DevSecOps In The Cloud Is Not Just CI/CD: Embracing Security Automation

Henrik Johansson
Security Specialist Solutions Architect
Amazon Web Services
@henrikjay
import re

= Security Automation
import re

= Security Automation

At Scale
Why/Who/Where/When/What
Why

Goals of DevSecOps
Why - Goals of DevSecOps

- Pace of Innovation...meet Pace of Security Automation
- Elastic and autonomous security validation of instance deployments
- Risk/rating based actions
- Automatic Incident Response Remediation
Why - Goals of DevSecOps

- Pace of Innovation...meet Pace of Security Automation
- Elastic and autonomous security validation of instance deployments
- Risk/rating based actions
- Automatic Incident Response Remediation

Security at scale
Who

Me?
Purpose

Security is a service team, not a blocker

Security is everyone's job

Allow flexibility and freedom

but control the flow and result.
Meet the new security team

Operations

Engineering

Application Security

Compliance
Meet the new security team

- Operations
- Engineering
- Development
- Application Security
- Compliance
Where

3(+) places
1. Security of the CI/CD Pipeline
   - Access roles
   - Hardening build servers/nodes

2. Security in the CI/CD Pipeline
   - Artifact validation
   - Static code analysis
CI/CD for DevOps

1. Commit to Git/master
2. Get / Pull Code
3. Code Config Tests
4. Version Control
5. Push
6. Generate
7. Install Create
8. Package Builder
9. Images
10. Artifacts Repo
11. Deployment templates for infrastructure
12. Test Env
13. Staging Env
14. Prod Env
15. Push Config
16. Install
17. Create
18. Artifact Repo
19. Distribution Builds
20. Run Tests in parallel
21. Send Build Report to Dev
22. Stop everything if build failed
CI/CD for DevSecOps

- Version Control
- CI Server
- Package Builder
- Promote Process
- Audit/Validate
- Continuous Scan
- Checksum
- Images
- Deployment templates for infrastructure
- Code Config Tests
- Get / Pull Code
- Block creds From git
- Scan hook
- Log for audit
- Send Build Report to Security
- Stop everything if audit/validation failed
What about my other stuff?
Where

3. Cloud scale Security
   aka all the other stuff people are really talking about

- Infrastructure as code
  - Split ownership
  - Pre-deploy validation

- Elastic security automation
  - API driven
  - Autoscaling groups – hooks
  - Execution layer scales with targets

- Run time security
  - Tag based targeting
  - Rip-n-replace
  - Continuous pentesting

- Immutable infrastructure
  - Validation and enforcement
  - Integrate with managed services

- ...
When
When

Easy

All the time!
When – Control and Validate

- Pre-event - When possible
  - Store infrastructure in code repository
    - Validate each push (git hooks)
    - Use managed microservices as execution engine
    - Scan cloud infrastructure templates for unwanted/risk valued configurations
    - Validate Container definitions
  - Validate system code early on
    - Find unwanted libraries etc.
  - Force infrastructure changes through templates
  - Block if needed/unsure
When – Control and Validate

- **Post-event - Always**
  - Follow-up on sensitive API’s
    - IAM, Security Groups/Firewall, Encryption keys, Logging, etc.
    - Alert/Inform
  - Use source of truth
    - Locked to execution function (Read Only)
  - Validate source
    - Human or Machine/CICD
  - Decide on remediation
When - Trigger

- **Trigger:**
  - Per change
    - API based
    - Event logs
  - Per day
  - Per framework
    - Overall infrastructure, components and resources
    - One component multiple frameworks
What

Give me some examples
Give me some examples

- Security validation in a elastic infrastructure
  - Implement -> Validate -> Decide
  - Terminate upon failure
- Automatic Incident Response Remediation
  - Autoheal logging
  - Disable offender
- Integrate host-based and cloud-based
  - Immutable infrastructure - Isolate instance
Example – Auto isolation

- Modify
  - /etc/pam.d/sshd

- Execute script upon logon
  - `session  optional  pam_exec.so /path/trigger.sh`

- Trigger cloud based event as marker
  ```bash
  #!/bin/bash
  DATE=$(date)
  aws ec2 --region $REGION create-tags --resources $INSTANCE_ID --tags "Key=Tainted,Value=$DATE"
  ```

- Execute cloud function on marker detection
  - Remove from load balancer/scaling groups (will auto-heal)
  - Block in/outgoing traffic using cloud controls
Example – Auto isolation

- Don’t forget safeguards!
  - How many instances can I isolate before
  - If isolated > x:
    - wake_human()
  - Remember, x could be 0
Example logging

- Detect
  - Cloud logging disabled
- Priority
  - Enable logging
- Forensics
  - Have this happened before
- Countermeasures
  - If num_disabled > x: # x could be zero based on type and user disable_user()
- Alert!
Cool...so I just fix things??

Well...yes...but...
Risks

- Failure is always an option, now at script speed
- *We forgot to tell you...*
- No proper alerting, logging or follow-up on automated events
- You got scripts...they got scripts

- *How do you minimize risk of failed remediation functions?*
Implement remediation framework
The anatomy of remediation

Know
- Continuous / Event based
- Execution constraints
- Will action risk breaking something
- Will change affect cost
- Is there a source of truth

Execute
- Priority Action
- Forensic
- Counter measures
- Alerts
- Log
At the end of the rainbow...

What are we trying to accomplish?
Goals

- Minimize relying on humans
  - Automation doesn’t sleep, eat or need coffee in the morning
- Prevent bad configurations before they are implemented
- Autocorrect/remediate violations where possible
- Daily/instant benchmark validation of infrastructure
  - Validate against industry frameworks
  - Extend to remediation
Your next step

- Look through your infrastructure security runbook
  - What can you automate?
  - How can you validate?

- Example: OSS validation for CIS AWS Foundation Framework
OSS Code to learn from

- **git-secrets** - Prevents you from committing passwords and other sensitive information to a git repository.
- **aws-security-benchmark** - Benchmark scripts mapped against trusted security frameworks.
- **aws-config-rules** - [Node, Python, Java] Repository of sample Custom Rules for AWS Config
- **Netflix/security_monkey** - Monitors policy changes and alerts on insecure configurations in an AWS account.
- **Netflix/edda** - Edda is a Service to track changes in your cloud deployments.
- **ThreatResponse** - Open Source Security Suite for hardening and responding in AWS.
- **CloudSploit** – Capturing things like open security groups, misconfigured VPCs and more.
- **Stelligent/Cfn_nag** – Looks for patterns in CloudFormation templates that may indicate insecure infrastructure.
- **Capitalone/cloud-custodian** - Rules engine for AWS fleet management.
Remember

*It’s actually not who, when, where or what...*

*It’s just how*