Application Secret Management with AWS

Emmanuel Apau

https://devopsdays.org/events/2019-washington-dc/program/john-allspaw/

- Twitter - https://twitter.com/technoGrouch
- Video Recording - https://youtu.be/CEoVxvwLhsQ
- Slides - https://thegrouchy.dev/general/2019/07/12/application-secrets.html
- Audio - http://traffic.libsyn.com/devopsdays/application-secrets-management.mp3

EMMANUEL: Let me change my screen settings. All right. That's much better. Cool. So today we're going to talk a little about application secret management with AWS. Raise your hands if you feel like your company does a good job in securing your application secrets. All right. Not a lot of positivity towards your companies' practices. But you know, it's -- there's a lot of different ways to handle application secrets and the different places that I've worked, I've seen people do it a good way, I've seen some bad practices, for example. Just because your secrets are in a private GitHub repo doesn't mean they're secure, but it's at least better having it in a public repo. Let's talk about who I am and what I'm going to be doing today. As mentioned earlier, my name is Emmanuel Apau part of the black code collective, if you are a fan of the T-shirt feel free to get one at this URL. That's my Twitter if you feel like listening to my rants about software development in general. So why do I carry about secret management? For one, it's you definitely don't want to wake up one morning and there's a headline in the Washington Post and you see a title like insert name here has left a huge trove of highly sensitive data. Sensationalized but it's never fun to wake up and get a surprise like that. It's one thing I've learned is I don't like surprises. So I want to talk about a few use cases in which you might manage your secrets. There's SSH keys in your secrets, SSL certificates configuration, if you're working with Kubernetes it might be -- API keys and database credentials, single usage, one-time secret for like, I don't know, if your IT has to send you a password for an account. And of course external service integration like Kubernetes secret syncing. Application secret management is important if you're in a start-up and you're about to be bought out during the due diligence face they come into your company and walk through a list of things you do well and things you need improvements on and handling of secrets tends to come up. I want to walk through a few examples how you can integrate those better using AWS specifically. Just in

case you're unaware of what the typical lifecycle is or the work flow is for a secret, you would have your application that makes a request to a vault. A vault could be all the different options. Your vault is going to take this request, look for the secret, decrypt it, kick it back to your application, let's say it's a database password and your app now has that secret that it can use in an authentication request to that database. Pretty straightforward. The problem is, the vault that's so many options. If you go on Google right now, what is one of the best secret vaults out there? You'll get a whole list of why everyone thinks their solution is the best solution. You have AWS secret manager and AWS parameter story. Why there's two I'm not sure. There's key vault, last pass, one password even base 64 encoding? Maybe? I don't know. I've seen it before, though. So let's dive into parameter store because that's where I want to dive into today. Let's dive into a quick demo in what is parameter store and what that looks like. AWS's system manager is going to be an option all the way at the bottom and shared resources. When you click that, it's going to give you your list of parameters. Pretty straightforward. And on the UI you're going see the number of your key, the tier that is standard, the description, key ID and the version. When it was modified and the last user. We're going to create a real quick secret just for demo purposes. We'll call this DevOpsDays test. Description don't need one. Leave it as standard. Use a secure string, the default key and call this test. All right. So I want to scroll down, select that.

Once you select it, you have an option of a history, you have a history menu where AWS where store up to 100 different versions of your secret. This is useful in the use case where your engineers are constantly making changes and all of a sudden you make a production deployment and things start breaking you can go back and restart your service an call it a day. And then you also have the ability to decrypt it to see what that secret would be and who was the person who modified it, to be honest that's the most important feature of a secret because when things do go wrong, you need to understand why they went wrong and the reason I like parameter store is the auditing behind it. If you're sharing things like last pass and shared note, you don't necessarily have that auditing feature or the revertibility feature. You have to hope your memory is good so you know how to revert quickly when things go wrong. On top of that, you encrypt your key using a KMS and a KMS key would be customer management key. I do recommend that you create your own KMS key when you are encrypting your secrets because then you can apply IM rules and permissions, accessibility rules behind who can encrypt, who can decrypt, who can read a secret and so on and so forth. The nice thing about parameter store, AWS has a monopoly when it comes to cloud engineering and you already probably have

IM users and rules clearly defined so you don't have to now rebuild your accessibility policies when you're handling your different secrets. You can reuse and update what you already have. If we're going to create a key let's call it SSM/dev for secrets. Click next. Don't need any tags. You can determine who the key administrators will be, whether that's your cyber team for now I'm going to use the user I created before. And say this user has the ability to use this key to encrypt and decrypt and it's going to give you a preview on what your IM rule is going to look like. And let me make this a little bigger. I realize it's going to be tough to see from the back. But what it does is it says that this user has the ability to create the key, describe the key, but most importantly, let's try and find, the encrypt and decrypt rules. This is what's going to be the main feature when you're using it for parameter store. So typically my work flow is that for dev, there's a key for stage, key for prod and a key for global secrets because sometimes you don't get the diversity of secrets and you have to use the same one across the board. So you can begin to fine tune who has access to what key and at what moment. So we're just going to finish that real quick. And that's that. Pretty straightforward. I like it because it's managed service. I didn't have to build this out myself. So there's no maintenance overhead. It's just configuration at this point. So a few things I like, just to reiterate, is that you can reuse IM policies and rules for access management. There's change management auditing. Cloud trail will track who's been updating so you know when things are deleted and shouldn't be. That could be propagated to a slack channel or email or so on and so forth. No maintenance overhead just configuration and it is encrypted with KMS. The work flow for parameter store is as follows. You have a user, a developer. They're typically going to be using the CLI or the UI. Not a big fan of the UI. I don't know if you guys have played around with AWS's UIs in general, they kind of suck across the board. A user will confirm that one, that they're allowed to even use the key that they're trying to encrypt the key with using their IM rules. Once that's been accepted, then you're going to update ECS parameter store then on the back end, when your application needs to use it, whether it's using the SDK or the CLI itself it's going to go through the same work flow where it confirms it has access to your secret and gets it back from the parameter store. I want to give a baseline on what we're going to be talking about in a few more minutes. Does anyone have a question on what they're seeing so far? Who has actually been using parameter store? Okay. So I'm not like I'm preaching to the choir over here, it seems like. Good stuff. Let's see. So yes. When I was finally got picked to do this talk, I've given this talk once been in February or March, something like that. Okay, cool, I can come here use the same slides call it a day. It's going to be simple. One of the great things about our jobs is that

technology moves really, really quickly. And a service that you were using maybe six months ago might have completely revamped either one their pricing structure or revamped different features or the tools that you were using to interact with the service becomes outdated because of AWS push. Okay. So I was like let's see what's new with the parameter store and they've introduced advanced parameters and that was posted on April 25th which as I've realized has changed completely how I would have initially pitched parameter store to you guys because it has pricing concerns. So, let's get right into it. What is a standard secret and what is an advanced secret? Standard they allow you up to 10,000 secrets versus 100,000. They have increased the size which you can store. It's still going to be a hundred history values, but one of the annoying things about parameter store, because there's always cons to these things, is that the transactions per second would cause you to have API rate limiting. If you were accessing secrets in real time for a lot of your applications. An example is let's say during a user flow, your application makes a call to database in real-time, it pulls the secret in real time to authenticate the user. If you scale this, you're going to hit that API limit pretty quickly. So I guess that was a concern and people complained about a lot and they've come up with an advanced secret which allows you 00 API transactions per second up to a thousand API transactions per second. But of course the difference is always about what is the best price for me. The parameter store before was pretty much free, until AWS came out with a second secret manager called AWS secret manager. I'm going to talk about this a little bit because it is important to understand the differences. They're pretty different products even though it seems like they're solving the same problem. You would typically use parameter store for API keys, possibly database credentials and miscellaneous key values like API keys and so on. You do get the versioning, you get the history and the cloud trail auditing. But with secret manager, you don't get the versioning, you don't get the history, but it does allow you to rotate your secrets if you happen to be using RDS. This is sweet in the fact you don't have to glue and pull together your custom scripts if this is a security policy you need to adhere to. So with secret manager, it's $0.40 per secret per month, versus parameter store which is free, but they do charge you on the transactions per second when you do interact with their API. Advanced would be $0.05 per secret. So in this respect, unless you have a reason to rotate passwords for your database or any other use case, parameter store is going to be the way to go. You would only use secret manager for rotation of secrets. All right. So because we talked a little about price, sometimes it's not always easy to understand how much that's truly going to cost for your particular use case. So let's math it out. Assume you have 5,000 parameters, and of which 500 are advanced parameters. And let's say

you interact with these parameters 24 times a day, 3.6 million transactions over a 30-day period. And assume you also have the higher throughput. Higher throughput is a new features introduced with advanced parameters, that allows you to introduce the 100 or 1,000 transactions per second that I spoke about earlier. So when you look at the cost, you're going to pay for the 500 advanced parameters, that would be 25 bucks and the 3.6 API interaction million API interactions would be multiplied by $0.05 per a thousand interactions so that would be 18 bucks. So total monthly cost is about 43 bucks. But everyone's use case is going to be different. And as an engineer this might be pretty important because you might have a, you know, cost limitations and things like that. So definitely make sure that when you do pick your secret option, do the math behind how much it's going to cost on a monthly bases, and that will help you determine whether you want to go with a managed service route or whether you want to go with an on-prem or individually built-out solution because there is going to be maintenance overhead and cost engineering time to maintain your own person solution versus using this service. Does that weigh out? I don't know. You would have to figure that out on your own.

So now we're going to talk a little bit how you would integrate your secrets into your application. So you can do this program atically. As I said earlier if your application is going to make database calls in real time. If you're using the SDK, you would use the SSN parameter, make the call to get the key but you are going to be subject to that rate limiting as you scale if you're using ECS, you have a dev stage and prod cluster, if they're on the same account, you're going to be hitting API limits on one account. If you're taking advantage of a multi-account set up like AWS organizations they have available, you might be able to manage that load if your clusters are spread out across different accounts. Then of course my personal favorite is on start-up. You can store year secrets a global environment variable and you wouldn't have to worry about accessing your second secret in the middle of an application work flow. On start-up, you can access them as you typically would through the application. And then in a micro service world, if you are using ECS, they have a pretty seamless integration between parameter store and ECS task definitions. In your container definition you would define in the secrets key this is my global environment variable that I want to set and this is its reference to it in parameter store and you would just use the fully qualified ARN to reference that key. Pretty straightforward. It happens on startup. Is anyone actually using ECS? Okay. I don't envy you. I just started using ECS from Kubernetes and it's lacking a lot of features. But I am very happy they do have seamless integration with parameter store. So then of course if you are using Kubernetes, if you're not familiar, this is what a simple service definition looks like. You would

have your spec that provides your container image that you're going to be using and your different environment variables. You could have the environment variables in plain text or you can reference Kubernetes they already have prebuilt. That secret manager that Kubernetes has also has its own template. It would be of the kind secret template and you would have your data. For example it would be a user name and that would be base 64 encoded value that you would then apply to your cluster. All your different containers in your cluster would make this reference to the Kubernetes secret manager and they would pull information that they need that way. But the thing is because Kubernetes is kind of an external I guess feature from AWS, you would have to create that glue between parameter store and Kubernetes to kind of sync the secrets from parameter store to Kubernetes. So just a few tips. Diversity of secrets per environments is key. You don't want to use the same secrets that you have in dev that you're using in production. I've seen time and time again when devs are developing in their different environments and now I don't know they accidentally committed their file to their GitHub. Diversity of secrets is key. Finely tuned die crypt access roles for admins, developers and PMs. Example last pass not pace bin. There's been many times I've been browsing pace bin and found developer configurations chilling in the open. As an exercise for you guys, I would be curious if you took some developer credentials that you currently have at your company and just Google for it out there and see what you come up with. And if anything, they should thank you, that you find a security hole, but don't blame me if they don't. Use temporary credentials where possible. RDS came out with token-based authentication which is pretty sweet, which is just temporary credentials it lasts for 15 minutes, so you don't have to give your devs static passwords that you have to rotate 90 days. They would get these tokens when they need it and they would refresh every 15 minutes during their work. And also, most importantly, for secrets management is to make sure everybody understands it. AWS has this idea for shared responsibility model. I feel like this is very important when it comes to cyber liability and DevOps in general. There is a shared responsibility model when it comes to securing your secrets between your DevOps guys and your engineers. They're going to be the ones interacting with it. You're possibly going to be handing over the keys to them. So the more people understand about your process, hopefully the more eyes will see holes. Before I move on, does anyone have questions or if you have your own tips in general that you've seen work in your different jobs and careers. All right. So it wouldn't be a talk about I didn't have something to sell, so to speak. So enforcer reloaded. Because this time it's personal. And personal in the fact that I just forked it into my personal repo and made some changes. I've been working with

parameter store for a while and the UI completely sucks. So I created CLI to help me expedite the uploading of secrets and so on and so forth. And right now it helps me to list secrets pretty quickly, create secrets and force the conventions that I needed developers to use when creating secrets whether that's tags for the secret, whether that is using defined KMS keys for dev, stage and prod and also sometimes we have really big secrets. Your kub config can be big. It helps me chunk out into 4K chunks and upload into the parameter store and merges them when I need to use them. And it also has Kubernetes synchronization helper function, just to kind of create that glue between Kubernetes and parameter store. A future want is to handle advanced secrets since that's now a thing. But as with open source development, if anybody is using parameter store and has an interest in a tool like this, feel free to three up a PR. I want to give a real quick demo on Enforcer real quick. Can everyone see the terminal?

Reloaded, list, interpolate values. Reload example using my Enforcer profile. We're going to use SSM dev. All right. Another important thing the secrets are storied in a file directory. You have to start with a forward slash and along with the name of your key.

It's going to create directory buckets for your secrets, so to speak. Example secret, test, and then the way this works is you copy the secret you want to copy and then it's going to show me this. Yes, yes, yes. It's going to be region US2, profile enforcer, use the KMS key SSM dev, upload pretty straightforward. Now I'm going to show you how it interpolates Kubernetes secret templates. So I have this template over here that takes user name, a password as an email. And from parameter store you're going to access a key for the account template. So if I look over here, you're going to see -- whoops. You see, this is why this UI sucks. I'm going to access this key over here. Account user name that just happens to have the value user name. It's going to be for the account bucket. So my template is going to be with a directory account for Kubernetes with a profile enforcer and you're going to see it spits out a Kubernetes secret template that I can pipe into a Kubernetes apply and call it a day. And you'll notice it base 64 encoded the password but it didn't because I didn't define the base 64. In case you're actually storing your secrets already base 64 encoded, you don't have to pipe it into anything.

So that is pretty much my spiel. Does anyone have any questions? Okay. There you go.

[Off microphone]

>> Yeah yeah yeah. If you go to GitHub/ -- you can see I forked it from where I used to work, check it out. My read me is pretty verbose but if it's not, you can hit me up on Twitter and say the read me sucks and I'll get to work. Any other questions? All right. And before I close out, I got to give the obligatory -- you know what I mean. We're hiring at c vent social tables, looking

for senior site reliability engineers and other engineers as well. So check us out. It's been cool.

Thanks for listening.

[Applause]