Getting Started

/ Autonomous Identity 2020.6.4

Latest update: 2020.6.4

Abstract

This guide is targeted to administrators who must maintain, troubleshoot, and monitor the Autonomous Identity system.

Table of Contents

Overview

Use this guide to get a quick, hands-on look at Autonomous Identity. Download the image, install, and play around with the software.

ForgeRock® Autonomous Identity is an entitlements analytics system that lets you fully manage your company's access to your data.

An entitlement refers to the rights or privileges assigned to a user or thing for access to specific resources. A company can have millions of entitlements without a clear picture of what they are, what they do, and who they are assigned to. Autonomous Identity solves this problem by using advanced artificial intelligence (AI) and automation technology to determine the full entitlements landscape for your company. The system also detects potential risks arising from incorrect or over-provisioned entitlements that lead to policy violations. Autonomous Identity eliminates the manual re-certification of entitlements and provides a centralized, transparent, and contextual view of all access points within your company.

Important

This guide is for developers, technical consultants, and ForgeRock partners who are familiar with Autonomous Identity. The deployment example in this guide is for evaluation purposes only. For production deployments, see the Installation Guide.

Quick Start

Autonomous Identity Features Learn about the Autonomous Identity features.	Architecture in Brief Learn about the Autonomous Identity architecture.	How It Works Learn how Autonomous Identity works with a simple example.
Deploy Autonomous Identity on CentOS Deploy a lightweight version of Autonomous Identity on a CentOS 7 server.	Next Steps Learn where to go from here.	FAQ Read some FAQs on Autonomous Identity.

For installation instructions, see the Installation Guide.

For a description of the Autonomous Identity UI console, see the Users Guide.

Chapter 1

Features

Autonomous Identity provides the following features:

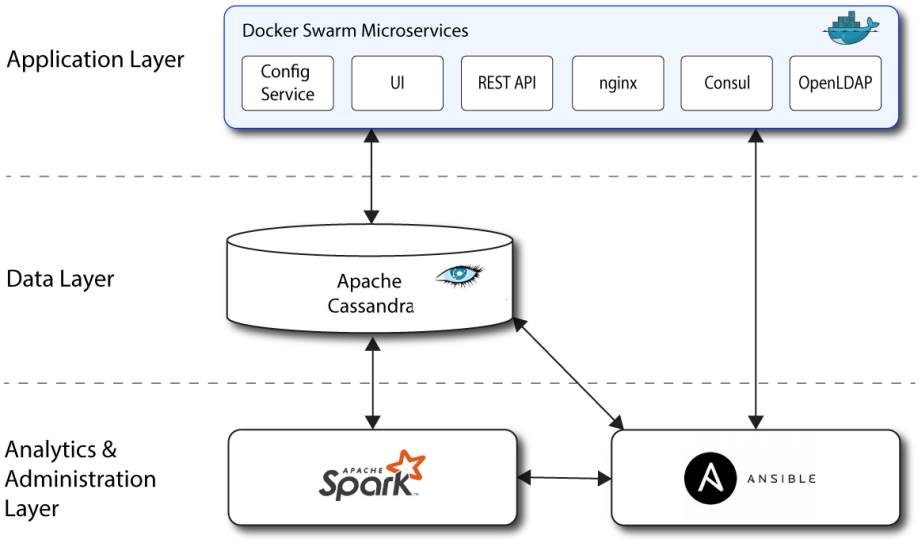
- **Broad Support for Major Identity Governance and Administration (IGA) Providers**. Autonomous Identity supports a wide variety of Identity as a Service (IDaaS) and Identity Management (IDM) data including but not limited to comma-separated values (CSV), Lightweight Directory Access Protocol (LDAP), human resources (HR), database, and IGA solutions.
- **Highly-Scalable Architecture**. Autonomous Identity deploys using a microservices architecture, either on-prem, cloud, or hybrid-cloud environments. Autonomous Identity's architecture scales linearly as the load increases.
- **Powerful UI dashboard**. Autonomous Identity displays your company's entitlements graphically on its UI console. You can immediately investigate those entitlement outliers that could potentially be a security risk. The UI also lets you quickly identify the entitlements that are good candidates for automated low-risk approvals or re-certifications. Users can also view a trend-line indicating how well they are managing their entitlements.
- **Automated Workflows**. Autonomous Identity reduces the burden on managers who must manually approve new entitlements, for example assigning access for new hires, by auto-approving high confidence, low-risk access requests and automate the re-certification of entitlements. Predictive recommendations lends itself well to automation, which saves time and cost.
- **Powerful Analytics Engine**. Autonomous Identity's analytics engine is capable of processing millions of access points within a short period of time. Autonomous Identity lets you configure the machine learning process and prune less productive rules. Customers can run analyses, predictions, and recommendations frequently to improve the machine learning process.
- **Powerful Explainable AI Algorithms**. The Analytics Engine provides transparent and explainable results that lets business users get insight into why the end-user has the access they have, or what access they should have.

Chapter 2
Architecture in Brief

The Autonomous Identity architecture has a simple three-layer conceptual model:

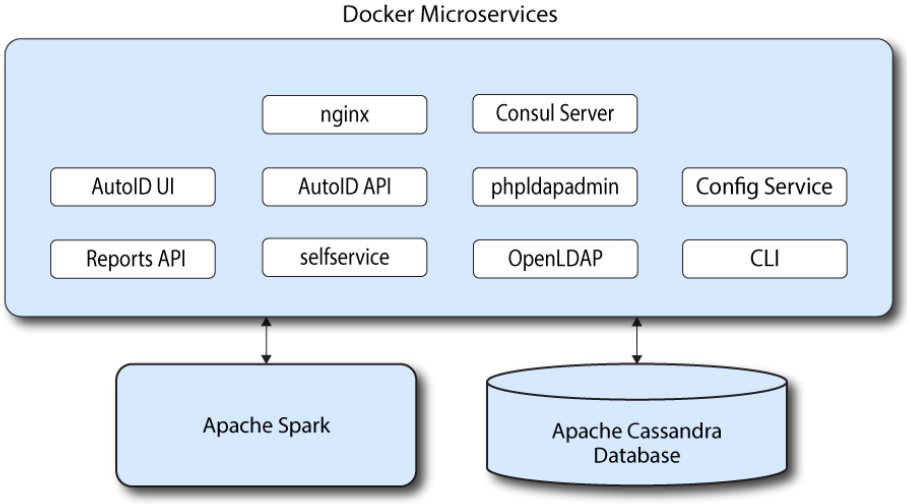
- **Application Layer**. Autonomous Identity implements a flexible Docker Swarm microservices architecture, where multiple applications run together in containers. The microservices component provides flexible configuration and end-user interaction to the deployment. The microservices components are the following:
  - **Autonomous Identity UI**. Autonomous Identity supports a dynamic UI that displays the entitlements, confidence scores, and recommendations.
  - **Autonomous Identity API**. Autonomous Identity provides an API that can access endpoints using REST. This allows easy scripting and programming for your system.
  - **Self-Service Tool**. The self-service tool lets users reset their Autonomous Identity passwords.
  - **Backend Repository**. The backend repository stores Autonomous Identity user information. To interface with the backend repository, you can use the **phpldapadmin** tool to enter and manage users.
  - **Configuration Service**. Autonomous Identity supports a configuration service that allows you to set parameters for your system and processes.
  - **Command-Line Interface**. Autonomous Identity supports a command-line interface for easy scripting and automation.
  - **Nginx**. Nginx is a popular HTTP server and reverse proxy for routing HTTPS traffic.
  - **Hashicorp Consul**. Consul is a third-party system for service discovery and configuration.
- **Data Layer**. Autonomous Identity uses a Apache Cassandra NoSQL database to serve predictions, confidence scores, and prediction data to the end user. Apache Cassandra is a distributed and linearly scalable database with no single point of failure.
- **Analytics and Administration Layer**. Autonomous Identity uses a multi-source Apache Spark analytics engine to generate the predictions and confidence scores. Apache Spark is a distributed, cluster-computing framework for AI machine learning for large datasets. Autonomous Identity also uses a deployer wrapper script to launch Ansible playbooks for easy and quick deployment of the components.

Figure 1: A Conceptual Image of the Autonomous Identity Architecture



Autonomous Identity's flexible architecture can deploy in any number of ways: single-node or multi-node configurations across on-prem, cloud, hybrid, or multi-cloud environments. For example, you can configure a two-server deployment with the minimum hardware and software requirements as shown below. Note that the example architecture is typically used for pilot deployments.

Figure 2: An Image of a Two-Server Autonomous Identity Pilot Architecture

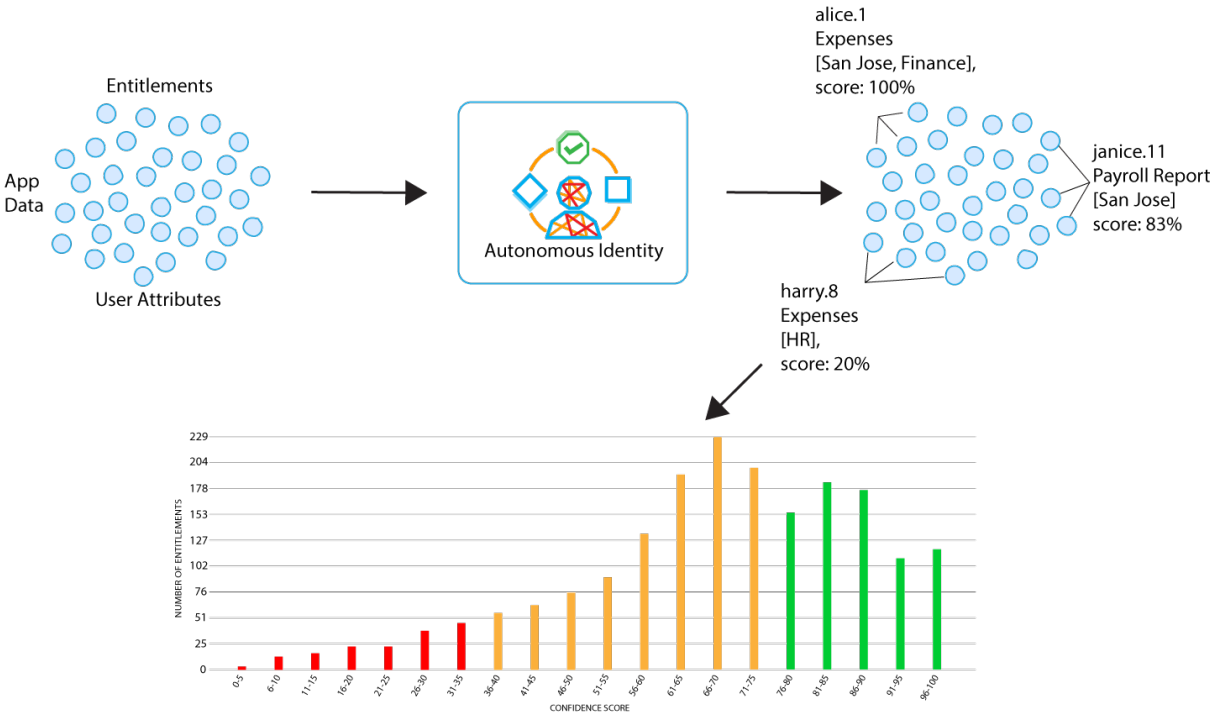


Chapter 3
How Autonomous Identity Works

Autonomous Identity is an AI-based analytics engine that discovers, analyzes, and generates a complete profile of your company's entitlements.

Autonomous Identity looks at each entitlement and its relationship to the assigned user within the company. These relationships are modelled and assigned a single confidence score (from 0 to 100%) indicating the strength of correlation between the model and the assigned entitlement. The results are displayed on the UI console.

Figure 2: A simple conceptual diagram of Autonomous Identity



Let's Run a Simple Example

Let's run a simple example to see how Autonomous Identity models the entitlements and calculates confidence scores. Each company can decide the level and scope of the analysis. However, in most cases, the more data you analyze, the better the entitlement models.

Autonomous Identity processes the data in the following steps:

1. "Data Ingestion"
2. "Training"
3. "As-is Predictions"
4. "Recommendations"
5. "Output to the UI Console"

Data Ingestion

The process begins with data ingestion, where data is exported from application services, HR databases, and other sources and imported into Autonomous Identity using comma-separated values (.csv) formatted files. There are three general types of data files needed for analytics. Note that, in actual deployments, you set up seven data files consisting of HR data, entitlement data, application data and other mapping files to export to Autonomous Identity.

- **User Profile Data**. Users profile data include those attributes that contain a subset of information from an HR system that is minimally required for entitlement analysis. Each user has a unique user ID and is represented in a single row within the file. Typically, the number of useful user profile attributes for successful modeling has been found to be between 7#15.

  The following table shows a simple example of user profile data:

Table 1: User Attributes

USER ID	CITY	DEPT
alice.1	San Jose	Finance
bob.2	San Jose	Finance
chris.3	San Jose	Finance
diane.4	San Jose	HR
ellen.5	San Jose	HR
fred.6	Austin	HR
gary.7	Austin	HR
harry.8	Austin	HR

USER ID	CITY	DEPT
irene.9	Dublin	IT
janice.10	Dublin	Finance
karen.11	San Jose	Finance

- **Entitlement Data**. In another file, you import a list of entitlement data and any other descriptive information. The file may include information, such as full entitlement name, application name, entitlement or role owner, or any other information that helps with the machine learning.

  The following table shows a simple example of entitlement data:

Table 2: Entitlement Data

Entitlement	APP_NAME	APP_ID
Payroll Report	ADP	adp-1
Expenses	Expensify	expensify-1

- **Access Data**. The third type of data is a mapping file that connects the user to their assigned entitlement. The data can come from your IAM/IGA system. If a user has multiple entitlements, each row represents a single assigned mapping to each entitlement. Some examples could have user ID, entitlement name, flag if it is a high risk entitlement, another flag if the entitlement is assigned, and last usage.

  The following table shows a simple example of user-to-entitlement data:

Table 3: User-to-Entitlement Mapping

USR_KEY	ENT	HIGH_RISK	LAST_USAGE
alice.1	Payroll Report	High	2020-06-06 17:11:07
alice.1	Expenses	High	2020-06-06 17:11:07
bob.2	Payroll Report	High	2020-06-06 17:11:07
bob.2	Expenses	High	2020-06-06 17:11:07
chris.3	Payroll Report	High	2020-06-06 17:11:07
chris.3	Expenses	High	2020-06-06 17:11:07
diane.4	Payroll Report	High	2020-06-06 17:11:07
ellen.5	Payroll Report	High	2020-06-06 17:11:07
fred.6	Payroll Report	High	2020-06-06 17:11:07
gary.7	Expenses	High	2020-06-06 17:11:07
harry.8	Payroll Report	High	2020-06-06 17:11:07
irene.9	Expenses	High	2020-06-06 17:11:07

USR_KEY	ENT	HIGH_RISK	LAST_USAGE
janice.10	Payroll Report	High	2020-06-06 17:11:07
karen.11	Expenses	High	2020-06-06 17:11:07

Training

Next, Autonomous Identity runs a two-stage training process to generate the association rules and confidence scores. An association rule is an IF-THEN rule that expresses patterns between random data variables in a large transaction set. For example, [San Jose, Finance] -> [Payroll Report] indicates that if a company's finance office is located in San Jose and a person works in that office and department, it is likely that they get access to the Payroll Report.

During stage one of the training, Autonomous Identity analyzes the user attribute data using machine learning algorithms to pattern-mine and create itemsets of rules. The frequency of occurrence is counted for each itemset. Only rules that appear three times or more are considered. Itemsets less than three are ignored.

Note

In a typical deployment, Autonomous Identity can create a million or more association rules for a company's dataset.

The following table shows the results of the initial training process:

Table 4: Itemsets of User Data

Itemset	Freq
[San Jose]	6
[Finance]	5
[HR]	5
[San Jose, Finance]	4
[Austin]	3
[Austin, HR]	3

During this training run, the analytics engine creates a unique row in a table for each user and their assigned entitlements. In the example below, alice.1, bob.2, and chris.3 have multiple rows, one for each assigned entitlement. Again, only frequency sets (freqUnion) of three or more are considered.

The following table shows the results of the second training process:

Table 5: Training Stage 2

USER ID	CITY	DEPT	ENT
alice.1	San Jose	Finance	Payroll Report
alice.1	San Jose	Finance	Expenses
bob.2	San Jose	Finance	Payroll Report
bob.2	San Jose	Finance	Expenses
chris.3	San Jose	Finance	Payroll Report
chris.3	San Jose	Finance	Expenses
diane.4	San Jose	HR	Payroll Report
ellen.5	San Jose	HR	Payroll Report
fred.6	Austin	HR	Payroll Report
gary.7	Austin	HR	Expenses
harry.8	Austin	HR	Payroll Report
irene.9	Dublin	IT	Expenses
janice.10	Dublin	Finance	Payroll Report
karen.11	San Jose	Finance	Expenses

Autonomous Identity applies the association rules to the entitlement mappings and calculates the risk confidence scores by dividing the freqUnion by frequency numbers (FreqUnion/Freq). The FREQ column show the number of occurrences of a rule from Table 5, for example, the rule [San Jose] appears 9 times. The FreqUnion is the union of a rule with an entitlement, for example, the union of the rule [San Jose] with the entitlement, Expenses, appears 4 times in Table 5. The confidence score indicates the scale from 0 to 100% to indicate the strength of each correlation. A confidence score of 100% indicates that the assigned entitlement is highly correlated to the user's job function. Only rules that appear three times or more are considered.

The following table shows the applied association rules:

Table 6: Applied Association Rules to Entitlements

RULE	ENT	FREQ	FreqUnion	Confidence
[San Jose]	Payroll Report	9	5	56%
[San Jose]	Expenses	9	4	44%
[Finance]	Payroll Report	8	4	50%
[Finance]	Expenses	8	4	50%
[HR]	Payroll Report	5	4	80%
[San Jose,Finance]	Payroll Report	7	3	43%

RULE	ENT	FREQ	FreqUnion	Confidence
[San Jose,Finance]	Expenses	7	4	57%

Next, Autonomous Identity must re-adjust the frequency numbers from the previous table as the occurrences of a rule are inflated due to multiple appearances of a user's entitlements (that is, one entitlement per row) as seen in Table 5. The re-adjustment provides a more accurate confidence score for each association rule.

The following are the results of the readjusted confidence scores:

Table 7: Applied Association Rules to Entitlements

RULE	ENT	FREQ (Corrected)	FreqUnion	Confidence
[San Jose]	Payroll Report	6	5	83%
[San Jose]	Expenses	6	4	67%
[Finance]	Payroll Report	5	4	80%
[Finance]	Expenses	5	4	80%
[HR]	Payroll Report	5	4	80%
[San Jose,Finance]	Payroll Report	4	3	75%
[San Jose,Finance]	Expenses	4	4	100%

As-is Predictions

After the training process has determined the association rules for each entitlement, the analytics engine runs through an *as-is predictions* process, where user accesses are mapped to each entitlement using the association rules.

In the example, the user `alice.1` has the following mapped entitlements, which are called *justifications* for their entitlement accesses.

The following table shows the results of the as-is-predictions process:

Table 8: Initial As-is Predictions for a User

USER ID	ENT	RULE	Confidence	FreqUnion
alice.1	Payroll Report	[San Jose]	83%	5
alice.1	Payroll Report	[Finance]	80%	4
alice.1	Payroll Report	[San Jose, Finance]	75%	3
alice.1	Expenses	[San Jose]	67%	4
alice.1	Expenses	[Finance]	80%	4
alice.1	Expenses	[San Jose,Finance]	100%	4

The as-is predictions filter the justifications from the previous step using confidence score properties that are set in the configuration file. The maximum confidence score is set by the `maxConf` property. The minimum confidence score is set by the `maxConf` minus the `pred_conf_window` property, which is set to 5% in the configuration file by default. Thus, for this example, the maximum confidence and minimum confidence score filters for each entitlement is as follows:

The following table shows the result of the as-is-predictions:

Table 9: As-is Predictions Filter for a User

ENT	maxConf	Min
Payroll Report	83%	78%
Expenses	100%	95%

Applying the filters in Table 9 to the mapped entitlements in Table 8, we get the following filtered assigned entitlements while discarding the rest. These filters are applied to all users in your analysis.

The following table shows the results for alice.1:

Table 10: Filtered As-is Predictions for a User

USER ID	ENT	RULE	Confidence	FreqUnion
alice.1	Payroll Report	[San Jose]	83%	5
alice.1	Payroll Report	[Finance]	80%	4
alice.1	Expenses	[San Jose, Finance]	100%	4

Finally, the highest `freqUnion` is used to find the users with a specific rule and entitlement access. All rules with the lower `freqUnion` values are filtered out to favor rules that apply to the largest number of employees within a company. This ensures that the most generalized rules are used for the analysis.

The following table shows the final as-is predictions for a user:

Table 11: Final As-is Predictions for a User

USER ID	ENT	RULE	Confidence	FreqUnion
alice.1	Payroll Report	[San Jose]	83%	5
alice.1	Expenses	[San Jose, Finance]	100%	4

Recommendations

The analytics process goes through a recommendations predictions process that takes the entitlement rules and identifies any users who should have access to the entitlement but do not. The analytics engine looks at each user's confidence score associated with the entitlement and if the confidence score exceeds a pre-configured hreshold value, the recommendation are made for the user.

The process begins by assigning the entitlements to all users and removing already existing accesses. Autonomous Identity assigns the rules and confidence scores for these new assignments.

The following table shows the recommendations assigned to users who do not have a particular entitlement:

Table 12: Recommendations

USER ID	ENT	RULE	Confidence
diane.4	Expenses	[San Jose]	67%
ellen.5	Expenses	[San Jose]	67%
fred.6	Expenses	[San Jose]	67%
gary.7	Payroll Report	[HR]	80%
harry.8	Expenses	[HR]	20%
irene.9	Payroll Report	no rule found	0%
janice.10	Expenses	[Finance]	80%
janice.11	Payroll Report	[San Jose]	83%

The analytics engine determines the rules and confidence scores that meet a threshold property, `conf_thresh`, which is set to 80% in the configuration file by default.

The following example shows the final recommendations:

Table 13: Final Recommendations

USER ID	ENT	RULE	Confidence
gary.7	Payroll Report	[HR]	80%
janice.10	Expenses	[Finance]	80%
janice.11	Payroll Report	[San Jose]	83%

The results will be uploaded to the Cassandra database as a recommended new entitlement and appears on the UI console on the Recommendations screen.

Output to the UI Console

The final step of the process is for Autonomous Identity to display the confidence scores graphically on the UI as a distribution from low, medium, to high scores. The console lets you immediately identify the low confidence scores that could pose a potential security risk as well as the high confidence scores that can be automatically approved or certified. Autonomous Identity displays the attributes that justified each confidence score as well as other data to help you manage your entitlements.

You can run the analytics weekly or monthly to ensure near realtime assessment of your entitlements. This ensures that some entitlements can immediately be flagged if it goes stale and is no longer necessary.

Chapter 4

Deploy a Lightweight Autonomous Identity

Autonomous Identity supports a flexible and scalable microservices architecture that you can deploy in any type of network environment: on-prem, cloud, multi-cloud, or hybrid cloud.

The deployment process is simple. Autonomous Identity provides a deployer script that you launch on a local server or laptop. The deployer pulls down the necessary images from the ForgeRock Google Cloud Repository (gcr.io) to install Autonomous Identity on the target node. From there, you run a few commands to complete the installation.

Important

To download the deployer, you must obtain a registry key to access the ForgeRock Google Cloud Repository. Only ForgeRock Autonomous Identity customers can obtain a registry key. For more information, contact ForgeRock.

For this example, we install a lightweight version of Autonomous Identity in a two-core 8GB virtual machine on a cloud platform, such as Google Cloud Platform (GCP), Amazon Web Services (AWS), Microsoft Azure, or others. The deployment includes the Docker-based microservices component and Apache Cassandra, but not Apache Spark. From there, you populate demo data by importing it into Cassandra and then display it on the Autonomous Identity UI console. This setup provides a way to deploy a system quickly for demonstration purposes.

This installation assumes that you set up the deployer script on a separate machine from the target. This lets you launch a build from a laptop or local server.

A simple diagram is shown below:

Figure 1: A lightweight single-node target deployment.

Important

This example setup is only for development or evaluation purposes and should not be used for production deployments.

Let's set up Autonomous Identity on CentOS 7. The following are some prerequisites:

- **Operating System**. The target machine requires CentOS 7. The deployer machine can use any operating system as long as Docker is installed. For this guide, we use CentOS 7 as its base operating system.

- **Memory Requirements**. Make sure you have enough free disk space on the deployer machine before running the `deployer.sh` commands. We recommend at least a 40GB/partition with 14GB used and 27GB free after running the commands.

- **Registry Requirements**. Autonomous Identity provides a Docker image that creates a `deployer.sh` script. The script downloads additional images necessary for the installation. To download the deployment images, you must first obtain a registry key to log into the ForgeRock Google Cloud Registry (gcr.io). The registry key is only available to ForgeRock Autonomous Identity customers. For specific instructions on obtaining the registry key, see How To Configure Service Credentials (Push Auth, Docker) in Backstage.

Set Up the Target Machine

1. The install assumes that you have CentOS 7 as your operating system. Check your CentOS 7 version.

   ```
   $ sudo cat /etc/centos-release
   ```

2. Set the user for the target machine to a username of your choice. For example, `autoid`.

   ```
   # sudo adduser autoid
   ```

3. Set the password for the user you created in the previous step.

   ```
   $ sudo passwd autoid
   ```

4. Configure the user for passwordless sudo.

   ```
   $ echo "autoid    ALL=(ALL)  NOPASSWD:ALL" | sudo tee /etc/sudoers.d/autoid
   ```

5. Add administrator privileges to the user.

   ```
   $ sudo usermod -aG wheel autoid
   ```

6. Change to the user account.

   ```
   $ su - autoid
   ```

7. Install yum-utils package on the deployer machine. yum-utils is a utilities manager for the Yum RPM package repository. The repository compresses software packages for Linux distributions.

   ```
   $ sudo yum install -y yum-utils
   ```

8. Create the installation directory. Note that you can use any install directory for your system as long as your run the **deployer.sh** script from there. Also, the disk volume where you have the install directory must have at least 8GB free space for the installation.

   ```
   $ mkdir ~/autoid-config
   ```

Set Up the Deployer Machine

Set up another machine as a deployer node. You can use any OS-based machine for the deployer as long as it has Docker installed. For this example, we use CentOS 7.

1. The install assumes that you have CentOS 7 as your operating system. Check your CentOS 7 version.

```
$ sudo cat /etc/centos-release
```

2. Set the user for the target machine to a username of your choice. For example, `autoid`.

```
# sudo adduser autoid
```

3. Set the user password for the user you created in the previous step.

```
$ sudo passwd autoid
```

4. Configure the user for passwordless sudo. Replace "autoid" with the username for your system.

```
$ echo "autoid    ALL=(ALL)  NOPASSWD:ALL" | sudo tee /etc/sudoers.d/autoid
```

5. Add administrator privileges to the user.

```
$ sudo usermod -aG wheel autoid
```

6. Change to the user account.

```
$ su - autoid
```

7. Install yum-utils package on the deployer machine. yum-utils is a utilities manager for the Yum RPM package repository. The repository compresses software packages for Linux distributions.

```
$ sudo yum install -y yum-utils
```

8. Create the installation directory. Note that you can use any install directory for your system as long as your run the **deployer.sh** script from there. Also, the disk volume where you have the install directory must have at least 8GB free space for the installation.

```
$ mkdir ~/autoid-config
```

Install Docker on the Deployer Machine

Install Docker on the deployer machine. We run commands from this machine to install Autonomous Identity on the target machine. In this example, we use CentOS 7.

1. On the target machine, set up the Docker-CE repository.

```
$ sudo yum-config-manager \
    --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

2. Install the latest version of the Docker CE, the command-line interface, and containerd.io, a containerized website.

```
$ sudo yum install -y docker-ce docker-ce-cli containerd.io
```

3. Enable Docker to start at boot.

```
$ sudo systemctl enable docker
```

4. Start Docker.

```
$ sudo systemctl start docker
```

5. Check that Docker is running.

```
$ systemctl status docker
```

6. Add the user to the Docker group.

```
$ sudo usermod -aG docker ${USER}
```

Set Up SSH on the Deployer

1. On the deployer machine, run **ssh-keygen** to generate an RSA keypair, and then click Enter. You can use the default filename. Enter a password for protecting your private key.

```
$ ssh-keygen -t rsa -C "autoid"
```

   The public and private rsa key pair is stored in `home-directory/.ssh/id_rsa` and `home-directory/.ssh/id_rsa.pub`.

2. Copy the SSH key to the `autoid-config` directory.

```
 $ cp ~/.ssh/id_rsa ~/autoid-config
```

3. Change the privileges to the file.

```
$ chmod 400 ~/autoid-config/id_rsa
```

4. Copy your public SSH key, `id_rsa.pub`, to the target machine's `~/.ssh/authorized_keys` file. If your system does not have an `/authorized_keys` directory, create it using **mkdir -p ~/.ssh/authorized_keys**.

5. On the target machine, set the privileges on your `~/.ssh` and `~/.ssh/authorized_keys`.

```
$ chmod 700 ~/.ssh && chmod 600 ~/.ssh/authorized_keys
```

6. On the deployer machine, test your SSH connection to the target machine. This is a critical step. Make sure the connection works before proceeding with the installation.

```
$ ssh -i ~/.ssh/id_rsa autoid@34.70.190.144
Last login: Sun Jun 14 23:23:36 2020 from 74.125.45.78
```

7. If you successfully accessed the remote server, enter **exit** to end your SSH session.

Install Autonomous Identity

1. On the deployer machine, change to the installation directory.

```
$ cd ~/autoid-config/
```

2. Log in to the ForgeRock Google Cloud Registry (gcr.io) using the registry key. The registry key is only available to ForgeRock Autonomous Identity customers. For specific instructions on obtaining the registry key, see How To Configure Service Credentials (Push Auth, Docker) in Backstage.

```
$ docker login -u _json_key -p "$(cat autoid_registry_key.json)" https://gcr.io/forgerock-autoid
```

3. Run the **create-template** command to generate the `deployer.sh` wrapper. Note that the command sets the configuration directory on the target node to `/config`. The **--user** parameter eliminates the need to use **sudo** while editing the hosts file and other configuration files.

```
$ docker run --user=`id -u` -v ~/autoid-config:/config -it gcr.io/forgerock-autoid/deployer:2020.6.2
 create-template
```

4. Make the script executable.

```
$ chmod +x deployer.sh
```

5. The **create-template** command creates a number of configuration files, including `ansible.cfg`. Open a text editor and edit the `ansible.cfg` to set up the remote user and SSH private key file location on the target node. Make sure that the remote_user exists on the target node.

```
[defaults]
host_key_checking = False
remote_user = autoid
private_key_file = id_rsa
```

6. Open a text editor and enter the target host's public IP addresses in the `~/autoid-config/hosts` file. Make sure to keep the Spark server entries blank as Spark is excluded from the deployment. The following is an example of the `~/autoid-config/hosts` file:

```
[docker-managers]
35.232.107.121

[docker-workers]
35.232.107.121

[docker:children]
docker-managers
docker-workers

[cassandra-seeds]
35.232.107.121

[cassandra-workers]
35.232.107.121

[spark-master]

[spark-workers]

[analytics]
```

7. If your external IP and internal IP addresses are different, for example, when deploying the target host in a cloud, define a mapping between the external IP address and the private IP address.

   On the deployer node, add the `private_ip_address_mapping` property in the `~/autoid-config/vars.yml` file. You can look up the private IP on the cloud console, or run **sudo ifconfig** on the target host. Make sure the values are within double quotes. The key should not be in double quotes and should have two spaces preceding the IP address.

   ```
   private_ip_address_mapping:
     external_ip:  "internal_ip"
   ```

   For example:

   ```
   private_ip_address_mapping:
     34.72.28.214:  "10.128.0.52"
   ```

8. Open a text editor to see the Autonomous Identity passwords, located at `~/autoid-config/vault.yml`.

   ```
   configuration_service_vault:
     basic_auth_password: Welcome123

   openldap_vault:
     openldap_password: Welcome123

   cassandra_vault:
     cassandra_password: Welcome123
     cassandra_admin_password: Welcome123
   ```

9. Encrypt the vault file that stores the Autonomous Identity passwords, located at `~/autoid-config/vault.yml`. Make note of the vault password. You will be asked to enter it when running the **encrypt-vault** command.

   ```
   $ ./deployer.sh encrypt-vault
   ```

The encrypted passwords are saved to `/config/.autoid_vault_password`. The `/config/` mount is internal to the deployer container.

10. Download the images. This step downloads software dependencies needed for the deployment.

```
$ ./deployer.sh download-images
```

11. Run the deployer. This step builds the Docker microservices and Cassandra servers.

```
$ ./deployer.sh run
```

Hostname Resolution

- On the target machine, open a text editor and add an entry in the `/etc/hosts` file for the Autonomous Identity self-service and UI services.

  For example:

```
34.70.190.144  autoid-ui.forgerock.com autoid-selfservice.forgerock.com
```

Check the System

1. Open a browser, and point it to `https://autoid-ui.forgerock.com/`.
2. Log in as a test user: **`bob.rodgers@forgerock.com`**. Enter the password for the user. On the Autonomous Identity UI console, notice that there is no data on the system yet. Logout when done.
3. Check the status of Apache Cassandra.

```
$ /opt/autoid/apache-cassandra-3.11.2/bin/nodetool status
```

```
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.128.0.6  624.21 KiB  256          100.0%            0a13aa7f-ea2e-43b2-9323-136bdec915d2  rack1
```

4. Enter **exit** to end your SSH session.

Set Up dbutils

- On the deployer machine, run the **install-dbutils** command. This command creates a `dbutils` directory in the target instance's Cassandra seed node.

```
$ ./deployer.sh install-dbutils
```

Import the Demo Data

If you are setting up a fresh install using the demo data provided by ForgeRock, follow these steps.

1. Contact ForgeRock to obtain the demo data.
2. On the target machine, change to the `dbutils` directory.

   ```
   $ cd /opt/autoid/dbutils
   ```

3. Use the **dbutils.py import** command to populate the Autonomous Identity keyspace with the demo data.

   ```
   $ python dbutils.py import /import-dir
   ```

   For example:

   ```
   $ python dbutils.py import ~/import/AutoID-data
   ```

4. Open a browser, and point it to `https://autoid-ui.forgerock.com/`.
5. Log in as a test user: **`bob.rodgers@forgerock.com`**. Enter the password for the user: `Welcome123`. You should see data populated on the Autonomous Identity UI.

Chapter 5
Next Steps

Once you have Autonomous Identity running and run the demo data, you can do the following:

- View the UI Console

  You can view the various page views of the Autonomous Identity UI. For more information, see the Users Guide.

- Manage Users

  You can add, remove, and change any passwords to Autonomous Identity. For more information, see the "*Creating and Removing Users*" in the *Admin Guide*.

- Using the Autonomous Identity API

  Autonomous Identity provides API for developers who want to create scripts or applications to access the endpoints. For more information, see the API Guide.

- Browse the Documentation

  The following documentation is available for this release:

  - Autonomous Identity Installation Guide
  - Autonomous Identity Administration Guide
  - Autonomous Identity API Guide
  - Autonomous Identity Users Guide
  - Autonomous Identity Release Notes

Chapter 6
Frequently Asked Questions

6.1. How is Autonomous Identity different to peer group analytics?

Peer group analysis compares a user to their peer group to identify whether they have any accesses that may be anomalous, relative to that peer group. Autonomous Identity differs from peer group comparisons in that it compares users in the same department, with the same job title, or with every single person in the company. This granular approach provides a more comprehensive and global view of your entitlements. Autonomous Identity can therefore identify many potential patterns that may be missed with peer group analysis.

6.2. Can we weight an attribute within the features file to be more important and influence the resulting confidence scores?

No. Autonomous Identity is entirely data-driven. All user attributes are equally weighted. This means that it is possible to have association rules that do not lead to an entitlement assignment, such as [Expenses, Finance] -> San Jose. In these cases, the rules are discarded. In general, weighted association rules would negatively impact the analytics results and thus are not implemented.

Glossary

anomaly report	A report that identifies potential anomalous assignments.
as-is predictions	A process where confidence scores are assigned to the entitlements that users have.
confidence score	A score from a scale from 0 to 100% that indicates the strength of correlation between an assigned entitlement and a user's data profile.
data audit	A pre-analytics process that audits the seven data files to ensure data validity with the client.
data ingestion	A pre-analytics process that pushes the seven .csv files into the Cassandra database. This allows the entire training process to be performed from the database.
data sparsity	A reference to data that has null values. Autonomous Identity requires dense, high quality data with very few null values in the user attributes to get accurate analysis scores.
data validation	A pre-analytics process that tests the data to ensure that the content is correct and complete prior to the training process.
driving factor	An association rule that is a key factor in a high entitlement confidence score. Any rule that exceeds a confidence threshold level (e.g., 75%) is considered a driving factor.
entitlement	An entitlement is a specialized type of `assignment`. A user or device with an entitlement gets access rights to specified resources.
insight report	A report that provides metrics on the rules and predictions generated in the analytics run.

recommendation	A process run after the as-is predictions that assigns confidence scores to all entitlements and recommends entitlements that users do not currently have. If the confidence score meets a threshold, set by the `conf_thresh` property in the configuration file, the entitlement will be recommended to the user in the UI console.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by an identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
stemming	A process that occurs after training that removes similar association rules that exist in a parent-child relationship. If the child meets three criteria, then it will be removed by the system. The criteria are: 1) the child must match the parent; 2) the child (e.g., [San Jose, Finance]) is a superset of the parent rule. (e.g., [Finance]); 3) the child and parent's confidence scores are within a +/- range of each other. The range is set in the configuration file.
training	A multi-step process that generates the association rules with confidence scores for each entitlement. First, Autonomous Identity models the frequent itemsets that appear in the user attributes for each user. Next, Autonomous Identity merges the user attributes with the entitlements that were assigned to the user. It then applies association rules to model the sets of user attributes that result in an entitlement access and calculates confidence scores, based on their frequency of appearances in the dataset.