of self-healing there where it will attempt to do a live migration of the VMs. Your storage is distributed to at least two nodes. You can configure that to be more, so even if you do lose a hypervisor, at least, there's two copies of the storage for the VM on two different nodes. But we never got that far in our prototype to actually test those things out completely.

Jose: But they are there. You know that they're there. You could use them.

Patrick: Yes, they are features of Harvester, so definitely something that could be used.

Doug: The other piece on that is everybody, sometimes, they pile on Kubernetes because it is such a big, complicated thing. I think that's why a lot of people are starting to run hypervisors with Kubernetes because just the platform itself has some of the self-healing built into it. Something fails on your cluster; it'll get restarted automatically. Same as Patrick said, same goes for Harvester, has that HA kind of built into it. That's why Kubernetes is kind of the orchestration platform that does everything. I'm not saying it is the app that everybody needs or what everybody should use, but that's why they kind of build all those features into Kubernetes, is to kind of have that capability and not have to really work hard making that a custom thing, or having to buy expensive software packages to perform the same thing. You just kind of get it for free with that platform.

Jose: Is it fair to say in the prototype that you put together that the machines that you were deploying to were just bare metal? There was nothing on them when you first started.

Patrick: Yes, that's correct. The idea here was to make the provisioning of these machines as painless as possible. Ideally, once they're in the field, this wouldn't be a concern, but this was a situation where you could possibly have a hundred machines you need to provision, so we wanted to automate that process as much as possible. And again, Harvester supported that well with its ability to be network-booted and automate the configuration of it during boot-up, so that when installing it, you wouldn't have to go through a typical Linux installer. It was just ready to go. It made that configuration.

Jose: And to follow up on that, since a lot of what you were using and deploying are mostly cloud-native tools, did you have to configure them or make them very specific to work in an edge environment like this, or they worked as expected off the shelf without having to make any changes to them?

Doug: Yes, I can take that. A lot of the cloud-native tooling, they're great because they work on the API. If some of the APIs are discoverable, it makes it easier for them to just start up and work. In some cases, though, for example, if you're going to something that has an S3 API, you'd have to point it to the local end point that points to that storage. The other thing that's also as a potential, some of these can be configured through DHCP, so that you can just point them and provide option 42 and DHCP. You can just put the host name to go look for that API, so the device is coming up and seeing that. They are cloud-native, but when we say cloud-native, there's always some sort of configuration with your initial cloud tool that you have to point to. You have to set your region, do the log in. When we had to do these, a lot of times, there wasn't a ton more configuration that you'd have to do than if you're going to a traditional cloud provider, which makes them a lot easier to set up and go for.

Jose: OK. We have a bare-metal server that's sitting with some resource constraints, possibly air-gapped. You're deploying cloud-native technologies onto this. Within this environment there's a lot going on here. I'm really interested in hearing about the challenges, if there were any, that you came across and building this out and how you took care of it. I'm assuming since there's so many parts, this is sort of an application of cloud-native tech that maybe is not seen often, you may have seen some challenges. You may have not. What do you guys think? Maybe Doug, we can start with you.

Doug: With any kind of large deployment, same thing we ran to here is trying to do this without connectivity. Because even though you kind of set all these devices up to work, without any connectivity, there's always that one thing that you're like, *Oh, well, hey, I'm trying to connect to something something.google.com*. That's always a problem that you have to do your testing to get that out of the way. If you are developing this sort of system, I know Patrick, he went through and tested this probably about a thousand times to make sure he got it right over a bunch of different systems. I think that's one of the big things that will get you. You miss some random thing and maybe it didn't come up before because you didn't have a feature turned

on, [and you need to] try to make sure that it is not reaching out to get something. And you're out in the field on the edge, so a lot of the stuff that Jeff had worked through, we automate all that package build. You still have to make sure that you have that edge package, either running on an external hard drive or a set of DVDs, and to be able to hook up your network gear and laptop together to connect to it. You still have to do your basic types of connectivity just to get online with the system. It seems like every time I've ever worked with disconnected environments, you always run into something that just didn't happen to be in your test case. That's something that you always need to plan for, especially if you're installing these devices somewhere. It is one thing if you're disconnected in a low side, high side environment. But if you have to drive two hours in the middle of nowhere, where there's no cell phone connectivity or gas stations, and then plug them in, and then you're like, *Oh, man, I forgot something*. It is critical to have as many test cases to test that out and make sure you're not missing anything.

Jeffrey: Yes, there's definitely challenges. And one of the challenges that I faced was when we were building these (our Zarf packages and CI/CD pipelines) is there's a lot of dependencies. And how do you track them? How do you make sure that no vulnerabilities sneak in there? One of the other great things about Zarf is you can export a software bill of materials, SBOMs, as we call them. And what an SBOM is, is it is basically an inventory of the dependencies in a software package, in this case, in a container image where things come from the source, the version, the license. A lot of our partners that we work with, they like to see a software bill of materials file. In this case, what we did was, to make it easier to track all these dependencies and make sure there's no vulnerabilities, is we exported as a SBOM using Zarf, as I mentioned. However, we wanted it in a format that actually was useful to us, so something like CycloneDX or SPDX. Those are the two formats that most people use for software bill of materials.

Jose: Does that capability come native in Zarf?

Jeffrey: The ability to create an SBOM comes native in Zarf. However, we took an additional step to convert that SBOM into CycloneDX format. And what we did from there was we pushed our SBOMs to a tool called OWASP Dependency-Track. What we're doing is we wanted to say, *Hey, here's a proof of concept of how you enforce policy*. A tool like OWASP Dependency-Track allows us to do that because you can say, *If there's a particular license that's not approved to use and on our edge device, throw a warning*. When we build a

new package, what it does is create the SBOM, changes to the CycloneDX format, pushes that to this tool, OWASP Dependency-Track. Check for any vulnerabilities. Check for any policy violations. Maybe there's a CV that comes up. It is like, *No, stop the bill. We don't want that*. Or a license, or a source for a package. Those are all things that could be used to enforce policy. Rule sets, basically, is what I'm talking about. And then report back to the pipeline if there's any violations and then proceed from there. The challenge was a lot of stuff in these Zarf packages. How do you make sure they're secure? How do you track versions and dependencies? And we did that through using a software bill of materials.

Jose: I want to come back to that aspect of security in a minute, but I'd like to give Patrick a chance, if there are any challenges you came across in your contribution on this prototype that you want to bring up.

Patrick: Sure. Probably the biggest challenges are just at the pre-boot environment, dealing with automating everything there. For example, when you configure Harvester, you need to give it a management interface or which network interface to use. If this is a server of one card, that's easy, but we were dealing with one that had up to eight; and you just had to figure out and decipher what the device was going to be named and figure out how to do that. Additionally, another issue we ran into is trying to automate the step of the initial user creation in the Harvester web UI. We didn't get past this part, and that could be fixed by now. But thankfully, another benefit of Harvester is that it is open source. If we really wanted to, we could have gone in there and maybe submitted a merge request to get that feature up and running. But I would say the biggest problems were just working—and this was a new experience for me—working at the pre-boot BIOS level and getting that all set up.

Jose: I want to go back to security. Jeff, you mentioned a lot about security steps that you took in this prototype overall. What were other security steps that were taken that may have been implemented or you would like to have implemented but you couldn't get to it in both the building and deploying itself? Did you apply DevSecOps in any of this to enforce some security controls or any other security aspects you want to bring up?

Jeffrey: Yes, I can jump in there and just talk from the CI/CD standpoint. There's a lot of great tools out there that you could pop into your CI/CD pipeline that are just best practices. Maybe you want to link your Terraform.

Doug could speak more to that. He's the expert on Terraform in this group, but things like lint my Helm chart. Scan the code. And if there's application code, it is our package. Run some sort of static analysis tool, that sort of thing. Just best practices. DevSecOps best practices. Use automation and tooling to achieve to scan your code and lint your code and that sort of thing, and look for vulnerabilities.

Jose: Were you able to put those into the prototype?

Jeffrey: They're in the prototype. Yes.

Doug: As Jeff mentioned, Terraform, there's a couple different open source tools that do Terraform linting, which are kind of helpful because one of the things that often happens is you get a provider upgrade and they add new features and things that you didn't realize that they added if you're used to using an older provider. I think, off the top of my head, tfsec will scan and say, *Oh, hey, you should probably do this to improve security*. Injecting those will help you find stuff that you may have missed. One of the things that I like is Ansible. If you're writing automation, there's an Ansible lint tool that helps you identify different things. For example, if you copy a file over to a system, it says, *Hey, there, you need to put an owner and a file node on this in a group*. It helps you think about security as you're creating that particular tool. And the other thing, finally, is we talked about using the Platform One, Big Bang package. One of the things that that is an open source package that the Air Force develops, and one of the nice things about it is it provides virtually all the default security controls that you need for operating in a federal or DoD environment. And then that's a lot with encryption standards and those type of things that, normally, if you're building a system from scratch, you'd have to do a lot of manual configuration, whereas on the Kubernetes platform, it's just, *Hey, here's the easy button*. Maybe you have to tweak it. Maybe you have to do things based on what the requirements for your environment are, but it just makes security so much easier than trying to roll your own right out of the gate. Security is a big importance in what we do. And really what everybody, whether they're commercial, government, or whatnot, especially with the uptick of different cyber activities and banks and hospitals getting taken over by software malware and ransomware. People might gripe at extra security because you have to do more, or type in more passwords or use different types of keys. But part of the reality of doing anything with IT, or computers, or software nowadays is with security, generally, the more, the better, unfortunately.

Jose: An open question for the three of you: when you're looking at the amount of security that is best practice in today's world, and you're applying it to an edge device like the one you worked with, does that overhead use significant resource consumption on that edge device, thus reducing what's available for the system that is actually supposed to be running on that device, or is it not important?

Jeffrey: If you don't mind, I'll answer that and give my two cents. I don't think there's any significant overhead. It is all done before the fact. One thing I will say is with an edge device, you want to tune in your security controls. For example, a vulnerability that comes up might not apply to an edge device. You essentially want to make sure that you can get rid of as much noise as possible in your security controls and only care about things that are applicable to your environment.

Doug: Yes, as Jeff said, if you're trying to work security in any of your products, you have to do a risk assessment. And that's important. Like Jeff said, most of the security controls that go on, they're not super overhead nowadays. You can buy a whole computer in a Raspberry Pi that basically five years ago would have been almost a supercomputer. You have a lot of processing power out there. But the key is that you still have to do your risk assessment. *Do we really need a giant firewall, a big iron firewall if we don't have any networks we're connecting to?* You just have to do that assessment and base your decisions off of those choices. And do you need all those security controls? Maybe, but you won't know until you run do that risk assessment and analyze what kind of threats you're trying to protect against.

Jose: One challenge I'd like to bring up that I didn't hear mentioned, and probably because it wasn't part of the prototype, but maybe you thought about it, is subsequent updates to this edge device. Once the initial deployment of the prototype is completed, does that follow the same process as what you put together, or is there anything different that occurs? You always want to be rebuilding, and reintegrating, and redeploying. Is it the same process as your prototype or is there any considerations that make it a little different?

Jeffrey: It is the same process. That's the automation, something new, a new version of the dependency comes out, you rerun your process and create new deployment artifacts.

Doug: What Jeff said, that's the easy button because you just redeploy new packages. Patrick, do you have any insight into the Harvester update? That wasn't one of the things we had to run into, I think, when we were deploying this, did we?

Patrick: No. We didn't test doing updates to the OS level. From what I could tell, since it is running on MicroOS underneath, which is an immutable operating system. I could see a world where you send out a bunch of hard drives that have the updates and apply them, and in theory, it just rolls everything to the new version. But our prototype window was so short, and there was never a time where we had a chance to practice that. But from a higher-level point of view, like Jeff said, like updating the actual deployments on there, Zarf handles that pretty easily. There's another package type, which we didn't get to play with too much, called differential, which basically looks at the current state of a package that's deployed and the new one and will apply updates as necessary. But from my point of view at the OS level, we never got to try that out.

Doug: That's the win with immutable operating systems. The goal is you send out maybe an ISO image, you run the update, and it basically builds a snapshot of what the new operating system is going to look like. And then part of the process is the next boot up of that system, it basically lets you swap to the new version, and then if it breaks, you can just reboot and swap back. Instead of having to apply thousands of packages and waiting an hour to install on the operating system you can just do this in the background. And then if you have an HA environment, you can just push everything to another node, reboot one node, then push everything back, and then do that update, so you don't really have any downtime. But obviously, in a single node edge environment, you'd have to take that downtime to do that reboot. But that's probably a good reason why I think people should look more towards immutable operating systems. It is easier, there's less downtime, and of course, since you can't change anything on them unless, since they're immutable, that's maybe an overall grand statement. But for the most part, it makes it easier to do this update and have things on the system without having a bunch of changes being made that shouldn't be made.

Jose: We've already covered what the prototype's origins were, how the prototype was built, how it looks now. As you all know, technology is always rapidly changing. It changes almost seemingly minute by minute. Given the

rapid change of environments in both software and hardware, how do you see this prototype being extended or enhanced to meet this rapidly changing environment? Is there something coming up the pipe that would have an impact on how your prototype or how this type of deployment currently functions or would have to be adapted to work with that?

Patrick: There are two big things that come to mind, and probably, the first one is what everyone's thinking is AI at the edge. I could see a world where maybe you want to run an LLM agent. Integrating the capability where you have GPU compute available so that you can run them and getting the models down to a size that they work. I mean, we are already running AI models on our cell phones. In theory, that wouldn't be too much work, but then there's power constraints and all that kind of stuff. But I'd say that's probably the biggest thing. Another is moving to a new CPU architecture. All our testing was done on our classic AMD64 architecture. A lot of the stuff we work with in the Linux world still runs and is compiled for that. But there has been a pretty big movement to move to ARM64 or AArch. And one of the deployments we mentioned, Big Bang, they're slowly working on getting a lot of their base images compatible with that. There's a bunch of companies out there that are building these servers that run off ARM64. And one of the big benefits of that is they are a lot more power efficient compared to their x86 or AMD64 counterparts, which is obviously a big benefit in the edge scenario where you are worried about your power constraints. If we were to extend this prototype, I would say those are the two big things would be looking at how do we integrate GPUs or see if there's any pains with that and seeing if we can get an ARM64 version of the system up and running.

Doug: I think Patrick hit the nail on the head. Right now, we're looking at AI, LLMs. How can we do inference on the edge? And then whether it is GPUs or if it is just regular compute, ARM is definitely more efficient. And a lot of times in the case, it is less expensive than x86 or AMD64 architecture. I think that's going to be moving along with everything else that's coming up with the AI revolution.

Jose: All right. Remembering that the Software Engineering Institute is a federally funded research and development center, one of the key things I always try to do is transition these ideas out to the community, to interested parties, to benefit from our findings. If the listeners of this podcast want to know more about edge environments and cloud infrastructures, where can they go to get more education and to learn more? Maybe Doug.

Doug: Our team at the SEI, our focus on our team is automation, infrastructure, CI/CD, and everything basically DevOps—we try to say DevSecOps for the security that's in the middle. Right now, we're working on more blogs, podcasts like this, and some videos. Feel free to tune into that. The SEI website has a lot of resources on there. Part of being an FFRDC, we are trying to give the government and DoD unbiased opinions and advice on how to do this thing in the best possible fashion. Feel free to reach out to info@sei.cmu.edu. And maybe we can help you and do work for you, or we can point you to someone who can help you out there. That's what we're here for, and part of this podcast, we're trying to give back more information to the community. People can't get education on things that don't happen. It is not the normal case where you just run over to the cloud and deploy some lambdas and go to town. We're just trying to give people options on different types of challenges that we run out to in the field, so yes, feel free to reach out.

Jose: All right. Excellent. This has been an excellent podcast. To close, I just want to ask each of you, you want to tell us what you're working on now, what you plan on working on next, what's next for you. Let's just go down the list here. We can start with Patrick.

Patrick: Currently what I'm working on is there's the new CMMC guidelines coming out, just kind of looking at getting up to speed on those and seeing if there's a possibility of maybe throwing some automation in there. Other things I'm working on is there is a current improvement proposal for Kubernetes where you are able to mount an image as a volume source that's currently being developed. It is only supported by one of the container runtimes. I'm looking to see if there's any way I could help move it along and get it working in containerd.

Jose: All right. Thank you. Doug?

Doug: Almost always, I'm doing some type of Terraform, or GitOps, or infrastructure as code. That's just kind of a given with what I do. But recently, I've been trying to get into doing some RMF attestation where you can automate and do checking on what sort of controls that you have in place. Instead of working on just building a hand building a list and analyzing everything is to try to automate some of that manual work that you have to spend a decent amount of time on. That's kind of my wheelhouse right now.

How about you, Jeff?

Jeffrey: Yes, thanks, Doug. I'm doing a lot of automation, as we all said. And a lot of my work lately has been in an air-gapped environment, how to run automated testing, automated test harnesses in an air-gapped environment, as well as static analysis in an air-gapped environment. A lot of my time has been focused on those two areas.

Jose: All right. Well, thanks, Patrick, Jeff, Doug, the three of you for taking time to come and talk to us today about this podcast. This is really exciting stuff. If the listeners here want to learn more, this podcast will be on all the available streaming services, the SEI's YouTube channel. If you have any questions, please, reach out to us, info@sei.cmu.edu. I'd like to thank everyone once again. I wish you all well. We'll talk again soon. Thank you very much.

Thanks for joining us, this episode is available where you download podcasts. Including SoundCloud, TuneIn radio, and Apple podcasts. It is also available on the SEI website at sei.cmu.edu/podcasts and the *SEI's YouTube channel. This copyrighted work is made available through the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense. For more information about the SEI and this work, please visit www.sei.cmu.edu. As always, if you have any questions, please don't hesitate to e-mail us at info@sei.cmu.edu. Thank you.*