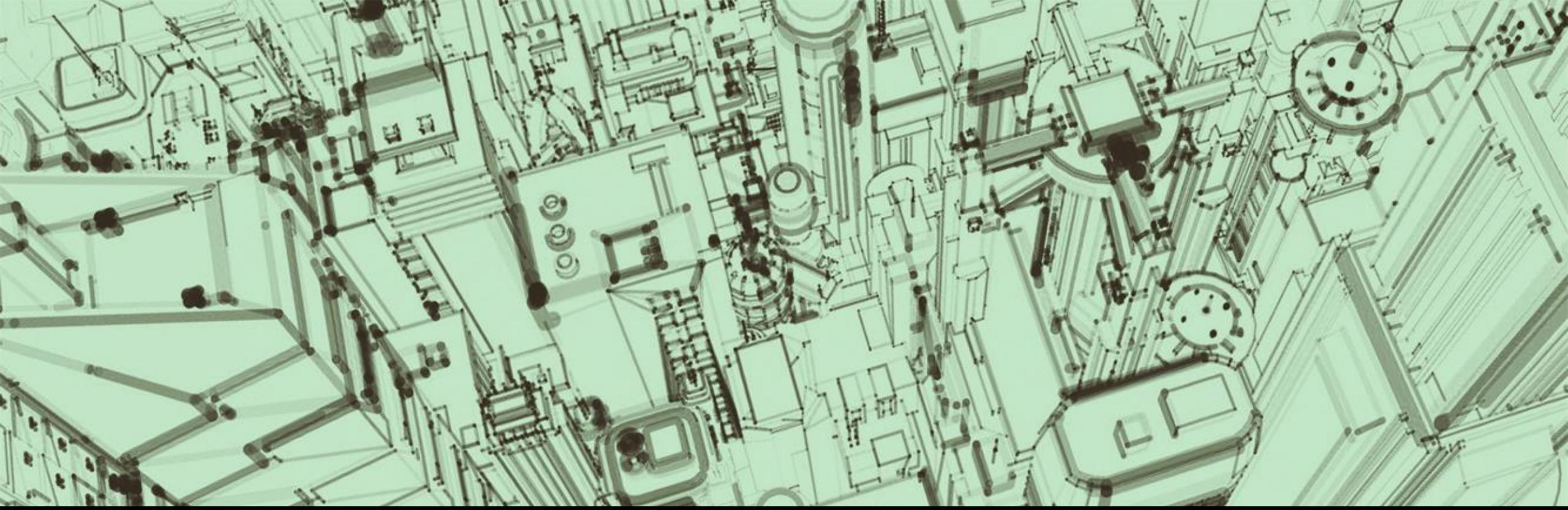


Container and Serverless Security

Cloud Security

What we will be covering today

- Our Journey
- Benefits of Serverless and Containers
- Comparison of Tech Stacks
- Containers: What are Containers and their Use Cases
  - Container Overview
  - Risks
  - Securing Containers
- Serverless Architecture: Use cases
  - Serverless Overview
  - Risks
  - Securing Serverless Architectures
- Auditing Considerations
- DevOps / SecDevOps
  - Devops Overview
  - Example: How to Implement DevOps

Our Journey

- Built and managed Data Centers supporting our product
- Lots of scale up problems, led to adoption of very expensive esoteric parallel processing hardware (Netezza, Terradata, Exadata)
- Transitioned to scale out architectures using open source (Hadoop, HBase) and various virtualization technologies (VMware)
- Cloud Revolution!
  - Clients started to demand Cloud-based Solutions
  - Adoption of Severless and PaaS capabilities (EMR, Lambda)
  - Adoption of Container capabilities (Docker, ECS, EKS, GKE)

Benefits of Serverless and Containers

- Rapid Deployment – Can be up and running in minutes
- Flexible – Supports wide array of use cases with common tech
- Scalable - Can expand and contract quickly
- Cost effective - You only pay for the time you are using the technology
- Portability – Can move workloads easily
- Availability – Inherently
- Use Case – Netflix (Discussion)
- Shifts a datacenter cost model from Capex to Opex
- Fewer things to patch (multiple containers can be sun up from a single image)

Compare Legacy/Container/Serverless

	Legacy Servers	Containers	Serverless
Rapid Deployment	No	Yes	Yes
Flexible	No	Yes	Yes
Easy to deploy DR	No	Yes	Yes
Easily Monitored	Yes	No	No
Easily Scalable	No	Yes	Yes
Native Security	No	No	Yes
Cost Model	Capex	Opex	Opex

What are Containers and their Use Cases?

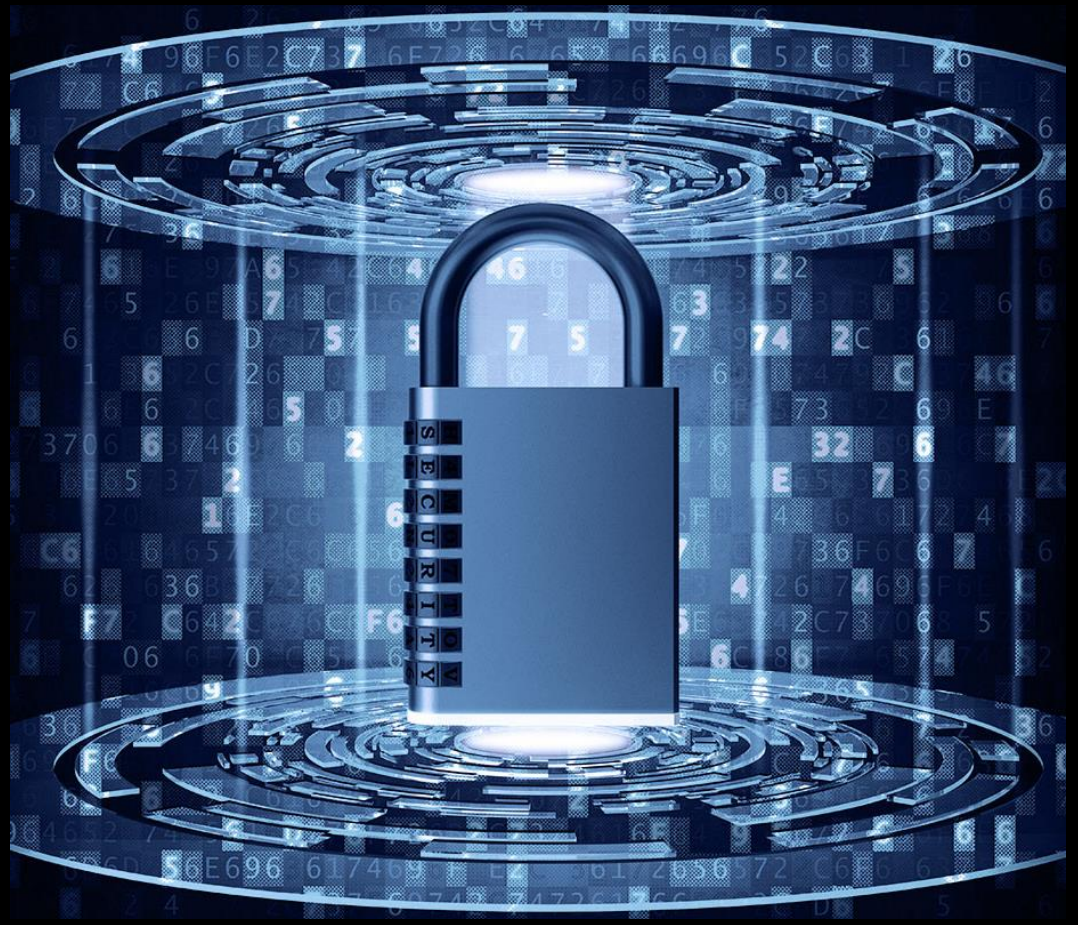
- Containers are a portable and lightweight way to package and run software
- Containers use a containerization engine or **runtime**, such as Docker or LXD to create and manage the container environment.
- Containers are based on **images** or pre-configured templates that define an application environment
- Containers can be easily moved between environments without changing underlying infrastructure
- Containers are widely used in modern application development and deployment, especially in cloud-native and microservices architectures

Containers – Risks

- Vulnerabilities can hide in containers
  - Example
- Ensure you have visibility into your containers, know what is running, know the versions and vulnerability status
- Traditional tooling may not work for containers (Specifically security and monitoring tools)
- Moving to a container model means you need to change your Security model and adopt new tech that is adapted for this technology stack

Securing Containers

- **Secure Container Runtime**
  - Isolate containers in separate networks to reduce the attack surface
  - Only expose ports that are required (Ex: HTTPS, SSH)
  - If using Docker leverage the Image policy plugin
- **Secure the Container Deployment**
  - Only use immutable deployments
  - Ensure the underlying Host OS is secured and patched
  - Leverage an Orchestration platform that provides role-based access control
- **Secure Container Images**
  - Remove all components not in use by the container (Ex: curl, ps, awk). Do not give an attacker leverage
  - Use known good images from known good repositories or create your own image. Places like Docker Hub can provide secure images

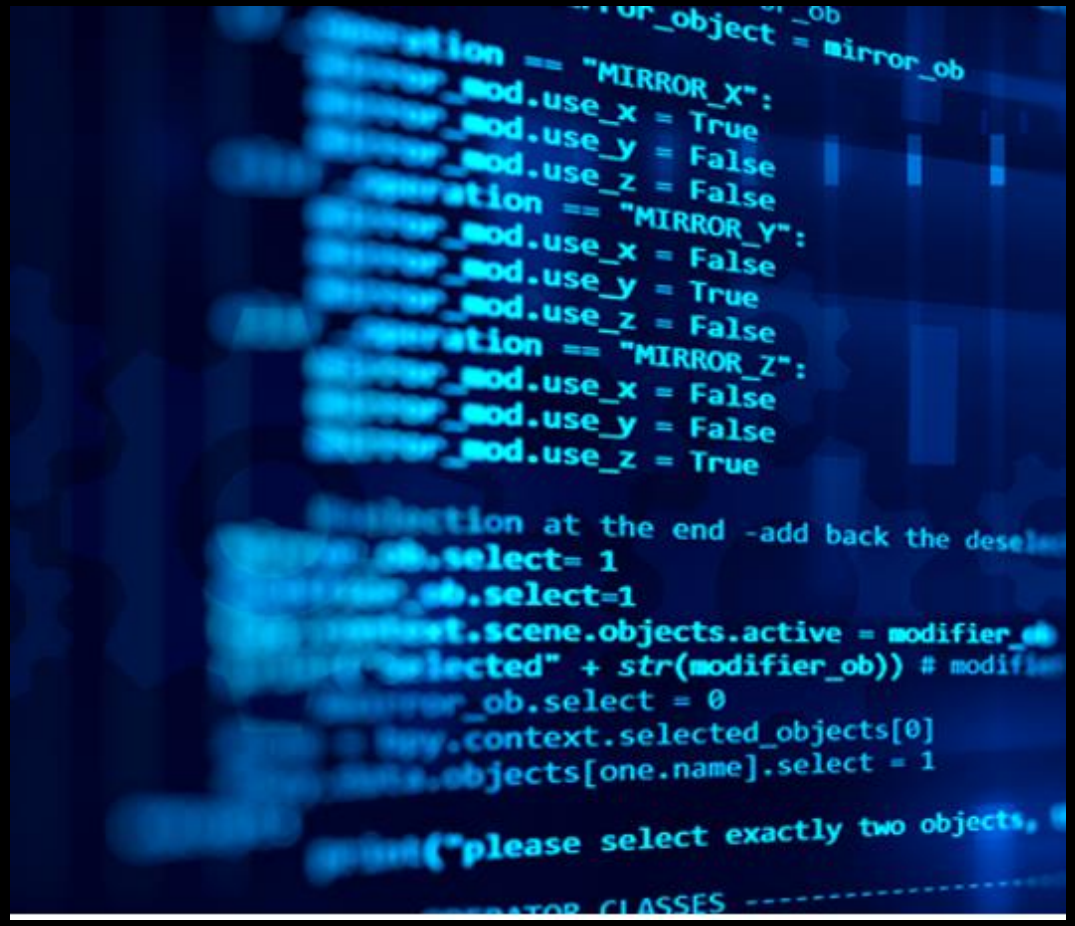


Serverless Architectures and Use Cases

- Technologies for running code, managing data and integrating applications without managing servers
- Highly scalable up and down (even scale to zero)
- Reduces traditional infrastructure management
- Generally lower cost but deployment models, scaling setting and usage has to be monitored

Serverless Risks

- Misconfigurations
  - Ensure that your baseline configuration is checked before moving to production
- Broken Authentication
  - Microservices manage internal auth
- Over Privileged Functions
  - Ensure you leverage a least privilege model
- Injection Attacks
  - Ensure you are SAST and DAST scanning your code
- Logging and Monitoring



Securing Serverless Architectures

- Secure Coding is the baseline for all serverless deployments
  - Ensure that input validation is in place before calls are processed,
  - Secure your API's, leverage an API gateway where possible
- Encrypt all data at Rest and in Transit, ensure you encrypt internal communication channels as well as external
- Log all server functions to an immutable location external to the instance (SIEM)
- Implement strong authentication and leverage things like OAuth, SAML, and MFA
- Create separate Production and Development environments
- Prioritize vulnerability management and react quickly to High and Critical vulnerabilities

Auditing Considerations

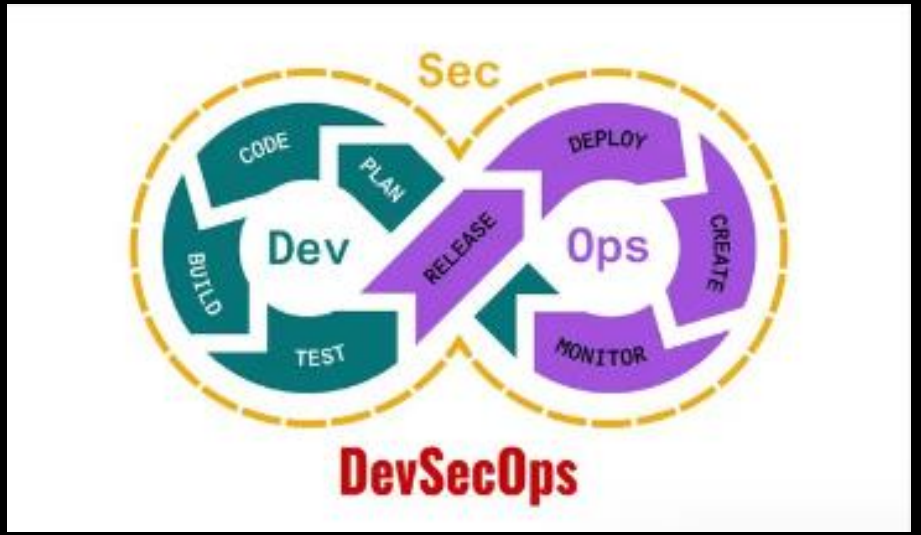
Both Serverless and Containers have Auditing considerations that you need to be aware of

1. Traditional audit tooling may not be appropriate or give you an accurate view of your risks
2. Due to the nature of this tech it is recommended that you perform internal audits quarterly to ensure your deployments are operating as expected and are secured to meet your policy. This can be automated
3. Leverage cloud native tools like AWS inspector and ECR scanning where possible, pull and review reports often

DevOps/DevSecOps Overview

While there is no single definition all can agree on for DevOps, there are some widely accepted thoughts / concepts:

- Think of DevOps as a collaboration combining tools, processes, and philosophies to increase a team's speed at delivering solutions and services.
- DevOps can be viewed as Product Management, Development, IT Operations, and even Information Security all working together and supporting one another.
- Do not think of DevOps as a new role or specific job title, but rather as a new way of working to foster a core framework aimed at achieving the best possible products/solutions/services
- DevOps also standardizes work across all parts of the team by using similar tools and techniques up and down the stack. An example would be that the Ops team would define infrastructure as Code using the same SCM systems the developers are using.
- Many would say that DevOps is a blend of Agile product management and philosophies such as lean thinking (especially when we talk about Continuous Delivery)



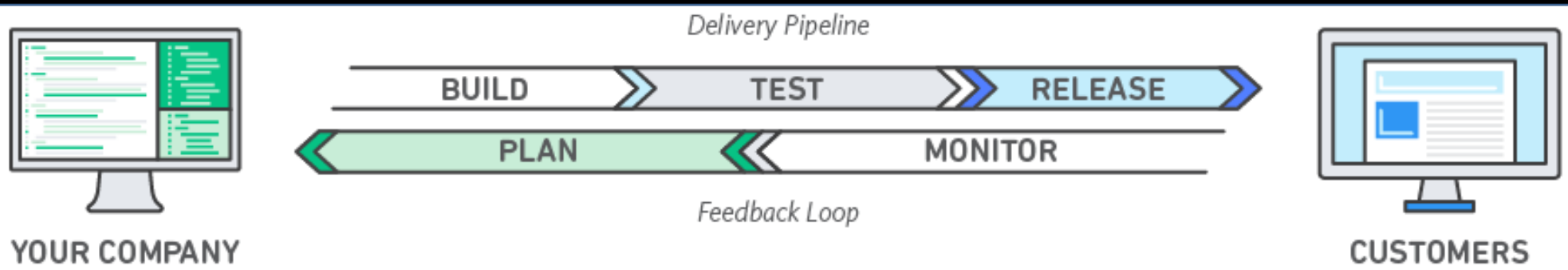
DevOps/DevSecOps Benefits

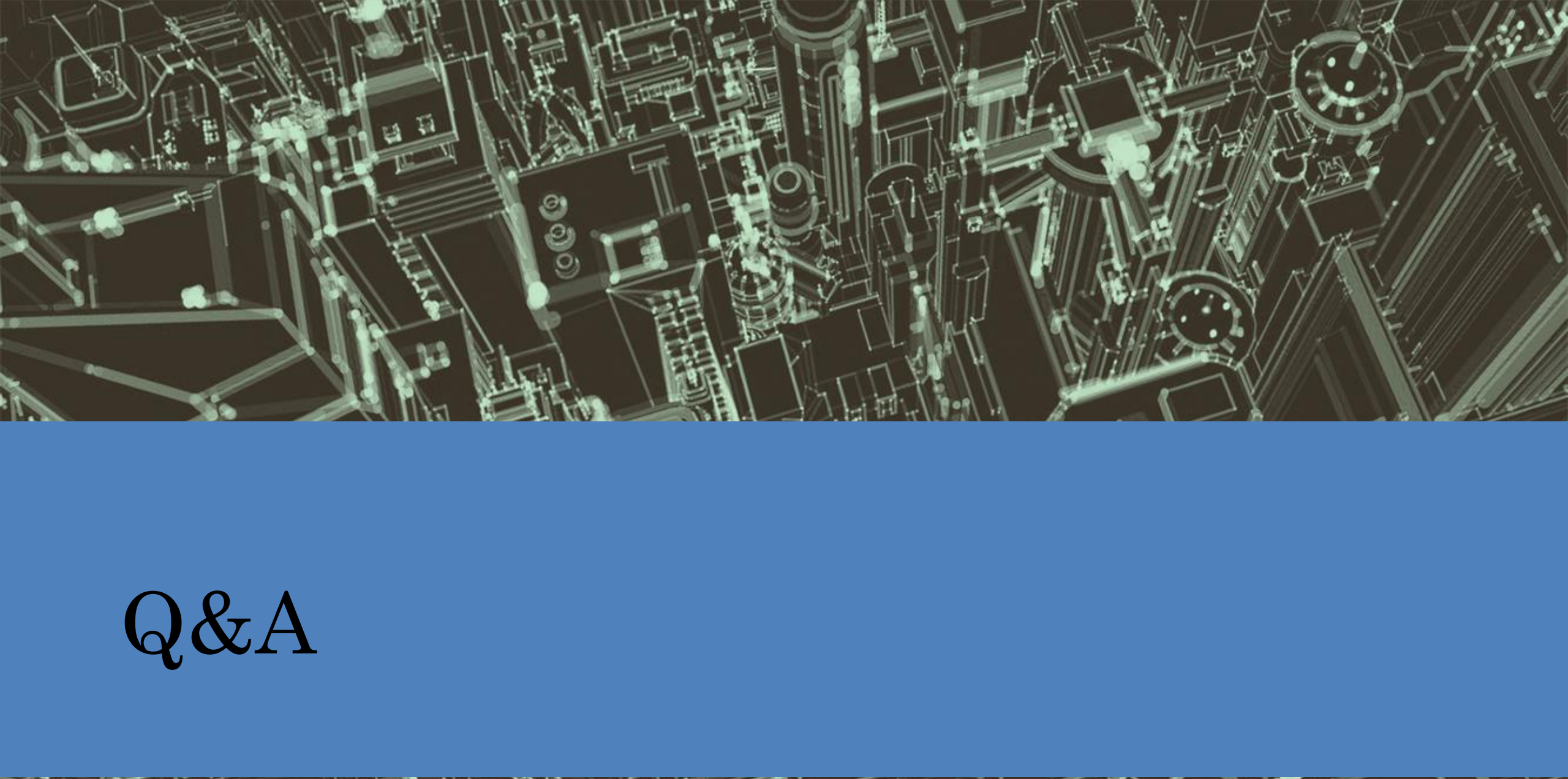
Using the DevOps Culture/Model, groups that are normally siloed are brought together to create a more unified team that works across the lifecycle of the solution or service. This team should include security and quality assurance members to produce a number of benefits:

- Faster Delivery – Features and bug fixes can be tested, validated and deployed quicker than ever
- Reliability – Now that Dev and Ops are collaborating, technical decisions can be solved early allowing for a solution/service to be more reliable. This is also where automation and CI/CD shine as they are the core framework that drives reliability.
- Security - Inclusion of Ops and Security from the beginning allows the overall team to focus on security as the lifecycle of the solution/service grows.
- Collaboration and Planning - Frequent cadence communication creates more collaboration. Heavy use of Agile principles and work management models including Scrum and Kanban are prevalent

How to implement a DevSecOps Model

- **C**ontinuous **I**ntegration is the process of regularly taking committed changes from a central repository and running them through automated builds and test.
  - Developers commit to shared repository
  - Code is scanned for vulnerabilities, common mistakes, duplicative code, etc
  - CI service automatically builds new code
  - CI service potentially tests code
- **C**ontinuous **D**elivery is the process by which all results of the CI process(code changes, artifacts, etc.) are built and delivered to a useable repository. Continuous Deployment will take the built and tested artifacts and deploy them to a testing or production environment.
  - Code is deployed for testing post build
  - Sets stage for further testing (automated or manual)
  - Discover issues before moving into production
- A CI/CD Pipeline automates the entire process





Q&A