Baseline Continuous Integration / Testing / Delivery Pipeline

# Context

The Baseline for the Telekom AWS Managed Services is a collection of Account settings, IAM objects, KMS keys, CloudWatch rules, CloudTrail settings, SNS topics and AWS Config rules being deployed in a managed AWS account. The provisioning of the Baseline in the target AWS account is an automated process coordinated by a Step function. This Step function is invoked in the AWS Organization Root Account and will perform the deployment of CloudFormation Stacks and Lambda functions in the target AWS account, as well as configuration changes in the Infrastructure AWS Accounts owned by Telekom (Root Account, DevSecOps Account, Billing Account), in order to accommodate and support the newly provisioned Baseline Account.

# Current approach to Baseline development and roll-out

At the moment, different CloudFormation StackSets containing the different Baseline versions (Prod, Dev, Test) exist on the AWS Root Account, each of these StackSets containing a subset of the managed AWS accounts, meaning that the managed Client AWS accounts are provisioned with different baseline versions.

The process of developing a new baseline version consists of updating the CloudFormation Template for Baseline\_Dev, updating the CloudFormation StackSet Baseline\_Dev and testing the changes on one of the AWS accounts part of this StackSet. After successful testing, the changes are being propagated to the CloudFormation Templates for Baseline\_Test and Baseline\_Prod and the CloudFormation StackSets will be updated.

Updating the Baseline version on an individual AWS Account consists of disconnecting the AWS Account from the current CloudFormation StackSet and attaching it to the CloudFormation StackSet containing the desired version.

The status quo approach to developing/testing/updating the Baseline has several disadvantages, the root cause being that the process of developing a new Baseline and rolling it out in all the client AWS Accounts is a slow and manual, thus unreliable, process. Because the process of updating a client AWS Account contains a phase where all CloudFormation objects in the Account are being deleted (by “disconnecting” the account from the “old” CloudFormation StackSet) and re-created by attaching the Account to the “new” CloudFormation StackSet, such an update procedure must be coordinated with each Client for each AWS Account individually. The result is that there is no uniform Baseline version being installed on the Client AWS Accounts, but they are rather spread over various versions. This renders the Baseline upgrade strategy as very difficult, since the starting point is not one common version but rather distributed over a spectrum of various Baseline versions, each with its own particularities.

# Automating the CI/CT/CD pipeline

In order to streamline the development process of the Baseline, my proposal is to create a partial or fully-automated CI/CT/CD pipeline which will ensure that the Baseline code is always fully tested and that the latest approved Baseline version is automatically being rolled out in all Client AWS Accounts with zero downtime, thus zero need for coordination with the individual Clients.

There are certain advantages that result out of such an automatic approach:

* All code must be saved in a common repository (no exceptions of scripts/configuration files existing only on e.g. the developer’s workstation drive), thus versioning and change tracking is enabled by default.
* Automated testing not only speeds up the test process, but also ensures that the code changes will not break existing features or introduce security vulnerabilities.
* Automated testing ensures that the code is always working end-to-end on a new account – as opposed to only working on upgraded AWS Accounts.
* Automated testing also ensures that the automatic configuration changes in the infrastructure AWS Accounts (Root/DevSecOps/Billing accounts) are successful for accommodating a new AWS Account.
* By continuously integrating and rolling out new features, the “delta” differences applied to the AWS Accounts become smaller and the risk related to these changes is more transparent and manageable. This directly results in an increased velocity in the development of new features and their deployment in the Client AWS Accounts.
* Having the same Baseline version in every Client AWS Account ensures a simplified support and upgrade process due to a uniform state over all Accounts.

# Versioning repository

A requirement for the automation of the CI/CT/CD pipeline is storing all code related to the Baseline and its deployment in a versioning repository. This includes, but is not limited to:

* Baseline code (CloudFormation Templates, Lambda functions, Step functions)
* Code deployed in the Infrastructure AWS Accounts (CloudFormation Templates, Lambda functions, configurations)
* Configuration files containing the Infrastructure AWS Account IDs, S3 Bucket names for e.g. CloudTrail, SOC/L1/L2 email addresses for SNS configuration, etc.
* Clean-up scripts for removing all the Baseline objects in an AWS Account and all the supporting configuration in the Infrastructure AWS Accounts.
* Test framework and test cases for the Continuous Testing

These files will be checked out (either by a regular process or automatically when a change in the repository takes place) to an S3 Bucket within one of the Infrastructure AWS Accounts. The CI/CT/CD pipeline will execute by always using the latest version of the files from either the repository or the “checkout” S3 Bucket. The CI/CT process will be triggered either manually or – preferably – automatically by the code commits to the repository.

At the moment, the GitLab repository hosted by the Telekom DevOpsLab team is being used for storing the CloudFormation Templates and Lambda functions. It is to be checked if this GitLab fulfills the requirements (e.g. if a connection to the AWS Services is possible), or if an alternative such as AWS CodeCommit is a better suitable solution.

# Continuous Cleanup

A pre-requisite to the CI/CT pipeline is to always start the Integration/Testing process on a clean AWS Account, very similar – preferably identical – to a freshly-created AWS Account, since any “left-overs” from previous tests can affect subsequent Integration/Test runs. Because AWS Accounts cannot be easily closed, we propose reserving certain AWS Accounts for testing only and developing a cleanup strategy for bringing them to a pre-defined state.  
The cleanup strategy also needs to undo the changes performed in the Infrastructure AWS Accounts (Root, DevSecOps, Billing) for this specific target AWS Account. This includes, but is not limited to the CloudWatch EventBuses permissions, the CloudTrail S3 Bucket policy, etc.

# Continuous Integration

The CI process will be triggered either manually or – preferably – automatically by a commit to the versioning repository. It is to be checked if this process can be achieved by triggers only, or if an orchestrating CI/CT/CD tool like Jenkins needs to be used.

This first step in the CI process is a cleanup of the target testing AWS Account and the supporting configuration in the Infrastructure AWS Accounts.

The input of the CI process can be either a standard or a customized configuration file, similar to a standard input request order for a Client AWS Account. At a minimum, this configuration file should contain the ID of the testing AWS Account and an email address used for SNS topic.

The CI process will continue by updating all the needed supporting structures (e.g. CloudFormation Template and StackSet in the Root Account, CloudTrail S3 Bucket policy in the DevSecOps account, etc.) and then start the Baseline provisioning process in the testing AWS Account.

Goal of the CI process is to confirm if the provisioning of a new account by using the newest Baseline code is successful.

# Continuous Testing

After a successful provisioning of the testing AWS Account by the CI process, the Continuous Testing process is being kicked off automatically in this AWS Account. Any change to the code will trigger the execution of the whole suite of functional, security and upgrade tests, ensuring quality and early-signaling of bugs or security problems.

The executed test cases will go through the following phases:

* Initialization – preparation of the required environment
* Execution – testing of a certain functionality/security feature
* Cleanup – undoing the changes

The Continuous Testing will cover the following areas:

* Security – ensuring that the delivered security policies (e.g. TSI\_Base\_Deny IAM policy) protect the Baseline objects from tampering, but allow full functionality of the AWS Services outside the scope of these security policies.
* Integration with Infrastructure AWS Accounts – ensuring that the integration with the central monitoring, logging, automation, billing, etc. services is successful, which implies that the configuration of the testing AWS Account as well as the Infrastructure AWS Accounts is properly executed.
* Functional tests within the test AWS Account – to confirm functionality of e.g. S3 DPC, automatic Tagging of resources, automation around backup/shutdown/startup, automated patch management, etc.
* Upgrade tests for existing AWS Accounts – a testing AWS Account will be brought to the same Baseline version level as the Client AWS Accounts, the newest Baseline code will be deployed in this Account and acceptance tests will be run.

A sample testing matrix for Security is being illustrated in the table below. All test-objects must be created with a “full access” role (e.g. “OrganizationAccountAccessRole” or “TSI\_Base\_FullAccess”), all test operations must be performed with an account user having two policies attached (“AdministratorAccess” and “TSI\_Base\_Deny”):



The result of the CT process is a list with all executed test cases and their success/failure status, including an overall status which will semaphore the deployment of the new code in the Client AWS Accounts.

# Continuous Delivery

The Continuous Delivery process enables the deployment of the latest tested and accepted Baseline code into all Client AWS Accounts. The deployment process is subject to approval through the Change Management Process.

The CD process will be manually started/approved for now, in the future an auto-deployment of the latest successfully tested Baseline code in the Client AWS Accounts is conceivable. It is possible to use the “branch merge” Git function as an approval signal of the current changes and as a trigger for the CD process.

The CD process must have proper logging and monitoring. As a control mechanism, the CD process must stop at the first encountered error and notify L3 support of the pending problem.