Cloud Native Project Documentation

Group: **CloudJB** (Jaromir Charles, Benjamin Herrmann)

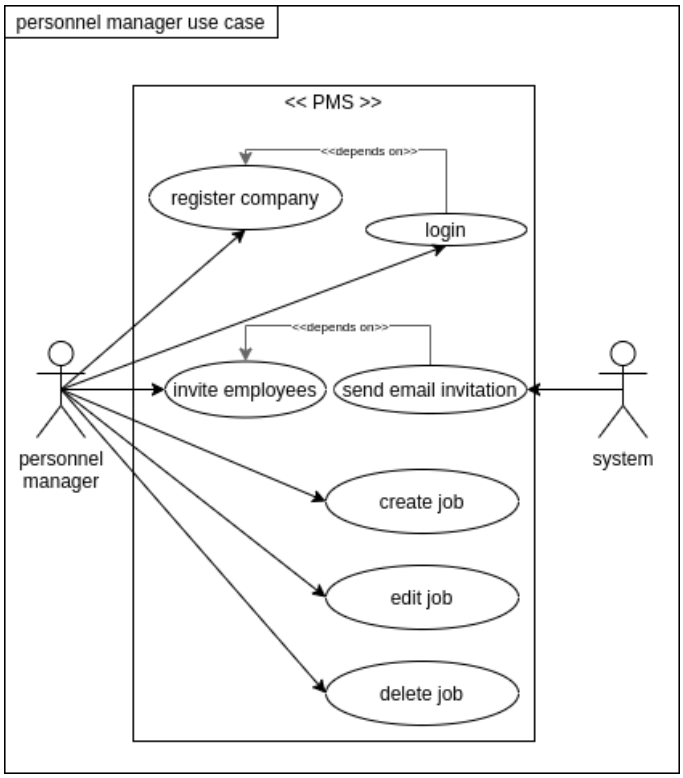
Table of Contents

Description

The main functionality of our multi-tenancy cloud native application is to allow:

- **tenants** (mainly the personnel managers) of a firm to organize their jobs and workers.
- **employees** to have a complete overview of their tenant's jobs.

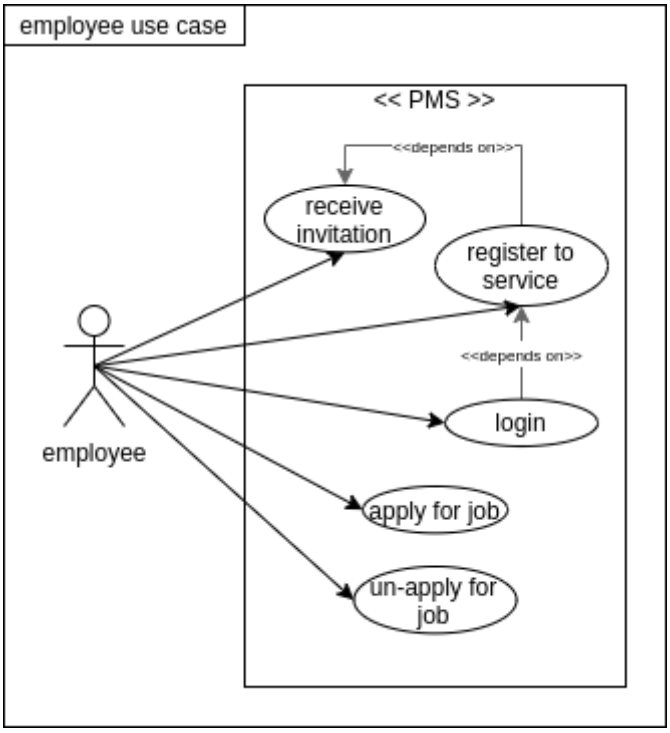
A better understanding of the functionality of the application can be seen within the following use cases.



The diagram above depicts the main functionalities of a tenant. A tenant has the ability to register his company to use the `PMS` service. Upon successful registration he/she can log into the system and use its functionalities. A tenant has basically two important functionalities:

- **organizing employees** - the personnel manager can invite employees to use their `PMS` service whereby after submitting an employee's email address, an email invitation will be sent to that employee via our system.
- **organizing jobs** - the other functionality available is to create, edit and delete jobs. While creating a job, fields such as the date, a title and description, location and start time and number of needed workers for that job must be filled out. Employees can apply for these

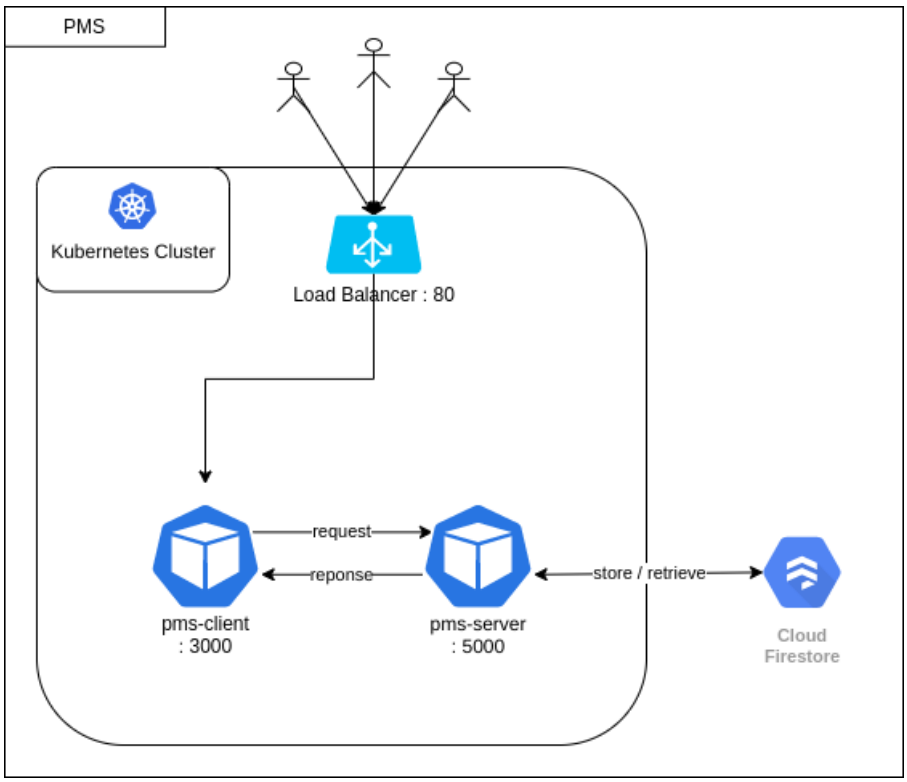
jobs, and the personnel manager can select from the provided list of applied workers who shall take part to accomplish said job.



The diagram above depicts the main functionalities of a user/employee. Upon retrieval of an email invitation, he/she can register themselves to use the `PMS` service of his/her company. After registration, when logged in, the employee is firstly greeted with a list of the company's current available jobs. He/she can then with free will apply and un-apply for desired jobs. A respective view for the applied and upcoming (jobs the personnel manager selected the employee to part take in) jobs are also available.

Application Architecture & Design

Components and Interfaces



The image above depicts the components and interfaces of the `PMS` application. The application is broken down into two services: the `client` and the `server`, both being able to run independently from one another. The application runs within a Kubernetes Cluster.

The `client` (developed with react JS) runs separately in a pod. The client entails the User Interface with which the user interacts with and is visible on port `3000`. The `server` (a Node JS Express server) serves the frontend with its backend logic. Its REST API handles the requests to be made to the database (`Cloud Firestore`) creating, updating and retrieving desirable information. The server also runs in a pod within the cluster and is visible to other pods on port `5000`.

`Cloud Firestore` is a NoSQL document database allowing the `PMS` application to easily store and query data. The database has been made available via a secure service key so that the application (server side) has full access to perform operations on the database.

Within the cluster, the application is not reachable from the outside world. To expose the application outside of the cluster, a service of type `LoadBalancer` has been created to make the pods reachable via the Internet. The Load Balancer service has an external IP and redirects incoming traffic from port 80 to the application's frontend running internally on port 3000.

Important use cases (dynamic view)

Cloud Provider Resources

The application is hosted on `Google Cloud` and uses a number of resources that Google Cloud provides.

- **Google Kubernetes Engine**
  - Google's Kubernetes Engine has been used to run the application in a containerized environment. Kubernetes automates the deployment, scales and manages the `PMS` application. The advantages of using Kubernetes is that it does most of the work. It scales the application up and down when the specified CPU threshold has been reach. It handles the recreation of new pods if one happens to fail.
  - Alternatives: docker

- **Firebase Firestore**
  - The application uses a NoSQL Document store database because of its flexibility. It supports agile software engineering making implementing the application a lot easier and more flexible to sudden changes.
  - A SQL database could have also been used to store the data but then the schema solubility would have been lost making agile engineering much harder.
- **Container Registry**
  - Google Cloud's Container Registry has been used to store manage and secure our docker container images. The application images how ever need to be available for Kubernetes, so that Kubernetes knows where to get it images to be used to create pods.
  - justification: images are secure and private
  - alternatives: docker hub could also be used to store our images, but we want our image to be private
- **Compute Engines**
  - Google's Compute engines are being used indirectly, as the Kubernetes cluster creates virtual machines for us to run the application.

The five essential characteristics of a cloud service see

1. `On-demand self-service`
   1. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
2. `Broad network access`
   1. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., **mobile** phones, tablets, laptops and workstations).

   The Load balancer which was set up gives our application an public external ip address which allows all users to access our service at any time from anywhere from any device which is connected to the Internet.
3. `Resource pooling`
   1. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state or datacenter). Examples of resources include storage, processing, memory and network bandwidth.

   Resources are shared whereby the application is accessiable to all tenants and users under the same ip address. The load balancer handles the request by up and down scaling the number of pods to be ran. Our storage database is shared by all tenants and users. The cpu processing is shared where by when a threshold is reached, new pods will spin up.
4. `Rapid elasticity`
   1. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

A Horizontal Pod Autoscaler object has been created which targets the client deployment which periodically adjusts the number of replicas of the scale target to match the average CPU utilization which was specified to 80%. A maximum of 5 replicas has been set and a minimum to 1, with a cpu utilization of 80%.

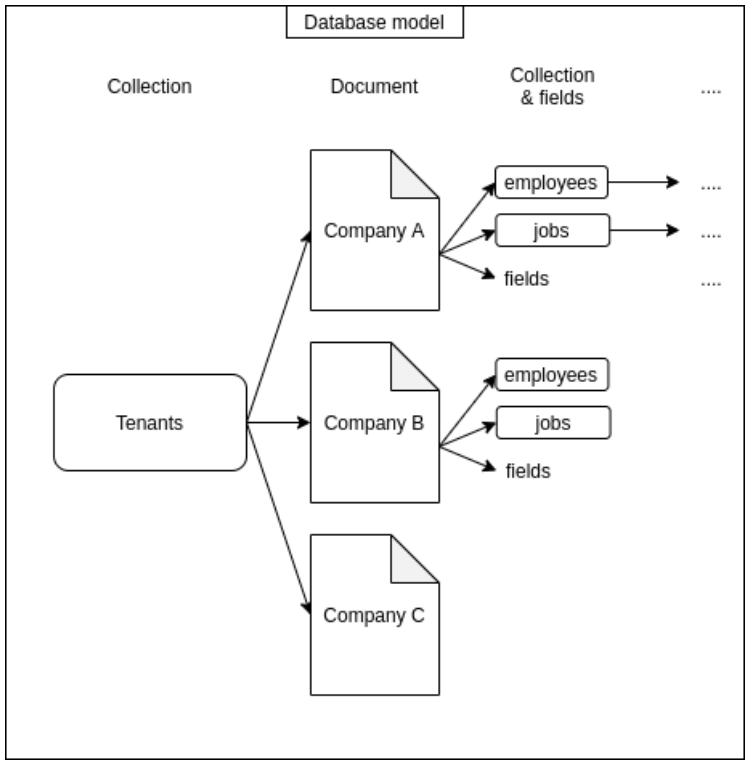
To achieve rapid elasticity, a Load Balancer has been set up on both client and server which will scale up when the user demands increase. The Load Balancer will also scale down if the resources are not utilized by the consumers.

Load Balancer? The load balancer allows us to scale up when a lot of requests come in and scale down when the requests retreat. A `Horizontal Auto scale Load balancer` is used to distribute the load over the client pods. The Load balancer runs on port `80` and redirects the incoming traffic to the client pods.

5. `Measured Service`
    1. Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth and active user accounts). Resource usage can be monitored, controlled and reported, providing transparency for the provider and consumer.

    WE dont have this

Multi-user Multi-tenancy



Different companies and users are using the same cloud provided instance of our application. Multi-tenancy/user is accomplished with our database model. With this approach we achieved lots of resource sharing, but little isolation. One collection is used for all tenants, whereby each tenant in stored in a separate document, thereby isolating the tenants from one another which is the first step in tenant isolation. A sketch of the database model can be seen above.

Since all tenants share the same database resource, ensuring that no data is being leaked is a priority. Within the collection, each tenant has its own document, basically its own name-space per say. And within each tenant's document, the tenant's jobs and employees and other fields are stored, making this information only visible for said tenant.

Another step taken to ensure that no information is being wrongfully leaked to a tenant or an employee, the application ensures that each request made from a tenant entails the respective tenant's name as well as requests made by employees contain the employee's email together with his/her associated tenant's name.

Describe why your application is cloud native, does it implement the 12F?

The application implements the twelve cloud native factors. **(1)** The application's `Codebase` is hosted on GitHub

Operations

Adding a new tenant

Installing application on the cloud provider

DevOps approach

As a team of 2 developers, we utilized `GitHub` to manage the changes made to the source code as well as for collaboration. To keep track of what needs to be done within the project, we used GitHubs project boards so that the team has an overview of what needs to be done, what is currently in progress, what needs to be reviewed and what has already been done. **INSERT PICTURE??**.

Cost Calculation & Business Model

Tenant: Register, unregister to PMS

As a tenant I would like to register/unregister my company to PMS.

Acceptance Criteria:

1. Possibility to register my company to use PMS's services.
2. Possibility to unregister my company to no longer use PMS's services.

Tenant: Invite workers

As a Tenant I would like to invite workers to join the company's "worker group" and also have an overview of the invited workers status.

Acceptance Criteria:

1. Text field to enter email addresses
2. Invite button to send invitation to the entered email addresses
3. A view of all workers as well as their invitation status, accepted or request pending.

Tenant: Create job

As a Tenant I would like to be able to create a new job listing., in order to add the job to the system as well as for the employees to be able to apply for the new job.

Acceptance criteria:

1. Creating a job is done with the click of a button `Create Job`.
    1. The ability to add the jobs title, location, description, start & end time, number of workers required.
2. The ability to either `Save` or `Cancel` creating the job.
3. The newly created job will appear in the list of jobs.

Tenant: View and edit jobs

As a Tenant I would like to see a list of all jobs, in order to have an overview of my company's jobs.

Acceptance criteria:

1. A list view of all jobs.
2. The ability to edit a job's information by clicking on the respective edit icon

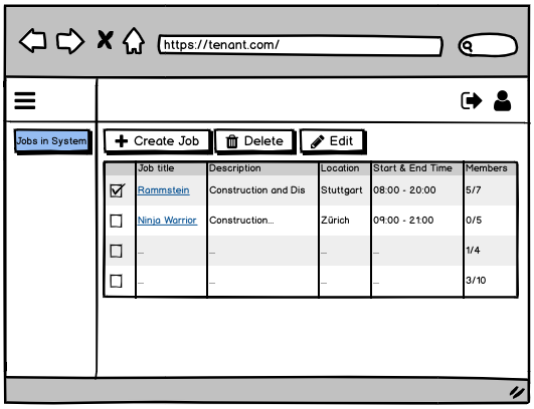
---

Tenant: Delete Job Listing

As a Tenant I would like to delete a job listing, so that employees can no longer apply for the job.

Acceptance criteria:

1. The ability to check mark a job and delete it.
2. The deleted job is no longer in the list of jobs.



Tenant: View and accept applicants

As a Tenant I would like to see which employees applied for which job, so that i can create a team to partake in the job.

Acceptance criteria:

1. Each row in the job listing gets a new field named `Members`
    1. Example the field will show `5/7` meaning 5 people applied from needed 7 workers.
2. When job is clicked, redirected to another page with the job's information and two lists with `applied` and `selected` workers.
3. The ability to move the workers between both `applied` and `selected` workers list.

Employee: Join Tenant's PMS service, edit profile

As an Employee I would like to accept my companies invite to use PMS's services, as well as to edit my profile information.

Acceptance criteria:

1. Fill out profile information with name, address, phone number, insurance number
2. View and edit profile information.

Employee: Available Job list

As an Employee I would like to see a list of all available jobs my employing company currently has and the ability to click on them to see all the information, in order to apply for jobs.

Acceptance criteria:

1. See a list of all available jobs under `Available jobs`.
2. The ability to click on the job's title and see the full job's information.
3. The ability to apply for a job with an `Apply` button.
    1. This job will no longer be shown under the `Available jobs` list.

Employee: Applied Job list

As an Employee I would like to see a list of all the jobs I applied to, in order to have a separate overview between all jobs and my applied jobs.

Acceptance criteria:

1. A view `Applied Jobs`
    1. This view contains all the jobs that were applied to in the `Available jobs` list section.
2. The ability to cancel a job application with a `cancel` button.

Employee: Accepted Job list

As an Employee I would like to see a list of all the jobs i got accepted to, in order to know my upcoming jobs.

Acceptance criteria:

1. A view `Upcoming Jobs`.
    1. View shows the upcoming jobs along with the jobs information.
2. The ability to `cancel` a job application with a `cancel` button.

##