Hey, I'm

Vedang Joshi

I am currently pursuing a Master's in Software Engineering with a focus on Scalable Systems at Carnegie Mellon University. I have a passion for reading, discussing, and developing software. Previously, I worked as an SDE-1 on a Cybersecurity team, where I specialized in building web applications and automating cloud processes.

👽About me

Hi, I'm Vedang, a computer enthusiast and student at Carnegie Mellon University (CMU), with a passion for distributed systems - there's something incredibly cool about solving complex, large-scale challenges. My journey into tech really took off during a two-year stint on the cybersecurity team at Innovaccer, a late-stage startup. I call it an "unfair advantage" because I had the opportunity to dive deep into all three major cloud providers, automate processes, build web applications, and develop a strong foundation in security-all while having a blast.
That experience sparked my interest in scalable, distributed systems, where I could combine technical depth with my desire to solve meaningful, impactful problems. I'm always full of energy, self-taught, and constantly looking for new things to learn. Whether it's hacking away at a new project or learning the latest tech, I'm eager to dive in and push the limits.

When I'm not programming, you'll probably find me immersed in a good book or vibing to music. I also have a knack for DIY projects - building things from scratch has been my first love for as long as I can remember. Whether it's crafting something new or tinkering with gadgets, I'm always looking for creative outlets. And of course, I love chatting about the latest trends in tech - there's always something exciting happening!

🎨Skills

Over the last two years, I've spent my days working with AWS and Azure, where I was the main person behind an in-house security solution that had a huge impact. I took this project from designing it, to building it, to keeping it running smoothly as it grew. For the same project, I integrated tools like Snowflake, Databricks, and New Relic, connecting everything with cron jobs, Python FastAPI, and a React frontend. I also got hands-on with CI/CD pipelines and Kubernetes. One of my proudest achievements was building a Terraform module that's deployed every time a new customer comes on board. It uses StackRox (now Red Hat Advanced Cluster Security) for protecting clusters, and I managed our own StackRox setup as if it were my own little garden.

In our team, I became the go-to person for fixing Snowflake connectivity issues. We had layers of security in our Snowflake setup, and I was one of the few who really understood how each piece worked. So whenever someone hit a connection problem, they'd come straight to me. I also took care of managing Satori Cyber clusters, handling updates, scaling, and monitoring them like I was growing a plant—always keeping an eye on their health.

I've also had the chance to do a lot of internships throughout college, and I credit internships and hackathons for a lot of my learning. I participated in Google Summer of Code, where I teamed up with some awesome Red Hat engineers to build an IoT-as-a-Service solution in Rust, which was a great learning experience.

I've interned twice at Innovaccer—once working on a schema API that let you change database definitions through a REST API, and another time building automations to simulate DOS attacks and run Nmap scans on open ports. On top of that, I even tried my hand at Android development, spending four months working with Kotlin and CameraX, and had a great time doing it.

All these experiences across different areas of tech have been amazing, but what excites me the most is diving deep into distributed systems. I'm eager to build and scale software that can solve real-world problems and reach people around the globe.

📂Projects

Some of the projects I built during college.

- Paradigm, is an automated testing tool. It is a plugin that converts whatever is taught in an online class to True/False and Fill in the blanks question and sends them to the students' smartphone in real-time. This helps improve attentiveness and retention in students. Demo
- Famulus, is an intelligent stock prediction tool. The users input a keyword, this tool analyses all the latest news articles to predict the change in the value of this stock in an interval of 1, 5, 7, 15 days. Read more
- Frisson, is a very simple android app to read stories about UFO sightings. It was built with Kotlin, following all aspects of MAD aka Modern Android Development.
- Music Wall, is a Web App, it creates a beautiful Music wall webpage for you, whenever you like a music video, it is added to this page, thus you can share your music interests with friends.
- Assistant, is a script that adds voice commands to simple tasks. It uses IFTT and thus works with Google Assistant, Alexa, or even automatic.

📻Contact

- Contact button

Hey, I'm

Vedang Joshi

I am currently pursuing a Master's in Software Engineering with a focus on Scalable Systems at Carnegie Mellon University. I have a passion for reading, discussing, and developing software. Previously, I worked as an SDE-1 on a Cybersecurity team, where I specialized in building web applications and automating cloud processes.

👽About me

Hi, I'm Vedang, a computer enthusiast and student at Carnegie Mellon University (CMU), with a passion for distributed systems - there's something incredibly cool about solving complex, large-scale challenges. My journey into tech really took off during a two-year stint on the cybersecurity team at Innovaccer, a late-stage startup. I call it an "unfair advantage" because I had the opportunity to dive deep into all three major cloud providers, automate processes, build web applications, and develop a strong foundation in security-all while having a blast.
That experience sparked my interest in scalable, distributed systems, where I could combine technical depth with my desire to solve meaningful, impactful problems. I'm always full of energy, self-taught, and constantly looking for new things to learn. Whether it's hacking away at a new project or learning the latest tech, I'm eager to dive in and push the limits.

When I'm not programming, you'll probably find me immersed in a good book or vibing to music. I also have a knack for DIY projects - building things from scratch has been my first love for as long as I can remember. Whether it's crafting something new or tinkering with gadgets, I'm always looking for creative outlets. And of course, I love chatting about the latest trends in tech - there's always something exciting happening!

🎨Skills

Over the last two years, I've spent my days working with AWS and Azure, where I was the main person behind an in-house security solution that had a huge impact. I took this project from designing it, to building it, to keeping it running smoothly as it grew. For the same project, I integrated tools like Snowflake, Databricks, and New Relic, connecting everything with cron jobs, Python FastAPI, and a React frontend. I also got hands-on with CI/CD pipelines and Kubernetes. One of my proudest achievements was building a Terraform module that's deployed every time a new customer comes on board. It uses StackRox (now Red Hat Advanced Cluster Security) for protecting clusters, and I managed our own StackRox setup as if it were my own little garden.

In our team, I became the go-to person for fixing Snowflake connectivity issues. We had layers of security in our Snowflake setup, and I was one of the few who really understood how each piece worked. So whenever someone hit a connection problem, they'd come straight to me. I also took care of managing Satori Cyber clusters, handling updates, scaling, and monitoring them like I was growing a plant—always keeping an eye on their health.

I've also had the chance to do a lot of internships throughout college, and I credit internships and hackathons for a lot of my learning. I participated in Google Summer of Code, where I teamed up with some awesome Red Hat engineers to build an IoT-as-a-Service solution in Rust, which was a great learning experience.

I've interned twice at Innovaccer—once working on a schema API that let you change database definitions through a REST API, and another time building automations to simulate DOS attacks and run Nmap scans on open ports. On top of that, I even tried my hand at Android development, spending four months working with Kotlin and CameraX, and had a great time doing it.

All these experiences across different areas of tech have been amazing, but what excites me the most is diving deep into distributed systems. I'm eager to build and scale software that can solve real-world problems and reach people around the globe.

📂Projects

Some of the projects I built during college.

- Paradigm, is an automated testing tool. It is a plugin that converts whatever is taught in an online class to True/False and Fill in the blanks question and sends them to the students' smartphone in real-time. This helps improve attentiveness and retention in students. Demo
- Famulus, is an intelligent stock prediction tool. The users input a keyword, this tool analyses all the latest news articles to predict the change in the value of this stock in an interval of 1, 5, 7, 15 days. Read more
- Frisson, is a very simple android app to read stories about UFO sightings. It was built with Kotlin, following all aspects of MAD aka Modern Android Development.
- Music Wall, is a Web App, it creates a beautiful Music wall webpage for you, whenever you like a music video, it is added to this page, thus you can share your music interests with friends.
- Assistant, is a script that adds voice commands to simple tasks. It uses IFTT and thus works with Google Assistant, Alexa, or even automatic.

📻Contact

- Contact button

Building CRD-Xray: An AI Agent for Kubernetes Custom Operators

June 11, 2025

MySQL on Statefulsets

April 4, 2025

Breaking down common attack vectors on AWS

November 11, 2024

How to reduce the time it takes to patch vulnerabilities?

October 12, 2023

MongoDB Aggregation for Data Analysis

September 14, 2023

Securing AWS using Terrascan

June 27, 2023

Access Management Best Practices

January 18, 2023

Secure coding practices

October 26, 2022

GSoC Final Report

August 16, 2021

GSoC Phase 1

July 17, 2021

Next →

Famulus - building a stock predictor

Author: Vedang Joshi

Published: May 9, 2021 · 4 min read

About a year back, my friends and I participated in a hackathon and received the following problem statement.

> Stock Market Trend Prediction using an Automated News Analysis

I won't narrate the story about the happenings of the hackathon, but I would surely say 2 things:

- NMIMS is the most luxurious college in Indore.
- We didn't win this hackathon, but we did secure 3rd place, unfortunately, there were prizes only for the top 2 :(

So, here is what we built.

Famulus

> Famulus means an assistant working for a scholar.

Famulus is an easy-to-use web application, where the user enters a brand name (company), our web app scraps news related to it, analyses it and predicts the change in the stock prize in an interval of 5, 7, 15, and 30 days. Displaying 3 things on the UI - Change in the stock price, predicted graph and the 3 point news summary.

How does it work ?

The tech stack used here is

- Python: machine learning stuff
- Flask
- React

Machine learning model

Various components are working together to give the final output.

1. Sentiment analyzer - This was an easy task, we took the very popular Twitter dataset and trained a model which outputs the sentiment score of a text, -1 to 0 being negative and 0 to 1 being positive. We tested it on a small sample of news headlines and it worked quite well.
2. We downloaded this Historic Indian news dataset and filtered out all business news. Then put this through our sentiment analyzer and got a sentiment score for each news headline. The final dataset we obtained here was `{ date, news_headline, sentiment_score }`
3. Then we downloaded Historic Sensex data from BSE (Bombay stock exchange) website and merged it with the dataset created above with the index being 'date', so finally, the dataset looked like this `{ date, (sensex_high + sensex_low)/2, sentiment_score}`
4. Now, comes an interesting part, We have two 2 values - sensex_avg and sentiment_score, to map both of them we can simply use linear regression, but do we need to predict these? NO. This is something we already know. What we need here is to use today's sentiment_score to predict the sensex_avg 5 days from now. We need to shift the sensex_avg column accordingly. So, the dataset now looks like `{ date, sentiment_score, sensex_avg_date+5, sensex_avg_date+7, sensex_avg_date+15, sensex_avg_date+30 }`
5. Most of the work is done. We used linear regression to map sentiment_score to sensex_avg_date+5, and similarly others. We create 4 linear regression models here.

Flask

Now, we use beautifulSoup to scrap the current news, put this news through the sentiment analyzer and obtain a sentiment score. This score is feed to the model and we get price predictions for 5, 7, 15, 30 days. Nice :)

Then we went on to create a summarizer, this takes in a few articles and outputs a 3 point summary for it.

Now, was the time to make all this deliverable. We create a Flask API exposing the prediction value, to plot the graph, we used Altair (Vega) which lets us send a JSON response that can be converted to a graph on the frontend.

Hackathon ends

All the work was done in the hackathon, there were a few patchworks and some things didn't work, but I loved this project. I returned home and it took me about a week to remove all the patchwork and fix things, I wrote test cases and also added a WebSocket endpoint using FastAPI. Here is the link to the repository.

- Backend
- Frontend

Note

Neither I nor my teammates are machine learning experts, so this approach has flaws, If you can provide a better approach, my inbox is always open, and as it always is

Thanks for reading!

Machine Learning | Hackthon Project | Python | Github

Hey, I'm

Vedang Joshi

I am currently pursuing a Master's in Software Engineering with a focus on Scalable Systems at Carnegie Mellon University. I have a passion for reading, discussing, and developing software. Previously, I worked as an SDE-1 on a Cybersecurity team, where I specialized in building web applications and automating cloud processes.

👽About me

Hi, I'm Vedang, a computer enthusiast and student at Carnegie Mellon University (CMU), with a passion for distributed systems - there's something incredibly cool about solving complex, large-scale challenges. My journey into tech really took off during a two-year stint on the cybersecurity team at Innovaccer, a late-stage startup. I call it an "unfair advantage" because I had the opportunity to dive deep into all three major cloud providers, automate processes, build web applications, and develop a strong foundation in security-all while having a blast.
That experience sparked my interest in scalable, distributed systems, where I could combine technical depth with my desire to solve meaningful, impactful problems. I'm always full of energy, self-taught, and constantly looking for new things to learn. Whether it's hacking away at a new project or learning the latest tech, I'm eager to dive in and push the limits.

When I'm not programming, you'll probably find me immersed in a good book or vibing to music. I also have a knack for DIY projects - building things from scratch has been my first love for as long as I can remember. Whether it's crafting something new or tinkering with gadgets, I'm always looking for creative outlets. And of course, I love chatting about the latest trends in tech - there's always something exciting happening!

🎨Skills

Over the last two years, I've spent my days working with AWS and Azure, where I was the main person behind an in-house security solution that had a huge impact. I took this project from designing it, to building it, to keeping it running smoothly as it grew. For the same project, I integrated tools like Snowflake, Databricks, and New Relic, connecting everything with cron jobs, Python FastAPI, and a React frontend. I also got hands-on with CI/CD pipelines and Kubernetes. One of my proudest achievements was building a Terraform module that's deployed every time a new customer comes on board. It uses StackRox (now Red Hat Advanced Cluster Security) for protecting clusters, and I managed our own StackRox setup as if it were my own little garden.

In our team, I became the go-to person for fixing Snowflake connectivity issues. We had layers of security in our Snowflake setup, and I was one of the few who really understood how each piece worked. So whenever someone hit a connection problem, they'd come straight to me. I also took care of managing Satori Cyber clusters, handling updates, scaling, and monitoring them like I was growing a plant—always keeping an eye on their health.

I've also had the chance to do a lot of internships throughout college, and I credit internships and hackathons for a lot of my learning. I participated in Google Summer of Code, where I teamed up with some awesome Red Hat engineers to build an IoT-as-a-Service solution in Rust, which was a great learning experience.

I've interned twice at Innovaccer—once working on a schema API that let you change database definitions through a REST API, and another time building automations to simulate DOS attacks and run Nmap scans on open ports. On top of that, I even tried my hand at Android development, spending four months working with Kotlin and CameraX, and had a great time doing it.

All these experiences across different areas of tech have been amazing, but what excites me the most is diving deep into distributed systems. I'm eager to build and scale software that can solve real-world problems and reach people around the globe.

📂Projects

Some of the projects I built during college.

- Paradigm, is an automated testing tool. It is a plugin that converts whatever is taught in an online class to True/False and Fill in the blanks question and sends them to the students' smartphone in real-time. This helps improve attentiveness and retention in students. Demo
- Famulus, is an intelligent stock prediction tool. The users input a keyword, this tool analyses all the latest news articles to predict the change in the value of this stock in an interval of 1, 5, 7, 15 days. Read more
- Frisson, is a very simple android app to read stories about UFO sightings. It was built with Kotlin, following all aspects of MAD aka Modern Android Development.
- Music Wall, is a Web App, it creates a beautiful Music wall webpage for you, whenever you like a music video, it is added to this page, thus you can share your music interests with friends.
- Assistant, is a script that adds voice commands to simple tasks. It uses IFTT and thus works with Google Assistant, Alexa, or even automatic.

📻Contact

- Contact button

Securing AWS using Terrascan

Author: Vedang Joshi
Published: June 27, 2023 · 5 min read

Using Static Code Analysis to Minimize Security Risk on AWS with Terraform and Terrascan

Developing software is a complex process. With each line of code, there is potential for error. Security risks, in particular, can be devastating, leading to data breaches, system downtime, and even monetary loss. To help mitigate these risks when deploying infrastructure using Terraform on AWS, we can utilize static code analysis. A critical tool in this process is Terrascan, which allows developers to automate the process of checking their Infrastructure as Code (IaC) for potential security vulnerabilities.

What is Terrascan?

Terrascan is an open-source static code analysis tool that scans Terraform code for security vulnerabilities and compliance violations. Terrascan comes with pre-written policies that check for security best practices in your IaC code. It uses Rego, a policy-as-code language, to perform these checks.

Terrascan has extensive support for AWS and other major cloud service providers, making it a highly versatile tool for cloud deployments.

What Policies are Checked by Terrascan?

There are several crucial policies that Terrascan checks. Let's focus on the most important ones related to AWS:

1. S3 Bucket Policies: It ensures that S3 buckets are not publicly accessible, which is crucial to preventing unauthorized access to stored data.
2. IAM Policies: It checks that minimal access policies are in place for your IAM roles, to avoid granting unnecessary permissions to entities.
3. Encryption Policies: It confirms that data at rest and in transit are encrypted, offering a necessary layer of security.
4. Logging and Monitoring Policies: It validates that AWS CloudTrail and CloudWatch are enabled, to monitor and record activities for security analysis.
5. Security Group Rules: It checks for overly permissive inbound and outbound rules in security groups, which could expose your resources to potential attacks.

Integrating Terrascan as a Pre-commit Hook

Having static code analysis as a pre-commit hook is a best practice that can save you from pushing insecure code to the repository. Here are the steps to integrate Terrascan as a pre-commit hook:

Step 1: Install `pre-commit`. You can do so by running the following command:

```
pip install pre-commit
```

Step 2: Create a `.pre-commit-config.yaml` file in the root of your repository and add the following:

```
repos:
  - repo: https://github.com/tenable/terrascan
    rev: v1.8.0  # Use the ref you want to point at
    hooks:
      - id: terrascan
        args: ['-f', 'yaml', '-o', 'results']
```

This configuration specifies that the Terrascan tool will be run before every commit, outputting the results in a YAML file.

Step 3: Install the pre-commit hook. Run the following command:

```
pre-commit install
```

Now, whenever you try to commit changes, Terrascan will automatically scan your Terraform code. If any violations are found, the commit will be blocked, and you'll need to fix the problems before the commit can proceed.

Employing static code analysis with Terrascan in your Terraform code development process can significantly minimize the security risks when deploying resources on AWS. By integrating it as a pre-commit hook, you'll ensure that every line of code committed to your repository adheres to security best practices, thus providing robust security for your cloud infrastructure.

Cutsom policies

Here's how you can create custom policies in Terrascan:

Step 1: Define Conditions: Start by defining the conditions that will trigger a policy violation. These conditions are written in the Open Policy Agent's (OPA) policy-as-code language, Rego. For example, you might create a condition that checks if an AWS S3 bucket is publicly readable.

Step 2: Create a Rule: After defining your conditions, the next step is to create a rule. A rule in OPA is a named collection of conditions. The name of the rule is the policy violation that will be reported by Terrascan when the conditions of the rule are met.

Step 3: Package the Rule: The rule you've created now needs to be part of a package. A package in OPA is akin to a namespace, grouping related rules together. This makes managing your policies easier, especially when dealing with a large number of rules.

Step 4: Test the Policy: Before incorporating your custom policy into Terrascan, you should test it to make sure it works as expected. This can be done locally using OPA's `opa eval` command, which lets you evaluate your policy against sample data to verify its correctness.

Step 5: Load the Policy into Terrascan: Once you're satisfied with your custom policy, it's time to load it into Terrascan. This can be done through the `--policy` path command line argument when you run Terrascan. Alternatively, you can place your custom policies into a directory and set the `POLICY_PATH` environment variable to the directory's path.

By following these steps, you can extend the capabilities of Terrascan with your custom policies, enabling more robust security checks that cater specifically to your organization's requirements.

Some usecases that require custom policies

Here are some scenarios where you might need to create custom policies:

1. Validating the existence of DDoS prevention mechanisms, such as by assessing the request per second (RPS) limit on Kubernetes ingress.
2. Ensuring that newly created resources adhere to appropriate tagging and naming conventions.

In conclusion, utilizing Terrascan's static code analysis for your Terraform AWS deployments, along with the ability to craft custom policies, brings robust and tailored security to your cloud infrastructure. It's an indispensable tool for automating security best practices and ensuring compliance in your infrastructure-as-code development process. As always, Thanks for reading!

Security Terraform DevOps

GSoC Final Report

Author: Vedang Joshi

Published: August 16, 2021 · 4 min read

The second evaluation for the Google Summer of Code 2021 has begun. This has been a wonderful experience so far. Fortunately, we have completed all the work promised in the previous blog post. Here are some quick links to the PRs, repos, and blogs related to my project.

- DRG
- Drogue IoT
- GSoC Proposal

PRs

- All commits
- #81 Added primary trust anchor support
- #86 Added --cert flag
- #87 Added ca key verification
- #89 Added RSA key generation and algo parameter
- #90 Added custom key support
- #91 Added test cases trust.rs
- RFC

Blog

- GSoC Phase 1
- GSOC, Drogue IoT & the Community

Implementation

We can authenticate our devices to Drogue IoT using either passwords or X.509 certificates. The latter method is better as it is scalable and more secure. To know more about the implementation kindly refer to the phase 1 blog.

In the past few weeks, I added support for generating RSA keys using the RSA crate. The private keys can be generated from any of the 3 algorithms - RSA, ECDSA, EdDSA. To specify an algorithm you would need to specify the `--algo` parameter.

```
drg trust create <app_name> --algo RSA
```

To save you the trouble of specifying the parameter every time, we leveraged the context functionality in drg. Thus you can specify a default signature algorithm for a context.

```
drg context set-default-algo RSA
```

Other than generating keys and certificates, users can also use existing keys. The only constraint here is it should strictly be in a PKCS8 format. To do this user can specify the `--key-input` parameter. To better understand the required format you can check out Ring's documentation.

Assuming you are using OpenSSL to generate

- RSA keys

```
openssl genrsa -out key.pem 2048
```

- ECDSA keys

```
openssl ecparam -name prime256v1 -genkey -noout -out key.pem
```

The only additional step is

```
openssl pkcs8 -topk8 -nocrypt -outform der -in key.pem > private.pk8
```

Now, you can use the `private.pk8` file to sign app or device certificates using

```
drg trust create <app_name> --key-input <key_file>
```

In the intial proposal, there were talks about creating a "key-store" to securely save all these private keys, but as we progressed the decision about where to store the keys was left to the user. That's when I thought, it would be a good idea to demonstrate how to store keys in a solution like LastPass and use it in drg. Frankly, it is just some 'terminal trickery' :).

Assuming you have LastPass cli installed, and you are logged in DRG and LastPassCli. To store app private key

```
printf "Private Key: %s\n" "$(drg trust create --app app_name)" | lpass
add --non-interactive --sync=now "app_name" --note-type=ssh-key
```

Now, to use that key to sign device cert+key

```
drg trust enroll d1 --ca-key <(lpass show app_name --field="Private Key")
--app app_name
```

In addition to this, we also tried writing unit test cases for various functions used by the `trust` subcommand. These test cases provide a ~92% code coverage in the `trust.rs` file. Required changes were also made to the ci file to run the test cases on each commit.

State of the project

The majority of functionality related to the `Trust-anchor management` has been implemented and merged to the main branch of DRG.

Challenges and learning

- The primary challenge in this project was acquiring knowledge, I was extremely new to cert-based authentication, so I learned many things on the way.
- We had to adjust by introducing a `--cert` flag, due to some device naming limitations, but we plan to overcome that by implementing alias functionality.
- RSA key generation was done using RSA crate. Initially, it was not compatible with the Ring crate we have used so far. This was an obstacle. I reported this on the RSA crate's community channel, they were already aware of this and had solved it on the main branch. Few days after this conversation, they made a new release and we were able to implement RSA key generation in our code.

The learnings I've got throughout this time are unparalleled and they will stay with me for a long time. I shall continue to make more such contributions.

Thank you

GSOC DrogueIoT IoT Github Open source

How to reduce the time it takes to patch vulnerabilities?

Author: Vedang Joshi
Published: October 12, 2023 · 5 min read

"Patch for CVE-2023-12345 that affected py-cool library from version v1.3 to v2.6 was released - v2.7" mentioned the release note of py-cool library on GitHub.

This statement can have two distinct reactions by all the people using py-cool. For a few teams, it happens to be a sigh of relief, as they can now see their CI colored green, while for a few other teams, it could just be a blip in the ocean - one more patch, one more vulnerability. What differentiates both these teams? which one is the better state to be in? and how can we go from being the second team to the first? Let's explore this.

A lot of what I am going to share in this article is just the elaboration of the following 2 words: PROACTIVE MAINTAINCE. Yes, that's it. If you already understand where we are headed, you can go back and sip your sangria.

Using third-party libraries is essential to software development. The important point is to always consider the tradeoff between the functionality the library offers vs. its maintenance overhead. This is our first key point:

- Choose Dependencies Wisely:
  - **Verify the Source**: Ensure that the source of your dependencies is reputable and reliable. Trustworthy sources are less likely to introduce vulnerabilities or abandon their projects.
  - **Verify Maintainability**: Assess the project's maintainability by checking the frequency of updates and the responsiveness of maintainers to issues and pull requests.
  - **Consider Overhead**: Every dependency adds complexity and potential maintenance overhead. Whenever possible, avoid dependencies that aren't critical to your project's functionality.

Once you have decided to go ahead and use the library. You need to constantly listen to what the maintainers have to say.

- Communicate with the Community
  - **Engage with the Community**: Dependencies with active communities are often more reliable. Interact with community members, ask questions, and provide feedback.
  - **Subscribe to Releases**: Stay informed about updates by subscribing to dependency releases. Regular updates can include critical security fixes and new features.
  - **Changelogs**: Pay attention to changelogs to understand what changes have been made in each update. This information can help you assess the impact of updates on your project.
  - **Direct Communication**: Don't hesitate to communicate directly with dependency maintainers. Building a rapport can be invaluable when troubleshooting issues or requesting assistance.

Now that you know who is writing the code you use. You need to stay updated with the new features and patches they are releasing.

- Plan Regular Dependency Reviews
  - **Set a Schedule**: Establish a routine for dependency reviews, aiming to conduct them at least every two weeks. Consistency is key to managing dependencies effectively.
  - **Update Direct Dependencies**: Prioritize updating direct dependencies as they directly impact your project's functionality. Pay attention to deprecation warnings or compatibility issues.
  - **Address Backlogs**: Don't let dependency updates accumulate. Tackling a backlog can be time-consuming and compounding, leading to increased risk.

While these are libraries that you introduced intentionally, there are tons of libraries working in the background. Docker, as we all know, does a great job and packaging these all the way to the kernel level. This ability of docker can be of great help, it allows us to use just what is required.

- Reduce Clutter in Docker Images
  - **Minimal Golden Images**: Start with minimal golden images as the base for your containers. These lightweight images minimize the attack surface.
  - **Trim Dependencies**: Modify your golden images to include only the dependencies required by your application. Reducing the number of dependencies can enhance security and resource efficiency. For example, use `--no-install-recommends` with `apt-get` in Debian-based images to avoid installing unnecessary packages.

While you do your best, there would still be a good chance that you have to keep support for outdated software. For example - You may be supporting a new version of a database but other microservices are still on the old versions, thus you can't update. The best thing to do here is to mark all the code that would be deprecated and assign alternatives to each.

- Separate Outdated Code from Your Core
  - **Modular Approach**: Maintain outdated or deprecated code as separate modules or packages. This allows you to update or discard them more easily when dependencies require changes.

Finally, testing is an investment you need to make

- End-to-End Tests are Your Best Friends
  - **Test Coverage**: Develop E2E tests that cover critical aspects of your application. These tests act as a safety net to catch issues caused by dependency updates.
  - **Automate Testing**: Automate your E2E tests and integrate them into your continuous integration (CI) pipeline. Automated testing ensures that tests are consistently executed.

In conclusion, We need to understand that software engineering is not a technical problem, it rather is a socio-technical problem. As developers, we can't afford to overlook the social aspect. People leaving the organization, new people joining, dependencies being maintained, dependencies being archived - These are not rare events, we need to make our products resilient to them.

The above steps are elementary and an initiation of good processes. Adhering to them can surely help us obviate vulnerabilities and reduce the time to patch them. Feel free to share what you think about the same, and as always

Thanks for reading!

Security Python DevOps

Access Management Best Practices

Author: Vedang Joshi

Published: January 18, 2023 · 4 min read

We are all aware of the importance of data in today's world. Anything valuable always comes with its security risks. The first step in the direction of protecting data is limiting who has access to it as much as possible. Identity and access management is a framework that helps us make sure that the right people have access to the right data and resources for the right duration.

In an organization, we work with many tools, for example - VMs, Databases, SAAS tools, etc. All these tools have their approach to user management. It is a cumbersome process to manage the users across all the platforms as the organization scales and the number of users increases. Thus having policies and procedures for user creation, audit, and removal helps reduce the security risk.

Improving the onboarding process

Centralizing access requests

The onboarding of each user on all platforms should be done by raising a request to a centralized system. The users should inform the team (that manages the tool) about why this particular access is required. This helps in tracking and compliance.

Enforcing MFA

The tools should be configured to let users access the resources only after an MFA device is used to authenticate.

Using emails as usernames

Emails are always unique in an organization. They should be used in place of generic usernames wherever possible. This practice can help us automate exit management.

For places where usernames can't be emails or for emails themselves, an organization should come up with a single pattern that is used to generate the username. This pattern must depend on something unique to the user like her employee ID. Example - <first_name>.<employee_id> etc.

Using SSO

SSO should be used wherever possible. SSO should also be configured to allow only members approved by the admin of the tool to access the resource.

Least Privilege Possible

This is considered a principal in the world of IAM. It refers to the practice of assigning minimum levels of access to users, only essential for their roles and duties. Every tool comes with RBAC - Role-based access control. These are policies that define what privileges a role has, a user is then given the required set of roles.

Enforcing strong password policies

Users should only be allowed to create their accounts when they have a strong password. A strong password is at least 8 characters long, uses special characters, and avoids obvious or guessable phrases.

Admin must apply password expiration policies.

Limiting programmatic access

Never commit credentials to Git

Credentials used to interact with the APIs of any tool should be stored in a secured vault or environment variables and should never be hard-coded or committed to version control. CI integrations can help enforce this.

Timely rotating keys

Access keys should be rotated timely and must be different for all environments in a single tenant setup.

Taking ownership

Each access key must be mapped to the human user who uses it.

Automate offboarding process

Daily user audits

Whenever a user leaves the organization, his email can be searched for and deleted from all the tools he has access. This process can be automated in tools that provide SSO, or it can be done through automation scripts.

Transferring ownership of access keys

After the offboarding of the user, if the keys are preserved they must be rotated and a new owner should be assigned to them.

Regular access audits and cleanup

The security team performs regular cleanup and audit activities to identify misconfigurations like:

- Over Privileged accounts
- Credential sharing
- Stale accounts
- Policy validation

Tips for users

- Users should inform the Admin if they no longer need particular access and get it deleted.
- Users should never share their credentials.
- Users should never use the same password for more than one tool.
- Users should never store their credentials on stick notes and slack channels etc

In conclusion, IAM revolves around people, each digital identity must have an owner - a human user. While teams benefit from using SSO for the majority of the places, it always helps to handle matters of access management with keeping the involved risk in mind.

Thank you

Security Access Management DevOps

MySQL on Statefulsets

Author: Vedang Joshi

Published: April 4, 2025 · 3 min read

The following kubernetes documentation and examples are outdated with the recent mysql version
https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

So I took it upon myself to run the latest MySQL version with 2 reader replicas and 1 write replica. In this post, I'll walk you through my approach step-by-step, explaining how I managed to set up a MySQL cluster using Kubernetes StatefulSets. Grab your favorite cup of coffee, and let's dive in!

The Overall Approach

Here's the quick rundown of what I did:

1. ConfigMap Magic:
   I started by creating a ConfigMap that bundles two key scripts:
   - `init-master.sh` : This script configures the master node by setting up replication parameters.
   - `init-slave.sh` : This script is designed for the slave nodes, ensuring they connect and replicate from the master.

   These scripts are vital as they initialize the replication setup for MySQL.
2. StatefulSet Setup:
   The heart of the deployment is a StatefulSet, which I configured to handle both the master and slave roles based on the pod hostname.
   - Init Container:
     Before the main containers kick in, an init container checks the hostname. If it's the first pod (i.e., hostname ends with `-0`), it's designated as the master; otherwise, it's recognized as a slave. It then copies the appropriate script (either `init-master.sh` or `init-slave.sh`) from the ConfigMap.
   - Main & Sidecar Containers:
     - Main Container: Runs MySQL 8.0 as expected.
     - Sidecar Container: Executes the copied script.
       Note: You can't merge the functionality of an init container with a sidecar because the sidecar has to run after MySQL is fully up and running. This separation ensures the scripts execute at the appropriate time for proper replication setup.
3. Service Configuration:
   Once the StatefulSet is up, I created two kinds of services:
   - Headless Service:
     This service lets you connect directly to any container. Specifically, I use it to connect to the master node via `mysql-svc-0` .
   - Replica Service:
     With a label selector of `replica` , this service is tailored to connect exclusively to the reader nodes.
4. Labeling the Readers:
   The final step was to apply specific labels to the two reader nodes. This labeling makes it easy to differentiate between the master and the replicas when routing read and write traffic..

Putting It All Together

Below is a snippet where you'd copy and paste your YAML files:

```
# -- ConfigMap with init-master.sh and init-slave.sh
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
data:
  master.cnf: |
    [mysqld]
    log-bin=mysql-bin
    binlog-format=ROW
    server-id=1
  slave.cnf: |
    [mysqld]
    server-id=2
  init-master.sh: |
    #!/bin/bash
    set -ex
    echo "Creating replication user..."
    mysql -h 127.0.0.1 -u root -p$(MYSQL_ROOT_PASSWORD) <<-EOF
    CREATE USER IF NOT EXISTS 'replication'@'%'
    IDENTIFIED WITH mysql_native_password BY '$(repl_password)';
    GRANT REPLICATION SLAVE ON *.* TO 'replication'@'%';
    FLUSH PRIVILEGES;
    EOF
  init-slave.sh: |
    #!/bin/bash
    set -ex
    echo "Waiting for master to be ready..."
    until mysql -h 127.0.0.1 -h mysql-0.mysql -u root -p$(MYSQL_ROOT_PASSWORD) -e "SELECT 1"; do
      echo "Master is not ready yet..."
      sleep 3
    done
    echo "Getting master position..."
    MASTER_STATUS=$(mysql -h mysql-0.mysql -u root -p$(MYSQL_ROOT_PASSWORD) \
-e "SHOW MASTER STATUS" --skip-column-names)
    MASTER_LOG_FILE=$(echo $MASTER_STATUS | cut -f 1 -d ' ')
    MASTER_LOG_POS=$(echo $MASTER_STATUS | cut -f 2 -d ' ')
    echo "Stopping replica IO thread if running..."
    mysql -h 127.0.0.1 -u root -p$(MYSQL_ROOT_PASSWORD) \
-e "STOP REPLICA IO_THREAD FOR CHANNEL '';"
    echo "Setting up slave with master log file: $MASTER_LOG_FILE \
and position: $MASTER_LOG_POS"
    mysql -h 127.0.0.1 -u root -p$(MYSQL_ROOT_PASSWORD) <<-EOF
    CHANGE MASTER TO
    MASTER_HOST='mysql-0.mysql',
    MASTER_USER='replication',
    MASTER_PASSWORD='$(repl_password)',
    MASTER_LOG_FILE='$MASTER_LOG_FILE',
    MASTER_LOG_POS=$MASTER_LOG_POS;
    START SLAVE;
    EOF
    mysql -h 127.0.0.1 -u root -p$(MYSQL_ROOT_PASSWORD) <<-EOF
    SET GLOBAL super_read_only = 1
    EOF
    echo "Slave setup complete!"
---

# -- StatefulSet definition for MySQL with init container and sidecar containers
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  serviceName: mysql
  replicas: 3
  template:
    metadata:
      labels:
        app: mysql
    spec:
      initContainers:
      - name: init-mysql
        image: mysql:8.0
        command:
        - bash
        - "-c"
        - |
          set -ex
          # Generate server-id based on ordinal index
          [[ $HOSTNAME =~ -([0-9]+)$ ]] || exit 1
          ordinal=${BASH_REMATCH[1]}
          # Only the first pod (ordinal 0) is master
          if [[ $ordinal -eq 0 ]]; then
            cp /mnt/config-map/master.cnf /etc/mysql/conf.d/
          else
            # Update server-id for slaves
            cp /mnt/config-map/slave.cnf /etc/mysql/conf.d/
            sed -i "s/server-id=2/server-id=$((ordinal + 1))/" \
/etc/mysql/conf.d/slave.cnf
          fi
        volumeMounts:
        - name: conf
          mountPath: /etc/mysql/conf.d
        - name: config-map
          mountPath: /mnt/config-map
      containers:
      - name: mysql
        image: mysql:8.0
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "rootpassword"
        ports:
        - name: mysql
          containerPort: 3306
        volumeMounts:
        - name: data
          mountPath: /var/lib/mysql
        - name: conf
          mountPath: /etc/mysql/conf.d
        - name: config-map
          mountPath: /mnt/config-map
        resources:
          requests:
            cpu: 500m
            memory: 1Gi
        livenessProbe:
          exec:
            command:
            - mysqladmin
            - ping
            - -u
            - root
            - -prootpassword
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
        readinessProbe:
          exec:
            command:
            - mysql
            - -u
            - root
            - -prootpassword
            - -e
            - "SELECT 1"
          initialDelaySeconds: 5
          periodSeconds: 2
          timeoutSeconds: 1
      - name: replication-init
        image: mysql:8.0
        command:
        - bash
        - "-c"
        - |
          set -ex
          # Wait for MySQL to be ready
          until mysqladmin ping -h 127.0.0.1 -u root -p$(MYSQL_ROOT_PASSWORD); do
            echo "Waiting for MySQL to be ready..."
            sleep 2
          done
          # Determine if master or slave based on hostname
          [[ $HOSTNAME =~ -([0-9]+)$ ]] || exit 1
          ordinal=${BASH_REMATCH[1]}
          if [[ $ordinal -eq 0 ]]; then
            echo "Initializing master..."
            # Execute the script content instead of trying to run the file
            bash -c "$(cat /mnt/config-map/init-master.sh)"
          else
            echo "Initializing slave..."
            # Execute the script content instead of trying to run the file
            bash -c "$(cat /mnt/config-map/init-slave.sh)"
          fi
          # Keep container running
          tail -f /dev/null
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "rootpassword"
        volumeMounts:
        - name: config-map
          mountPath: /mnt/config-map
      volumes:
      - name: conf
        emptyDir: {}
      - name: config-map
        configMap:
          name: mysql-config
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 10Gi
---

# -- Service definitions for headless service and replica service
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  ports:
  - port: 3306
    name: mysql
  clusterIP: None
  selector:
    app: mysql
---

apiVersion: v1
kind: Service
metadata:
  name: mysql-read
  labels:
    app: mysql
    app.kubernetes.io/name: mysql
    readonly: "true"
spec:
  ports:
  - name: mysql
    port: 3306
  selector:
    app: mysql
    replica: "true"
---
```

```
# -- Commands for applying labels to reader nodes
kubectl label pods mysql-1 replica=true
kubectl label pods mysql-2 replica=true
```

Final Thoughts

This is a decent approach for using MySQL with StatefulSets. It leverages MySQL's default replication mechanism to create a small MySQL cluster. For a user application, say a Django project, this setup provides database routers that allow you to specify separate configurations for reader and write databases.

Just a quick heads up:
Be mindful that all nodes are in the same Availability Zone (AZ) to avoid incurring extra data transfer charges. While this isn't a production-ready solution, the intention was more to showcase how Kubernetes StatefulSets can be utilized to run MySQL deployments—an idea that's often misunderstood.

Hope this gives you a clearer picture of how you can run MySQL on Kubernetes with a modern twist. Happy coding and clustering!

Kubernetes | Database

Secure coding practices

Author: Vedang Joshi

Published: October 26, 2022 · 5 min read

Hello Everyone! we all have come across tweets like these - Security vulnerabilities being detected and exploited. What causes security vulnerabilities in the first places ? Can it be prevented ? and most importantly, what do we learn from these ?

Security vulnerabilities occur when a program does something it isn't meant to do - A bug. It was there all along, maybe in the code you didn't even know existed. For hackers, it is an opportunity and they would surely take advantage of it. Here, I would discuss some best practices that avoid common kinds of security vulnerabilities.

Common pitfalls

Dynamically executed code

If a program is taking input from the user and executing it without any sanitization, it is like serving the integrity of the application on a silver platter to the bad actors. In the case of python, the use of eval statements is highly discouraged. These types of attacks are commonly addressed as injection attacks. An attack could inject SQL statement, HTML, or Javascript which if executed on the server side gives the attackers full control. Log4j was an example of code injection. XCKD sums this up in a humorous comic.

Developers need to make sure that there is no room for user input to get executed. Here user input can be input from the end user or even from another developer.

SQL Injection can be avoided by using an ORM instead of writing raw queries. Mostly all popular ORMs come with built in sanitization features.

Major Web frameworks like Django and Flask also come with their sanitization functions. This prevents potentially malicious HTML inputs.

Other than this, Static application security testing (SAST) tools can also detect lines of code that execute a user input and flags them. Using a SAST check either on the local system or in the CI pipeline is recommended.

Hardcoding credentials

While building an application, it is a common requirement to fetch some data from an external API. It may be some payment gateway integration, cloud provider, or any other API. Generally, these APIs are authenticated and we need to specify the authentication key in the headers. While programming, developers often hardcode the credentials in the code itself. This becomes a serious issue when this code is pushed to a git repository because now the keys are visible to everybody who has access to the git repo. This problem amplifies when this git repo is public. Which means the keys are exposed to the entire world.

Not only this practice is a blunder in security but it hinders the developer's productivity too. Once the code becomes a part of a container image and is pushed to production, it becomes difficult to track and replace the credentials if needed.

It is quite easy to avoid this

1. Passing the credentials as env variables solves this problem. Developers can create a .env file if the number of variables is large. This .env file should never be checked out to git.
2. Implementing secret scanning in pre-commit hooks. This can make sure no credentials are hardcoded before every commit. Similarly, secrets scanning can be added as a CI stage as well.

Parsing

Developers often need to deal with XML and YAML files. The first step in using any of these files is to parse them. There are a few common attacks through these parsers.

- XML parsing
  Python XML standard library warns the users that using the etree, DOM, xmlrpc is not secure. An attacker can create a DOS-style attack. This attack references a piece of data in the same file and doing it over and over increases the RAM usage exponentially. Thus crashing the server.
  An attacker can also take advantage of an XML parser trying to fetch and parse resources from an external URL. This can be used to circumvent firewalls.
  The use of defuseXML is recommended as a fix.
- YAML parsing
  Using the `yaml.load` method can be used to make system calls, thus leaving you wide open to attack.
  The use of `yaml.safe_load` is recommended as a fix.

General Tips

Updating dependencies

Your python program surely uses one or the other libraries from PyPI. These libraries may contain some security vulnerabilities. If the library is maintained then these security vulnerabilities would be fixed in no time, once it is fixed it is our responsibility to update our dependencies to the proper version. Developers may find things breaking when a version upgrade on a dependency is done, but it is better than having a known vulnerability in your software.

It is recommended that developers subscribe to the GitHub releases of the packages they use in their code and update the dependencies whenever a stable version is released.

Trusting opensource

No doubt, we all love open-source software, but while installing a package from PyPI for your project, we need to always verify that the library is well-maintained. Installing unknown or outdated packages can introduce security risks. Install either the popular libraries only or review the library before installing.

That's all from my side.

Thank You

Security Python DevOps

Building CRD-Xray: An AI Agent for Kubernetes Custom Operators

Author: Vedang Joshi
Published: June 11, 2025 · 5 min read

Github

If you're running Kubernetes seriously, chances are you've built a bunch of custom operators for your organization. These operators work wonders. But I've always felt something was missing and that is context. The kind of context that helps new engineers understand what's happening, or helps me debug when things break. That's where the idea for **CRD-Xray** started.

My goal is simple: collect as much useful data from custom operators as possible, feed it into an AI agent, and use that agent to answer questions like, "What does this CRD do?", "Which controller manages it?", or "Why is it failing?"

The 4 Problems I'm Trying to Solve

To explain what CRD-Xray aims to solve, here are four specific pain points I've faced:

1. No Documentation for Custom Operators

Open-source tools from CNCF are well-documented, and you can find answers on forums and StackOverflow. But when it comes to internal, organization-specific operators, well generally that's not a happy path. There's usually zero documentation or outdated README files that don't help.

2. Debugging is a Nightmare

There's no built-in mapping in Kubernetes between a controller and the CRDs it manages. Tracing logs, understanding how different tools interact, and piecing it all together manually is painful. Imagine if we could debug across all tools and logs in one go, I know it sounds far-fetched, but I wanted to try.

3. MCP Integration

Since I'm already maintaining a database of CRDs, their controllers, logs, and resources, this makes a great base layer of context for future AI agents. By exposing it via MCP (Model Context Provider), I could let even smarter agents use this data later on.

4. CRD Metrics

We have Prometheus exporters for most core components, but custom CRDs often go unmonitored. Having a way to track their versions, usage patterns, and metrics across clusters could be super helpful.

Step 1: Data Collection

To solve any of the above, I had to first collect the right data. I started by building a Kubernetes controller using the **Kopf** framework. It runs inside the cluster and periodically collects:

1. All the CRDs installed on the cluster
2. All events involving those CRDs
3. All the CRs (custom resources) for each CRD
4. All pods — this list gets passed to an LLM to figure out which pod might be the controller for a given CRD
5. Logs from the identified controller pod (fetched every hour for now)

I store metadata like CRD schemas, controller names, and resource names in a **SQLite** database. For everything else like logs, manifests, etc. I use **Qdrant** as a vector database.

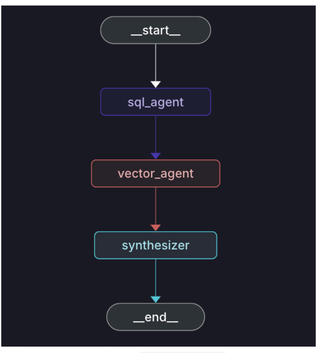
Why Kopf?

Choosing Kopf was an architectural decision. I wanted this controller to live *inside* the cluster. That way, it can continuously monitor and collect data without relying on external scripts or cron jobs. In the future, the idea is that you could just `helm install` this tool, and boom your AI agent is up and running.

Step 2: The LangGraph Agent

Once the controller is up and collecting data, the next step is building an agent that can answer queries. I used **LangGraph** to create a simple state-based AI agent with this flow:

```
user input -> SQL agent -> Vector agent -> Synthesizer -> Final output
```



- **SQL Agent:** Uses LangChain's `SQLDatabaseToolkit` to generate and run SQL queries based on natural language input
- **Vector Agent:** Uses `langchain_qdrant` to search the vector store for relevant logs or manifests
- **Synthesizer:** Combines the results from the SQL and vector search to generate a final answer using Claude (via `langchain_anthropic`)

It's surprisingly effective for questions like:

- "How many CRs exist for `FooBar`?"
- "What does the controller for `FooBar` do?"
- "Show me logs from the controller when resource X failed."

The Agent in Action

The whole point of combining structured (SQL) and unstructured (logs) data is to get deeper, more helpful answers. The agent takes in the query, runs SQL to extract basic info, does a vector search using that context, and then synthesizes a human-readable answer. It's not perfect yet, but it's already saving me time.

What Didn't Work (Yet)

I also played around with **LangGraph Studio** and **LangSmith**. While both are promising, they still need quite a bit of manual setup and tuning. I've shelved that for now, but will revisit once the core system is solid.

What's Next?

There's a lot I want to improve:

- Make the agent more robust and conversational
- Build a proper test suite
- Create a Helm chart for easy deployment
- Test it on more and more custom CRDs in the wild

This is still very much a work in progress, but I'm excited about where it's headed. If you've dealt with the same pain points, or just find this idea interesting, I'd love to hear your thoughts.

Thanks for reading!

AI-Agent Langgraph Kubernetes

Breaking down common attack vectors on AWS

Author: Vedang Joshi
Published: November 11, 2024 · 5 min read

With the proliferation of cloud computing, organizations are increasingly adopting services like AWS to scale their operations. However, the cloud presents unique challenges in securing assets. Notably, public IP addresses and user credentials emerge as some of the most significant attack vectors in cloud environments. This blog delves into the intricacies of these attack vectors, drawing from notable breaches to illustrate risks and explore effective mitigation strategies.

High-Profile AWS Breaches: Lessons Learned

Some major breaches underscore how severe misconfigurations and permissions issues can be within AWS and cloud environments in general. **Companies like Dropbox, Capital One, Twilio, and Mobikwik** have suffered substantial data breaches due to cloud mismanagement. Key factors in these breaches included:

- **Over-privileged IAM credentials:** Attackers often exploit excessive permissions granted to users or applications, allowing them to move laterally within a network or access sensitive data.
- **Misconfigured Firewalls:** Many organizations fail to properly restrict traffic, inadvertently exposing their services to the open internet.
- **Public S3 Buckets:** Misconfigured S3 storage buckets, accessible without authentication, have led to unauthorized access to sensitive data.

These incidents highlight critical areas in cloud security that are often overlooked, reinforcing the need for strict security configurations and continuous monitoring.

Core Attack Vectors in the Cloud: Public IP Addresses and User Credentials

1. Public IP Addresses

Public IPs allow resources to be accessible from anywhere, making them highly vulnerable if not properly secured. Key risks associated with public IPs include:

- **Exposed EC2 Instances:** Misconfigured security groups can leave EC2 instances open to the internet, making them vulnerable to attack.
- **Public S3 Buckets and RDS Instances:** Publicly accessible storage or database instances are critical vulnerabilities, especially if they contain sensitive data.

Severity Assessment:

- **Critical:** EC2 instances with misconfigured security groups are at high risk and should be secured immediately.
- **High to Low:** While public S3 buckets and RDS instances are less commonly exposed, they still represent significant risks. Public access to these services, even in test or POC environments, can lead to unintended data exposure.

2. User Credentials

Compromised credentials are one of the most frequent causes of cloud breaches. In AWS, this includes:

- **Credentials Exposed in Code Repositories:** Hard-coded credentials or secrets accidentally pushed to public repositories (like GitHub) expose systems to direct attacks.
- **Integration with Third-Party Tools:** Tools like New Relic, Databricks, and Snowflake often have API access to AWS. If these tools are compromised, attackers can pivot into AWS resources.
- **Credentials Stored on EC2 Instances or EKS Pods:** If an EC2 instance or containerized application with embedded credentials is compromised, attackers gain access.
- **Weak Credentials in Communication Channels:** Credentials shared in tools like Slack (e.g., Slack tokens) present vulnerabilities if the chat application is breached.

Severity Assessment:

- **Critical:** Credentials leaked in code repositories or used by third-party tools pose immediate risks.
- **High to Low:** Credentials stored within EC2 or shared across Slack channels are also high-risk but less immediately exploitable.

Mitigation Strategies

Mitigating Public IP Risks

1. Technical Solutions
   - **Automated IP Monitoring:** Develop scripts to periodically fetch and update public IPs, ensuring a current inventory.
   - **Public Exposure Flagging:** Flag any IPs associated with publicly exposed services and assess exposure risks.
   - **NMAP Scans on Public IPs:** Regularly perform scans to detect open ports and vulnerabilities.
   - **Automated IP List Refresh:** Configure the script to refresh IP lists automatically, keeping your monitoring up to date.

2. Process Improvements
   - **Approval Requirements for New IPs:** Require security team approval for all new public IP provisions to ensure each IP is justified and secure.

3. Enhanced Security Monitoring
   - **Request Logging:** Log each request to publicly accessible IPs to monitor for unusual access patterns.
   - **Web Application Firewall (WAF):** Implement a WAF to help mitigate attacks on exposed services.
   - **Rate Limiting and DoS Simulation:** Apply rate limits to public IPs and periodically simulate DoS attacks to test the effectiveness of rate limiting.

Mitigating Risks from User Credentials

1. Technical Solutions
   - **Multi-Factor Authentication (MFA):** Enforce MFA for all human users, making it significantly harder for attackers to exploit stolen credentials.
   - **Role-Based Access Control:** Avoid granting IAM or administrative privileges to service accounts, restricting elevated privileges only to human users with clear need.
   - **Credential Rotation:** Rotate IAM service account credentials every 90 days, minimizing the impact of compromised credentials.
   - **Vault Integration for Secret Management:** Use Vault or similar tools for securely managing and rotating credentials, reducing the risk of exposure.

2. Monitoring and Usage
   - **CloudTrail and SIEM Integration:** Enable CloudTrail logging and integrate it with a SIEM tool for real-time monitoring and alerting on anomalous activity.
   - **Activity Monitoring:** Continuously monitor IAM user activity to detect unusual usage patterns that may indicate compromised credentials.

3. Process Improvements
   - **Distinguishing Human and Automated Users:** Clearly separate human and automated user accounts to control permissions more effectively.
   - **Exit Notifications:** Notify IT, Security, and relevant teams when a user exits to prevent any orphaned access.
   - **OAuth-Only Tools:** Procure only tools that support OAuth to enhance security in third-party integrations.

Summary

1. **Primary Attack Vectors:** The most significant attack vectors in cloud environments are public IP addresses and user credentials, as they represent entry points that attackers can exploit directly.
2. **Mitigation through Review and Monitoring:** Regularly reviewing and securing public IPs, combined with robust credential management, can significantly reduce exposure risks.
3. **Layered Security for Credentials:** Keeping credentials updated, using tools like Vault, enforcing MFA, and integrating monitoring tools like CloudTrail create a layered defense strategy that strengthens cloud security.

By understanding and addressing these attack vectors, organizations can fortify their cloud environments against common threats. The cloud provides substantial flexibility and scalability, but with it comes the responsibility of vigilant, continuous security.

Security Cloud

Building CRD-Xray: An AI Agent for Kubernetes Custom Operators

June 11, 2025

MySQL on Statefulsets

April 4, 2025

Breaking down common attack vectors on AWS

November 11, 2024

How to reduce the time it takes to patch vulnerabilities?

October 12, 2023

MongoDB Aggregation for Data Analysis

September 14, 2023

Securing AWS using Terrascan

June 27, 2023

Access Management Best Practices

January 18, 2023

Secure coding practices

October 26, 2022

GSoC Final Report

August 16, 2021

GSoC Phase 1

July 17, 2021

Next →

GSoC Phase 1

Author: Vedang Joshi

Published: July 17, 2021 · 5 min read

In the past month, I worked on a feature in DRG, which enables a user to create and add Trust anchors to the application object, and use them to sign the device certificates.

Implementation

Drogue cloud tries to make it as easy as possible for devices to connect to the cloud. In today's IoT world, a convenient and secure solution to authenticate devices is to use digitally signed certificates. We first need to obtain a public and private key pair which can be generated by algorithms like RSA, DSA, or ECDSA. An X.509 certificate is the envelope that holds a public key along with some other information about the usage and ownership of the certificate and the issuer. In Drogue IoT, we have applications (aka apps), which are containers for devices. The flow for X.509 authentication looks like this:

- We create a new key for an application. This is the root CA key, it needs to be stored securely. Let's name it - app key.
- We create a certificate and sign it using the app key. This is a self-signed certificate, and it would work are the Trust anchor or root CA.
- We store this certificate in the application object on the cloud side.

Now, we have a trust anchor set up and ready to sign our device certificates. To do that:

- We download the Trust anchor certificate from the application object. This helps us add "issuer" information on the device certificate and also verify the private key.
- We take the app key as input.
- Then, we create a CSR i.e certificate signing request and sign it with the app key to create the device certificate.
- We can now export the certificate and private key for the device either to a file or to the terminal.

This concludes the flow, now we can use this device certificate and its key to authenticate and send data to the drogue cloud. This may sound like a lot to do, especially if you are not familiar with tools like OpenSSL, don't worry, DRG is here to rescue. DRG abstracts this entire process in not more than 2 commands.

- `drg trust create` : Creates an app key, Trust anchor, and adds it to the application object.
- `drg trust enroll` : Uses the app key to sign and export device certificate and key.

We have focused on having a pure rust implementation for all the cryptography we have used here. `rcgen` crate is just amazing and makes handling X.509 certificates very easy. Currently, We using the ECDSA algorithm to generate the key pairs, this is the default for the rcgen crate, but we are working on generating and using RSA key pairs as well.

While implementing this, I started by creating a script that uses OpenSSL to generate certificates and authenticate devices. For a few days, I would create a Trust Anchor correctly but couldn't authenticate any device. On sharing this with mentors, we discovered that it is because of a broken dependency but it got fixed pretty quickly. Once, my bash script worked successfully, I knew exactly what I had to do with `rcgen` . The majority of the work on this feature has been done in this PR. I am happy to say it is now merged to the master branch.

Here's a quick demo

To use X.509 authentication, please follow the following steps. Assuming that you are logged into drogue cloud if not use `drg login <drogue-cloud-endpoint>` .

- `drg create app <app_name>` - This commands create an application.
- `drg create device <device_name> --app <app_name> --cert` - This command creates a new device. Notice, the `--cert` flag used here, it formats the device name according to what is needed by Drogue cloud for certificate authentication.
- `drg trust create <app_name> --key-output <app-key-filename.pem>` - This command generates a new key and a certificate, uses the key to sign the cert, finally uploads the certificate to the application object, and writes the key to the file. Optionally, you may specify `--days <no_of_days>` argument, for the validity, by default it is 365.
- `drg trust enroll <device_name> --app <app_name> --ca-key <app_key_filename.pem> --key-output <deivce_key_filename.pem> --out <deivce_cert_filename.pem>` - This commands uses the app key to sign a newly generated device certificate + key. This certificate and key are exported to the respective files.

Now, we can use this device certificate and key to authenticate the devices. An example to do the same over MQTT using the MQTT-CLI (v4.6.2) tool would look like.

```
mqtt pub -h <mqtt_host> -p 443 --cert device-cert.pem --key device-key.pem
```

To Do

This implementation has many things to improve, some of them are:

- RSA Key Pair, This is currently "work in progress".
- Using existing private keys and existing certificates, in that case, DRG works only as a medium to upload them to the cloud.
- Trying to use it with a key store solution like LastPass, could be a good topic for a blog post.
- Improving the implementation.

Thank you

Thanks to @jbtrystram, @ctron and everybody in the community for helping me and as always Thanks for reading.

GSOC DrogueIoT IoT Github Open source

Tag: Machine Learning

Famulus - building a stock predictor

May 9, 2021

MongoDB Aggregation for Data Analysis

Author: Vedang Joshi

Published: September 14, 2023 · 4 min read

Why did MongoDB apply for a job? Because it wanted to find its true aggregate potential!

In the vibrant world of database management, MongoDB has carved a space for itself through its non-relational, document-oriented architecture, which offers a high level of flexibility and scalability. Among its myriad of functionalities, MongoDB's aggregation framework stands tall, offering a potent tool in data analysis. Let's delve deeper to explore this powerful feature and understand its underpinnings.

Understanding Aggregation

Aggregation refers to the operation process used to process data that retrieves the computed results and summarized data. What sets MongoDB's aggregation framework apart is its pipeline mechanism. This mechanism enables developers to break down complex operations into a series of smaller, more manageable stages. Each stage transforms the documents as they pass through the pipeline, using the output of one stage as the input for the next. This approach simplifies queries and enhances efficiency, making the aggregation process akin to a well-oiled assembly line in a manufacturing unit, with each stage adding value to the end product.

Essential Aggregation Functions

Within the MongoDB aggregation pipeline, various functions play pivotal roles in data analysis. Let's familiarize ourselves with some of them:

- $unwind: This function deconstructs an array field from the input documents to output a document for each element. Each output document replaces the array with an element value.
- $addField: As the name suggests, `$addField` adds new fields to documents, providing a pathway to add computed fields, establish conditional fields, or even reset existing field values.
- $group: This function groups documents by specified expression and outputs a document for each distinct grouping. It's immensely useful in data analysis to aggregate information based on particular criteria.

The Philosophy of In-Database Processing

MongoDB promotes the philosophy of carrying out all data processing operations within the database itself. Consequently, pipelines run when we hit refresh on the UI, a practice that has transformed application development dynamics. This approach has facilitated the creation of a minimalist business logic layer that merely acts as a proxy between the database and the UI, steering clear of any hefty operations.

This "in-database processing" ideology not only streamlines application development but also evades the necessity for Python notebooks or ad-hoc script works, heralding a simplified and efficient developmental cycle. The ease of writing these pipelines means that developers can swiftly sail through processes that would otherwise be labor-intensive, fostering a development environment where efficiency meets excellence.

In our exploration, we stumbled upon an invaluable Grafana plugin designed specifically for MongoDB. Despite its lack of ongoing maintenance, the plugin operates seamlessly, facilitating a smooth connection between our MongoDB instance and the Grafana pod orchestrated through docker-compose. The setup process is wonderfully straightforward: add MongoDB as a data source, and you're set. This simplicity extends to creating dynamic and visually appealing dashboards, a task made effortless as one can directly input aggregation pipelines into the data panel on Grafana dashboards. Thus, this plugin not only stands as a testament to the versatility and compatibility between MongoDB and Grafana but also opens up a vista of possibilities in dashboard visualization by leveraging MongoDB as a potent data source.

The Backbone: Implementation at the Database Level

The powerhouse behind MongoDB's aggregation functionality is its implementation at the database level. At this juncture, the operations are close to the metal, leveraging low-level optimizations that bring about faster data retrieval and processing. Index utilization, for instance, ensures that the pipeline stages are executed with optimal efficiency, conserving system resources. By handling complex operations at the database tier, MongoDB ensures that data analysts and developers can wield the full might of the database's analytical prowess without stepping out of the MongoDB environment.

Moreover, the database's robust structure accommodates aggregation commands that foster complex queries and generate extensive reports, establishing a firm ground for data analytics that thrives on precision and efficiency.

Conclusion

In summation, MongoDB's aggregation framework emerges as a formidable tool in the data analysis arena. By envisioning operations as a series of interconnected stages, MongoDB personifies the mantra of "divide and conquer," simplifying complex tasks to facilitate seamless data analytics. Its functions, like `$unwind`, `$addField`, and `$group`, work in harmony to transform raw data into valuable insights, underpinning a data infrastructure that is streamlined, potent, and remarkably easy to work with.

As we continue to forge ahead in the data-centric world, understanding and leveraging MongoDB's aggregation for data analysis will not only make our lives tremendously easier but will also pioneer a pathway towards efficient and insightful data management.

MongoDB Python Data Analysis

GSOC, Drogue IoT & the Community

June 1, 2021

Famulus - building a stock predictor

May 9, 2021

Hello World!

April 14, 2021

← Previous

Tag: Github

GSoC Final Report

August 16, 2021

GSoC Phase 1

July 17, 2021

GSOC, Drogue IoT & the Community

June 1, 2021

Famulus - building a stock predictor

May 9, 2021

Tag: Hackthon Project

Famulus - building a stock predictor

May 9, 2021

Tag: Python

How to reduce the time it takes to patch vulnerabilities?

October 12, 2023

MongoDB Aggregation for Data Analysis

September 14, 2023

Secure coding practices

October 26, 2022

Famulus - building a stock predictor

May 9, 2021

Tag: Kubernetes

Building CRD-Xray: An AI Agent for Kubernetes Custom Operators

June 11, 2025

MySQL on Statefulsets

April 4, 2025

Tag: DevOps

How to reduce the time it takes to patch vulnerabilities?

October 12, 2023

Securing AWS using Terrascan

June 27, 2023

Access Management Best Practices

January 18, 2023

Secure coding practices

October 26, 2022

Tag: AI-Agent

Building CRD-Xray: An AI Agent for Kubernetes Custom Operators

June 11, 2025

Tag: Langgraph

Building CRD-Xray: An AI Agent for Kubernetes Custom Operators

June 11, 2025

Tag: Terraform

Securing AWS using Terrascan

June 27, 2023

Tag: Open source

GSoC Final Report

August 16, 2021

GSoC Phase 1

July 17, 2021

GSOC, Drogue IoT & the Community

June 1, 2021

Tag: IoT

GSoC Final Report

August 16, 2021

GSoC Phase 1

July 17, 2021

GSOC, Drogue IoT & the Community

June 1, 2021

Tag: DrogueIoT

GSoC Final Report

August 16, 2021

GSoC Phase 1

July 17, 2021

GSOC, Drogue IoT & the Community

June 1, 2021

Tag: Security

Breaking down common attack vectors on AWS

November 11, 2024

How to reduce the time it takes to patch vulnerabilities?

October 12, 2023

Securing AWS using Terrascan

June 27, 2023

Access Management Best Practices

January 18, 2023

Secure coding practices

October 26, 2022

Tag: Cloud

Breaking down common attack vectors on AWS

November 11, 2024

GSOC, Drogue IoT & the Community

Author: Vedang Joshi
Published: June 1, 2021 · 4 min read

This year, I got an opportunity to work as a Google summer of code student at Drogue IoT, which is a part of the JBoss community (Red Hat or IBM).

Understanding the project

While looking for organizations on the GSOC website, I came across the idea list of the JBoss community. It had only 3 ideas all from an organization named Drogue IoT. I read those, but barely comprehended any, but I was curious so I went to their Github, blog, and matrix channel. To help myself understand what their project is, I started reading some of their earliest blogs. I read that, and one line just clicked me, which was

> "Plus, given it's a known problem domain with at least one known solution to me, I can concentrate on the implementation, not on the specification and design."

So, It's okay to do everything by hand when we have one device. But we can't do this at scale. There needs to be a system that sits in between devices and applications, to handle everything, from configuring devices to sending them commands. This is what Drogue Cloud does,

> Device <-> Drogue Cloud <-> Applications.

I loved this idea. Something more fascinating was the tech stack - Rust, Knative, Kafka, etc. I introduced myself on the community channel. I worked to understand each component of the tech stack. I started to play around and learn Rust, the best resource to do so is undoubtedly The Rust Book. Everything was new to me, so I was extremely happy to get to know so much.

Drogue IoT also focuses on the device side of things. Drogue device brings reusable components to the embedded side, this is done by an async actor framework written in Rust. I don't understand much of it but I want to learn more and contribute to it at some point, maybe after GSOC. ;-)

Community bonding and contributions

When I came across this organization around mid-January, I was interested in a "Hey Rodney" project, which is based on using voice commands to control Kubernetes. I researched about it but soon my interest faded. I came across a repository named DRG. It is a command-line client to communicate with Drogue Cloud API, which helps us manage devices and apps. I discovered some issues which I could solve, so I started right away.

I love the community and the people at the organization. Everyone is immensely experienced, and an inspiration. I think it is a privilege to get an opportunity to work with such amazing people, that too sitting at home.

I contributed to some patches in DRG, working with Jean-Baptiste Trystram. He was very supportive and guided me with my PRs. I think this experience is larger and better than any internship I may have got, and this is the beauty of Open-source.

I look forward to contributing and giving my best in the GSOC period and later too.

My project

I would write a separate blog entirely focused on my project, but here is a little glimpse of it. My project is about facilitating a DRG user with the ability to create X.509 certificates for a device, and a Trust Anchor (root CA) for the apps.

> Apps: Applications are isolated containers for a collection of devices.

> Devices: Devices ideally map to a physical device, and are owned by exactly one application.

Now, these devices can be authenticated using Username and password combinations, but it is not a good idea to do this at scale. The solution lies in the trust anchor flow. An app (which owns the device) has its root certificate and a private key. This key is used to sign the certificate flashed on the device. Whenever the device connects, it presents its certificate which is cryptographically verified with the App's public key.

This helps establish a One-to-Many mapping and provides a secure authentication mechanism for any number of devices.

Drogue cloud already has the support for X.509 certs. DRG can abstract away many parts of this process. So, this is what we have to do.

More blog to come on this topic, and as always

Thanks for reading.

GSOC DrogueIoT IoT Github Open source

Tag: MongoDB

MongoDB Aggregation for Data Analysis

September 14, 2023

Tag: Access Management

Access Management Best Practices

January 18, 2023

Tag: Database

MySQL on Statefulsets

April 4, 2025

Tag: Data Analysis

MongoDB Aggregation for Data Analysis

September 14, 2023

Hello World!

Author: Vedang Joshi

Published: April 14, 2021 · 4 min read

It was a summer afternoon, back in 2014. I was restless, pacing around the hallway of the school corridor, I looked at the clock and then at my friend - Aniruddha. The prize distribution ceremony was about to begin, we rushed our way downstairs, looking for seats. The ceremony had started, I was nervous, the names of the winners were called out one after the other. Now it was time for our “Theme”, and suddenly Aniruddha’s name was called, he secured 3rd position, I was happy for him. One more name who secured 2nd position was called which was not me. I could hear my heart beating now. It was a moment longer than usual, Anirudhda returned after collecting his prize and sat beside me, I had my eyes closed, my leg shaking.

> “Vedang Joshi for Automation of warehouse systems” the anchor called.

Suddenly, all anxiety vanished. I had won this state-level science exhibition, all the talent and sweat I put into my project has fructified now. I shook hands with Aniruddha and went to collect my prize.

This was the first time I had achieved something out of my comfort zone. This project is not on my resume but in retrospect, this incident has always encouraged me to work hard and choose challenging tasks.

Let’s get to technology now, I would tell you what was the idea of this project.

Automation of Warehouse

The idea was pretty simple: We paste RF-ID tags on each product stored in a warehouse. A robot equipped with an RF-ID reader would scan that tag, pick up that product and hand it over to us. Thus, we get a completely automatic warehouse.

I started with a toy JCB and connected it to a motor driver (L293D) and then to an Arduino UNO. I played out with this setup for a while. I now had to control this over the internet (or local wifi for the demo). I plugged in an ESP-01, and using AT commands, initiated a local server. Now, I should be able to send commands from chrome to this server, but unfortunately, it didn’t work. I debugged it and found ESP’s server turned down the moment I requested it from my browser, good people on the Arduino forum told me it was because of the low current from Arduino.

I bought a simple 3.3v regulator and plugged the ESP directly into a battery, it still didn't work. I tried a few more things before I gave up on ESP. 😀 I switched to the HC-05 Bluetooth module and it was pretty awesome, I found an app on the play store - Bluetooth terminal, it was just a serial monitor but over Bluetooth. It’s time for the RF-id sensor now, I bought a few plastic boxes and pasted RF-ids on them, these would resemble the products in the warehouse. Soon, I realized I was using the RF-tags working on a different frequency than the reader (MFRC522). I had to change that.

Now, somehow with tons of patchwork, everything seemed to be working. It’s time to learn JAVA and create an Android app. The purpose of the app was pretty simple, you see a list of products, you choose any product that interests you, and click on “Order”. This sends a signal to the toy JCB, which scans and looks for the proper RD-id, picks up the right product, and gets it to you. I had no idea of what databases are, so for the 10 products I had, I created a long chain of if-else statements.

Now the main challenge was to send a signal to the Arduino which required knowledge of Bluetooth, which was obviously out of my league. So, I went back to my favorite app - Bluetooth terminal extracted its apk, unboxed it to dex files, and converted it to dex2jar. Thus reverse engineering my way, I was able to simply copy and paste the code in my app. After hours of pain, everything was working and I was ready for the demo.

Hopefully, this was just the beginning of my computer science journey.

Arduino ESP other

Tag: GSOC

GSoC Final Report

August 16, 2021

GSoC Phase 1

July 17, 2021

GSOC, Drogue IoT & the Community

June 1, 2021

Tag: Arduino

Hello World!

April 14, 2021

Tag: other

Hello World!

April 14, 2021

Tag: ESP

Hello World!

April 14, 2021