DevSecOps – Security and Test Automation

Vibha Dhawan
Rock Sabetto

March 2019

The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions, or viewpoints expressed by the author.

Approved for Public Release; Distribution Unlimited. 19-0769
Purpose

- Clearly describe how Security and Testing can be integrated into a DevSecOps environment without compromising speed, security, or quality

- Provide a baseline of the terminology, methodologies, processes, environments, and automation technologies used in DevSecOps programs
DevSecOps Value Proposition
- Programs can realize significant value by implementing DevSecOps. But, testing and security should not be sacrificed

Shift Left
- Programs must truly shift Security and Test to the left to realize time and cost savings

Agile and DevSecOps go together
- DevSecOps must be fed by Agile software development. Security user stories must be part of each sprint

Automation is key
- Security and test automation can reduce delivery time, improve quality and security, and eliminate human error
Outline

- DevSecOps Background
- Security and Testing Considerations for DevSecOps
- DevSecOps Processes and Technical Considerations
- Platform Deployment Options in DevSecOps
- Conclusion
DevSecOps Background
DevSecOps

- **DevOps** – the union of people, process and tools to achieve building, testing and releasing of software more frequently and reliably.
- DevOps can also be referred to as **DevSecOps** to emphasize the importance of security.
- DevSecOps is not Agile software development. Agile feeds new code / functionality into DevSecOps.

© 2019 The MITRE Corporation. All rights reserved.
What is Agile + DevSecOps

- **No silos** exist between Development, Test, and Operations
  - More teamwork and information sharing
  - Better integration throughout the lifecycle

- **Iterative development and deployment**
  - Design, develop, test, and deploy incremental changes
  - Deploy changes to business users faster

- **Automate as much as possible**
  - Reduce delivery time
  - Improve quality and security
  - Eliminate human error

- **Streamlined, repeatable, routinized** processes
  - Faster delivery cycles – satisfied customers

- **Culture, Practices and Tools** all part of the DevSecOps equations
  - Empowered, trained teams leverage technologies to make it happen
## DevSecOps Value Proposition

<table>
<thead>
<tr>
<th>Traditional Development Challenges</th>
<th>DevSecOps Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ Repetitive, Manual Processes</td>
<td>➢ Automated configuration and software deployment</td>
</tr>
<tr>
<td>➢ Deployment requires Days to Weeks</td>
<td>➢ Deployment takes Minutes</td>
</tr>
<tr>
<td>➢ Not Repeatable – Error prone</td>
<td>➢ Continuous and repeatable process</td>
</tr>
<tr>
<td>➢ Human intervention causes inconsistencies</td>
<td>➢ Consistent</td>
</tr>
<tr>
<td>➢ Frequent downtime</td>
<td>➢ Minimum downtime</td>
</tr>
<tr>
<td>➢ Easier – less technical skill required</td>
<td>➢ Harder – more technical skill needed</td>
</tr>
<tr>
<td>➢ Teams work in silos</td>
<td>➢ Continuous collaboration</td>
</tr>
<tr>
<td>➢ Early security testing not performed on the code</td>
<td>➢ Early, automated security testing during coding</td>
</tr>
</tbody>
</table>
DevSecOps is a union of Culture, Practices and Tools providing continuous delivery to the end user.

**Culture**
- Trust
- Collaboration
- Transparency
- Communication
- Incentive and responsibility alignment
- Governance

**Practices**
- Continuous Integration
- Continuous Testing
- Continuous Delivery
- Continuous Monitoring
- Configuration Management

**Tools**
- Build
- Test
- Deploy
- Monitor
- Security
- Logging

Security Built-in Throughout
DevSecOps applies automation to deliver functionality: Speed, without sacrificing security and test rigor

Traditional Development
- Handoff Dev Builds
- Manual Testing
- Long Deployment Cycle
- Dev/Integration/Prod Environment Impedance
- Manual Configuration

Automation Enablers
- Continuous Integration
- Automated Testing
- Infrastructure as Code
- VM/Container
- Environment Provisioning

DevOps
- Faster Deployment
- Continuous Delivery
- Continuous Deployment
- Repeatable Process
- Consistent/Compliant Environments
DevSecOps Opportunities and Risks

Opportunities
• Speed and Repeatability
  • Automation of testing
  • Automation of security policy enforcement
  • Continuous improvement
• Agility
  • Can be integrated seamlessly with Agile development
  • Removes post-Agile sprint/release chokepoints

Risks
• Security and Test
  • Organizations must fundamentally relook their value proposition.
• Physical Ownership
  • Infrastructure is no longer the model (exceptions include private / community clouds)
• Shared Responsibilities:
  • Security and test responsibilities are now shared by programs, CSOs, and third parties (e.g. 3PAO, CASB)
• System architecture
  • Must address security and test equities

© 2019 The MITRE Corporation. All rights reserved.
Security and Test Considerations for DevSecOps
Security Considerations

- Security should be built into the entire DevSecOps process
  - Agile process that feeds DevSecOps also must be secure
  - Security user stories must be in the backlog

- Embed security throughout software lifecycle to identify vulnerabilities earlier, perform faster fixes therefore reduce cost

- Continuous monitoring to ensure devices, tools, accounts, etc. are continuously discovered and validated
Shift Left: Security and Test Considerations

Security
• Security processes
• Security tools
• Security access (i.e., to DevSecOps environment)
• Security tool visibility (i.e., across the pipeline)
• Security reporting

Test
• Test events
• Test environments
• Test tools
• Test access (i.e., to DevSecOps environment)
• Test data
• Test reporting

Programs must: plan/budget for these items, integrate them into architectures, and write them into RFPs

They won’t magically appear at a program Operational Readiness Review (ORR) 😞
## Test Oversight Influence Areas

<table>
<thead>
<tr>
<th>Type</th>
<th>Artifacts to Influence</th>
<th>Proactive Measures</th>
</tr>
</thead>
</table>
| **Input**  | Acquisition Strategy, SOW Technical Requirements, Program TEMP                          | • **Acquisition.** Develop and communicate the Test Strategy (including security test activities), including: major test events, automation strategy and requirements (e.g., needed tools / standards), required access to Dev/Test environments, plan for test data  
• **Required Test Artifacts.** Ensuring the contract(s) mandates test plans, test cases, test reports, traceability matrices, shared with the govt. Formats, ability to comment are important  
• **Testability.** Requirement for testability of contractor-derived requirements, testable code including security  |
| Pre-Develop| Architecture, Use Cases, Scenarios, System/Functional Requirements                      | • **Interfaces.** Understand and define interfaces, both internal and external systems  
• **Test Environment.** To model / influence the test environment to closely mirror production (and development)  
• **Test Data.** Identify test data sources and ability to access (or emulate); security use cases  |
| During-Develop| Design Specs, Demos, Test Events, Test Cases                                             | • **New Interfaces and Data Sources.** What is the developer changing? Understand how the developer is deriving requirements, interfaces, and functionality  
• **Traceability.** Do the developer’s changes align to the system-level requirements and architecture?  
• **Observe.** Automated testing, live test events / demos  
• **Risk Assessment.** Is the evolving design going to work? What new risks have been introduced?  |
| Output     | Test Reports, Working Software, Data Model(s)                                            | • **Review of Test Outputs.** Increased visibility to stakeholders of metrics around tests (automated test suite vs manual test time, code coverage, etc.)  
• **Recommendations.** How can we reduce risk without killing the benefits of “agility”? TEST AUTOMATION!  |
### Test Event Levels, Challenges, DevSecOps Considerations

<table>
<thead>
<tr>
<th>Test Level</th>
<th>Conducted By</th>
<th>Overseen By</th>
<th>Focus Area</th>
<th>Challenges (bolded words are important)</th>
<th>DevSecOps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit</strong></td>
<td>Contractor</td>
<td>FFRDC / SETA</td>
<td>Code</td>
<td><strong>Automation</strong> Access and Tools</td>
<td>• Automate unit tests</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test Output Access</td>
<td>• Any failed unit tests fail the DevSecOps Pipeline</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test <strong>Traceability</strong></td>
<td>• Development, security, and test work together</td>
</tr>
<tr>
<td><strong>Integration</strong></td>
<td>Contractor</td>
<td>Oversight body: DT&amp;E</td>
<td>Interface / API</td>
<td><strong>Interfaces</strong> / Interface Design</td>
<td>• User Stories are the “requirements” to be tested</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test <strong>Environment</strong></td>
<td>• Each User Story should have corresponding automated tests and acceptance criteria, including Security User Stories</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test <strong>Data</strong> External Systems</td>
<td></td>
</tr>
<tr>
<td><strong>System</strong></td>
<td>Mission Owner</td>
<td>Oversight body: DT&amp;E / OT&amp;E</td>
<td>End-to-End Functionality</td>
<td>Test Environment Test Data External Systems</td>
<td>• Automated user functional tests via tools (e.g., UFT, Selenium, OWASP Zap)</td>
</tr>
<tr>
<td><strong>Acceptance</strong></td>
<td>Operator</td>
<td>Oversight body: OT&amp;E</td>
<td>End-to-End Operations</td>
<td><strong>Not slowing everything down!</strong> Timely Validation, Feedback Loop <strong>Feasibility</strong> (what can actually be changed)</td>
<td>• Automated acceptance tests</td>
</tr>
<tr>
<td><strong>Release</strong></td>
<td>Contractor</td>
<td>Oversight body: OT&amp;E</td>
<td>Deployment</td>
<td>Successful delivery of working software</td>
<td>• Minimize manual system installation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Treat Infrastructure as Code and use deployment automation</td>
</tr>
</tbody>
</table>
# Continuous Delivery Testing – Software Release Approaches

<table>
<thead>
<tr>
<th>Continuous Delivery Test Techniques</th>
<th>Description</th>
</tr>
</thead>
</table>
| • **Blue Green Deployment**        | This requires 2 identical infrastructures to host the application.  
• Green environment runs the current version of the application.  
• Blue environment hosts the new version of the software to be tested.  
• User load is then incrementally shifted from the previously accepted version to the new version.  
  • If there are any issues encountered in the new version, rollback can be done easily to the older accepted version.  
  • This technique increases availability and reduces risk of the application. |
| • **Canary Releases**              | This testing is often automated and includes a limited set of early adopter users.  
These users assist in identifying issues before the application is released to a wider range of users. |
| • **A/B Testing**                 | This method compares two versions of a single webpage or app to determine which one performs better over the other.  
A/B testing is an experiment in which 2 variants of a page are shown to users randomly and then determine which version performs better. |
Additional Test Types – Leveraged as Needed

- Smoke Testing
- Functional Testing
- Security Testing
- Performance Testing
  - Load Testing
  - Stress Testing
  - Spike Testing
- Regression Testing
- Compliance Testing
DevSecOps Processes and Technical Considerations
Agile + DevSecOps Pipeline

**Agile Development**
Developing functionality based upon user needs

- Product Backlog
- 2 weeks Sprint Backlog
- Final Product

**DevSecOps Pipeline**
Automating build, secure / test, deploy, and monitor

- Continuous Integration
- Continuous Delivery
- Security
- Logging and Monitoring

Security: is part of the entire process, from Backlog through Production Operations

© 2019 The MITRE Corporation. All rights reserved.
Agile SCRUM – Team Composition

- **Design Decisions.** Many design choices are made by the Agile team. Programs need to ensure that these decisions are:
  - Consistent with the program architecture
  - Compliant with the security approach
  - Testable
- **Variance.** Organizational composition and roles will vary from program to program
- **Multiple Teams (e.g., Scaled Agile Framework (SAFE)).** Most large programs will have multiple Agile development teams contributing to a common architecture.

**Others Stakeholders:**
- Operators
- External System Owners and Operators
- External Government or Commercial Data Providers / Consumers
- IT Administrators
- Cloud Service Providers
- Security: CASBs, MSSPs
- SETA Contractors
- FFRDCs

**Legend:**
- Team Name / Level – Blue Font
- Government – White Font
- Contractor – Black Font
- Leader – Underline Font
Agile SCRUM – Test and Security

- **Contractor Testing.** Resources are embedded in the SCRUM. Test coverage includes application functionality and security.

- **Government Testing.** Should take place at the end of each sprint, and can be done via a test event or other verification method (e.g., demo, report, etc.). Test coverage should include application functionality and security.

- **Testing Environments.** Government can test in either (or both) Test and Pre-Prod environments.

- **Tailoring Roles.** Government vs. Contractor roles and responsibilities should be adapted for specific program needs.
Continuous Integration (CI)

Agile Scrum Team*

Development Team – Builds and Tests secure functionality

Source Control → Build → Static Code Analysis → Unit Test → Packaging → Artifact Repository

Check in → Feedback

Security

Logging and Monitoring

Feedback loop ensures continuous error correction and vulnerability remediation at each stage in the DevSecOps pipeline

*DoD programs will typically have multiple Agile teams developing in parallel. Security user stories are part of the backlog.
Continuous Delivery (CD)

Continuous Delivery promotes the working software from lower environments to higher environments after security and tests are satisfied.
DevSecOps Tools - Examples

- **Security**
  - Snort, Splunk, Fortify SCA, Vault, OWASP Zap, SonarQube

- **Source Control**
  - GitHub, GitLab, Bitbucket, Artifactory

- **Continuous Integration Tools**
  - Jenkins, Bamboo

- **Testing Tools**
  - JUnit, Selenium, JMeter, TestNG, SoapUI

- **Config/Provisioning Tools**
  - Ansible, Chef, Puppet

- **Logging and Monitoring Tools**
  - ELK (Elasticsearch, Logstash & Kibana) Stack, Splunk

- **Release Orchestration**
  - Kubernetes, OpenShift

- **Containers**
  - Docker, Docker Swarm

**Security**: Tools used throughout the process, regardless of the specific tools being used

**Example**: Snort signatures are applied to all flows that are visible

**Example**: Splunk collects and aggregates all logs that are available throughout the process
CI-CD on Cloud

Application Development (CI/CD)
- Develop / Check In Code
- Compile Code
- Unit Test
- Package
- Deploy Application

Environment Provisioning
- Build Infrastructure Code
- Build and Test VMs and Containers
- Automate
- Deploy

Multiple Environments
- Dev
- Test
- Pre-Prod
- Prod

CSPs (IaaS/PaaS)

© 2019 The MITRE Corporation. All rights reserved.
CI-CD System in Operations

Application Development (CI/CD)
- Develop / Check In Code
- Compile Code
- Unit Test
- Package
- Deploy Application

Environment Provisioning
- Build Infrastructure Code
- Build and Test VMs and Containers
- Automate
- Deploy

Multiple Environments
- Dev
- Test
- Pre-Prod
- Prod

User
- Risk: User Identity Compromised
  Mitigation: Ensure all accounts and devices are continuously validated

- Risk: Privilege Accounts compromised
  Mitigation: Deploy identity governance and PAM tools

- Risk: Misconfiguration of SaaS
  Mitigation: Perform scans to identify and fix potential misconfigurations and identify shadow IT

- Risk: API is not secure
  Mitigation: API security testing and continuous monitoring

- Risk: CAP becomes bottleneck
  Mitigation: Direct connect to CSP (policy dependent), or use VPN with split tunnel + whitelisted IPs

CSPs (IaaS/PaaS)
- Amazon Web Services
- Microsoft Azure

© 2019 The MITRE Corporation. All rights reserved.
Security in DevSecOps

- Embed security throughout software lifecycle to identify vulnerabilities earlier, perform faster fixes therefore reduce cost.
- Different aspects of DevSecOps security in the software lifecycle including tools
  - Static Code Analysis – Scans for vulnerabilities in the code after coding but before unit testing during development (e.g., SonarQube)
  - Configuration Management and Compliance – Know how your application is configured and whether it follows your policies (e.g., Ansible, Chef, Puppet)
  - Dynamic Code Analysis – Scan your code for vulnerabilities in how it performs. Execute unit tests to find errors (e.g., SonarLint, VeraCode)
  - Vulnerability Scanning – Automatically identify known issues in your application for penetration testing (e.g., Nessus)
  - Infrastructure as Code – Ensures the application is deployed securely and without errors in a repeatable manner (e.g., Ansible)
  - Continuous Monitoring – Information on how the application is running, collected and monitored to identify issues and feed future improvements. This is done in production environment. (e.g. Splunk, AppDynamics)
  - Container Security – monitor and protect containers (e.g., BlackDuck)
DevSecOps Security Tools – Examples

<table>
<thead>
<tr>
<th>Security Tool</th>
<th>Description</th>
<th>Focus Area</th>
<th>Test Oversight</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snort</td>
<td>It is a Network intrusion detection and prevention system. Scrutinizes each packet on the network for anomalies and monitors traffic real time.</td>
<td>IDS</td>
<td>OT&amp;E</td>
<td>Yes</td>
</tr>
<tr>
<td>Fortify SCA</td>
<td>Static code analyzer helps to identify security vulnerabilities efficiently in source code during development.</td>
<td>Code Security</td>
<td>DT&amp;E</td>
<td>No</td>
</tr>
<tr>
<td>Gauntlt</td>
<td>Gauntlt provides hooks to a variety of security tools and puts them within reach of security, dev and ops teams to collaborate to build rugged software.</td>
<td>Security Test Automation</td>
<td>DT&amp;E</td>
<td>Yes</td>
</tr>
<tr>
<td>HashiCorp Vault</td>
<td>Improves how software teams store important keys, tokens, passwords, and other secrets in their projects. Vault is an environment- and infrastructure-agnostic open toolset for secrets management.</td>
<td>Credential Protection</td>
<td>DT&amp;E</td>
<td>Yes</td>
</tr>
<tr>
<td>Sonar Qube</td>
<td>Continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities.</td>
<td>Code Security</td>
<td>DT&amp;E</td>
<td>Yes</td>
</tr>
<tr>
<td>OWASP Zap</td>
<td>Used to identify security vulnerabilities in an application while it is being developed. Useful in penetration testing.</td>
<td>Vulnerability Scanning</td>
<td>DT&amp;E and OT&amp;E</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## DevSecOps Testing Tools – Examples

<table>
<thead>
<tr>
<th>Testing Tool</th>
<th>Description</th>
<th>DT&amp;E Applicability</th>
<th>URL</th>
<th>Focus Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUnit</td>
<td>Open source, automated unit test framework for Java programming language</td>
<td>Applicable for DT and OT</td>
<td><a href="http://junit.org">http://junit.org</a></td>
<td>Unit Testing</td>
</tr>
<tr>
<td>Selenium</td>
<td>Suite of tools to automate web application testing across many platforms. Supported by many popular browsers such as Firefox, Chrome. Robot framework built on top of Selenium enables continuous testing.</td>
<td>Applicable for DT and OT</td>
<td><a href="http://docs.seleniumhq.org">http://docs.seleniumhq.org</a></td>
<td>Unit, System, Integration Testing</td>
</tr>
<tr>
<td>SoapUI</td>
<td>Open-source web service testing application framework for SOAP and REST APIs</td>
<td>Applicable for DT and OT</td>
<td><a href="https://www.soapui.org">https://www.soapui.org</a></td>
<td>Unit, Functional and Integration Testing</td>
</tr>
<tr>
<td>JMeter</td>
<td>Load testing tool for analyzing and measuring performance of services, with a focus on web applications</td>
<td>Applicable for DT and OT</td>
<td><a href="http://jmeter.apache.org/">http://jmeter.apache.org</a></td>
<td>Performance (Load) Testing</td>
</tr>
<tr>
<td>TestNG</td>
<td>Testing framework to cover all categories of tests: unit, functional, end-to-end, integration etc.</td>
<td>Applicable for DT and OT</td>
<td><a href="http://testng.org/doc/index.html">http://testng.org/doc/index.html</a></td>
<td>Unit and Integration Testing</td>
</tr>
</tbody>
</table>
Cloud Native (AWS, Azure) DevSecOps Testing and Security Tools

- **DevSecOps Pipeline**
  - AWS CodePipeline
  - Azure DevOps

- **Infrastructure Provisioning**
  - AWS Cloud Formation
  - Azure Automation, Azure Resource Manager

- **Security**
  - AWS Inspector, AWS GuardDuty, AWS CloudWatch
  - Azure Security Center, Azure AD, Azure Application Insights
Platform Deployment Options: Containerization verses Virtualization
Containerization vs. Virtualization

- In DevSecOps, software applications can be deployed in Containers or Virtual Machines (VMs)

- **VMs**
  - Self-contained computing unit with host operating system (OS)
  - Each application runs dedicated software binaries/ libraries (bins/libs) and a guest OS
  - Managed by a hypervisor

- **Containers**
  - All applications share the OS and software bins/libs
  - Containers are managed by a controller. Example: Docker Daemon (which sits in a sibling container)
Containers vs. VMs

**VM Environment**
- App 1
- Bins/Libs
- Guest OS
- Hypervisor
- Host Operating System
- Infrastructure (Server)

**Container Environment**
- App 1
- Bins/Libs
- Container(s)
- Docker Engine*
- Operating System
- Infrastructure (Server)

*Docker Engine is the runtime controller for container and images
Source: Docker.com
# Containers vs. VMs Comparison Criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Virtual Machines</th>
<th>Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Popular Examples</td>
<td>VMWare vSphere, Microsoft Hyper-V</td>
<td>Docker, Google Kubernetes, Red Hat OpenShift</td>
</tr>
<tr>
<td>Hosting Environment</td>
<td>On or Off Premise cloud environments</td>
<td>On or off premises cloud environments</td>
</tr>
<tr>
<td>Runtime Environment</td>
<td>Full OS with dedicated resources; one or more microservices per VM</td>
<td>Shared OS, resources per container; single microservice per container</td>
</tr>
<tr>
<td>Portability</td>
<td>Microservice portability is tied to the portability of selected VMs</td>
<td>Microservices are decoupled from the OS, allowing greater portability</td>
</tr>
<tr>
<td>Security</td>
<td>VM security tools and procedures are more mature</td>
<td>Larger number of services and interfaces to monitor and protect</td>
</tr>
<tr>
<td>Scalability</td>
<td>VMs can be automatically scaled based on demand</td>
<td>Containers can be automatically scaled based on demand</td>
</tr>
<tr>
<td>Performance</td>
<td>Dedicated resources in a VM mean more overhead</td>
<td>Better performance than VMs due to smaller footprint than VMs</td>
</tr>
<tr>
<td>Admin Burden</td>
<td>Less time/effort to spin up and configure vs. physical machines.</td>
<td>Simpler packaging and deployment vs. VMs</td>
</tr>
<tr>
<td></td>
<td>However, more time to spin up than containers</td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td>VMs with separate OSs may complicate cross-service communications, plug-and-play interoperability</td>
<td>Single-OS microservice deployments are more interoperable</td>
</tr>
<tr>
<td>Agility</td>
<td>Requires some degree of planning and coordination</td>
<td>Single-function containers can support faster development lifecycle</td>
</tr>
<tr>
<td>Market Trend</td>
<td>Still popular but losing ground to container deployment</td>
<td>Increasingly popular option for app migrations and microservice deployment</td>
</tr>
</tbody>
</table>
Containers vs. VMs – Security Examples

VMs

- App 1
  - Bins/Libs
  - Guest OS
- App 2
  - Bins/Libs
  - Guest OS
- App 3
  - Bins/Libs
  - Guest OS

- Hypervisor
- Host Operating System
- Infrastructure (Server)

- Container(s)
  - App 1
    - Bins/Libs
  - App 2
    - Bins/Libs
  - App 3
    - Bins/Libs
  - App 4
    - Bins/Libs

Docker Engine

Operating System

Infrastructure (Server)

AWS, Azure:
- Infrastructure Provisioning: AWS Cloud Formation, Azure Automation, Azure Resource Manager
Conclusions

- **DevSecOps Value Proposition**
  - Programs can realize significant value by implementing DevSecOps. But, test and security should not be sacrificed.

- **Shift Left**
  - Programs must truly shift Security and Test to the left to realize time and cost savings

- **Agile and DevSecOps go together**
  - DevSecOps must be fed by Agile software development. Security user stories must be part of each sprint.

- **Automation is key**
  - Security and test automation can reduce delivery time, improve quality and security, and eliminate human error
Appendix
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3PAO</td>
<td>Third Party Assessment Organization</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>CAP</td>
<td>Cloud Access Point</td>
</tr>
<tr>
<td>CASB</td>
<td>Cloud Access Security Broker</td>
</tr>
<tr>
<td>CSO</td>
<td>Cloud Service Offering</td>
</tr>
<tr>
<td>CSP</td>
<td>Cloud Service Provider</td>
</tr>
<tr>
<td>DT</td>
<td>Development Test</td>
</tr>
<tr>
<td>DT&amp;E</td>
<td>Developmental, Test and Evaluation</td>
</tr>
<tr>
<td>FFRDC</td>
<td>Federally Funded Research and Development Center</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSSP</td>
<td>Managed Security Service Provider</td>
</tr>
<tr>
<td>ORR</td>
<td>Operational Readiness Review</td>
</tr>
<tr>
<td>OT</td>
<td>Operational Test</td>
</tr>
<tr>
<td>OT&amp;E</td>
<td>Operational Test and Evaluation</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PAM</td>
<td>Privileged Access Management</td>
</tr>
<tr>
<td>RFP</td>
<td>Request for Proposal</td>
</tr>
<tr>
<td>SAFE</td>
<td>Scaled Agile Framework</td>
</tr>
<tr>
<td>SETA</td>
<td>Systems Engineering and Technical Assistance</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
</tbody>
</table>
Periodic Table of DevSecOps Tools

© 2019 The MITRE Corporation. All rights reserved.
References

- https://xebialabs.com/
- https://www.docker.com/resources/what-container
MITRE is a not-for-profit organization whose sole focus is to operate federally funded research and development centers, or FFRDCs. Independent and objective, we take on some of our nation's—and the world’s—most critical challenges and provide innovative, practical solutions.

Learn and share more about MITRE, FFRDCs, and our unique value at www.mitre.org