Molnify Security Q&A

December 2020

How does data flow between a client and Molnify?

For the SaaS based solution (and not on-prem), there are two general data flows - one when the application is created/updated on the platform, and one when the user interacts with the application. See Exhibit 1 for a simplified overview of the architecture.

- Application creation/upload
    - The client develops an Excel model with potentially sensitive data and this model is saved in a file that is residing somewhere on client network (e.g., on a file server or on a client machine)
    - The file is uploaded to Molnify over an encrypted channel (HTTPS)
    - The file is processed within Molnify's datacenter (hosted on GCP), from which two components are built - one user interface component, and one backend component. The former contains the look and feel of application with named inputs, outputs etc. but no calculation logic. The latter contains the actual calculation logic
    - The two components, and the file, is stored and processed by Molnify until either the client terminates the agreement or choses to delete/update the application (that can be done by an admin from the client)
- Application interaction
    - A user signs in to Molnify using one of the supported methods (see authentication policy above)
    - Molnify validates that the user has access to view and use the application (and which role the user has). In every case where an application is not open for anybody to access, encrypted channels are enforced (i.e., everything is served over HTTPS)
    - When the user interacts with the application, changes to the application is transferred bidirectionally over HTTPS

What policy is used for electronic logging?

Molnify consists of a number of different components, in most cases deployed as separate Docker containers on a Kubernetes cluster. The components utilize a central logging system, from which we can both create logging metrics, review historical usage as well as stream logs live. See Exhibit 1 for an example of one of the views we are using. See also Exhibit 2 for an overview of the different technical components and how data flows between them. Standard log retention policy is 30 days, but can easily be changed

What process is followed for Change Control?

While there is no formal change control policy, primarily driven by our small size and dependency on key people, it works like the following in practice

- Bugs, feature enhancements and other proposed changes are typically tracked through an issue tracker. See Exhibit 3 for an example of documentation and data from the issue tracker
- All source code is without exception in a source code repository where we can track changes from the first rows of code written
    - Branches are typically used for new development
    - We view configuration as code, and hence configurations are also included in the source code repository
- Features typically have both unit tests and feature tests. See Exhibit 4 for a simple example. In exceptional cases, we also have tests for non-functional requirements. Test coverage is tracked with automatic tools and tests automatically run in each build/deploy
- Deployment to production for all components follow the following steps
    - Tests are run
    - Source code is committed
    - Deployment to a staging environment is done, which itself is versioned, but the software is not yet deployed in the sense it is running
    - Configuration is updated so that the staging environment uses the new version
    - Often, manual tests are performed so that features work as intended

As both code, configuration and deployments are versioned, this most notably means that software can easily be rolled back to know working configuration. Rollback generally takes less than a minute, regardless of component

What vulnerability management is in place?

See “Software dependency management” in the Information Security Policy. In addition to this we have

- Vulnerability scanning enabled and automatic software updates on all managed software (e.g., OS for Docker images)
- Vulnerability scanning enabled for all components of the software that scans running production code. Latest run on December 8th, see Exhibit 5 for additional information
- Static code analysis, to help detect weaknesses in the software that we develop ourselves. We use SpotBugs, which checks for 400+ typical bug patterns
- Test code analysis, to help detect untested software components that we develop ourselves. This analysis includes detailed analysis of which actual

rows of code that is tested and which branches that are run in tests to help identify areas to improve testing in
- Awareness of OWASP with active efforts to mitigate the top 10 vulnerabilities as they apply to different components in our platform

What authentication policy is used for user accounts?

We have 6 ways of authenticating users, with somewhat different policies

1) User & password
2) Google account (SSO)
3) Microsoft account (SSO)
4) Custom SSO, e.g., SAML-based from client system
5) Government based identification (BankID, with either chip and reader or biometrics which is granted based on human validation of a users identity)
6) Token based authentication

For 1-3, we use the default settings on Firebase, which is a service acquired and now run by Google. To the best of our knowledge, Firebase does not fully lock out the user ID after five invalid attempts but may require additional proof - e.g., email validation, two-factor authentication, captchas and/or put limitations on number of allowed incorrect attempts during a certain time period. For 4, it is up to the client's internal login policies. For 5, it is a central policy that handles limits in incorrect login attempts as well as allowed concurrency. For 6, we log exceptional instances and contact the client (our customer) that uses this to discuss potential actions.

Molnify uses login method 2 for all internal accounts on Molnify. Accounts are provisioned by checking photo ID. Additionally two-factor authentication is enforced on the domain, unexpected account behaviour monitored, strong passwords are enforced and password reuse is not possible

What encryption policy is used?

- All data on all client devices are encrypted
- All communication between client devices and the datacenter is encrypted, using Google standards - such as two factor authentication that opens up a VPN connection over which data flows
- All data between a logged in user's client and the platform is encrypted using industry standards
- Sensitive configuration of the platform is often, but not always, using Kubernetes secrets for additional security, as opposed to clear text configuration in source code

What background checks are done for employees?

- Education confirmation (no time limit)
- Former employment confirmation (most recent employments)

- Criminal background check-up through the police (maximum time possible)

How does Molnify handle incidents?

General incidents shall be handled adhering Atlassian's handbook for incident management. Incidents are classified as Critical, High, Medium or Low. Up to today's date, no incidents have been classified as Critical and the platform has never had downtime affecting a significant amount of users.

Security incidents, where information may be compromised and particularly where personal data may be at risk, shall additionally adhere to the following guidelines:

- Systems not deemed to be safe or potentially compromised shall be shut down for external access
- Data Controllers shall be notified as soon as possible and potentially additional requirements should be adhered to
- The board shall be notified as soon as possible
- An incident report shall be sent, if deemed necessary, to the Swedish Authority on Data Protection ("Datainspektionen"), at the latest 72 hours after the incident as first discovered
- Depending on the incident and DPA, individual data subjects may be notified, depending on who is the controller of data
- Root cause analysis shall always be done and shall include immediate response as well as mitigating actions to prevent a similar incident from arising again

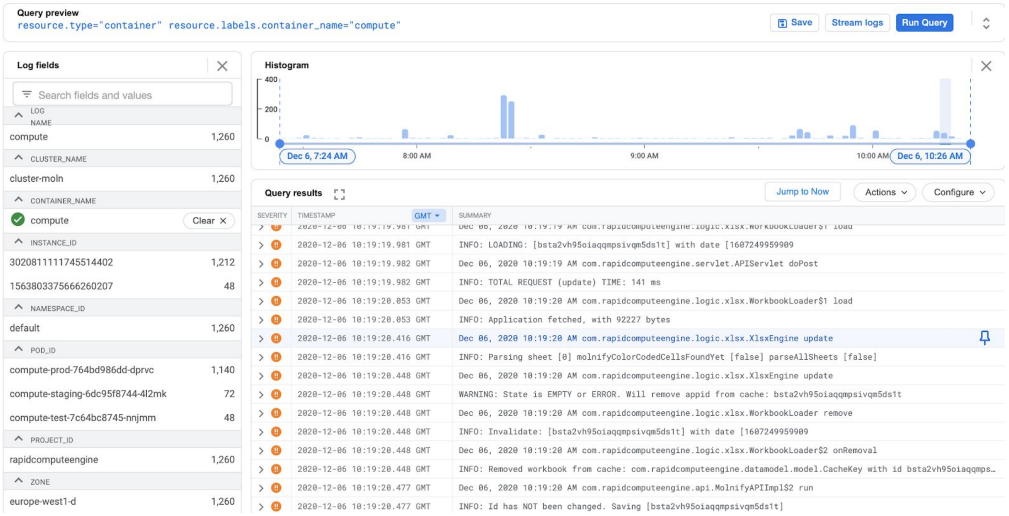

Exhibit 1 *One of the views available on the platform to view, search and audit logs*

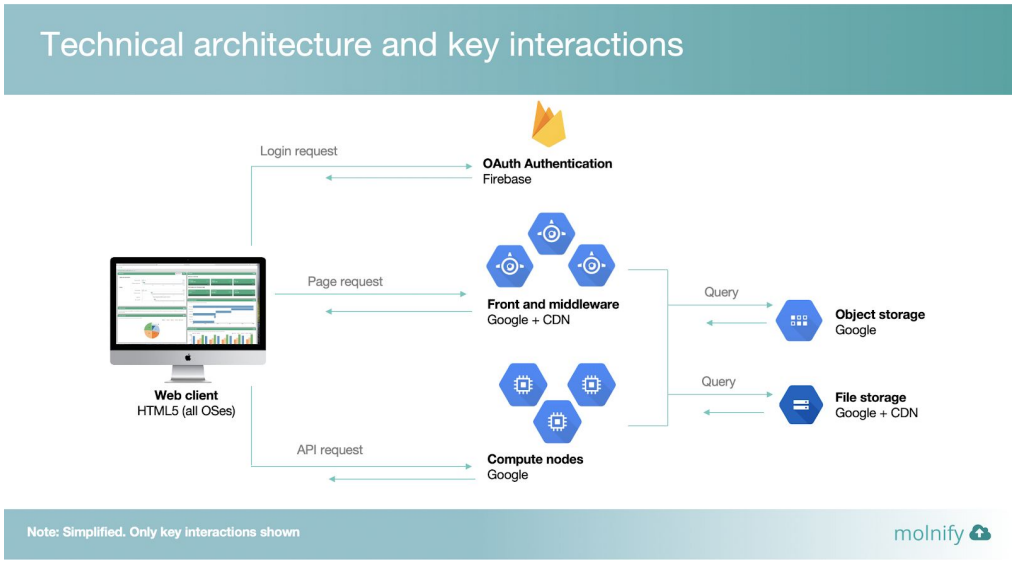

Exhibit 2 *Simplified view of technical architecture*

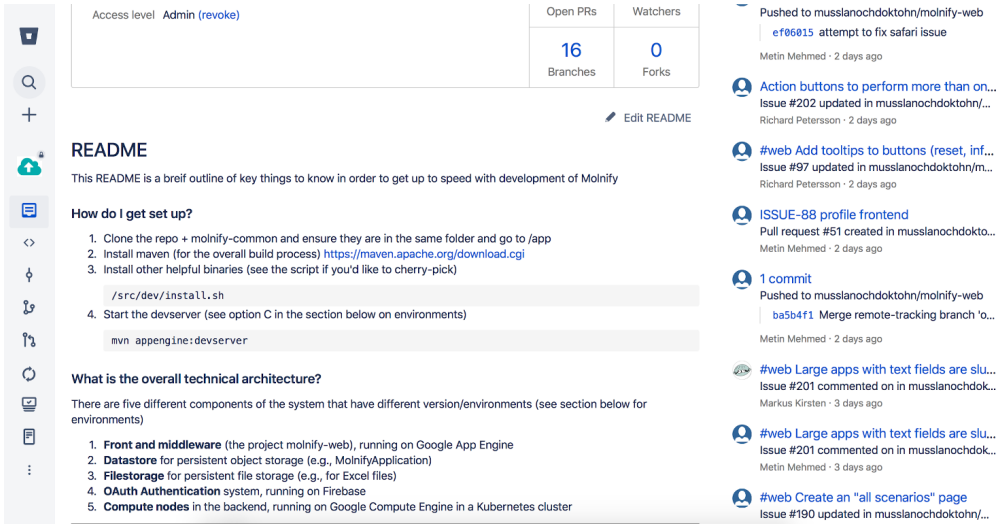

Exhibit 3 *Example of documentation, including integration with issue tracking system*

```java
@Test
public void testTableWithString() throws Exception {
    MolnifyApplication tableApp = get("internal-table-with-string");

    // Get some string data from the table
    String tableData = tableApp.getOutputs().get(0).getValue();
    assertTrue(tableData.length() > 10);
    assertFalse(tableData.indexOf("Major rodent") > 0);
    assertTrue(tableData.indexOf("Beaver") > 0);
    tableData = null; // So we won't use it by mistake

    // Make updates
    ChangeList changes = new ChangeList();
    changes.addChange(new InputOutputItemShort("Blad1!B3", "5"));
    Engine impl = EngineFactory.getImplementation(tableApp);
    ChangeList result = impl.calculate(tableApp, changes);
    assertEquals(1, result.getChanges().size());

    // Ensure we can read updated string data from the table
    String updatedTableData = result.getChanges().get(0).getValue();
    assertTrue(updatedTableData.indexOf("Major rodent") > 0);
    assertTrue(updatedTableData.indexOf("Beaver") > 0);
}
```

Exhibit 4 *Example unit test*

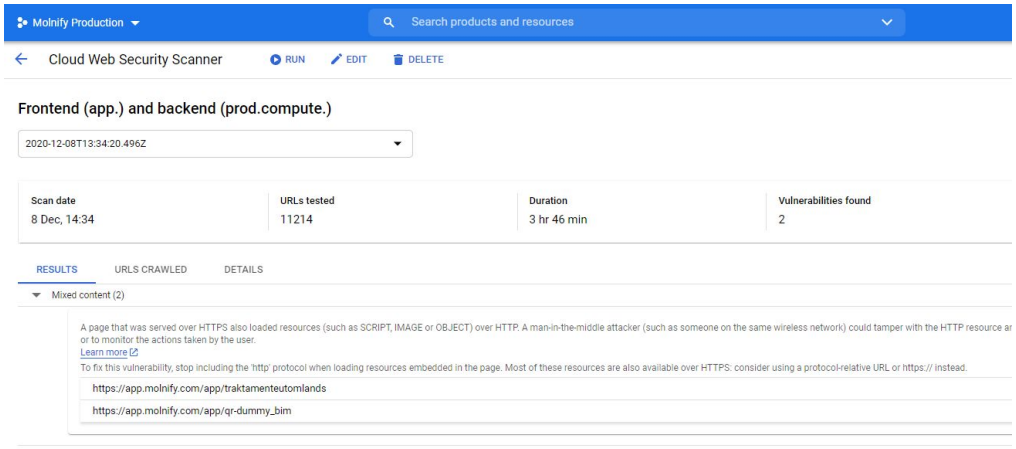

Exhibit 5 *Summary of a vulnerability scan run on December 8th 2020. Out of 11214 URLs tested with Google's standard attack vectors, two vulnerabilities were found - both relating to individual applications running on the platform that is that selected image content is served over HTTP while the application and all other resources are served over HTTPS. These two (customer) applications do not affect any other applications of the rest of the platform.*