

Documentation

on

**PingAccess**

**Submitted By:** Chirag Gaur

**Document Information**

|  |  |
| --- | --- |
| **Document Name** | Ping Access |
| **Classification** | For Deloitte use only |
| **Version** | 1.1 |
| **Effective Date** | 13-04-2022 |
| **Document Owner** | Chirag Gaur |
| **Status** | Draft |

**History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Revision Authors** | **Notes** |
| 12-04-2022 | 1.0 | Chirag Gaur | Draft |
| 13-04-2022 | 1.1 | Chirag Gaur | Updated the Document |

**Table of Contents**

[1. What is OAuth? 3](#_Toc100331629)

[2. OAuth Versions 3](#_Toc100331630)

[3. Principle of OAuth 2.0 3](#_Toc100331631)

[4. Understanding OAuth with an example 4](#_Toc100331632)

[5. OAuth 2.0 Terminologies 4](#_Toc100331633)

[6. How OAuth 2.0 Works? 5](#_Toc100331634)

[7.1 Authorization Code Flow 7](#_Toc100331635)

[7.2 Implicit Flow 8](#_Toc100331636)

[8. Problems with OAuth 2.0 for authentication 9](#_Toc100331637)

[9. OpenID Connect 9](#_Toc100331638)

[10. Introduction to Ping Access 9](#_Toc100331639)

[11. Ping Access V/s Ping Federate 9](#_Toc100331640)

[12. Why Ping Access? 10](#_Toc100331641)

[13. Why Ping Access is a Policy Engine? 10](#_Toc100331642)

[14. Terminologies in Ping Access 10](#_Toc100331643)

[15. How Ping Access Works? 11](#_Toc100331644)

[16. Gateway V/s Agent Based Deployment 12](#_Toc100331645)

[16.1 Gateway 12](#_Toc100331646)

[16.2 Agents 12](#_Toc100331647)

[17. Appendix-1: Ping Access Installation 13](#_Toc100331648)

[17.1 Setting up a Virtual Machine 13](#_Toc100331649)

[17.2 Downloading JAVA 13](#_Toc100331650)

[17.3 Downloading License Files 14](#_Toc100331651)

[17.4 Ping Access Installation 15](#_Toc100331652)

**List of Figures**

[Figure 1: Example of Valet Parking System 4](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780388)

[Figure 2: OAuth 2.0 Complete flow 5](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780389)

[Figure 3: Request sent from client to authorization server 6](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780390)

[Figure 4: Authorization code sent by authorization server 6](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780391)

[Figure 5: Exchange token is returned in place of code 6](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780392)

[Figure 6: Client makes an API call to resource server through access token. 7](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780393)

[Figure 7: Implicit flow of OAuth 2.0 8](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780394)

[Figure 8: OpenID Connect and OAuth 2.0 relation 9](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780395)

[Figure 9: Working of Ping Access 11](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780396)

[Figure 10: Downloading JAVA for Ubuntu 13](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780397)

[Figure 11: Downloading License files 14](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780398)

[Figure 12: Downloading License Keys 14](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780399)

[Figure 13: Download the Product distribution ZIP file 15](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780400)

[Figure 14: Extract the ZIP file 15](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780401)

[Figure 15: Adding JAVA\_HOME path 16](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780402)

[Figure 16: Running chmod command 16](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780403)

[Figure 17: Running the Ping Access 16](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780404)

[Figure 18: Importing License Files 16](https://amedeloitte-my.sharepoint.com/personal/chigaur_deloitte_com/Documents/Desktop/Reports/PingAccess.docx#_Toc100780405)

# What is OAuth?

* Delegation Authorization problem lead to the discovery OAuth protocol. Delegation Authorization is an ability for a service or application to gain access to a user's resources on their behalf.
* OAuth stands for Open Authorization. It is a standard protocol which allows an application or website to have access to resources which are hosted by some other application.
* It allows a client application to perform some actions on the resources on behalf on user without having user’s credentials.
* OAuth is an authorization standard and not an authentication one. It was designed for authorization between services

# OAuth Versions

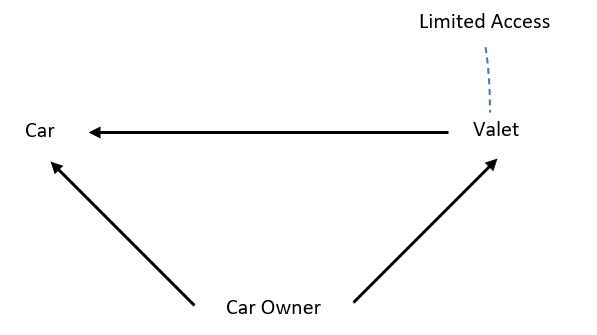
There are two versions of OAuth

1. OAuth 1.0: It was the earlier version of OAuth and used Initially by Google.
2. OAuth 2.0: It is the latest and most widely used version.

# Principle of OAuth 2.0

* The principle of OAuth 2.0 is based on authorization. OAuth is designed for granting access to a set of resources.
* OAuth uses access tokens to perform authorization. Access token is a data which allows to perform authorization to gain access of resource on behalf of the end user.

# Understanding OAuth with an example



Ex: Valet Parking System

In valet parking system, when a car owner hands over the car to valet then the car is given with limited access as key given to valet is a valet key which can only handle basics functionalities of car. The main key remains with car owner known as Master key.

Figure 2: Example of Valet Parking System

OAuth also works in a similar way where services can authorize each other on behalf of the user.

# OAuth 2.0 Terminologies

1. Resource: It is something that is protected and needs to be accessed by some different service. It is also referred as protected resource.
2. Resource Owner: It is an entity who has the access to the resource.

* An entity capable of granting access to a protected resource.
* Generally, user is the resource owner.

1. Resource Server: It is the Server that is hosting the Resource.
2. Client: It is an application making protected resource requests on behalf of the resource owner and with its authorization
3. Authorization Server: Resource server is coupled with an Authorization Server. Authorization Server is responsible for making sure that whoever is accessing the resource server is authorized. Authorization server receives the request for access token from client and after successful authentication, it grants them the access token.

1. Redirect URI: Authorization server redirects the user to a location once application has been successfully authorized and granted an authorization code using URL known as Redirect URI. This process is called as Callback.
2. Access Tokens: Access token is a piece of some data which allows to perform authorization to gain access of resource.
3. Scope: Scopes are list of permissions requested from client side. Scopes are sent from client to authorization server while sending the request to access the resource. They are used to specify the type of access (i.e. read only access or read and modify etc.) to be granted to client. Multiple scopes can also be sent.
4. Consent: Consent is used by Authorization server to user asking whether client can do the actions mentioned in the scope.
5. Back channel: It is a Highly secure channel and used to send request from user’s server to other API server from backend. No one can decrypt the information.
6. Front channel: It is Less secure channel and used to send request from browser. As browser might have loopholes like putting secret key in web app inside html then one can see it using view source with inspect element or with chrome developer tools etc.). It is mainly used for interaction with user.

# How OAuth 2.0 Works?

Timeline

Description automatically generated

Figure 3: OAuth 2.0 Complete flow

Let’s understand the OAuth 2.0 standard with the help of a use case.

1. Resource owner (i.e. user) will be at Client website and for our use case let’s assume the client website to be **yelp.com**. Now this website yelp.com needs to access the contacts present in Gmail (Google Accounts).
2. When Resource owner selects the option **‘Connect with Google’** in client website, it is redirected to Authorization server (accounts.google.com be the Authorization Server) and While Redirecting, configuration such as Redirect URL, Response type and Scope is passed from Client to Authorization Server.

Text

Description automatically generated

Figure 4: Request sent from client to authorization server

1. Authorization Server requests consent from resource owner to either grant or deny the permission.



1. If consent is granted from the resource owner, then Authorization server redirects to an URL known as Redirect URL along with authorization code.

Graphical user interface, text

Description automatically generated

Figure 5: Authorization code sent by authorization server

1. Client goes back to authorization server and exchanges this code for access tokens.

Text

Description automatically generated

Figure 6: Exchange token is returned in place of code

1. Graphical user interface, application

   Description automatically generatedClient contacts the resource server with access token that contains the information regarding actions (read only) and after successful validation of access tokens by the resource server, client can access the requested resource.

Figure 7: Client makes an API call to resource server through access token.

**Note:** Authorization Code comes back to redirect URI over the browser (browser is a front channel). Since someone can interpret the code as front channel is less secured. Exchange request of access token happens from back channel as it is safe. Thus, token exchange request is sent from one backend server to authorization server which also contains secret key that only backend know so even if hacker has Authorization code, they don’t have the secret key.

1. **OAuth Flows**

## Authorization Code Flow

The Authorization Code grant type is used by web and mobile apps. It differs from most of the other grant types by first requiring the app launch a browser to begin the flow. At a high level, the flow has the following steps:

* The application opens a browser to send the user to the OAuth server
* The user sees the authorization prompt and approves the app’s request
* The user is redirected back to the application with an authorization code in the query string
* The application exchanges the authorization code for an access token

## Timeline Description automatically generatedImplicit Flow

Figure 8: Implicit flow of OAuth 2.0



* The Implicit flow was a simplified OAuth flow previously recommended for native apps and JavaScript apps where the access token was returned immediately without an extra authorization code exchange step.
* It is not recommended to use the implicit flow (and some servers prohibit this flow entirely) due to the inherent risks of returning access tokens in an HTTP redirect without any confirmation that it has been received by the client.
* Public clients such as native apps and JavaScript apps should now use the [authorization code](https://oauth.net/2/grant-types/authorization-code/) flow
* Implicit flow has some of the drawbacks which are mentioned below:
* Not as secure
* Primarily used with short lived access tokens
* Used with JavaScript apps.

# Problems with OAuth 2.0 for authentication

* There is no standard way for getting user’s information. OAuth was designed only for authorization.
* No common set of scopes.

# OpenID Connect

Figure 9: OpenID Connect and OAuth 2.0 relation

* It is an extra identity layer added on top of the OAuth. It allows to check the identity of the user based on authorization done by authorization server.
* In this we can fetch additional info about end user
* OpenID connect is for Authentication and OAuth 2.0 is for authorization.
* OpenID adds the following to OAuth 2.0:
* ID token
* If more information is required, then it connects with user information endpoint.
* Standardized implementation
* Standard set of scopes

# Introduction to Ping Access

* Ping Access is alternative to the Web Access Management Solution.
* Ping Access is used to protect websites, APIs, and other web resources using rules and other authentication criteria. It allows both internal and external users to access web applications securely.

# Ping Access V/s Ping Federate

* Ping Federate takes care of authentication whereas Ping Access takes care of authorization.
* Ping Federate uses SAML protocol due to which granular access is not possible in Ping Federate whereas Ping Access uses OAuth 2.0 which allows to have a granular access in Ping Access.

# Why Ping Access?

* Some Applications that don’t support SAML or may need granular access control. For that purpose, Ping Access is required.
* Ping Access can be deployed as a reverse proxy for all traffic to the application. After authentication by PingFederate, Ping Access can provide session data to the application through HTTP headers, JWT tokens or proprietary token injection.
* When combined with PingFederate, Ping Access enhances security by applying resource policies (URL) and transaction-specific policies with a wider range of contextual data used for authorization.
* Ping Access can validate authentication tokens by communicating with PingFederate in predetermined time intervals.

**Note:** First, user needs to be authenticated using Ping Federate. So, authentication requests are sent to Ping Federate and Ping Federate sends the authentication response. Virtual hosts will generate a URL for which application configuration to be triggered.

# Why Ping Access is a Policy Engine?

Policy Engine is an enforcing mechanism that allows to enforce certain rules. Hence, Use of Ping Access is to enforce certain set of rules to provide access to resources for some set of users. That’s why Ping Access acts as Policy Engine.

**Note:** Need for OIDC in Ping Access is describe below:

* SAML allows to protect your application but it does not allow to have granular level of control.
* OIDC allows to authenticate user then authorization occur using OAuth.
* OIDC works in integration with OAuth

# Terminologies in Ping Access

1. Virtual Hosts: Virtual Hosts is the receiving end which receives the request in Ping Access. For user point of view, it is the URL for which application need to be triggered.
2. Web session: Web Sessions allows Ping Access to communicate to Ping Federate

* In web session There is a client id which allows to map Ping Access to Ping Federate

1. Token Provider: In Ping Federate, token provider provides the OIDC tokens
2. Site: It points to an application server. Site configuration sends the request outwards from Ping Access. Ping Access, Ping Federate and the application is running on diff servers. So that’s why site is configured which is used to send the user to network URL which has some servers.

# How Ping Access Works?

Figure 10: Working of Ping Access

* When a user requests a Web resource from Ping Access, Ping Access inspects the request for a PA Token.
* If the PA Token is missing, Ping Access redirects the user to an OpenID Connect Provider (OP) for authentication.

Note: OAuth must be configured.

* The OP follows the appropriate authentication process, evaluates domain-level policies, and issues an OpenID Connect (OIDC) ID Token to Ping Access.
* Ping Access validates the ID Token and issues a PA Token and sends it to the browser in a cookie during a redirect to the original target resource. Upon gaining access to the resource, Ping Access evaluates application and resource-level policies and optionally audits the request.
* Note: Ping Access can perform [Token Mediation](https://documentation.pingidentity.com/pingaccess/pa40/pa_c_Token_Mediation.html) by exchanging the PA Token for the appropriate security token from the PingFederate STS or from a cache (if token mediation occurred recently).
* Ping Access forwards the request to the target site.
* Ping Access processes the response from the site to the browser (step not shown).

# Gateway V/s Agent Based Deployment

Ping Access can be deployed using Agents, as a Gateway (or reverse proxy), or using a combination of both. Before deciding on a deployment, it is important to understand the pros and cons of each deployment scenario and determine how they impact your strategy.

## Gateway

Pros:

* Fewer number of deployed components that require maintenance
* Independent of target application platform
* No impact on web/app server processing and performance
* Able to work with existing security token types (such as creating 3rd party WAM tokens)

Cons:

* Requires networking changes
* Requires strategy for securing direct access to backend web/app servers (network routing or service level authentication)
* Depending on the application, may require content / request/response rewriting
* Another layer that requires HA/DR planning

## Agents

Pros:

* No networking or server level authentication changes required
* Tight integration with web server handling requests
* Scales with application

Cons:

* High cost of ownership when many agent instances deployed, although should be upgradable/patchable independently of Ping Access (policy) server
* Policy evaluation is cached; although it is periodically flushed/re-evaluated (for new sessions, updates to session token, etc.) it isn't quite is "real time" as proxy
* Tight dependency on web server version & platform

# Appendix-1: Ping Access Installation

## Setting up a Virtual Machine

Download any Virtual Machine Software and set up any Linux distribution of your choice.

* Virtual Machine - VMware Workstation
* Linux Distro - Ubuntu

## Downloading JAVA

PingFederate requires a Java runtime to be installed on your server and has been tested in the following Java environments:

* Amazon Corretto 11
* Amazon Corretto 8
* OpenJDK 11
* Oracle Java SE Development Kit 11 LTS
* Oracle Java SE Runtime Environment (Server JRE) 8

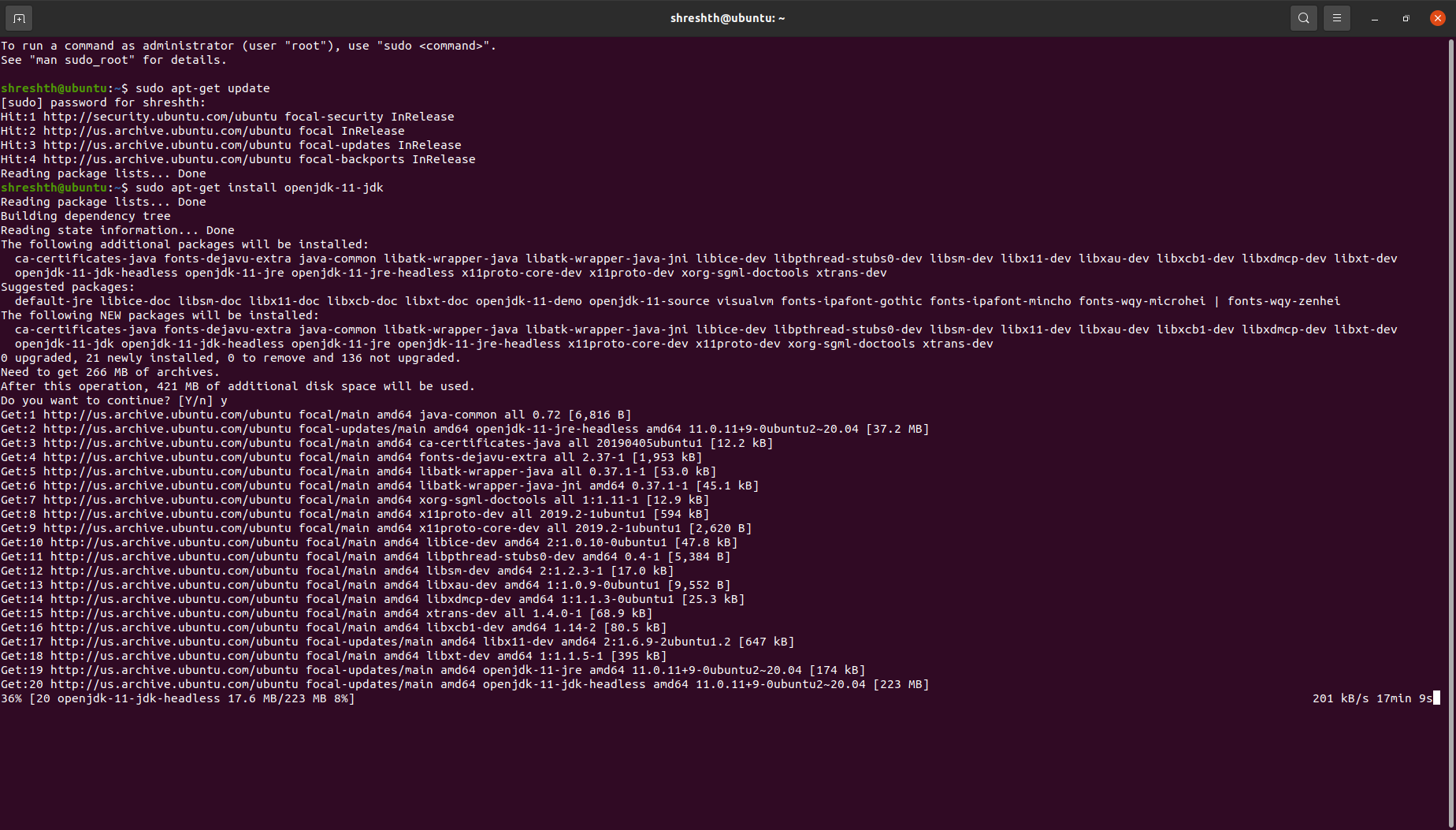
One can install OpenJDK 11 on an Ubuntu by running **sudo apt-get install openjdk-11-jdk** in the terminal

Figure 11: Downloading JAVA for Ubuntu

## Downloading License Files

Request your license files for PingAccess, PingFederate and Ping Directory, one at a time, on this URL <https://www.pingidentity.com/en/account/request-license-key.html>

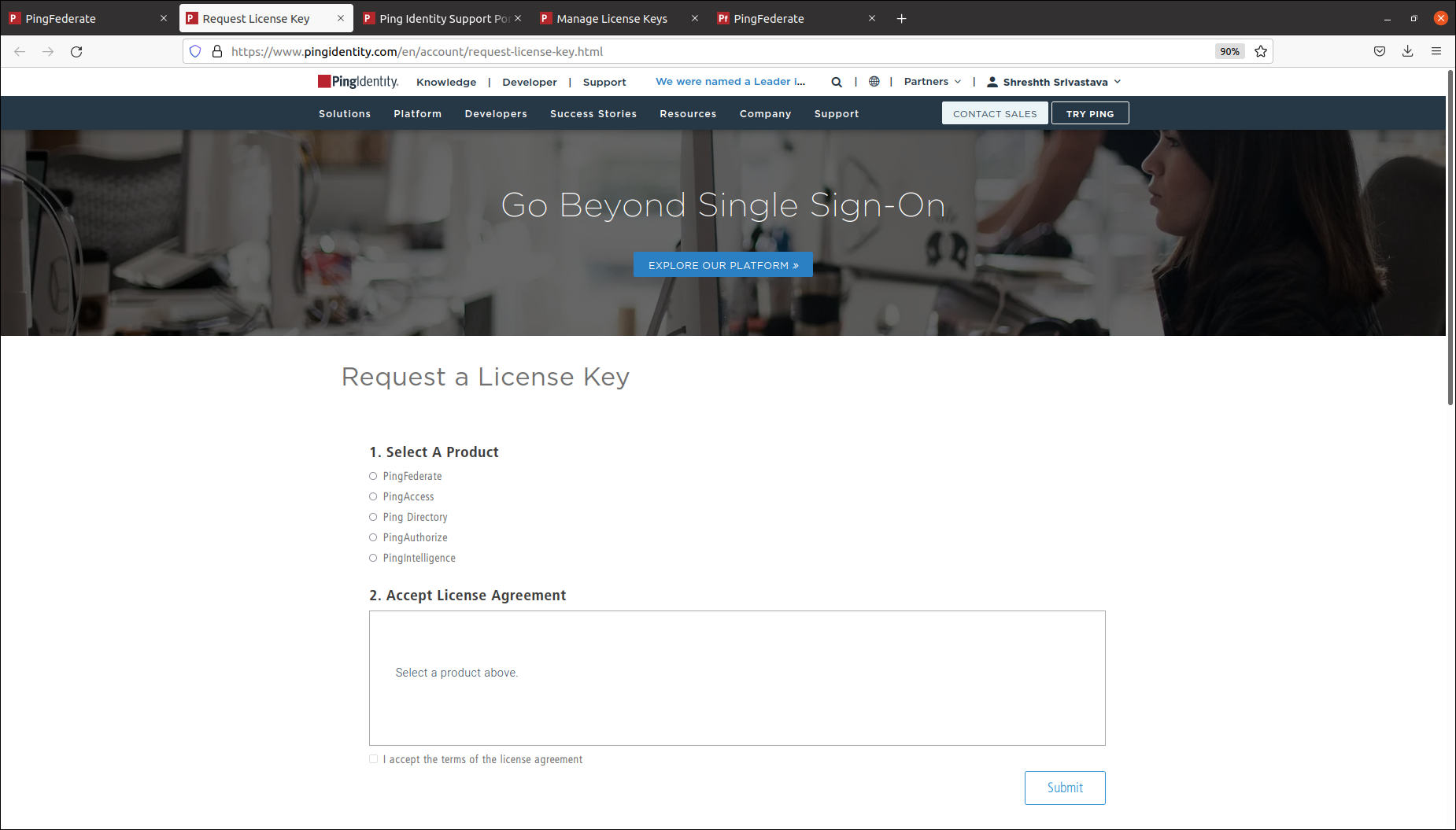


Figure 12: Downloading License files

Once requested for all three license keys, download them by going to the Support Dashboard from this URL <https://support.pingidentity.com/s/>

Click on **MANAGE LICENSE KEYS → VIEW → DOWNLOAD** and download all the three requested license files respectively

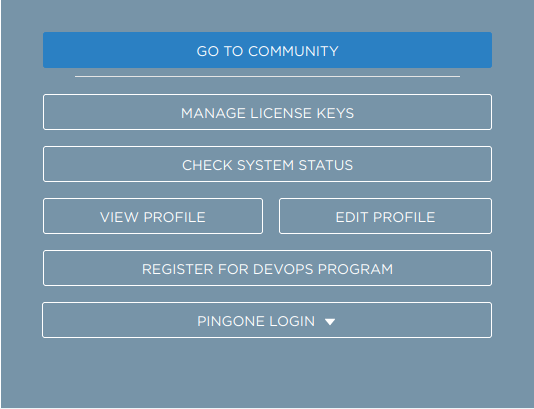
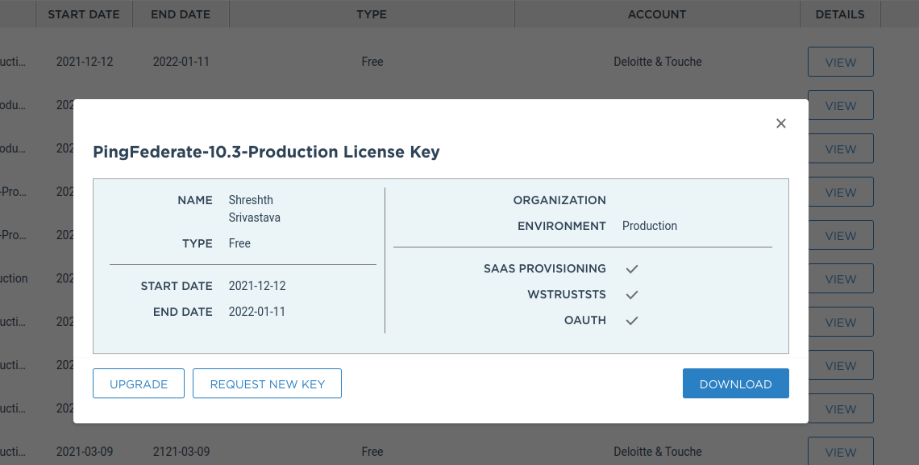


Figure 13: Downloading License Keys

## Ping Access Installation

* Inside your VM, hit the URL - <https://www.pingidentity.com/en/resources/downloads/pingaccess.html> and download the Linux-based Product Distribution (ZIP) file.

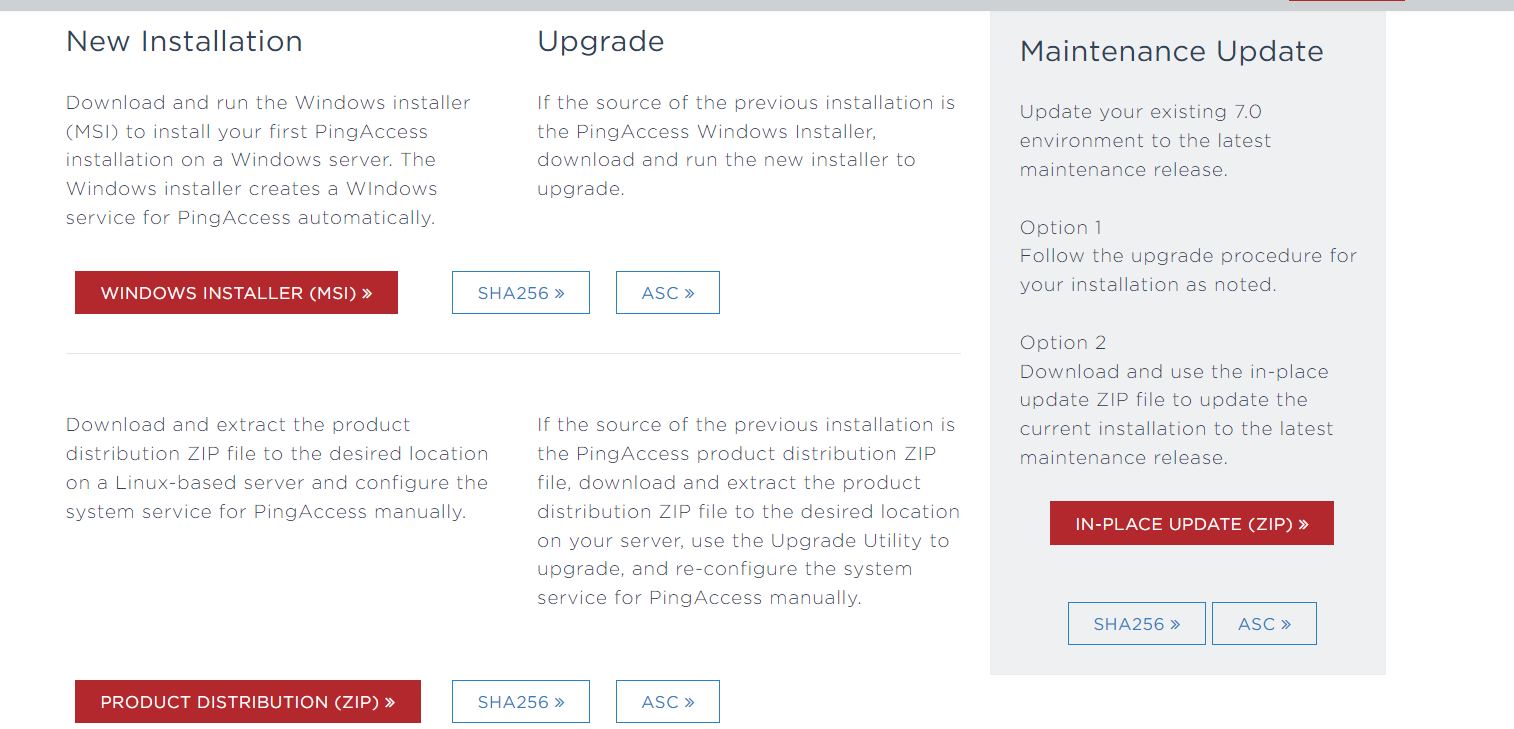


Figure 14: Download the Product distribution ZIP file

* Extract the downloaded ping access zip file

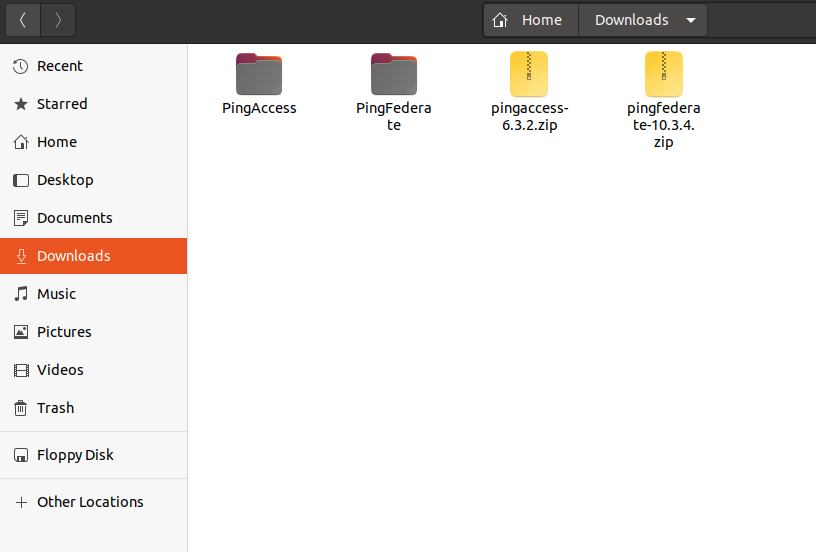


Figure 15: Extract the ZIP file

* Go to the bin folder inside the extracted PingFederate folder and add the following line at the beginning of run.sh file

**JAVA\_HOME="/home/chigaur/Downloads/openlogic-openjdk-11.0.8+10-linux-x64"**

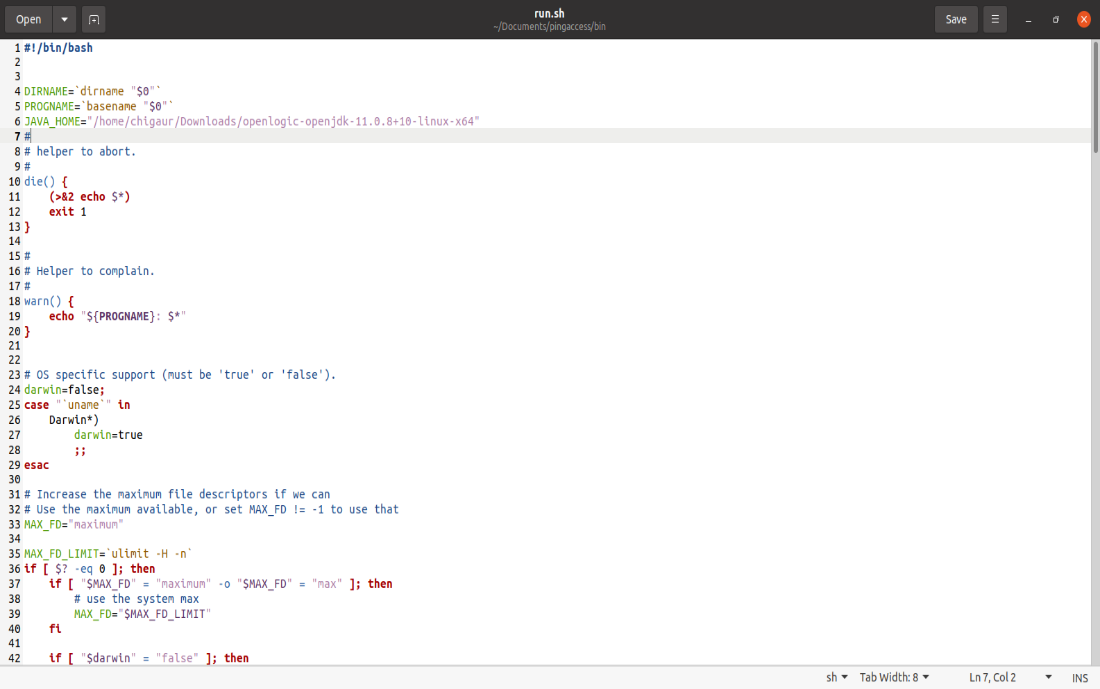


Figure 16: Adding JAVA\_HOME path

* Then open the terminal inside the bin folder and type the following command

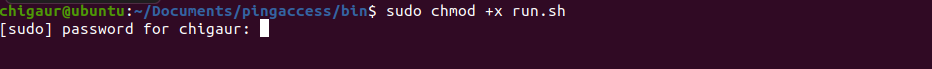


Figure 17: Running chmod command

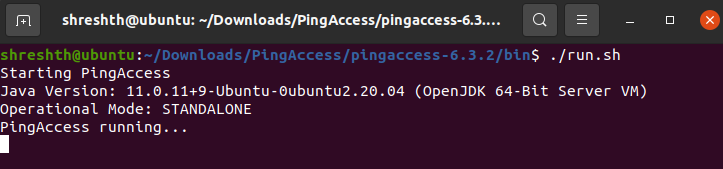
* Run PingAccess by running the **run.sh** file in the terminal as root

Figure 18: Running the Ping Access

* Open <https://localhost:9000/> to access the PingAccess console application

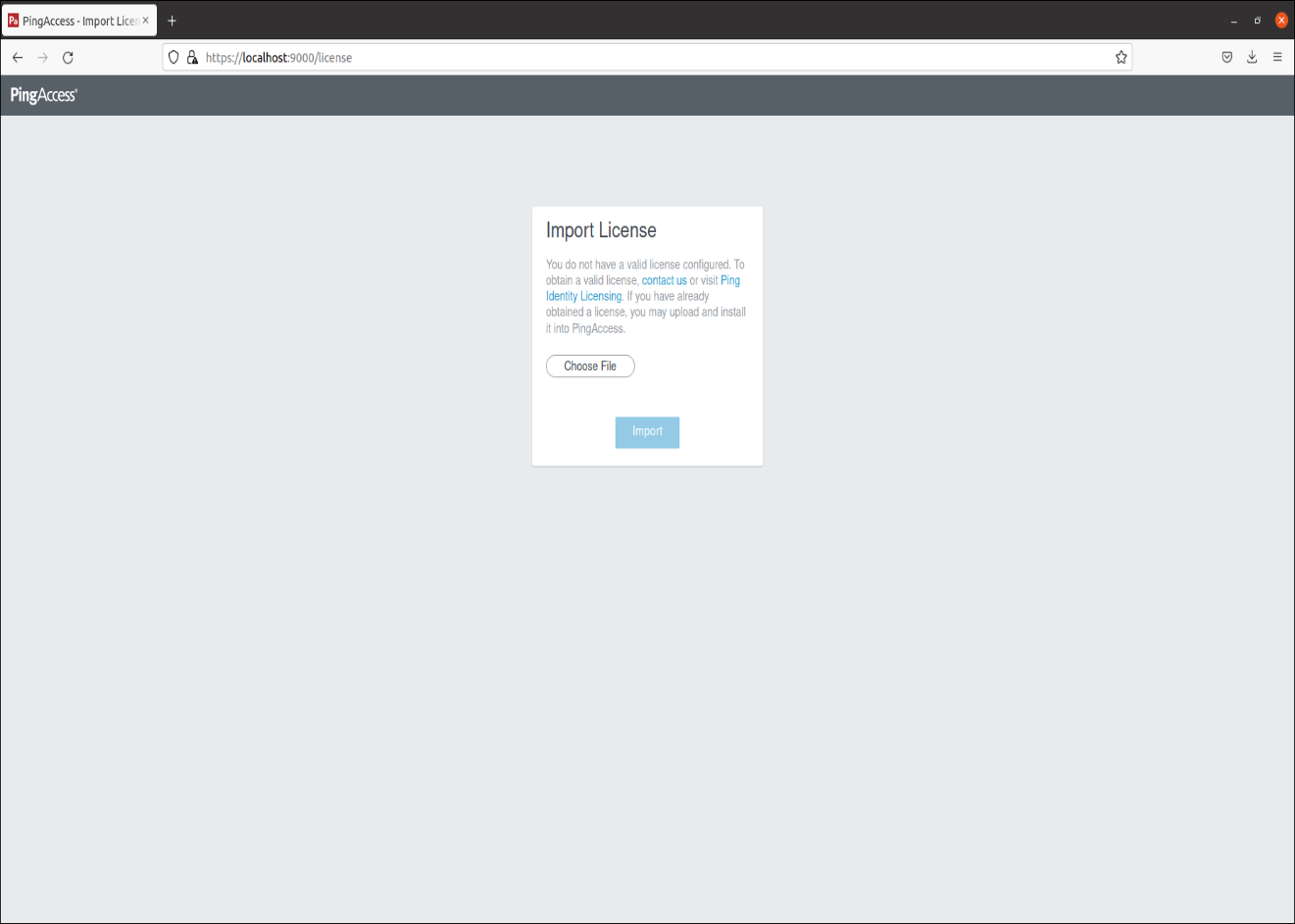


Figure 19: Importing License Files

* Select your previously downloaded license file and import it

**REFERENCES**

* <https://www.youtube.com/watch?v=t4-416mg6iU>
* <https://www.youtube.com/watch?v=3pZ3Nh8tgTE>
* <https://oauth.net/2/grant-types/authorization-code/>
* <https://developer.okta.com/blog/2018/04/10/oauth-authorization-code-grant-type>
* <https://www.pingidentity.com/en/resources/content-library/data-sheets/pingaccess-data-sheet.html>
* <https://www.pingidentity.com/en/pingone/pingaccess.html#:~:text=PingAccess%20is%20a%20centralized%20access,access%20the%20resources%20they%20need>.
* https://openid.net/connect/