# DevSecOps

Is the integration of security practices within the DevOps process. DevSecOps involves creating a 'Security as Code' culture with ongoing, flexible collaboration between developers, release engineers and security teams.

# What would a DevSecOps workflow look like?

* A developer creates code within a version control management system. ​
* The changes are committed to the version control management system. ​
* Static, dynamic and dependency code analysis to identify any security defects, bugs and code quality issues. ​
* An environment is then created, using an infrastructure-as-code tool. The application is deployed, and security configurations are applied to the system. ​
* A test automation suite is then executed against the newly deployed application, including back-end, UI, integration, API and security tests. ​
* If the application passes these tests, it is deployed to a production environment. ​
* This new production environment is monitored continuously to identify any active security threats to the system.



# What are the DevSecOps scenarios?

* **Credential Scanning**

Integrating the ability to detect any secrets (user accounts, passwords, connection strings, certificates, etc.) in the work flow is essential to protecting data in the cloud. As teams move to agile development with frequent deployments/releases, exposing secrets in code can be easy to overlook. Good automation to detect issues keys/secrets is crucial to successful DevSecOps.

* **Credential Management**

Use Key Vault to safeguard and manage cryptographic keys and secrets used by cloud applications and service within the CI/CD and build a strategy for secret rotation.

* **Dependency and Container Scanning**

Integrating scanning application dependencies, and scan images (container/VM) for known vulnerabilities, licensing compliance in the workflow process before code updates are integrated into upstream branches. Components that were not vulnerable at build time may become vulnerable later as new vulnerabilities are discovered and exploited. Components, Builds, and Images need to be scanned for vulnerabilities on a regular basis continually as well.

* **Static Code Analysis**

With Code analysis tools integrated early in the workflow, teams have a better understanding of how dependencies, technical debt, bugs, and other code quality factors change with every change to the code.

* **Penetration Testing**

New vulnerabilities surface every day. New types of attacks that seek to exploit runtime weaknesses or clever buffer overflow/SQL injection attacks can leave apps that are using secure coding practices vulnerable. That’s why it is important to add dynamic application security testing to test the application in its running state along with the static code analysis.

* **Secure Optimized Workflow**

Pull requests are a central activity in adopting Git and Agile based workflows that leverage Continuous Integration (CI). DevSecOps seeks to shift security left and introduces several scenarios/processes/tools needed to have broad security coverage. These additional scenarios during CI can have an impact on developer productivity. This scenario seeks to improve/optimize workflow during Pull Requests.

In the past, where teams were in a waterfall development model, security testing tended to occur during late testing/staging phases of the development lifecycle. The concept of shifting left is to move the security testing to an earlier stage of development to get feedback more quickly. In an agile delivery model that leverages Git as the version control platform, the pull request becomes a central activity for shifting tasks left and earlier in the cycle.

* **Organization Policy Enforcement**

Policy injection offers the opportunity to enforce policy during pipeline execution. In Azure DevOps, policy is set at the organizational level. There are several scenarios and tools that can be used to enforce these policies either through Pipeline decorators or Azure DevOps component governance.