Multi-Tenancy in Cloud Applications on the Example of PROCEED

by

Felipe Trost

Matriculation Number 456129

A thesis submitted to

Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
Department of Telecommunication Systems
Service-centric Networking

Bachelor's Thesis

August 6, 2024

Supervised by:
Prof. Dr. Axel Küpper

Assistant supervisor:
Kai Grünert

Eidestattliche Erklärung / Statutory Declaration

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Berlin, August 6, 2024 Chuck Norris' son

Abstract

In this thesis, we show that lorem ipsum dolor sit amet.

Zusammenfassung

Hier kommt das deutsche Abstract hin. Wie das geht, kann man wie immer auf Wikipedia nachlesen `http://de.wikipedia.org/wiki/Abstract`...

Contents

1 Introduction **1**

2 Research Questions **5**

3 Task List **6**

4 Related Work **8**
4.1 OAuth 2.0 and OpenID Connect 8
4.1.1 OAuth 2.0 Roles 8
4.1.2 Authorization Grants 9
4.1.2.1 Implicit 9
4.1.2.2 Resource Owner Password Credentials 9
4.1.2.3 Client Credentials 9
4.1.2.4 Authorization Code 9
4.1.3 OpenID Connect 10
4.2 PROCEED's role system 10
4.2.1 MS's Role System Terminology 10
4.2.2 MS's resources and actions 11
4.2.3 MS's roles in CASL 11

5 Concept and Design **13**

6 Implementation **16**

7 Evaluation **17**

8 Conclusion **18**

List of Tables **19**

List of Figures **20**

Appendices **21**

Appendix 1 **23**

1 Introduction

In today's digital age, businesses heavily rely on cloud applications, software tools that are accessed and run entirely over the internet. Cloud applications offer many advantages:

- Accessibility: They can be accessed anywhere from anywhere with an internet connection.
- Cost-Efficient: Most cloud applications offer a subscription model, where you have to pay a monthly price, instead of paying for the whole software upfront.
- Collaboration: Typically, collaboration is easier since everything can be found in one place, instead of having to send files back and forth.
- Data safety: All files are stored by the application in the cloud, and they don't have to be stored in the user's device, which could be lost, stolen or damaged.
- Device agnostic: many cloud applications can be accessed through different device types.
- No IT overhead: users don't have to setup the application on their own, which would require technical knowledge.

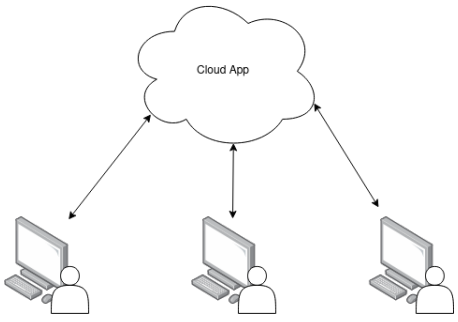

Figure 1.1: Cloud Application.

One very common feature that makes these benefits possible is called "multi-tenancy". Multi-tenancy is a software architecture in which one single instance of an application can be used by many different users or organizations at the same time. Without multi-tenancy, each user or organization would need to run the application on their own servers or computers, largely negating the numerous benefits listed earlier.

Think of it like a big apartment building. Each tenant (user or organization) has their own private space (their assets), but they're all using the same building (the cloud application).

Many popular cloud applications use this approach. For example, when you use Microsoft Teams, Slack, or Asana, you're sharing the application with many other companies, but you only see and interact with your own team.

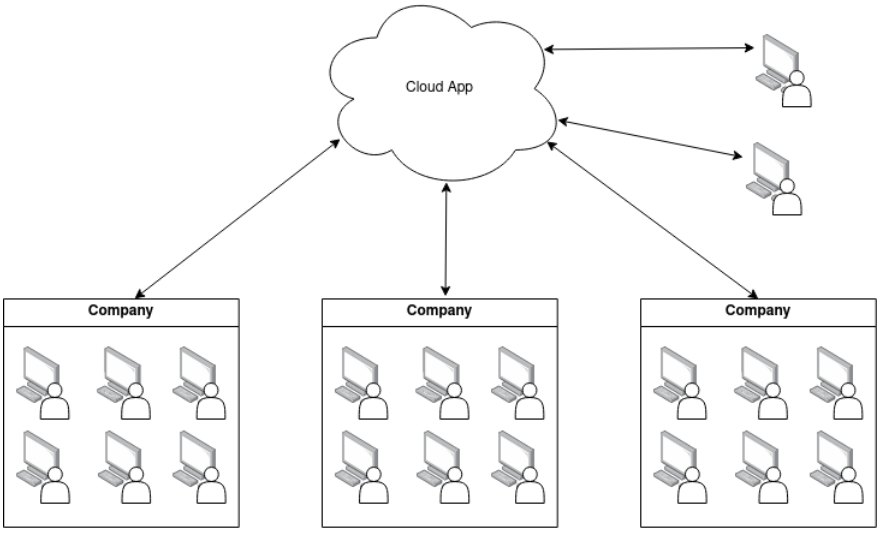

Figure 1.2: Multi tenancy in cloud applications.

PROCEED (short for PROCess EnginE in a Distributed environment) is a decentralized Business Process Management System. PROCEED uses BPMN at its core to model and execute business processes. BPMN (Business Process Model and Notation) is a standardized graphical notation used for documenting business processes. BPMN is typically used inside of organizations to illustrate sequences of tasks, decision points, and interactions within various business processes, providing a standardized visual representation.

PROCEED offers two products:

- Distributed Process Engine (DPE for short): the DPEs execute bpmn processes in a distributed fashion.
- Management System (MS for short): the MS is a cloud application that gives users a graphical interface to work on their BPMN processes and deploy these to the DPEs.

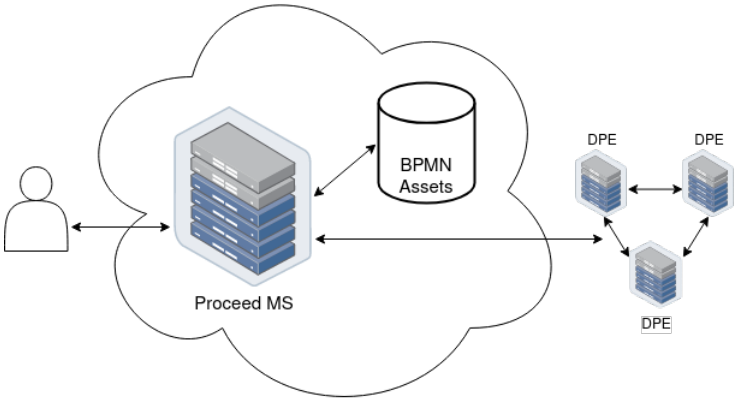

Figure 1.3: Overview of PROCEED.

Currently the MS lacks full multi-tenancy support, it only supports individual users and doesn't fully support organizations. For organizations to be supported, members of the organization need to be able to have a shared workspace, where they can work on the same assets. This could technically achieved if members of the organization shared all assets with each other. However, PROCEED only supports universal sharing, meaning that all users would be able to see the shared assets. Furthermore, even if it was possible to share assets only to a group of users, it would be very cumbersome and error-prone. For this reason, this thesis implements multi-tenant functionality into the PROCEED MS by introducing the concept of Environments.

Each tenant will be able to work in their own isolated environment, which lives in a central instance of the Management System. Each account will automatically have its own Environment allowing users to work on personal projects. Organizations will be able to create environments where multiple members can work together.

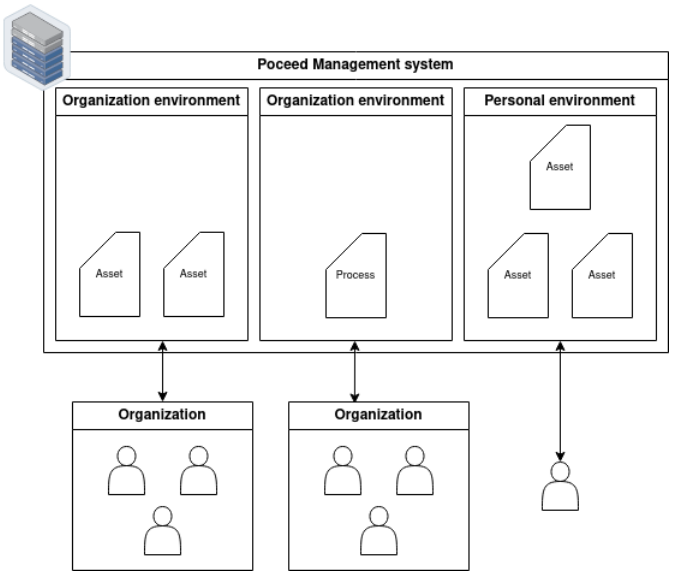

Figure 1.4: Environments in PROCEED.

Additionally, environments will include a folder structure to improve the organization of assets. Currently the MS only provides labels for organizing assets. Assets can be labeled with one or more labels, which can be used to filter assets. Users cannot add, delete nor modify these labels. Folders will provide more flexibility than labels.

2 Research Questions

1. Environment Representation: How can we model environments within the existing PROCEED database schema to ensure data integrity, isolation, and efficient access?
2. Asset-Environment Association: How can we associate assets with their respective environments while maintaining a clear and consistent data model?
3. Folder Hierarchy: What data structure (e.g., tree, graph, nested sets) is best suited for representing folder hierarchies within environments? How can we efficiently store and query this structure in the database?
4. How can we extend the existing PROCEED role system to incorporate environment-specific permissions?
5. How can we apply different permissions to assets based on what folder they are stored in?
6. What mechanisms are needed to ensure that role-based access control is consistently enforced?
7. How can the frontend user interface dynamically adapt to a user's roles and permissions within different environments?
8. What user interface elements and interaction patterns are most effective for navigating and managing folders and environments?
9. What software abstractions (e.g., classes, interfaces, functions) can we create to simplify the process of identifying and managing a user's environment in the backend code?
10. How can we minimize the impact on existing code while ensuring seamless integration?

    Performance and Scalability: How will the addition of environments impact the performance and scalability of the PROCEED Management System? What optimizations can be made to ensure that the system remains responsive and efficient even with a large number of users and environments? Migration Strategies: How can we seamlessly migrate existing PROCEED data and users into the new environment-based structure? What strategies can we employ to minimize disruption during the transition? Security: What additional security measures are needed to protect data within environments? How can we prevent unauthorized access to environments and their assets?

3 Task List

(1) Functional
  (I) The MS has to support two types of environments: personal and organization environments.
    (A) Every asset in the MS must be stored in one and only one environment.
    (B) Assets stored in one environment can only be accessed by members of that environment.
    (C) Every user has to have a personal environment, of which only he can be a member.
    (D) Organization environments must be able to have multiple members and roles that control what each member can do.
    (E) Environments must have a folder system to store assets.
      (i) Find a suitable abstraction to represent folders in a database.
      (ii) Ensure privacy between environments.
      (iii) The MS's preexisting role system must be adapted to fit environments: The proceed management system already has a role system in place to manage user's access to resources, these roles need to be modified for them to work with environments.
        (a) Ensure roles are always enforced in the backend.
        (b) The frontend UI must adapt to a user's roles, by only showing options that the user has permission to do.

        depending on the folder the asset is in.
  (II) The MS must be able to hold multiple environments and let users access them concurrently.
  (III) Users must be able to be members of multiple environments and carry out actions in each one of them.
(2) Non functional
  (I) keep changes to the existing codebase to a minimum.
  (II) The same data structure should be used for both personal and organization environ-

ments.

(III) The user interface for navigating and managing folders and environments should be intuitive and easy to use.

(IV) Prioritize developer experience by creating clear abstractions and APIs.

  (A) Create simple abstractions for the backend code of the MS, that allow to acknowledge a user's environment with minimal effort.

  (B) Create a simple abstraction for the frontend, that facilitates adapting the Interface for each.

4 Related Work

4.1 OAuth 2.0 and OpenID Connect

Oauth is an open standard for access delegation, commonly used as a way for users to grant client applications access to their information on other applications. Oauth was born as a necessary security measure, to avoid sharing plaintext credentials between applications. Plaintext credential sharing, as outlined in [**?**] has many security risks:

1. Applications are forced to implement password authentication, to support the sharing of plaintext credentials.
2. Third party applications gain overly broad access to the user's account.
3. Users cannot revoke access to specific third party applications.
4. If any of the third party applications are compromised, the user's account is at risk.

Oauth adresses these issues by decoupling the client application from the role of the resource owner, meaning that the client application will not get a full set of permissions to the user's account. Insetead of handing his crendentials to the third party application, the resource owner signs in, in the application's website which then issues an access token to the client application. This method avoids the user having to share his credentials with third party applications.

4.1.1 OAuth 2.0 Roles

Oauth 2.0 defines four roles for participants in the protocol flow:

1. Resource owner: The entity that can grant access to a protected resource, typically this would be an end user of a web application.
2. Resource server: The server hosting the protected resources.
3. Client: The application requesting access to the protected resources. OAuth 2.0 distinguishes between two types of clients: confidential and public clients. Confidential clients are capable of keeping their credentials confidential, while public clients, like browser-based applications, cannot.
4. Authorization server: The server that issues access tokens to the client after the resource owner has been succesfully authenticated.

The resource server and the authorization server can be the same entity, but they are not required to be.

4.1.2 Authorization Grants

Authorization Grants are credentials that are issued to clients, which can be exchanged for an access token. This access token can be used to access the protected resources on the resource server. Oauth 2.0 defines four authorization grants with different flows.

4.1.2.1 Implicit

The implicit grant is very helpful for public clients, as it doesn't require confidential client credentials. This is very helpful for browser-based clients, as they can't store confidential credentials securely. In the implicit grant users are redirected to the authorization server, where they authenticate thmeselves and authorize the client. Afterwhich the authorization server issues an access token directly to the client, this is done so with a HTTP redirect, where the access token is embedded in the redirect URL, this way the client can extract the access token from the URL.

In this flow the resource owner only authenticates with the authorization server, thus never having to share his credentials with the client.

Implicit grants have many security risks, as the access token is exposed in the URL and can be intercepted by a malicious attacker. This is why PKCE (Proof Key for Code Exchange) was later introduced as an addition to the implicit grant [**?**].

4.1.2.2 Resource Owner Password Credentials

This grant type requires the resource owner to share his password credentials with the client. The resource owner's password credentials represent an authorization grant, which the client can exchange them for an access token. Even though this grant type requires the resource owner to share his credentials with the client, these are only used for one request and don't have to be stored.

4.1.2.3 Client Credentials

The client credentials grant is used when the client is the resource owner. Clients are typycally issued crendentials, which they can use to authenticate themselves. Clients send these credentials to the authorization server and are issued an access token.

4.1.2.4 Authorization Code

The Authorization Code grant is the most common grant type used in OAuth 2.0, it is similar to the implicit grant 4.1.2.1, as it also uses HTTP redirects and it doesn't require the resource owner to share his credentials with the client. In the authorization code grant, the client redirects the resource owner to the authorization server. There the resource owner authenticates

himself and authorizes the client. afterwhich the authorization server redirects the resource owner back to the client with an authorization code. The client then authenticates itself with his confidential credentials on the authorization server and exchanges the authorization code for an access token. As the client needs confidential credentials, this flow is only suitable for confidential clients. The exact steps are shown in figure 4.1.

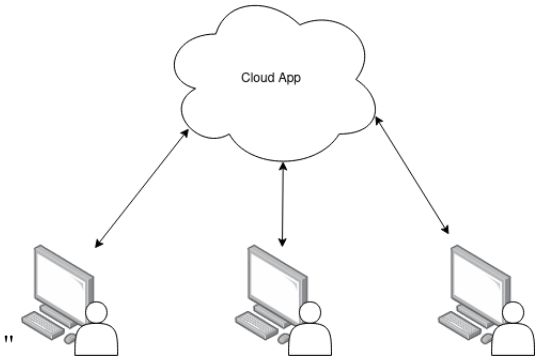

Figure 4.1: Cloud Application.

4.1.3 OpenID Connect

OpenID (OIDC for short) Connect is an identity layer built on top of the OAuth 2.0. While OAuth 2.0 focuses on authorization (granting clients access to resources), OIDC extends this to authentication. Since OIDC deals with authentication, I will call the resource owner the user from now on.

OIDC uses as its base either the Authorization Code flow 4.1.2.4 or the Implicit flow 4.1.2.1 and it introduces a new type of token called an ID Token. This ID Token is a JSON Web Token (JWT) that contains minimal information about the authenticated user.

This token is sent along side the Access Token in the Authorization Code flow to the client after the user has authenticated himself.

In the OIDC flow, after the client obtains the Authorization Code, it exchanges it for both an Access Token (as in OAuth 2.0) and an ID Token. The client can then validate the ID Token to ensure it's genuine and extract the user information contained within.

In essence, OIDC allows clients to obtain information about the authenticated user in a standardized way.

4.2 PROCEED's role system

The PROCEED MS uses a Role-Based Access Control (RBAC) system to mange user authorization, i.e. to determine what actions a user can perform. Roles can be seen as bundles of permissions, which are granted to users. A user can have multiple roles. Typically roles are assigned to users based on their job function. RBAC can be advantageous since they can be assigned to multiple users and don't change often, making them easier to manage than individual permissions.

4.2.1 MS's Role System Terminology

The following terms are important to understand the role system in the PROCEED MS:

- Resource: A resource is any protected entity in the management system, that can be accessed by users.
- Action: An action is a specific operation that can be performed on a resource.
- Permission: A permission is a tuple of resource type and action, which specifies that a user can perform the action on the resource instances. Optionally a permission can have conditions that have to be met the by resource instances, for the user to be able to perform the action.
- Role: A role is a set of permissions. Roles can be assigned to users, which then inherit the role's permissions. Roles can have expiration dates, after which all permissions are revoked.

4.2.2 MS's resources and actions

The following are the resource types that are used in the PROCEED MS: `Process`, `Project`, `Template`, `Task`, `Machine`, `Execution`, `Role`, `User`, `Setting`, `EnvConfig`, `RoleMapping`, `Share`, `Folder`.

These are the actions that can be performed on these resources: `none`, `view`, `update`, `create`, `delete`.

4.2.3 MS's roles in CASL

The PROCEED MS uses [1]CASL to implement Rules. CASL is an isomorphic authorization JavaScript library. To enforce authorization CASL has abilities, which are assigned to users. Abilities expose functions to check weather a user can perform an action on a resource. Abilities are defined by four parameters: user action, subject, fields, conditions. User actions and subjects are analogous to actions and resources 4.2.1.

CASL differentiates between subject type and subject instance. A subject instance is a specific instance of a subject type, e.g. a specific process users are working on, is an instance of the resource type "Process".

Fields are used to specify which fields of a resource instance an action can be performed on, e.g. a user can update a process's name, but not its id, or creation date.

Conditions are used to specify additional conditions that have to be met by a resource instance, for a user to be able to perform an action on it. E.g. a user can only update a process if he created it.

```
1 import { defineAbility } from '@casl/ability';
2
3 class User {
4   constructor(id) {
```

[1] https://casl.js.org/v6/en/

```
    this.id = id;
  }
}

class Process {
  constructor(user, name) {
    this.authorId = user.id;
    this.createdOn = new Date();
    this.name = name
  }
}

function abilityForUser(user){
  return defineAbility((can, cannot) => {
    can('delete', 'User', {id: user.id});

    can('update', 'Process', ['name'], {authorId: user.id});
  });
}

const user1 = new User(1);
const user1Ability = abilityForUser(user1);
const user1Process = new Process(user1, 'some process');

const user2 = new User(2);
const user2Ability = abilityForUser(user2);

user1Ability.can('update', 'Process'); // true
user1Ability.can('update', user1Process, 'name'); // true
user1Ability.can('update', user1Process, 'createdOn'); // false

user1Ability.can('delete', user1); // true
user1Ability.can('delete', user2); // false

user2Ability.can('update', 'Process'); // true
user2Ability.can('update', user1Process); //false
```

Listing 4.1: CASL example

If there exist any possible resource instance, where the user has permission to perform an action, then the user has permission to perform the action on the resource type. E.g if a user has permission to view some process in the MS, then he has permission to view .

5 Concept and Design

At the moment the Proceed Management system stores and manages all user assets the same for all users. There is no option to create assets such that only a specific group of people have access to it, meaning it is unsuitable for companies and large teams for which collaboration is important. This behaviour will be changed by the implementation of the beforementioned workspaces.

Workspaces are abstractions that enable asset isolation. A fundamental principle of this system is that every asset belongs exclusively to one workspace. Workspaces contain hierarchical structures, similar to folders, in which assets can be stored. The existing implementation of the Management System supports Role-based access control (RBAC). Which means that access to resources is determined by a user's roles. The implemantation of workspaces will transform the existing roles, so that each role only belongs to one workspace, and so that it only affects the assets in that workspace. Roles will also be adapted for BPMN assets, they will be applied on the folders that the BPMN assets are stored in, instead of being defined globally, where they apply to each asset. Roles will cascade down the folder structure and be applied to all the assets in these folders as before. Furthermore users will now be able to see BPMN assets

We distinguish between two types of workspaces: company workspaces and personal workspaces. Personal workspaces will offer a more simplified feature set compared to company workspaces.

Every user is automatically provided with a personal workspace, offering an individualized space within the system. Within his workspace, a user will be granted the admin role. Roles in a personal workspace will not be able to be modified. Personal workspaces also cannot have members, the only way to enable others to work on it's assets will be to share them with the proper permissions.

Company workspaces will introduce roles that are associated with a hierarchical structure and cascade down through it, enabling role-based access control at multiple levels. Company workspaces can accommodate multiple members in contrast to personal workspaces.

In order to facilitate cooperation, the implementation will also allow sharing assets to users outside or inside the asset's workspace and granting temporary access to user's that aren't members of the workspace to the workspace.

Company workspaces can be created by normal users, for this, they will have to input data about their company. After the workspace is created, the user is granted the role of administrator and can add new members.

The frontend will include a interface for creating company workspaces, for adding members to the workspace, for managing the structure and roles of the workspace. In the context of company workspaces, when a user is created through the company workspace's interface, the user will be a regular Proceed user, with the only distinction being, that he will automatically be part of the company workspace. The Management System should also have the ability to import or interact with existing user Databases. This will facilitate team managers, to create all the accounts they need for their team at once. The Management system frontend will show users only the contents of one workspace at the time. It will also include a simple way to view the workspaces a user belongs to and switch between them. The frontend will include an interface for sharing assets and managing the permissions of the shared assets.

Checklist

1. Interface for creatring and deleting company workspaces.
2. Adapt roles to workspaces.
   - Each role has to belong to one workspace
   - Roles have to cascade down the folder structure of workspaces
   - Roles have to roughly keep the same functionality as before.
3. Adapt asset storage for workspaces.
   - Find a suitable solution to store assets.
   - Ensure privacy between workspaces.
4. Adapt the Management System's API to take into account workspaces.
   - Make sure that a the requester specifies the workspace he is refering to.
   - Make sure that the requester belongs to the workspace where the asset he is trying to access is located.
   - Implement an endpoint for users to get the workspaces they belong to.
   - Implement an endpoint for users to manage the workspaces they belong to.
5. The Management System frontend will only have one active workspace at a time.
   - All actions that manipulate assets, roles or create users, will be performed on the active workspace.
   - Only the assets of the active workspace will be shown.
   - There will be a clear indication of what worspace is active
   - There will be an easy way to switch between workspaces.
6. Interfaces for managing different aspects of company workspace's members.
   - Option for creating and deleting users.
     - Basic creation with manual input.
     - Import existing user Databases.
   - Option for inviting users to the workspace.
   - Option to manage users' roles.
7. Users should be able to share assets.

- Allow users to share assets with other users.
  - A person who shares something will be refered to as a sharer and the reciepient as a sharee
- Users will only be able to share assets if their roles allow it.
- Each shared object has associated perimissions with it, which restrict the sharee's ability to access and manipulate it.
- Sharees will be able to view and select all workspaces where they have at least one shared asset.
  - Sharees sill see the folder structure of the workspace, but they won't see any of the assets in them, besides the ones shared with him
- The frontend will implement a Interface to share assets.
- The frontend will implement a Interface to manage shares.
- The frontend will indicate to the sharer, in the asset overview that an asset is shared.
- Users that are allowed to view an asset will also be able to see the users that it has been shared to.

6 Implementation

Describe the details of the actual implementation here...

7 Evaluation

The evaluation of the thesis should be described in this chapter

8 Conclusion

Describe what you did here

List of Tables

List of Figures

1.1 Cloud Application. 1
1.2 Multi tenancy in cloud applications. 2
1.3 Overview of PROCEED. 3
1.4 Environments in PROCEED. 3

4.1 Cloud Application. 10

Appendices

Appendix 1

```
for($i=1; $i<123; $i++)
{
    echo "work harder! ;)";
}
```