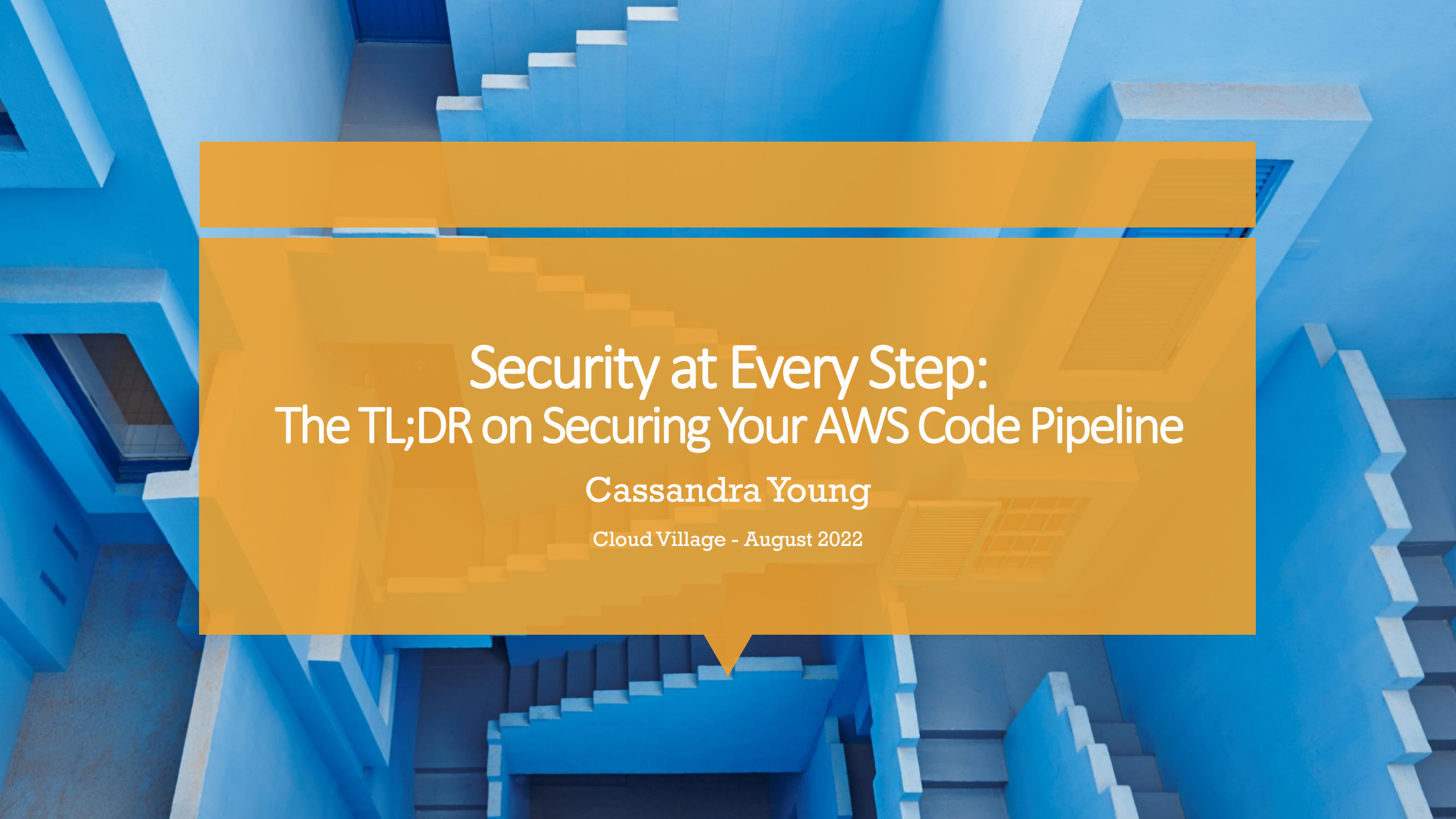


Security at Every Step:
The TL;DR on Securing Your AWS Code Pipeline

Cassandra Young

Cloud Village - August 2022

Cassandra Young (aka muteki)

Senior Scientist, Cloud Security
@ Security Risk Advisors

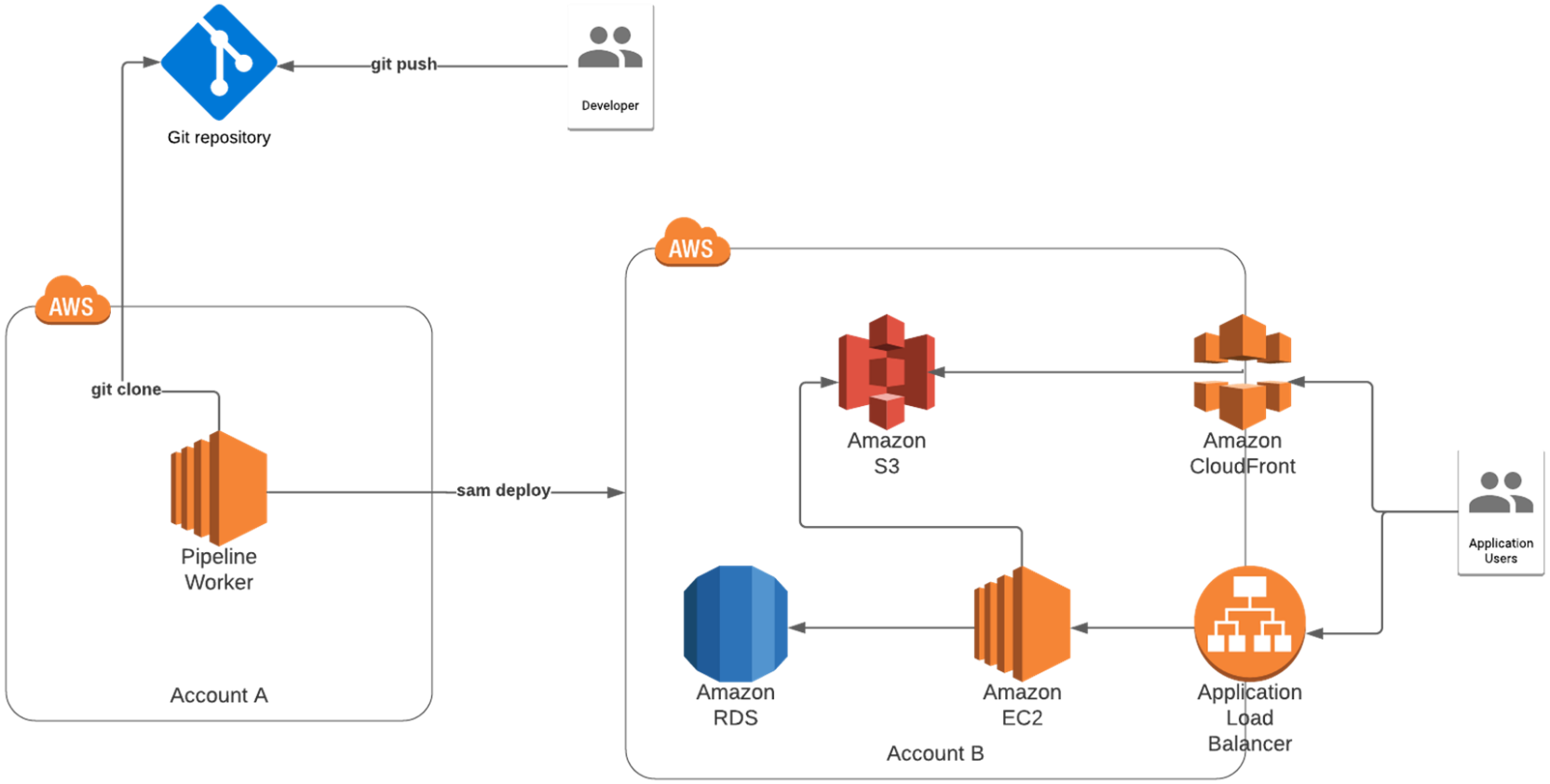
Master's of Computer Science
@ University of Pennsylvania

Director & Meet-a-Mentor Lead
@ Blue Team Village

Lives for international travel,
aurora chasing & scuba diving,
Star Trek & jigsaw puzzles

Cassandra Young (she/her)
Twitter: @muteki_rtw
GitHub: github.com/muteki-apps
LinkedIn: linkedin.com/in/cassandray

Secure What?



From Code to AWS

Secure What? All the Things!

- Credentials used for development/deployment
  - Developer repo access and permissions to dev & prod
  - Pipeline credentials used to authenticate to AWS, which have admin-level privileges to deploy infra, components
- Code deployed to AWS services
  - If unsecured, access to the underlying host may result in privilege escalation
  - Case: Lambda functions, string injection
- VM (EC2) and container (ECS) base images
  - Use trusted, hardened images or validated public images
  - Image scanning is available through Amazon's Elastic Container Repository

Local Code Development

Local Code Development: Environment

- Multifactor authentication must be enforced for all relevant accounts.
- Follow endpoint security best practices, policies and processes
  - Least privilege, endpoint protection, and other best practices protect code from the start
- Bonus: Isolate via VM dev environment
  - Run VM dedicated to development, accessed via IDE
  - Limit necessary admin privileges to within isolated environment instead of host

Local Code Development: Checks & Commits

- Restrict access to repositories
  - Follow least privilege for access to code repositories
  - Control access with SSO and MFA
- Local checks
  - Ensure that no secrets are used within local development environments
  - Pre-commit hooks can be used at the local level
- Signed Git commits
  - Mitigates impersonation as each commit is signed by the developer, validates code authenticity
  - GPG keys used for code signing can be stored in an external service such as AWS Secrets Manager

Continuous Integration

Continuous Integration: Pull Requests

- Pull request validation/analysis:
  - Example: Implement pull request validation and analysis with SonarQube (QualityGate) and/or Checkmarx
- Review:
  - Pull requests should be reviewed and approved by two or more people

Continuous Integration: Static Code Analysis

- Static Code Analysis:
  - Example: SonarQube can be incorporated into both GitHub and Jenkins
  - Scans can be added as a step of the build process
  - Use tools to also scan Infrastructure as Code templates (ie. Checkov)
- Dependency Analysis:
  - Run open-source dependency analysis (ie Checkmarx xOSA) to identify vulnerabilities in 3rd-party libraries
- In AWS, implement Security Standards for Elastic Container Services (ECS) and Elastic Container Registry (ECR) that include scanning images on push.

Continuous Integration: Permissions

- Access to CI/CD services should follow least privilege
- Manage secrets securely within the pipeline
  - Example: Use GitHub encrypted secrets, Hashicorp Vault, AWS Secrets Manager, etc. with scoped access
- Follow best practices for supporting infrastructure
  - Example: Jenkins running on EC2 instance must follow EC2 security best practices, with separate dev/prod deployments
  - Networking, IAM permissions, etc are crucial due to the high access level that the pipeline has

Continuous Delivery/ Deployment

Continuous Delivery/ Deployment

- Wherever possible, scope IAM permissions of accounts deploying to AWS
- Ideally, use roles or native/supported integration instead of provisioning access keys for the pipeline

Continuous Delivery/ Deployment

- Scan running instances
  - Example: Amazon Inspector for EC2 instances
- Any resources exposing OS must run endpoint security tooling

Future State & Takeaways

Future State: Infrastructure as Code

- Infrastructure as Code allows you to store infrastructure design and configuration as a stateful blueprint:
  - Benefits include centralization, version control, and standardization across your AWS environment
  - Provides portability and helps prevent configuration drift
- AWS CloudFormation is an infrastructure automation platform for AWS that deploys AWS resources in a repeatable, testable and auditable manner.

Secure CI/CD Pipeline & Automation: Future State

- Cloud configurations are deployed using Infrastructure as Code (IaC) templates such as Terraform, CloudFormation
- An automated CI/CD pipeline is used to manage and deploy IaC to the AWS environment
- IaC templates are reviewed and approved by a peer/supervisor prior to being deployed
- Third-party code libraries are reviewed for security risks
- The Software Development Lifecycle (SDLC) contains steps to assess software security including design reviews, code reviews, and static/dynamic application security testing
- Source code scanning is built into the CI/CD pipeline and causes deployments to fail when security issues exist
- IaC template scanning is built into the CI/CD pipeline and causes deployments to fail when configurations do not meet security requirements

So Long and Thanks for All the Fish!

Cassandra Young

Twitter: @muteki_rtw

GitHub: github.com/muteki-apps

LinkedIn: linkedin.com/in/cassandray