Limehouse DevSecOps

# What is Infrastructure as Code

Infrastructure as code is a way to provision and configure your computer infrastructure, including servers, networks and databases as well as other resources. Traditional ways were to deploy infrastructure by hand one step or one resource at a time, or to write a code blueprint that explains how to build it. IaC is like writing that blueprint to build that infrastructure.

This means that you have something that is repeatable and will always deploy exactly how you specified it. The key benefits are that you can

* Place under version control just like other source code
* reuse your infrastructure across multiple environments such as development, test and production
* deploy and destroy rapidly to save costs on infrastructure especially in a cloud development environment
* Ensure all environments will be configured equally across all environments
* Use automation to deploy your environment
* Allow multiple members to work on the same setup regardless of environment

There are different type of IAC, some are imperative such as Ansible, others are declarative such as Cloudformation or Terraform. In general declarative is what many experts prefer as you just declare your end state and the system and providers work in the background to provide it for you.

Personally I am an expert mainly in Terraform having coming from Cloudformation background, I like it because rather than being locked into one cloud provider you can use the same scripting language HCL on multiple platform and even on resources within the AWS platfrom such as Kubernetes using the Kubernetes provider. Personally I prefer to use since it stops users from building incredible snowflake servers which are built by experts but can never be reproduced as developers make very bad documenters. IAC is also good for documentation.

# Observability

In the context of microservices, observation refers to the process of

* monitoring the performance of the microservices
* analyzing the behavior of individual microservices,
* examining the interactions between different microservices

The goal is to observe and understand the system's performance, behavior, and interactions in real-time, allowing for fast detection of issues and efficient troubleshooting.

We are essentially interested in metrics

* At the service level - CPU, Usage, memory usage, request-response times and throughput
* At the Instance level – Number of instances running, CPU per instance , memory per instance
* At the Network level – Request volumes, latency metrics , loos of packets , network throughput
* At the UI level or interaction level – HTTP requests , API calls etc

The challenges we face include

* As microservices become **scattered** over systems geographically or system wise it can make it difficult to monitor
* Growing **complexity** – with the use of different technologies monitoring in a single way can be difficult
* **Service discovery** - In a distributed environment, it becomes challenging to discover and communicate with individual microservices.
* **Network latency** – can have a big impact on communication between microservices

We can use tools such as

* Open Tracing or Jaeger to analyse distributed transactions
* Or implement a service registry like etcd in Kubernetes to help discovery maybe use a tool like Consul
* Platforms such as Grafana , Prometheus or datadog/New relic to centralise montiroing
* Kubernetes is an ideal orchestrator tool and integrates well with many logging tools

Finally we can analyse logfiles there are many tools to dissect and make nice charts and graphs and help identify failures.

# AWS Security

Is a big area of concern, I know because I have spent many years doing risk assessment and security plans . However, there are several areas where you can quickly minimize risks of breaches

1. Tightening up IAM policies
   1. Limiting users abilities – minimize what they can do, least privilege we call it
   2. Use more IAM roles instead of hardcoded credentials
   3. Use more temporary credentials
   4. Limit API credentials
   5. Enforce strict credential rotation
   6. Monitor metrics and logs
   7. Ensure MFA is used especially for privileged accounts
2. Use the tools such as
   1. AWS config to track and audit changes on resources
   2. Monitor the cloudtrail, check for potential risks
   3. Use IAM access analyzer to check your users
   4. Enforce Guard duty to protect/monitor against security breaches
3. Ensure all users are identifiable
   1. Via SSO system or
   2. Corporate AD system
   3. No generic accounts

IAM is at the core of the system and many issues can be removed by having strong policies, even resources use IAM like policies on S3 and KMS and other such resources. In away you deny everything and only allow what you want to happen

AWS invests a lot of energy into providing the SaaS for security, so rather than buying expensive tools we should use what they provide and this ensure we are always up to date.

The biggest issue is identifying who has done what, so always ensure you only allow identified person to perform actions, most of these will then be in the Cloudtrail which records the API actions