Consultingwerk
software architecture and development

DevOps Security Best Practices in Application Deployment

Clive Pain

- DevOps Engineer at Consultingwerk
- Working with Progress since 1994 version 6, in a variety of industry sectors and IT roles throughout this time.
- More recently, designing and maintaining CI/CD. Utilizing Jenkins/Ant/Groovy/PCT along with my OpenEdge skills.

Consultingwerk Software Services Ltd.

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany, subsidiaries in UK, USA and Romania
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration
- Experts in OpenEdge Application Modernization

Services Portfolio, Progress Software

- OpenEdge (ABL, Developer Tools, Database, PASOE, …)
- Telerik DevCraft (.NET, Kendo UI, Angular, …), Telerik Reporting
- OpenEdge UltraControls (Infragistics .NET)
- Telerik Sitefinity CMS (incl. integration with OpenEdge applications)
- Kinvey Platform, NativeScript
- Corticon BRMS
- Whatsup Gold infrastructure, network and application monitoring
- Kemp Loadmaster
- …

Services Portfolio, related products

- Protop Database Monitoring
- Combit List & Label
- Web frameworks, e.g. Angular
- .NET
- Java
- ElasticSearch, Lucene
- Amazon AWS, Azure
- DevOps, Docker, Jenkins, ANT, Gradle, JIRA, …
- …

Agenda

- Why Security Matters
- Database Security
- Binary-only Deployments
- PASOE Encryption
- Development Workflow
- Jenkins Pipeline
- Artifact Storage

Why Security Matters

The Increasing Threat Landscape

- “The global cost of cybercrime is projected to reach $10.5 trillion annually by 2025.”
- DevOps pipelines are prime targets due to their automated nature and integration of multiple systems.
- Threats include pipeline poisoning, code tampering, and unauthorized access.

Common Security Issues in DevOps

- Data breaches
  - Exposed credentials, unsecured APIs
- Code tampering
  - Malicious actors inserting vulnerabilities
- Unsecured CI/CD
  - Lack of proper access controls

Understanding the DevOps Attack Surface

- Vulnerabilities exist at multiple levels
  - Source code
  - Infrastructure
  - Deployment pipeline
  - Database access
- Each stage of the pipeline (build, test, deploy) is a potential attack vector

Security Complexity in Modern CI/CD Environments

- As deployment frequency increases, so does the complexity of managing security
- More automation means more integration points, each potentially vulnerable

Regulatory Pressures and Compliance Needs

- GDPR
  - Data protection and privacy requirements.
- PCI-DSS
  - Securing financial transactions.
- Failure to comply can lead to heavy fines and reputational damage

Business Impact of Poor Security

- Data loss
  - Irretrievable customer information
- Financial penalties
  - Millions in fines due to non-compliance
- Reputation
  - Customer trust can take years to rebuild
- Business disruption
  - Revenue loss

DevSecOps

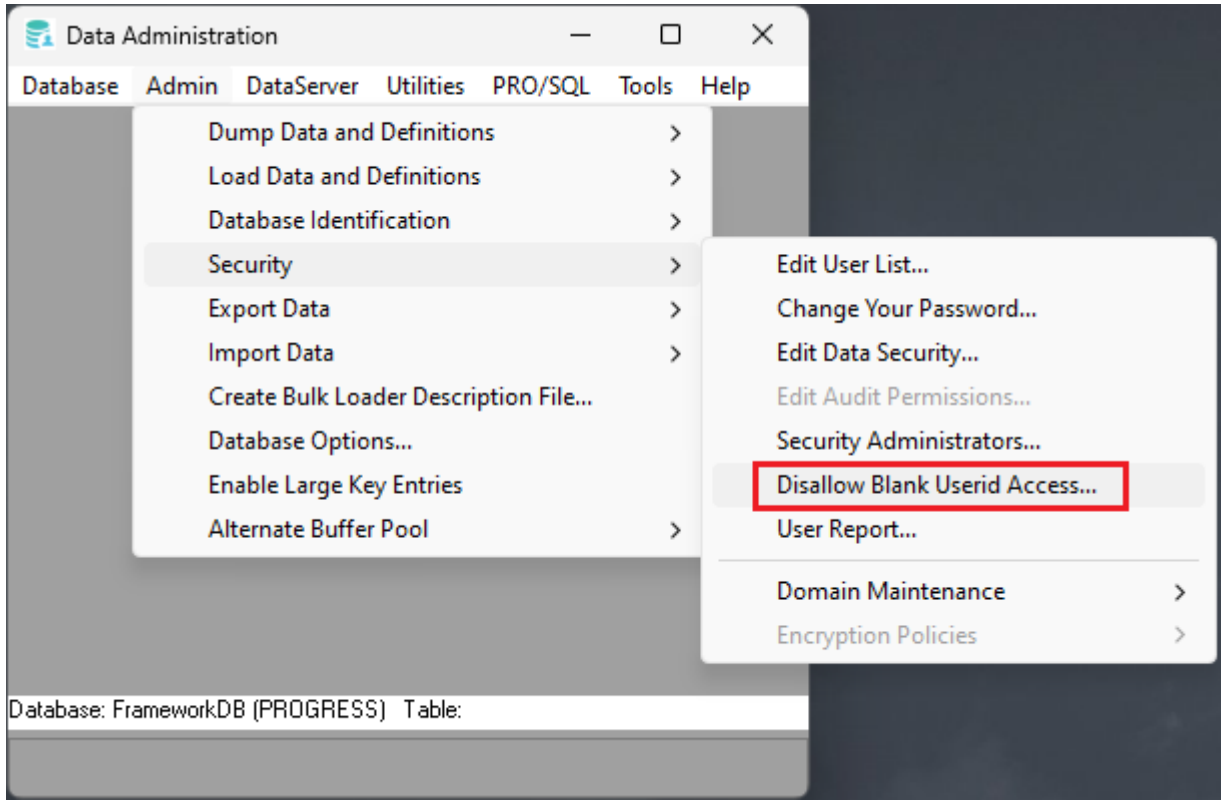
- Combines advanced DevOps and security practices
- Embeds security practices directly into the DevOps process
- Continuous integration of security tools and automated testing

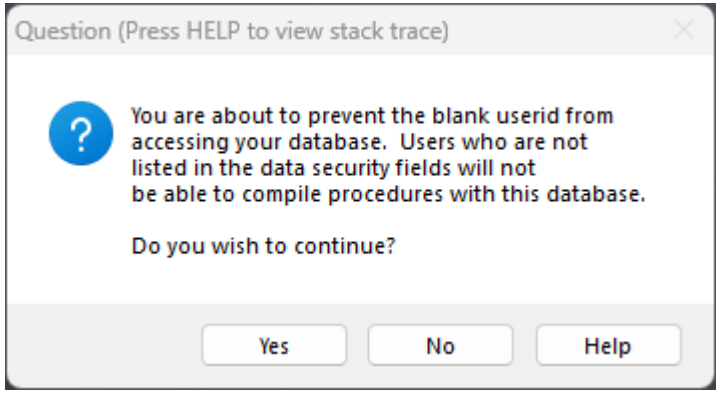
Database Security

Database Security

- Switching on/off OpenEdge Database security
- Utilizing the ‘Disallow Blank Users Access’ at run-time security
- Updates the Database’s meta-schema
- Two ways we can achieve this:
  - Data Administration menus
  - Programmatically – the DevOps way

Data Administration menus





Programmatically – Switching On

```
do transaction:

  for each _File where _File._Tbl-Type = "T":u:

    assign
      _File._can-create = "!,*":u
      _File._can-write  = "!,*":u
      _File._can-read   = "!,*":u
      _File._can-delete = "!,*":u
      _File._can-load   = "!,*":u
      _File._can-dump   = "!,*":u.

  end.

  catch oError as Progress.Lang.Error :
    message oError:GetMessage(oError:NumMessages).
    return "1":u.
  end catch.

end.
```

Programmatically – Switching Off

```
do transaction:

  for each _File where _File._Tbl-Type = "T":u:

    assign
      _File._can-create = "*":u
      _File._can-write  = "*":u
      _File._can-read   = "*":u
      _File._can-delete = "*":u
      _File._can-load   = "*":u
      _File._can-dump   = "*":u.

  end.

  catch oError as Progress.Lang.Error :
    message oError:GetMessage(oError:NumMessages).
    return "1":u.
  end catch.

end.
```

Considerations

- Still able to connect to the Database
  - Just no longer have access to the data
- Switching off the Database security is only recommended as a temporary measure:
  - DB Maintenance
  - Cloning of Database for Test/Demo environments
- Separate the Database Security from the standard deployment workflow
  - Fail the Standard deployment workflow if DB security is switched off

Managing Database Credentials

- Create an _user record as part of switching on Database Security is essential
- During deployment we can use the credentials provided to do this
- Maintaining credentials after Database Security has been switched on
  - Updating of the _user's Password
  - Prompt for old password and new password
- Switching off Database Security should involve the deletion of the _user record

Database Connection

- Deployment scripts
  - Use of dynamically generated pf files to access the database
    - Loading schema
    - Loading data
  - These are temporary files that are housekept as soon as possible
  - You will now require the use of the -U and –P connection parameters
- PASOE
  - Openedge.properties – not recommended
  - Programmatically - recommended

Deployment Scripts – pf files

- Temporary files that are purged after usage during deployment of a release
- Still require encryption of passwords
- Recommend usage of standard Progress genpassword utility
  - https://docs.progress.com/bundle/openedge-security-keys-and-certificates/page/genpassword.html
- No manual decryption required
  - Encrypted password generated is recognized when used directly with the -P connection parameter

Example - ant script macro using genpassword utility

```
<!--
  * Encrypt OpenEdge passwords
  *-->
<macrodef name="EncryptOpenEdgePassword">
  <attribute name="Password"/>
  <attribute name="outputproperty"/>
  <sequential>

    <exec dir="." executable="cmd" outputproperty="@{outputproperty}">
      <arg value="/C ${MainOpenEdgeInstallationLocation}\bin\${genpassword} -prefix ${OpenEdge.Encryption.Prefix} -password @{Password}"/>
    </exec>

  </sequential>
</macrodef>
```

Database Connection - Programmatically

- Store the Database credentials in a hidden properties file where they are encrypted
- Benefits:
  - You can apply any type of encryption for both UserID and Password
  - If credentials are stored in the Openedge.properties file
    - Limited to only encrypting the password using the genpassword utility
- Within your PASOE startup program you can dynamically set the credentials via the use of setuserid

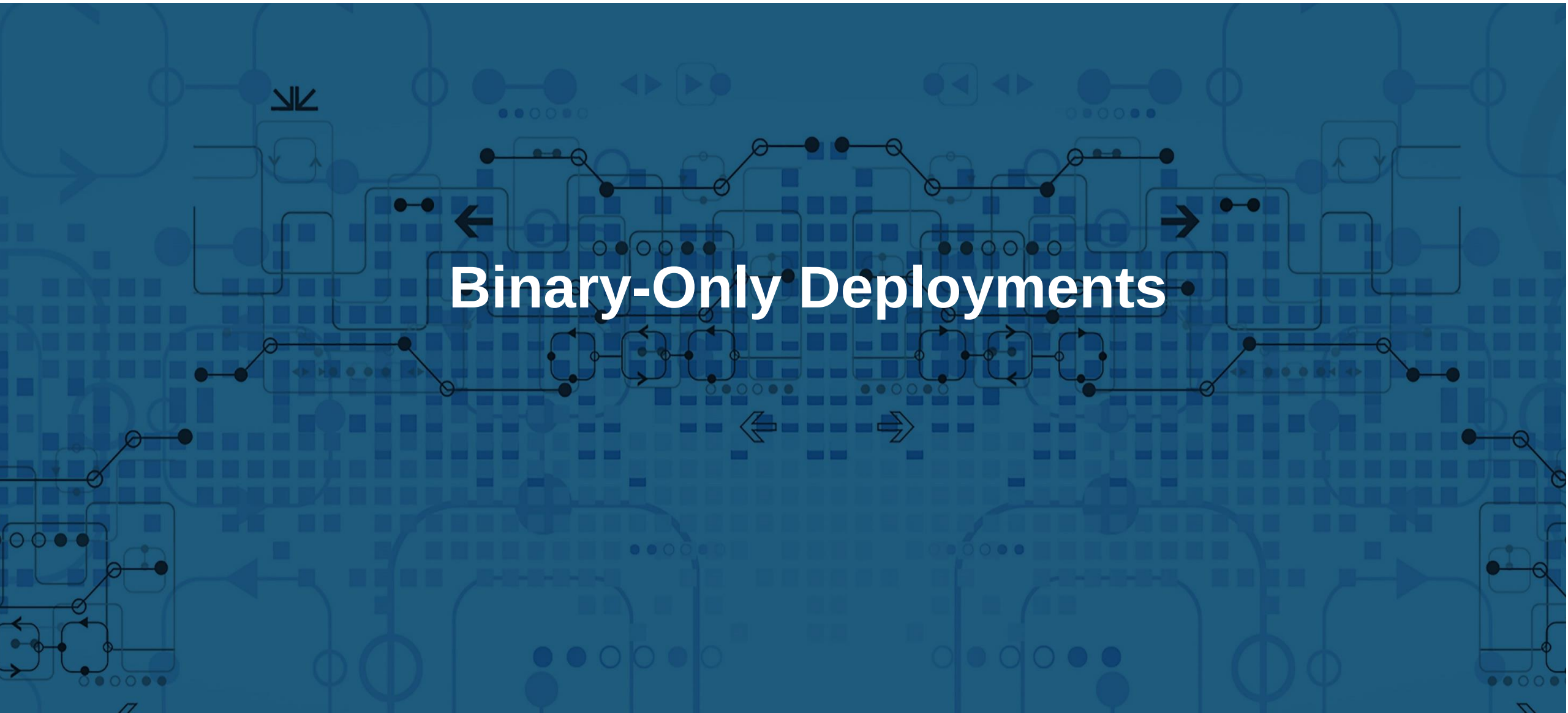
Example – property file entries

```
"databaseAuthenticationList": "DB1,DB2",
"DB1DBUser": "siwkRgLHi0+Y968UYe19Jg+hYw3JsX48Fie0rM7SQzIQ==",
"DB1DBPassword": "HNgyeZJTp6YDg0VAbaminQ:zdWuuHS5NuW2QUvydzkNtQ",
"DB1DBEncrypted": "true",
"DB2DBUser": "EQfguvFXY8SaXXJBRyXYkw:GE9cK0wUWHT8M3YUg19Xdw",
"DB2DBPassword": "Teg37538XM7X0nrMlXjV9A:k+f3+QzPknmJFnmSoqFhww",
"DB2DBEncrypted": "true",
```

```
method public void SetDatabaseAuthentication ():

  define variable cUserID                as character             no-undo.
  define variable cPasswd                as character             no-undo.
  define variable cLDBnam                as character             no-undo.
  define variable cDBList                as character             no-undo.
  define variable i                      as integer               no-undo.


  cDBList = ConfigHelper:GetApplicationSetting ("databaseAuthenticationList":U).

  if num-entries(cDBList) > 0 then
  do i = 1 to num-entries(cDBList):

    cLDBnam = entry(i, cDBList).

    if not connected(cLDBnam) then
      next.

    assign cUserID = ConfigHelper:GetApplicationSetting (cLDBnam + "DBUser":U)
           cPasswd = ConfigHelper:GetApplicationSetting (cLDBnam + "DBPassword":U).

    if ConfigHelper:GetApplicationSetting (substitute ("&1DBEncrypted":U, cLDBnam)) = "true":U then
    do:
      cUserID = this-object:DecodeValue (cLDBnam, "UserID", cUserID).
      cPasswd = this-object:DecodeValue (cLDBnam, "Password", cPasswd).
    end.

    // Set the DB credentials
    if not setuserid(cUserID, cPasswd, cLDBnam) then
      undo, throw new Exception (substitute ("Error authenticating database &1":U, cLDBnam)).

  end.

end method.
```

Considerations

- You need to run your encryption program for the credentials prior to storing them securely in the properties file
- Creating new databases during deployment for the purpose of applying delta.dfs
  - Requires re-applying of security
  - Use of a flag per Database to determine the security status (on/off)
  - If on then re-apply security
- Always take a database backup prior to undertaking any maintenance, especially in this area of security



Binary-Only Deployments

Benefits

- Protect source code
- Efficient licensing
  - Run-time only
- Efficient deployment packaging
  - Use of PLs
    - Reduced Propaths

Utilizing PCT functionality

- Deployments often require database management
  - Updating of Schema
  - Updating of data
- Achieve this via the following PCT functionalities:
  - PCTDynRun
  - PCTLoadSchema

PCTDynRun

- Allows you to run compiled Progress procedures only
- Run from deployment ant scripts
- This temporary procedure is launched by an Exec task (using either prowin, prowin32 or _progres executable), and with the specified parameters.
- https://github.com/Riverside-Software/pct/wiki/PCTDynRun

Example - ant script using PCTDynRun utility

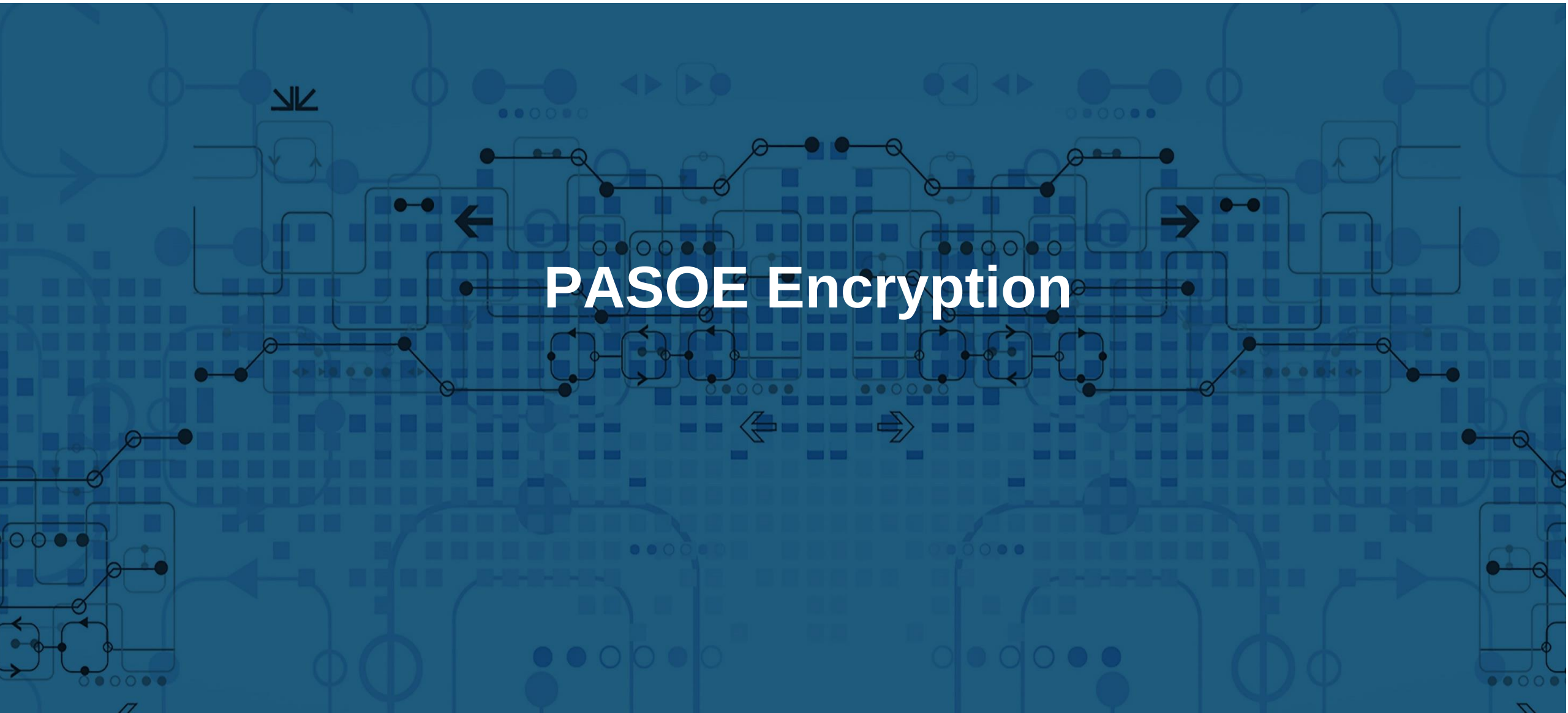
```
<PCTDynRun
  procedure="${run.procedure}"
  graphicalMode="false"
  dlcHome="@{dlcHome}"
  baseDir="@{basedir}"
  cpinternal="${Default.CpInternal}"
  cpColl="${Default.Collation.Casing}"
  cpstream="${Default.CpStream}"
  inputchars="${RunOptions.InputChars}"
  token="${RunOptions.Token}"
  msgBufferSize="${RunOptions.Message.Buffer.Size}"
  paramFile="${pf.file}">
  <options/>
  <Parameter name="Param1" value="@{Param1}" />
  <Parameter name="Param2" value="@{Param2}" />
  <OutputParameter name="@{OutputParam}" />
  <propath refid="${run.propath@{procedure}}"/>
  <DBAlias if:blank="@{deploymentPropath}" name="smartdb" value="FrameworkDB"/>
  <PCTRunOption name="-rereadnolock" />
  <PCTRunOption name="-reusableObjects" value="${RunOptions.ReusableObjects}"/>
  <PCTRunOption name="-tmpbsize" value="${RunOptions.Temp.Table.Block.Size}"/>
  <PCTRunOption name="-TB" value="${RunOptions.Table.BlockSize}"/>
  <PCTRunOption name="-TM" value="${RunOptions.Table.Number.Blocks}"/>
  <PCTRunOption name="-errorstack" />
</PCTDynRun>
```

PCTLoadSchema

- Allows you to load the Database Schema
- To achieve this with run-time only licenses, we require the following parameter:
  - clientMode="rx“
- https://github.com/Riverside-Software/pct/wiki/PCTLoadSchema

Example - ant script using PCTLoadSchema utility

```
<PCTLoadSchema srcFile="@{df}" dlcHome="${MainOpenEdgeInstallationLocation}" commitWhenErrors="true" clientMode="rx">
  <PCTConnection dbName="${FrameworkDatabaseLocation}/FrameworkDB" singleUser="true" />
</PCTLoadSchema>
```



PASOE Encryption

Encrypting PASOE Credentials

- Another vulnerability is keeping PASOE or Tomcat credentials in plain text
- During the deployment process we can encrypt these credentials to mitigate this risk
- The two areas we need to focus on to achieve this are:
  - PASOE tomcat-users.xml
  - PASOE server.xml

PASOE tomcat-users.xml

- Taking the supplied credentials given during the deployment process
- We can encrypt the password using the digest command
  - Specifying the tomcat encryption algorithm
    - For example - SHA-256
- We can then write these encrypted credentials to the PASOE tomcat-users.xml file

Example - ant script macro using digest utility

```
<!--
  * Encrypt Tomcat User passwords
  *-->
<macrodef name="EncryptTomcatUserPassword">
  <attribute name="Password"/>
  <attribute name="outputproperty"/>
  <sequential>
    <local name="digest.output.file"/>
    <local name="digest.output.guid"/>

    <generateguid property="digest.output.guid" />
    <tempfile property="digest.output.file" prefix="${digest.output.guid}." destDir="${WorkingDirectory}"/>

    <exec dir="." executable="cmd">
      <env key="CATALINA_HOME" value="${MainOpenEdgeInstallationLocation}\servers\pasoe"/>
      <arg value="/C ${MainOpenEdgeInstallationLocation}\servers\pasoe\bin\${digest} -a ${Tomcat.Encryption.Algorithm} -h org.apache.catalina.realm.MessageDigestCredentialHandler @{Password} > ${digest.output.file}"/>
    </exec>

    <loadfile property="@{outputproperty}" srcFile="${digest.output.file}">
      <filterchain>
        <!-- Remove 'MyPassword:' prefix -->
        <replaceregex pattern="@{Password}:" replace="" />
        <!-- Remove trailing whitespace, including carriage return -->
        <striplinebreaks />
      </filterchain>
    </loadfile>

    <delete quiet="true" verbose="${Build.Verbose}">
      <fileset file="${digest.output.file}"/>
    </delete>
  </sequential>
</macrodef>
```

PASOE server.xml

- After encrypting our Tomcat credentials we now need to insert our tailored CredentialHandler into our PASOE server.xml file
- Two considerations:
  - What algorithm to use
    - Specify the same algorithm used for encrypting the credentials
    - For example - SHA-256
  - Where to insert the CredentialHandler
    - Must be nested inside the relevant Realm component
    - For example - UserDatabase

Example - ant script macro inserting CredentialHandler

```
<!--
  * Insert CredentialHandler into Tomcat server.xml
  *-->
<macrodef name="TomcatCredentialHandler">
  <attribute name="ServerFileLocation"/>
  <sequential>
    <xmltask source="@{ServerFileLocation}/conf/server.xml" dest="@{ServerFileLocation}/conf/server.xml" indent="true">
      <replace path="/Server/Service/Engine/Realm/Realm[@resourceName='UserDatabase']">
        <![CDATA[
          <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase">
            <CredentialHandler className="org.apache.catalina.realm.MessageDigestCredentialHandler" algorithm="SHA-256"/>
          </Realm>
        ]]>
      </replace>
    </xmltask>
  </sequential>
</macrodef>
```

Example - ant script macro updating tomcat credentials

```
<!--
 * update tomcat.users file
 * -->
<macrodef name="update-tomcat-users">
  <attribute name="installdir"/>
  <attribute name="pasoename"/>
  <attribute name="verbose" default="${Build.Verbose}"/>
  <sequential>
    <log message="###### @{installdir}/Configuration/Templates/tomcat-users.xml to @{installdir}/@{pasoename}/conf/" mode="verbose"/>
    <copy file="@{installdir}/Configuration/Templates/tomcat-users.xml" tofile="@{installdir}/@{pasoename}/conf/tomcat-users.xml" overwrite="true" force="true"/>
    <EncryptTomcatUserPassword password="${tcPasswd}" outputproperty="EncryptedtcPasswd"/>
    <replace file="@{installdir}/@{pasoename}/conf/tomcat-users.xml">
      <replacefilter token="${tomcat.user}" value="${tcUser}"/>
      <replacefilter token="${tomcat.passwd}" value="${EncryptedtcPasswd}"/>
    </replace>
    <TomcatCredentialHandler ServerFileLocation="@{installdir}/@{pasoename}"/>
  </sequential>
</macrodef>
```

Considerations

- Any changes in configuration require restarting of your PASOE Tomcat service
- Deployment workflow of new PASOE Tomcat service:
  - Stop service
  - Unregister service
  - Delete service
  - Config new service
  - Deploy new service
  - Start new service

Development Workflow

Development Workflow

- To avoid malicious/vulnerable code being added to the application
- We can introduce important steps into the development workflow to mitigate this risk
  - Unit Testing
  - Peer Review
- Have security in mind and included in the Definition Of Done (DOD)

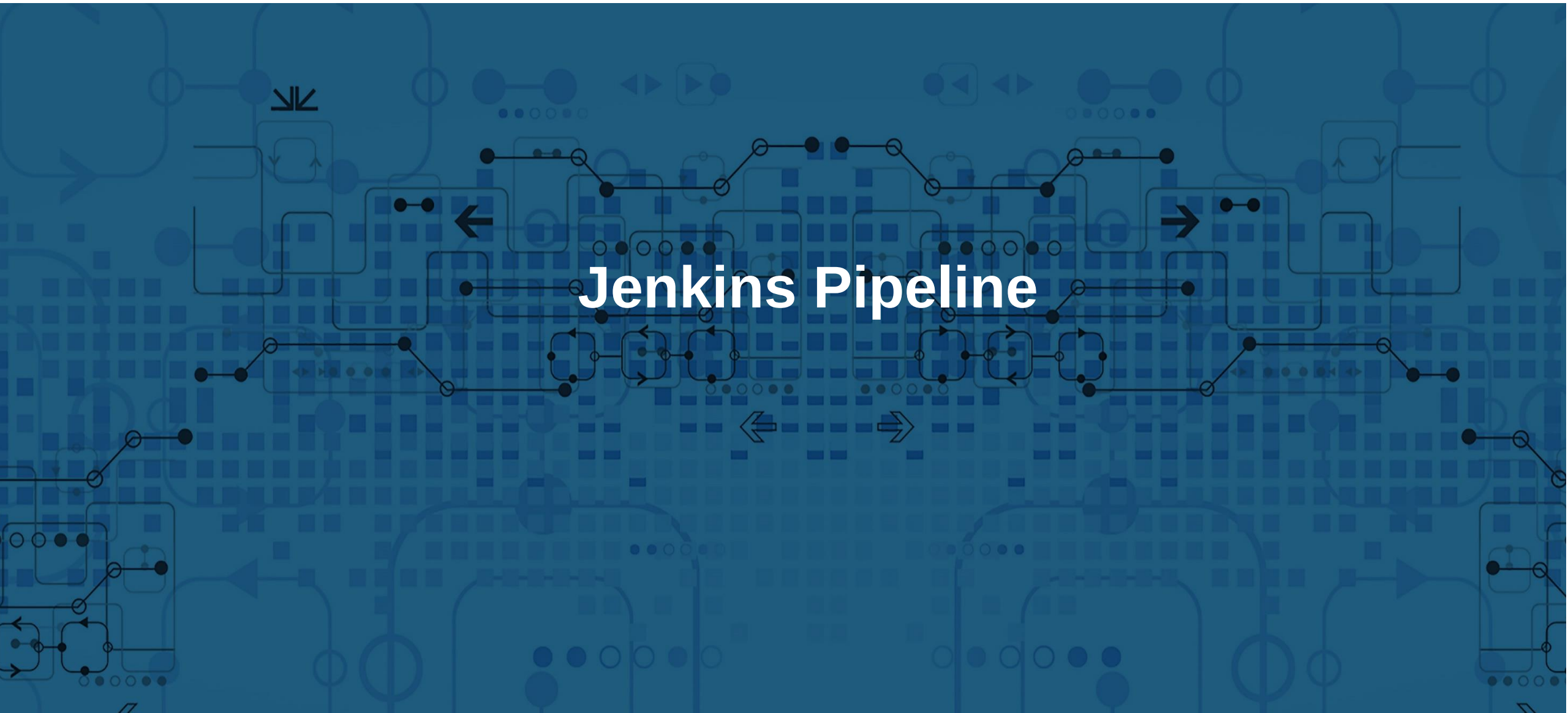
Testing

- Write Unit Test code that will be fired during the pipeline builds automatically
- Adopting a shift-left on security testing into the initial phases of design and coding
- Dynamic application security testing (DAST)
  - Build in security Testing to be run as part of the pipeline builds
- Static application security testing (SAST)
  - For example SonarQube

Peer Review

- Ensuring Adherence to Security Best Practices
- Catch intentional or unintentional introduction of backdoors or malicious code by any developer
- Enforcing organizational security policies
- Detecting insecure dependencies
  - If the code introduces or updates dependencies (e.g., third-party libraries), peer reviewers can check if those dependencies have known security vulnerabilities and flag the use of insecure versions

Consultingwerk
software architecture and development



Jenkins Pipeline

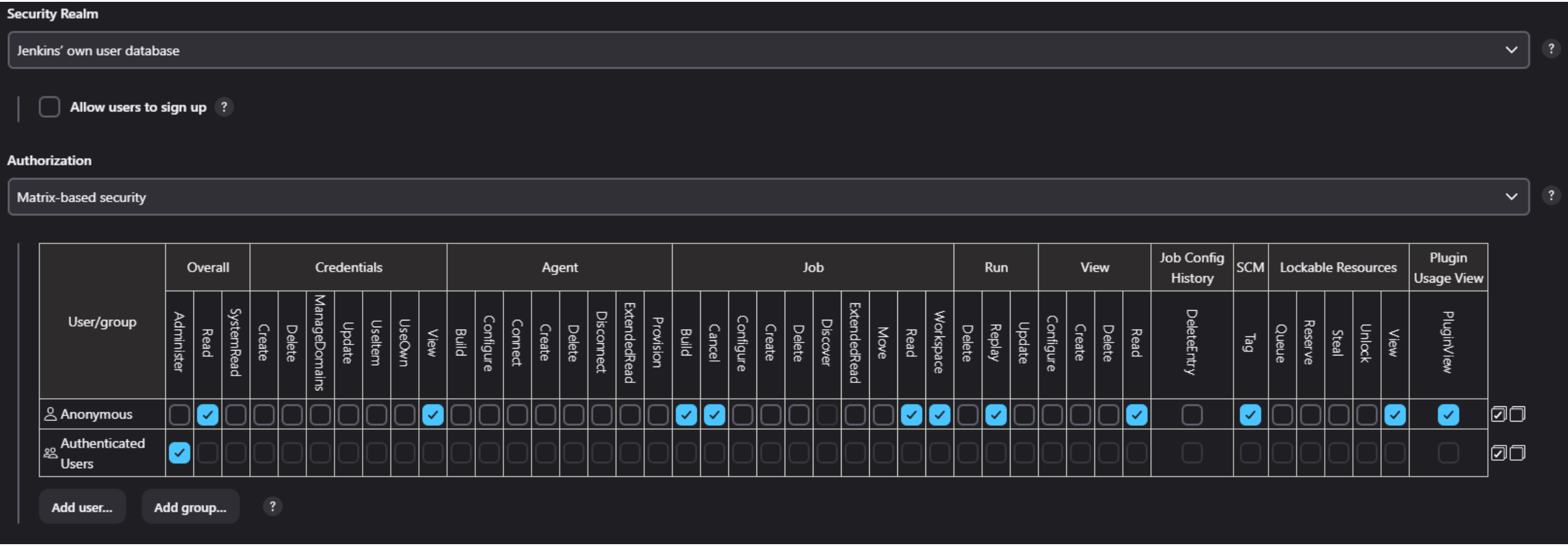
Jenkins Pipeline

- Utilize the Matrix-based security
- Prohibit access to the Script Console
  - Malicious Groovy scripts can be run from here
- Blackbox the Jenkins server
- Configure Credentials
- Keep Jenkins and plugins up to date

Matrix-based Security

- Anonymous and authenticated users
- Anonymous users
  - Restricted authorization
  - Minimize access to only allow the bare minimum
- Authenticated users
  - Logged in users can do anything
    - Sensible only if tightly managed
  - Add Users/Groups for specific roles
    - Necessary for untrusted users

Example - Matrix-based Security



Script Console

- Very powerful tool that can potentially cause widespread damage
- Restrict access to Jenkins Administrator
- Restrict access to Groovy plugin when using the "Execute system Groovy script" step
- Script Security plugin
  - In-process Script Approval
    - https://plugins.jenkins.io/script-security/
  - Any new scripts encountered during a build will fail and require approval before being allowed to run

No pending script approvals.
You have 493 script approvals with deprecated hashes: Clear Deprecated Approvals
Script approvals are stored in Jenkins as the hashed value of the script. Old approvals were hashed using SHA-1, which is deprecated. Because only the hash of the script is stored, they cannot be immediately converted to use a new hash algorithm. Instead, they will be automatically rehashed when the script is next used. To minimize potential security risks, you can immediately revoke all script approvals that were hashed using SHA-1. **This will cause all jobs and features that use those scripts to fail until they are reconfigured and then approved by a Jenkins administrator.**
You can also remove all previous script approvals: Clear Approvals

No pending signature approvals.
Signatures already approved:

```
field hudson.model.Slave name
field hudson.plugins.git.GitSCM GIT_BRANCH
field hudson.plugins.git.GitSCM GIT_COMMIT
field hudson.plugins.git.GitSCM GIT_LOCAL_BRANCH
method com.cloudbees.jenkins.plugins.sshcredentials.SSHUserPrivateKey
getPassphrase
method com.cloudbees.jenkins.plugins.sshcredentials.SSHUserPrivateKey
getPrivateKey
method com.cloudbees.plugins.credentials.common.IdCredentials getId
method com.cloudbees.plugins.credentials.common.UsernameCredentials
```

Signatures already approved assuming permission check:

Signatures already approved which may have introduced a security vulnerability (recommend clearing):

```
method groovy.lang.GroovyObject getProperty java.lang.String
method groovy.lang.GroovyObject invokeMethod java.lang.String java.lang.Object
method java.net.URL openConnection
method org.jenkinsci.plugins.workflow.support.steps.build.RunWrapper
getRawBuild
new java.io.File java.lang.String
new java.io.FileOutputStream java.lang.String
new java.io.FileWriter java.lang.String
staticMethod hudson.model.Hudson getInstance
staticMethod java.lang.System getenv java.lang.String
```

You can also remove all previous signature approvals: Clear Approvals Or you can just remove the dangerous ones: Clear only dangerous Approvals

No pending classpath entry approvals.
Classpath entries already approved:
No approved classpath entries.

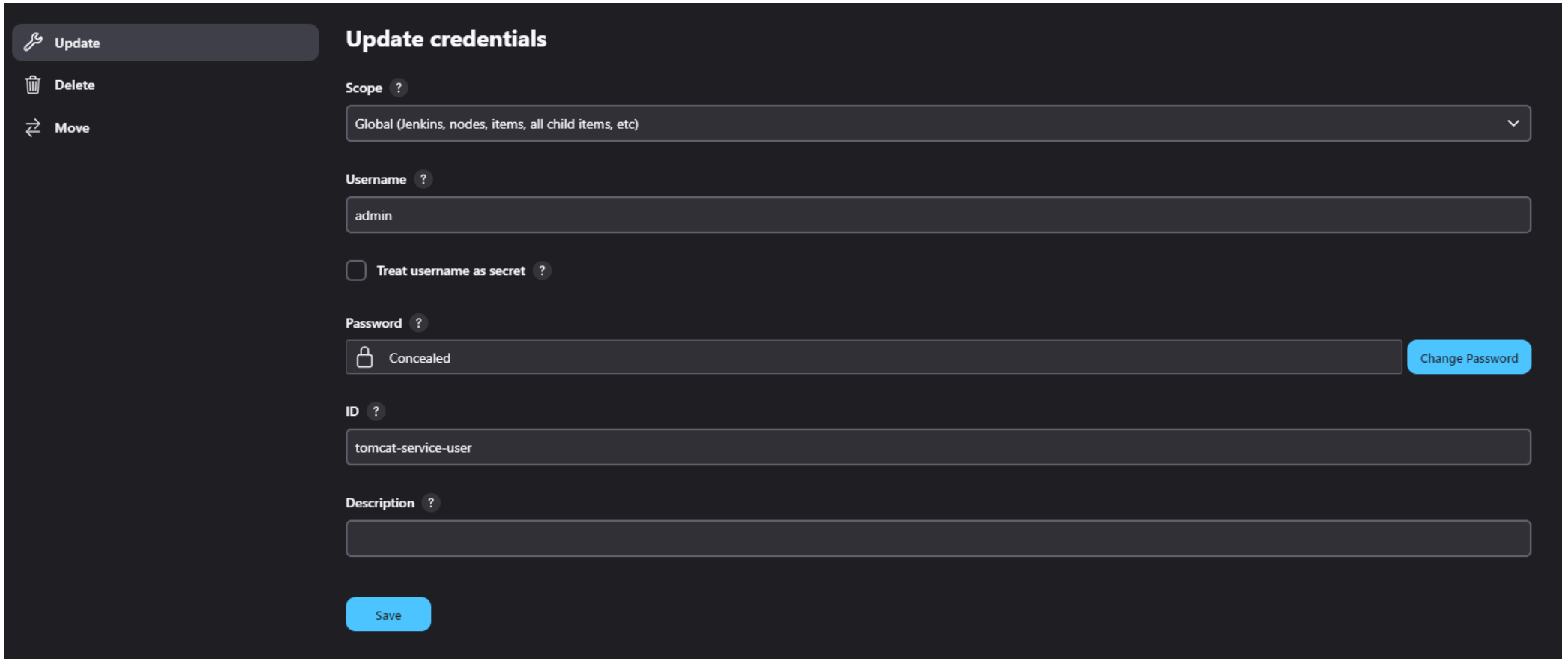
Blackbox the Jenkins server

- Restrict RDP user access
- Restrict internet access by placing behind firewall or VPN
- Jenkins and its agents should never be run as the System Administrator of the OS

Configure Credentials

- For our internal deployments - we can store our credentials in Jenkins
- We can then access these credentials via our groovy scripts and pass them to our relevant ant scripts
- From our ant scripts we adopt the same encryption approach we have used for our deployment scripts

Example - Configure Credentials



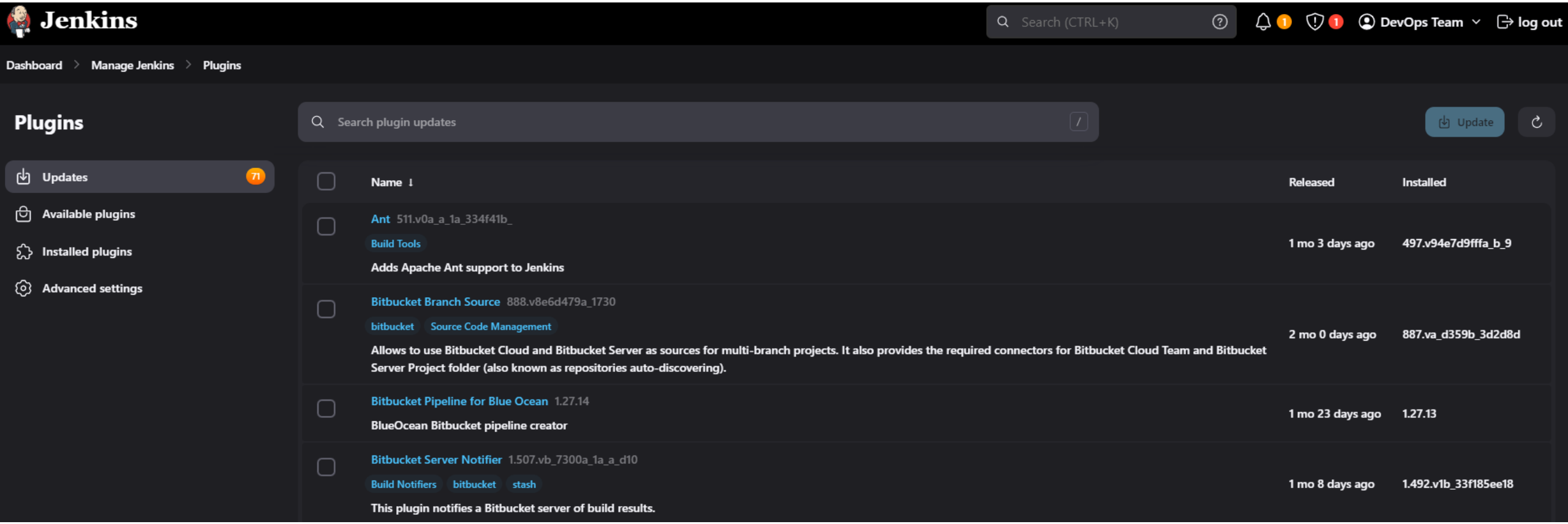
Example - Groovy script

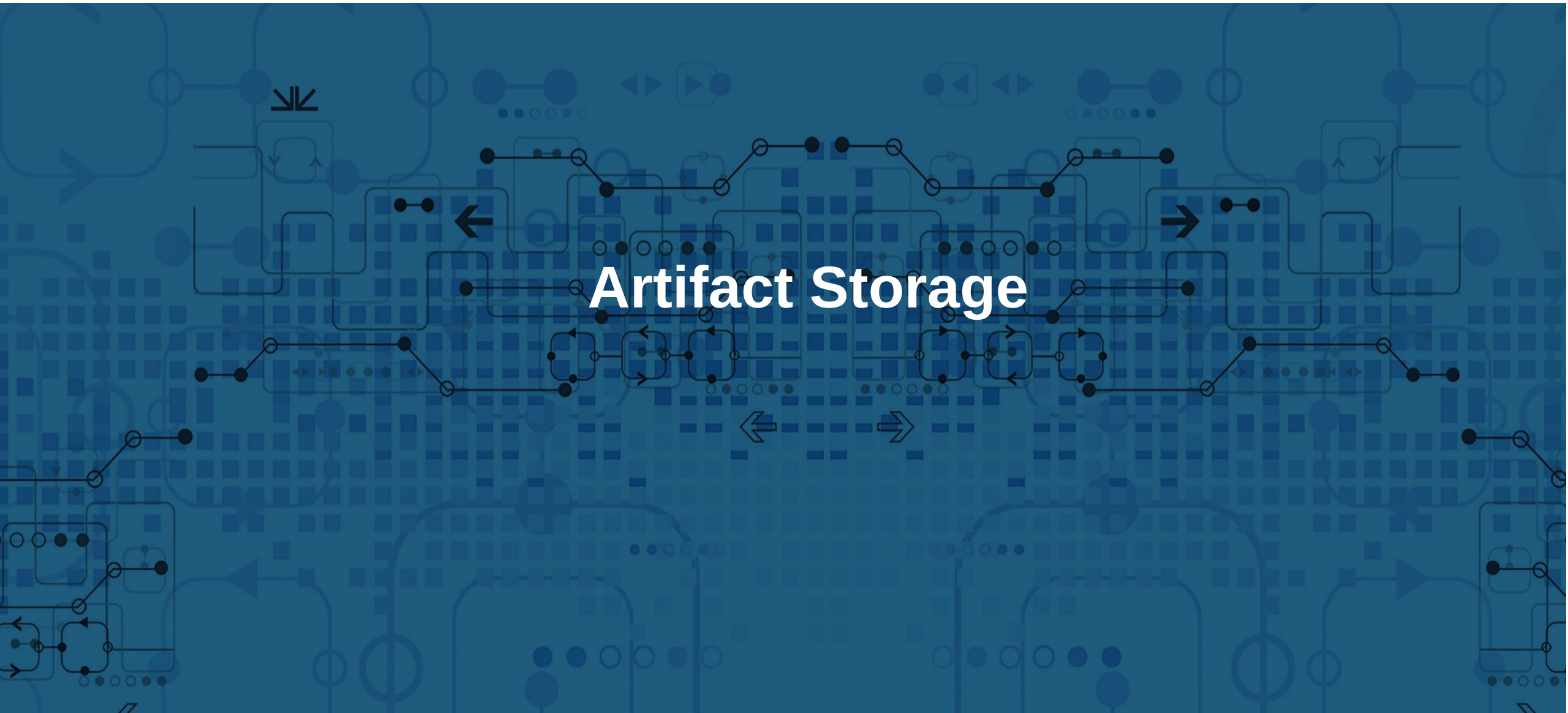
```
/*
 * deployTomcat.groovy
 */
withCredentials([usernamePassword(credentialsId: 'tomcat-service-user', passwordVariable: 'password', usernameVariable: 'username')]){

  // Delete existing Tomcat and re-deploy it from scratch
  dir("Tomcat") {
    osiv3g.unlockFolder(cTomcatPath)

    bat(script: 'ant deploy.tomcat -Dtomcat.username="' + username + '" "-Dtomcat.password=' + password + '" -Dtomcat.archive.folder=' + env.WORKSPACE + '/Output/Archives' )
  }
}
```

Keep Jenkins and plugins up to date

- Can easily get out of date
  - Finding time to test the impact of updates properly
  - Having many plugins to maintain
- Use of a Test Jenkins instance before upgrading the Production instance
- Use of Plugin Manager
  - Lists all available upgrades
  - Select relevant plugins for upgrade

Example – Applying upgrades via Plugin Manager





Artifact Storage

Repository Management Systems

- Role-Based Access Control (RBAC)
- Secure Data Transfer
- Caching External Artifacts to avoid malicious versions

Role-Based Access Control

- Users and Roles
  - Allows the creation of users, groups, and roles to control access. Users can be assigned specific roles that determine what they can view or modify.
- Granular Permissions
  - Permissions can be defined at a granular level, allowing fine control over actions such as creating, reading, updating, or deleting artifacts in specific repositories.

Example – Nexus managing roles

Roles

Manage roles

ID	NAME	DESCRIPTION
nx-admin	nx-admin	Administrator Role
nx-anonymous	nx-anonymous	Anonymous Role
nx-api	nx-api	Let the users trigger the APIs
nx-deploy	nx-deploy	upload components to hosted repositories

Secure Data Transfer

- HTTPS/SSL
  - Supports HTTPS for secure communication between clients and the server. This ensures that sensitive data (such as credentials and artifact files) are encrypted during transit.
- Repository Proxies
  - When using a Repository Management System as a proxy for external repositories (e.g., Maven Central), you can pull artifacts securely over HTTPS, ensuring that downloads from public repositories are encrypted.

Caching External Artifacts

- Acts as a proxy for external repositories. When a Repository Management System proxies an external repository, it caches the artifacts (packages) locally within your organization.
- Serves the cached version of the package, improving performance and reducing the risk of downloading malicious updates that may have been introduced after a package is cached.
- This is particularly useful because open-source packages are sometimes hijacked, or their maintainers could unknowingly publish vulnerable or malicious updates.

Conclusion

- Database Security
- Binary-only Deployments
- PASOE Encryption
- Development Workflow
- Jenkins Pipeline
- Artifact Storage