riskfocus INSIGHT

Data Masking: A must for test environments on public cloud

by Subir Grewal | Oct 29, 2019

Eat your own cooking

Why mask data? Earlier this month, the security firm Imperva announced it had suffered a significant data breach. Imperva had uploaded an unmasked customer DB to AWS for “test purposes”. Since it was a test environment, we can assume it was not monitored or controlled as rigorously as production might be. Compounding the error, an API key was stolen and used to export the contents of the DB.

In and of itself, such a release isn’t remarkable; it happens almost every day. What makes it unusual is that the victim was a security company, and one that sells a data masking solution; Imperva Data Masking. This entire painful episode could have been avoided if Imperva had used their own product and established a policy to require all dev/test environments be limited to masked data.

The lesson for the rest of us is that if you’re moving workloads to AWS or another public cloud, you need to mask data in all test/dev environments. In this blog post, we will consider how such a policy might be implemented.

Rationale for Data Masking

Customers concerned about the risk of data loss/theft seek to limit the attack surface area presented by critical data. A common approach is to limit sensitive data to “need to know” environments. This generally involves obfuscating data in non-production (development, test) environments. Data masking is the process of irreversibly, but self-consistently, transforming data such that the original value can no longer be recovered from the result. In this sense, it is distinct from reversible encryption and has less inherent risk if compromised.

As data-centric enterprises move to take advantage of pubic cloud, a common strategy is to move non-production environments first; the perception is that these environments present less risk. In addition, the nature of the development/test cycle means that these workstreams can strongly benefit from the flexibility in infrastructure provisioning and configuration that public cloud infrastructure provides. For this flexibility to have meaning, dev and test data sets need to be readily available, and as close to production as possible so as to represent the wide range of production use cases. Yet, some customers are reluctant to place sensitive data in public cloud environments. The answer to this conundrum is to take production data, mask it, and move it to the public cloud. The perception of physical control over data continues to provide comfort (whether false or not). Data masking makes it easier for public cloud advocates to gain traction at risk-averse organizations by addressing concerns about the security of data in the cloud.

Additionally, regulations like GDPR, GLBA, CAT and HIPAA impose data protection standards that encourage some form of masking in non-production environments for Personal Data, PII (Personally Identifiable Information) and PHI (Personal Health Information) respectively. Every customer in covered industries has to meet these regulatory requirements.

Base Requirements

Masking solutions ought to provide some number of the following requirements:

1. **Data Profiling:** the ability to identify sensitive data across data-sources (eg. PII or PHI)
2. **Data Masking:** the process of irreversibly transforming sensitive data into non-sensitive data
3. **Audit/governance reporting:** A dashboard for Information Security Officers responsible for meeting regulatory requirements and data protection

Building such a feature set from scratch is a big lift for most organizations, and that’s before we begin considering the various masking functions that a diverse ecosystem will need.
Masked data may have to meet referential integrity, human-readability or uniqueness requirements to support distinct test requirements. Referential integrity is particularly important to clients who have several independent datastores performing a business function or transferring data between each other. Hash functions are deterministic and meet the referential integrity requirement, but do not meet the uniqueness or readability requirements. Several different algorithms to mask data may be required depending on application requirements. These include:

1. **Hash functions:** e.g., use a SHA1 hash
2. **Redaction:** (truncate/substitute data in the field with random/arbitrary characters)
3. **Substitution:** with alternate “realistic” values (a common implementation samples real values to populate a hash table)
4. **Tokenization:** substitution with a token that can be reversed, generally implemented by storing the original value along with the token in a secure location

Data Masking at public cloud providers

AWS has several white-papers, reference implementations, including:

- An AI powered masking solution for personal health information (PHI) which uses API gateway and Lambda to retrieve and mask PHI in images on S3, and returns masked text data posted to API gateway
- A design case-study with Dataguise to identify and mask sensitive data in S3
- A customer success story of a PII masking tool built using EMR and DynamoDB
- An AWS whitepaper which describes using Glue to segregate PHI into a location with tighter security features.

However, none of these solutions address masking in relational databases or integrate well with the AWS relational database migration product, DMS.

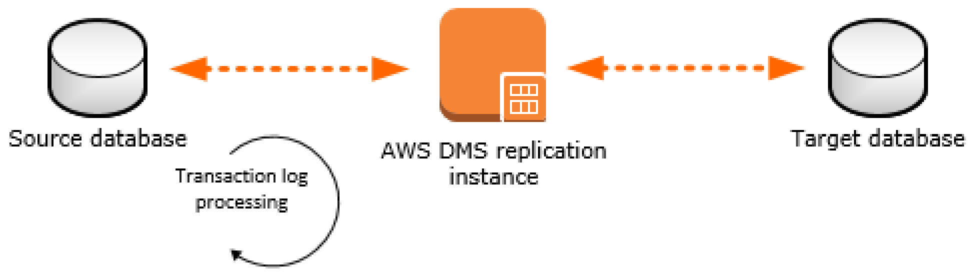
Microsoft offers both versions of its SQL Masking product on Azure:

- Dynamic Masking for SQL Server: which overwrites query results with masked/redacted data
- Static Masking for SQL Server: which modifies data to mask/redact it.

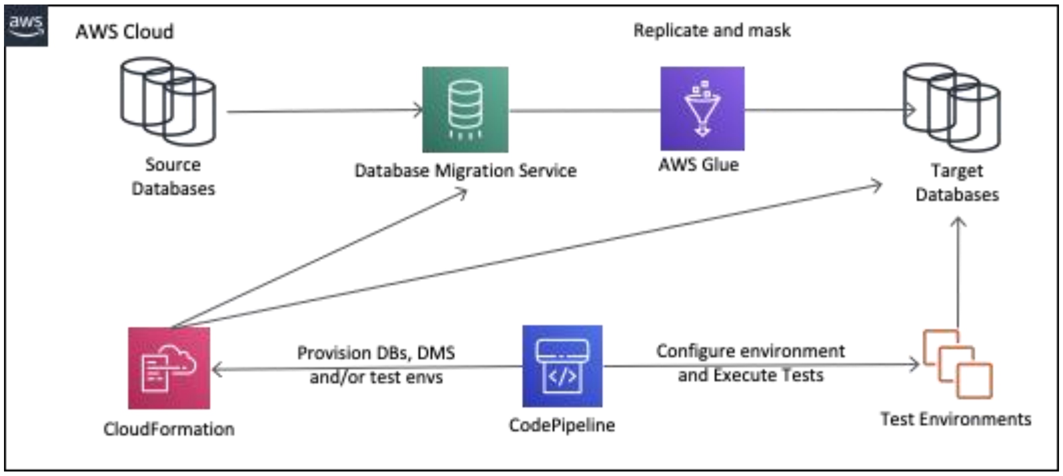
For the purposes of this discussion, we focus on what Microsoft calls “static masking” since “dynamic masking” leaves the unmasked data present on the DB, failing the requirement to shrink the attack surface as much as possible. We will also focus this discussion to AWS technologies to explore cloud-native versus vendor implementations.

Build your own data masking solution with AWS DMS and Glue

AWS Data Migration Service (DMS) currently provides a mechanism to migrate data from one data source to another, either at one time or via continuous replication as described in the diagram below (from AWS documentation):



DMS currently supports user-defined tasks that modify the Data Definition Language (DDL) during migration (e.g. dropping tables or columns). DMS also supports character level substitutions on columns with string type data. A data masking function using AWS’ ETL solution Glue could be built to fit into this framework, operating on field level data rather than DDL or individual characters. An automated pipeline to provision and mask test datasets and environments using DMS, Glue, CodePipeline and CloudFormation is sketched below:



When using DMS and Glue, the replication/masking workload is run on AWS, not in the customer's on-premises datacenter. Un-masked or un-redacted data briefly exists in AWS prior to transformation. This solution does not address security concerns around placing sensitive data (and accompanying compute workloads) on AWS for clients who still gingerly approach public clouds. Still, for firms that look for a cloud-native answer, the above can form a kernel of a workable solution, when combined with additional work around identification of data needing masking and reporting/dashboarding/auditing.

Buy a solution from Data Masking Vendors

If the firm is less concerned about cloud-native services, there are several commercial products that offer data masking in various forms which meet many of these requirements. This includes IRI Fieldshield, Oracle Data Masking, Okera Active Data Access Platform, IBM Infosphere Optim Data Privacy, Protegrity, Informatica, SQL Server Data Masking, CA Test Data Manager, Compuware Test Data Privacy, Imperva Data Masking, Dataguise and Delphix. Several of these vendors have some form of existing partnership with cloud service providers. In our view, the best masking solutions for the use case under consideration is the one offered by Delphix.

Buy: Data Masking with Delphix

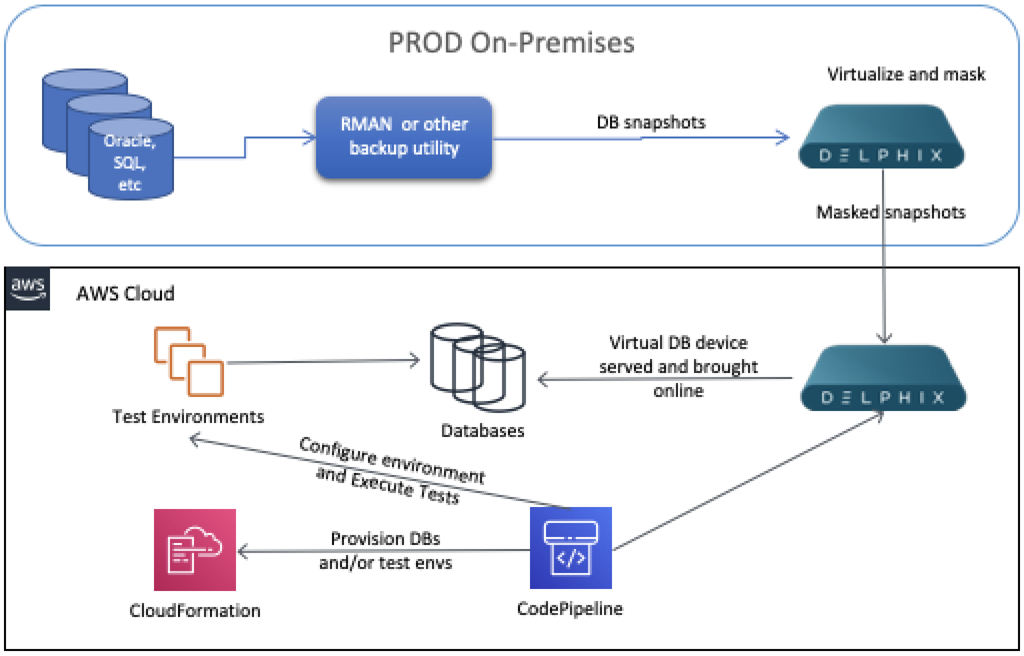
This option leverages one of the commercial data masking providers to build data masking capability at AWS. Delphix offers a masking solution on the AWS marketplace. One of the benefits of a vendor solution like Delphix is that it can be deployed on-premises as well as within the public cloud. This allows customers to run all masking workloads on-premises and ensure no unmasked data is ever present in AWS. Some AWS services can be run on-premises (such as Storage Gateway), but Glue, CodeCommit/CloudFormation cannot.

Database Virtualization

One of the reasons Delphix is appealing is the integration between its masking solution and its “database virtualization” products. Delphix virtualization lets users’ provision “virtual databases” by exposing a filesystem/storage to a database engine (eg. Oracle) which contains a “virtual” copy of the files/objects that constitute the database. It tracks changes at a file-system block level, thus offering a way to reduce the duplication of data across multiple virtual databases (by sharing common blocks). Delphix has also built a rich set of APIs to support CI/CD and self-provisioning databases.

Delphix’s virtualized databases offer several functions more commonly associated with modern version control systems such as git. This includes versioning, rollback, tagging, low cost branch creation coupled with the ability to revert to a point along the version tree. These functions are unique in that they bring source code control concepts to relational databases, vastly improving the ability of CI/CD pipeline to work with relational databases. This allows users to deliver on-demand, masked data to their on-demand, extensible public cloud environments.

A reference architecture for a chained Delphix implementation utilizing both virtualization and masking would look like this:



Conclusion

For an organization with data of any value, masking data in lower environments (dev, test) is an absolute must. Masking such data also makes the task of migrating dev and test workloads to public clouds far easier and less risky. To do this efficiently, organizations should build an automated data masking pipeline to provision and mask data. This pipeline should support data in various forms, including files and relational databases. Should the build/buy decision tend toward purchase, there are several data masking products that can provide many of the core masking and profiling functions such a masking pipeline would need, and our experience has led us to choose Delphix.